

Specification and Analysis of Software Systems with Configurable Real-Time Behavior

Luthmann, Lars
(2020)

DOI (TUprints): <https://doi.org/10.25534/tuprints-00017363>

License:



CC-BY 4.0 International - Creative Commons, Attribution

Publication type: Ph.D. Thesis

Division: 18 Department of Electrical Engineering and Information Technology

Original source: <https://tuprints.ulb.tu-darmstadt.de/17363>



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Specification and Analysis of Software Systems with Configurable Real-Time Behavior

VOM FACHBEREICH ELEKTROTECHNIK UND INFORMATIONSTECHNIK
DER TECHNISCHEN UNIVERSITÄT DARMSTADT
ZUR ERLANGUNG DES AKADEMISCHEN GRADES
EINES DOKTOR-INGENIEURS (DR.-ING.)
GENEHMIGTE DISSERTATION

VON

LARS LUTHMANN

GEBOREN AM

21. JULI 1990 IN WOLFSBURG

REFERENT: PROF. DR. RER. NAT. ANDY SCHÜRR

1. KORREFERENT: PROF. MOHAMMAD REZA MOUSAVI, PH.D.

2. KORREFERENT: PROF. DR. RER. NAT. HABIL. MALTE LOCHAU

TAG DER EINREICHUNG: 2020-09-15

TAG DER DISPUTATION: 2020-12-02

DARMSTADT 2020

Luthmann, Lars

Specification and Analysis of Software Systems with Configurable Real-Time Behavior

Darmstadt, Technische Universität Darmstadt

Year of publication at TUpriints: 2020

URN: urn:nbn:de:tuda-tuprints-173639

URL: <https://tuprints.ulb.tu-darmstadt.de/id/eprint/17363>

Disputation date: 2020-12-02

Published under CC BY 4.0 International

<https://creativecommons.org/licenses/>

©2020, Lars Luthmann

ABSTRACT

Nowadays, non-functional properties and configurability are crucial aspects in the development of (safety-critical) software systems as software is often built in families and has to obey real-time requirements. For instance, industrial plants in Industry 4.0 applications rely on real-time restrictions to ensure an uninterrupted production workflow. Modeling these systems can be done based on well-known formalisms such as *timed automata (TA)*. In terms of configurability, *software product line engineering (SPLE)* is used for developing variant-rich systems by integrating similar behavior into a product-line representation. In SPLE, we map core behavior and variable behavior to Boolean *features* representing high-level customization options, thus facilitating traceability between configuration models and behavioral models. However, only few formalisms combine real-time behavior with configurability. In particular, *featured timed automata (FTA)* support Boolean variability, whereas *parametric timed automata (PTA)* instead utilize numeric *parameters*, allowing us to describe infinitely many variants. Here, PTA facilitate an increased expressiveness as compared to FTA by using a-priori unbounded time intervals.

Unfortunately, there does not exist a formalism for real-time SPLs supporting traceability of Boolean features *and* infinitely many variants being available through parameters. Hence, we introduce *configurable parametric timed automata (CoPTA)*, combining the advantages of Boolean features and numeric parameters. Therewith, we are able to model SPLs comprising an infinite number of variants while supporting traceability between configuration model and behavioral model.

For analyzing real-time properties of CoPTA, we cannot directly apply product-based approaches anymore due to the (possibly) infinite number of products. Hence, we develop quality-assurance techniques for CoPTA models. Here, *sampling* (i.e., the derivation of a subset of variants) still allows us to perform product-based analyses even in case of infinitely many products. To this end, we introduce a strategy specifically tailored to boundary cases of time-critical behavior.

Moreover, we introduce *family-based* techniques for quality assurance of CoPTA. For black-box analysis (where the behavioral model is unavailable), there already exist approaches for systematically reusing test cases among different configurations by accumulating configuration-specific information. However, these approaches only consider features, whereas we enhance these approaches by also considering parameters, allowing us to derive *complete* finite test suites satisfying product-based coverage criteria even in case of infinitely many variants. Additionally, our framework for test-case generation also covers boundary cases in terms of time-critical behavior. In case of white-box analysis, we introduce a formalism for a *decidable* check of timed bisimilarity, and we lift timed bisimulation to CoPTA.

We illustrate the concepts presented in this thesis by using a bench-scale demonstrator of an industrial plant as an example, and we evaluate our approaches based on a prototypical implementation, revealing efficiency improvements (in cases where we can compare our approach to other approaches) and applicability.

KURZFASSUNG

Nichtfunktionale Eigenschaften und Konfigurierbarkeit sind wesentliche Aspekte bei der Software-Entwicklung, da Software häufig familienbasiert entwickelt wird und Echtzeitanforderungen hat. Beispielsweise haben Anwendungen im Bereich Industrie 4.0 Echtzeitbeschränkungen, um sicherzustellen, dass der Produktionsablauf nicht verzögert wird. Diese Systeme können dabei auf Basis bekannter Formalismen, wie z. B. *Timed Automata (TA)*, modelliert werden. Im Bereich Konfigurierbarkeit ist *Software Product Line Engineering (SPLE)* ein wichtiger Aspekt für die Entwicklung variantenreicher Systeme, wobei ähnliches Verhalten in einer Produktlinie zusammengefasst wird. Im SPLE wird Kern- und variables Verhalten auf boolesche *Features* abgebildet, die Anpassungsoptionen darstellen und Artefakte aus Konfigurations- und Verhaltensmodell verknüpfen. Jedoch kombinieren nur wenige Formalismen beide Aspekte. Zum Beispiel unterstützen *Featured Timed Automata (FTA)* boolesche Variabilität, während *Parametric Timed Automata (PTA)* numerische Parameter verwenden, mit denen eine unendliche Anzahl von Varianten beschrieben werden kann. Hier bieten PTA durch die Nutzung von a priori unbegrenzten Zeitintervallen eine höhere Ausdrucksmächtigkeit als FTA.

Bisher gibt es kein Modell für Echtzeitsysteme, das die Vorteile boolescher Variabilität und numerischer Parameter unterstützt. Daher führen wir *Configurable Parametric Timed Automata (CoPTA)* ein, die die Vorteile beider Ansätze kombinieren. Damit können wir Systeme modellieren, die unendlich viele Varianten umfassen und Artefakte aus dem Konfigurations- und Verhaltensmodell verknüpfen.

Bei Nutzung von CoPTA können wir aufgrund der (potentiell) unendlichen Anzahl von Produkten keine produktbasierte Analyse mehr direkt anwenden, weshalb wir außerdem Techniken zur Qualitätssicherung von CoPTA vorstellen. Produktbasierte Ansätze können allerdings weiterhin unter Verwendung von *Sampling* durchgeführt werden. Jedoch existieren bisher keine Sampling-Strategien für unendliche Konfigurationsräume und Echtzeit-Produktlinien. Daher stellen wir eine Strategie vor, die das Extremverhalten von Echtzeitsystemen berücksichtigt.

Des Weiteren stellen wir *familienbasierte* Techniken zur Qualitätssicherung von CoPTA vor. Im Bereich der Black-Box-Analyse (bei der das Verhaltensmodell nicht verfügbar ist) gibt es für das Testen bereits Ansätze zur Wiederverwendung von Testfällen zwischen Konfigurationen, welche jedoch nur boolesche Variabilität berücksichtigen. Deshalb erweitern wir diese Ansätze um Parameter, wodurch wir für CoPTA mit unendlich vielen Varianten *vollständige* Test-Suites ableiten können. Darüber hinaus deckt unsere Testfallgenerierung auch zeitkritisches Extremverhalten ab. Bei der White-Box-Analyse führen wir eine *entscheidbare* Überprüfung von TA-Bisimilarität ein und definieren Bisimulation für CoPTA.

Wir veranschaulichen die in dieser Thesis vorgestellten Konzepte am Beispiel eines Demonstrators einer Industrieanlage und evaluieren unsere Ansätze anhand einer prototypischen Implementierung. Unsere Evaluationen zeigen dabei eine verbesserte Effizienz und generelle Anwendbarkeit unserer Ansätze.

CONTENTS

1	INTRODUCTION	1
1.1	Challenges and Contributions	3
1.2	Outline	8
2	BACKGROUND AND MOTIVATION	9
2.1	Motivating Example	9
2.2	Modeling Real-Time Systems	11
2.3	Modeling Product Lines of Real-Time Systems	15
2.3.1	Software Product Lines	16
2.3.2	Featured Timed Automata	19
2.3.3	Parametric Timed Automata	22
2.4	Sampling Product Lines of Real-Time Systems	25
2.5	Semantics and Analysis of Real-Time Systems	30
2.5.1	Semantics of Timed Automata	31
2.5.2	Semantics of Featured Timed Automata	37
2.5.3	Semantics of Parametric Timed Automata	38
2.5.4	Testing Real-Time Systems	38
2.6	Bisimulation of Real-Time Systems	40
3	MODELING PRODUCT LINES WITH PARAMETRIC REAL-TIME CONSTRAINTS	47
3.1	Extended Feature Models	50
3.2	Configurable Parametric Timed Automata	54
3.3	Transformation of CoPTA into Extended PTA	58
3.4	Related Work	63
3.5	Conclusion and Future Work	65
4	SAMPLING PRODUCT LINES WITH INFINITE CONFIGURATION SPACES	67
4.1	Sampling Product Lines with Infinite Configuration Spaces	70
4.1.1	Black-Box Sampling Strategies for Infinite Configuration Spaces	70
4.1.2	White-Box Sampling Strategies for Infinite Configuration Spaces	74
4.2	Semantics of Product Lines with Parametric Real-Time Constraints	79
4.3	Sampling Product Lines with Parametric Real-Time Constraints	83
4.3.1	Best-Case/Worst-Case Execution Times for TA Locations	85
4.3.2	Minimum/Maximum Delay Coverage	86
4.3.3	Minimum/Maximum Delay Sampling	89
4.4	Experimental Evaluation	94
4.5	Related Work	100
4.6	Conclusion and Future Work	101
5	TESTING PRODUCT LINES WITH PARAMETRIC REAL-TIME CONSTRAINTS	103
5.1	Test-Case Generation from Timed Automata	105
5.1.1	Test Cases for Timed Automata	105
5.1.2	Location Coverage for Timed Automata	108
5.1.3	Generating Test Suites for Complete Location Coverage	110
5.2	Family-based Test-Case Generation from CoPTA	113
5.2.1	Variability-aware Test Cases for CoPTA	115

5.2.2	Family-based Location Coverage for CoPTA	120
5.2.3	Generating Test Suites for Family-based Location Coverage	122
5.3	Family-based Test Coverage of BCET/WCET Behavior	128
5.3.1	M/MD Test Cases for CoPTA	129
5.3.2	M/MD Coverage for CoPTA	130
5.3.3	Generating Test Suites for Complete M/MD Coverage	133
5.4	Experimental Evaluation	137
5.5	Related Work	143
5.6	Conclusion and Future Work	145
6	BISIMILARITY OF PRODUCT LINES WITH PARAMETRIC REAL-TIME CON- STRAINTS	149
6.1	Checking Bisimulation for Timed Automata	152
6.1.1	Modeling the History of Real-Time Behavior	154
6.1.2	Checking Timed Bisimilarity of Deterministic TA	157
6.1.3	Checking Timed Bisimilarity of Non-Deterministic TA	168
6.1.4	Bounded Checking of Timed Bisimilarity	175
6.2	Checking Bisimulation for Parametric Timed Automata	178
6.2.1	Parametric Timed Bisimulation	180
6.2.2	Parameter-Abstracted Timed Bisimulation	186
6.3	Checking Bisimulation for Configurable Parametric Timed Automata	188
6.4	Experimental Evaluation	191
6.5	Related Work	198
6.6	Conclusion and Future Work	199
7	CONCLUSION AND FUTURE WORK	201
7.1	Summary	201
7.2	Future Work	204

ACRONYMS

BCET	B est-Case Execution Time
CA	C overing Array
CIT	C ombinatorial Interaction Testing
CoPTA	C onfigurable P arametric Timed Automaton
EFM	E xtended Feature Model
FM	F eature Model
FODA	F eature-Oriented Domain Analysis
FTA	F eatured Timed Automaton
LTS	L abeled Transition System
M/MD	M inimum/ M aximum Delay
PLT	P arametric Linear Term
PEPTA	P arameter-Extended P arametric Timed Automaton
PTA	P arametric Timed Automaton
PPU	P ick and P lace Unit
SPL	S oftware Product Line
SPLE	S oftware Product Line Engineering
TA	T imed Automaton
TLTS	T imed Labeled Transition Systems
WCET	W orst-Case Execution Timed
xPPU	extended P ick and P lace Unit
ZHG	Z one- H istory G raph

INTRODUCTION

Nowadays, non-functional properties and configurability are crucial aspects in the development of (safety-critical) software systems as software is often built in families and has to obey real-time requirements. For instance, many industrial plants in Industry 4.0 applications rely on real-time properties (e.g., for moving workpieces) to ensure that the production is not delayed or even interrupted [122, 197]. Furthermore, machines not satisfying real-time requirements might result in damaged products. In addition to non-functional properties, also highly configurable systems are an essential aspect in modern software engineering. For instance, some software systems of industrial plants or even whole industrial plants can be tailored specifically to the needs of customers [88, 142, 84, 127]. Besides these examples, also other application areas, such as automotive software and medical devices, adopt the ideas of variant-rich systems [202] and need to obey real-time requirements. In the following, we start by giving an overview on the research areas covered in this thesis, followed by a discussion about the challenges that we tackle (see Section 1.1).

An important concept for developing and handling these variant-rich systems is so-called *software product line engineering (SPLE)* [75, 163]. The goal of SPLE consists of integrating similar (software) systems into a product-line representation. Such a *software product line* consists of two parts. First, the core behavior describes the behavior being present in *every* variant of the product line. For instance, a product line for industrial plants might contain a crane, which can move a workpiece, in every variant. Second, variable behavior describes the behavior being present in *some* variants of the product line. For example, the crane of the industrial plant might have different real-time requirements for moving the workpieces, depending on the material of the workpiece.

Both the core behavior as well as the variable behavior of an SPL are encapsulated in so-called *features*. In particular, a feature is “a prominent or distinctive aspect, quality, or characteristic of a software system or systems” [108]. Hence, features represent high-level customization options such that customers can tailor product lines to their specific needs, and every feature is mapped to given artifacts. Then, a specific product of a product line can be derived by selecting the desired features and composing the implementation artifacts mapped to these features. For instance, a customer might select an industrial plant handling workpieces made of plastic. In this case, we select the core feature (which contains a crane) and the feature for plastic (which has an end-effector that is suitable for grabbing plastic pieces). As plastic workpieces are relatively light (as, e.g., compared to metal), the workpieces can be moved around fast.

Next, we want to analyze a product line in terms of quality assurance, where we eliminate possible faults by utilizing test-case generation and verification (e.g., equivalence checking w.r.t. a specification). A first naive approach for applying these analysis techniques could be deriving all possible variants and then perform quality assurance for the product line in a variant-by-variant way with existing techniques. However, this approach is usually infeasible due to the large number of configurations [53]. A more sophisticated strategy for analyzing a product line consists of deriving a representative subset of variants, such that we only analyze these variants. Thereafter, we can generalize the analysis results of the variants for the product line. This approach is called *sampling*.

One of the challenges of utilizing sampling for quality assurance of a product line is to find a strategy resulting in a representative, yet sufficiently small set of variants. For instance, a sample containing almost all of the variants might be representative but cannot be analyzed anymore (due to the large number of variants) while a sample only containing a single variant is small but most likely not representative. One mature sampling strategy for finding representative variants is (pairwise) *combinatorial interaction testing (CIT)* [150]. Here, the idea is to cover every (pairwise) combination of all features. Additionally, CIT requires covering every combination of selection and deselection for each pair of features. For instance, assume that the features *metal* and *plastic* are two features of our industrial-plant SPL. For pairwise CIT, our sample then needs to include a product where both features are selected, both features are deselected, metal is selected, and plastic is selected (if these combinations are valid w.r.t. the SPL). It should be noted that these four products may also cover other combinations of further pairs of features, such that this strategy can be used to significantly reduce the number of variants [150]. Having derived the variants with CIT (or other sampling strategies), we then analyze only these variants (with variant-based techniques) for quality assurance as representatives of the SPL.

In this thesis, we consider black-box and white-box analysis techniques, where the behavioral model is unavailable in case of black-box analysis. One of the critical analysis tasks in software engineering is (black-box) *testing*, describing “the process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or component” [1]. In particular, we consider *model-based testing* as an approach for systematic automated coverage-based test-case generation, test execution, and evaluation of test results based on formal modeling languages [190]. A *test case* (for a single variant) imitates behavior of the environment of the *implementation under test (IUT)* in terms of sequences of actions (or test inputs) injected into the IUT [190]. We then compare the reactions (or test outputs) of the (black-box) IUT to the expected reactions. In model-based testing, these test cases are systematically derived from our model (e.g., *timed automata*) until every *test goal* described by a *coverage criterion* (e.g., every state) is reached by at least one test case. As we consider time-critical systems, we also need to specify allowed time intervals in our test cases and record the amount of time used for actions during test execution. For instance, a test case for the industrial plant mentioned above might be concerned with the tasks of the crane. To this end, a test case may describe functional behavior

in terms of picking up a workpiece, moving the crane, and placing the workpiece. Furthermore, this test case contains information about non-functional behavior in terms of the allowed timeframe for each action.

When evaluating product lines (in terms of quality assurance) by utilizing testing, we can apply sampling as described above and provide results for the product line by testing the variants contained in the sample. A different goal might be that we want to analyze all variants that we produce for customers, such that we can be sure that all products actually being in use have been tested. In this scenario, we could, again, generate test cases in a variant-by-variant approach (for the products in use). However, if we have a product line being deployed for many customers, we have to generate test cases for common behavior over and over again, which might be infeasible due to the large number of configurations. In this case, we can instead use the product-line representation of our software and generate test cases based on this representation [53]. In particular, we record all features being traversed by a test case. Then, the test case is valid for all configurations satisfying the recorded feature selection. As a result, we can generate a complete test suite based on the product-line representation and then derive test suites for products being used by customers. For instance, a test case might be valid for configurations of the industrial plant where workpieces made of plastic or metal are moved. Here, the test case covers behavior of (at least) two variants.

A further issue of model-driven software engineering that we are concerned with in this thesis is so-called (*timed*) *bisimulation*, constituting a white-box analysis technique. For instance, this technique may be applied for verification of (real-time) systems. In model-driven software engineering, there usually is an initial model to describe the behavior of the specification. During the development process, this model is further and further refined. Throughout this process, we have to ensure that the (time-critical) behavior of our model remains the same as in the original model, such that we do not accidentally introduce faults or unwanted behavior. Hence, we compare the resulting model of each development step with the original model by utilizing (timed) bisimulation, where we compare the behavior of the adapted model with the original model step by step. Moreover, we can check for *similarity* in settings where the task is to ensure that the adapted model contains *at least* or *at most* the behavior of the original model (instead of exactly the same behavior in case of bisimilarity).

Next, we motivate open challenges in the research areas introduced above, and we give an overview on the contributions present in this thesis.

1.1 CHALLENGES AND CONTRIBUTIONS

In this thesis, we improve the state of the art in model-driven software engineering of product lines with real-time behavior in several aspects.

MODELING So far, there exist two formalisms for modeling variant-rich systems having real-time behavior, namely *featured timed automata (FTA)* [70] and *parametric timed automata (PTA)* [15]. FTA (as depicted in the top row of Figure 1.1) utilize a feature model relying on Boolean features as a *configuration model*. Here, the

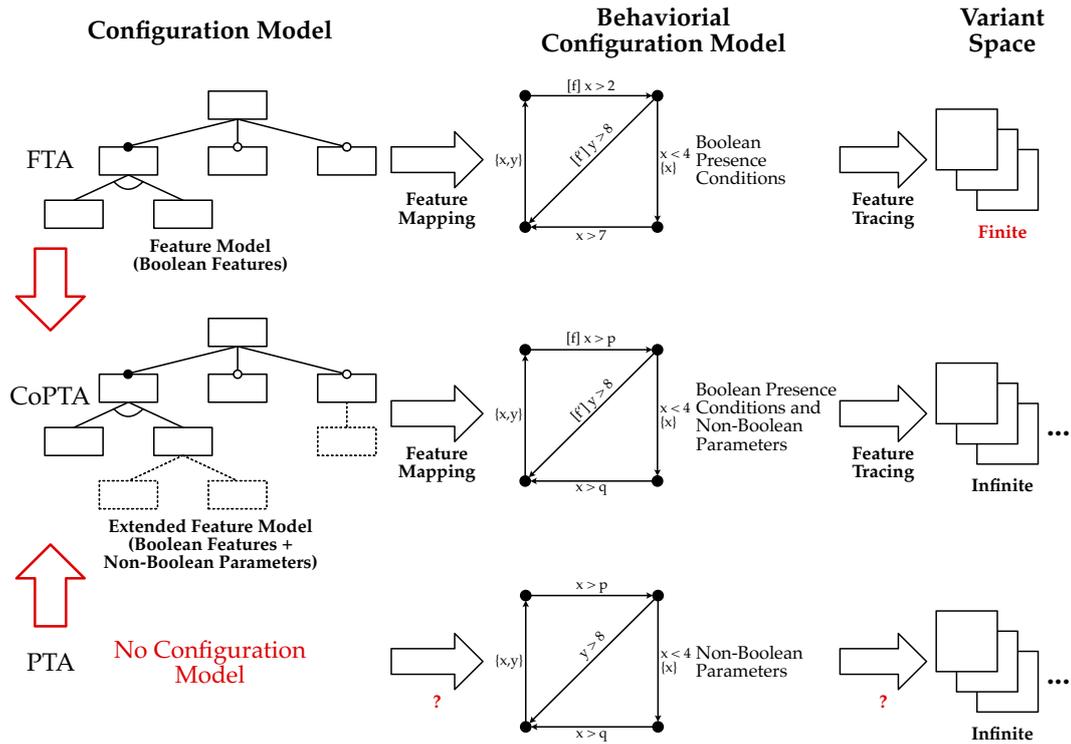


Figure 1.1: Schematic Overview on the Contributions for Modeling Software Systems with Configurable Real-Time Behavior

configuration model describes the *problem space*. These features can be used to annotate parts (e.g., real-time constraints) of the *behavioral configuration model* (i.e., the automaton itself) with Boolean presence conditions, where the behavioral configuration model is part of the *solution space*. For obtaining a variant in the *variant space* (also being part of the solution space), we then select a set of features such that exactly the corresponding parts of the model are included in our variant. Hence, by using this feature mapping, FTA facilitate traceability between artifacts in the problem space and the solution space. In contrast, PTA (as depicted in the bottom row of Figure 1.1) do not have a configuration model. Instead, PTA utilize variable parameters as bounds for real-time constraints (in the behavioral configuration model) such that we derive variants by replacing these a-priori unbounded numeric parameters by constant values. Hence, PTA have an increased expressiveness as compared to FTA by allowing the usage of unbounded time intervals.

However, both modeling approaches have weaknesses (highlighted with red question marks in Figure 1.1) which we tackle in this thesis. On the one hand, FTA do not support numeric parameters, such that we can only specify models with a finite number of variants in the variant space. Hence, it is not possible to have a model where some time constraints may be set to very specific values chosen by engineers for each configuration individually. For instance, a user might want to fine-tune clock constraints for moving workpieces in the industrial plant according to very specific aspects of the plant (e.g., age of the plant, size of the crane, and size of the workpieces). On the other hand, PTA do not facilitate traceability between

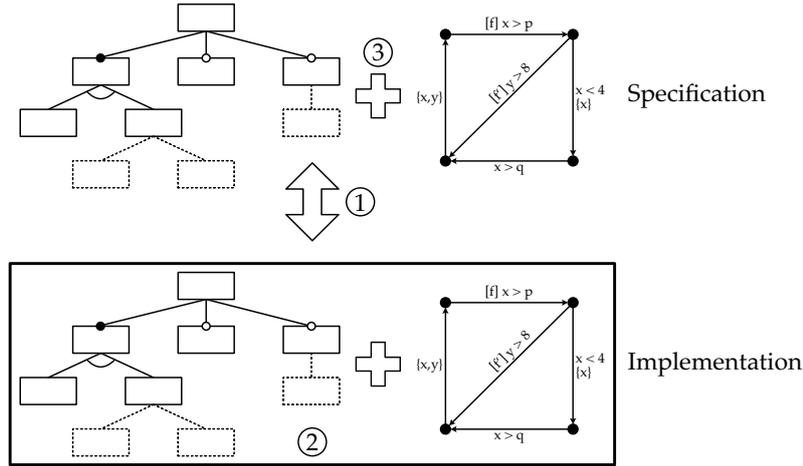


Figure 1.2: Schematic Overview on the Contributions for Analysis of CoPTA

artifacts in the problem space and the solution space as there does not exist a configuration model.

Hence, in this thesis, we improve the state of the art of modeling variant-rich systems having real-time behavior by combining the advantages of FTA and PTA (middle row in Figure 1.1). In particular, we introduce a novel formalism enriching FTA by configurable parametric variability, which we call *configurable parametric timed automata (CoPTA)*. To this end, we extend the feature model as used for FTA by constraints over numeric parameters. Furthermore, we also use these parameters in our behavioral configuration model. Therewith, we obtain a model that supports traceability between the problem space and the solution space by utilizing Boolean features while also supporting unbounded numeric variability of parameters (resulting in a possibly infinite number of variants). However, analyzing CoPTA causes the additional challenge that we have to solve inequations over numeric parameters (in addition to solving Boolean formulae as done for FTA).

SPL ANALYSIS Based on the CoPTA representation of product lines with configurable parametric real-time constraints, we introduce several analysis methods throughout this thesis, where the numbers correspond to the numbers in Figure 1.2. These methods can be structured in the following categories (where we discuss the different instantiations afterwards in more detail). In particular, we consider different (1) analysis techniques (indicated by the arrow in Figure 1.2), (2) analysis settings (indicated by the box around the implementation), and (3) analysis strategies (indicated by the plus sign between feature model and CoPTA).

1. **Analysis techniques:** In this thesis, we consider two different analysis techniques. In particular, we utilize (a) model-based test-case generation for testing allowed behavior w.r.t. a specification and (b) timed bisimulation for checking behavioral equivalence.
2. **Analysis settings:** For our analysis approaches, we consider (a) black-box settings as well as (b) white-box settings.

3. **Analysis strategies:** In terms of strategies, we may consider (a) product-by-product approaches for behavioral analysis, (b) product-sampling approaches, and (c) family-based approaches for behavioral analysis.

This results in an overall number of twelve combinations, when we pick one approach of each category for each combination. For instance, we might choose test-case generation as an analysis technique and the black-box setting, such that we can combine this selection with the family-based analysis strategy. As a result, we obtain the combination of family-based test-case generation in case of a black-box setting. However, not all of these combinations are applicable in case of CoPTA. In particular, all combinations using a product-by-product strategy are inapplicable as CoPTA comprise, in general, an infinite number of variants. Furthermore, timed bisimulation is not applicable in a black-box setting as checking bisimilarity involves comparing the behavior of states of a system. Hence, there are eight remaining combinations which we consider in this thesis. In the following, we give an overview on our contributions in terms of analysis approaches, where the structure follows the structure of this thesis.

SAMPLING Utilizing CoPTA facilitates precise formalization of analysis techniques for configurable real-time software systems. However, we cannot apply product-based approaches anymore due to the (possibly) infinite number of products. Nonetheless, product-based analyses can be performed indirectly by first applying sampling, describing the derivation of a representative subset of variants of an SPL. Unfortunately, CIT techniques for sampling, as described before, only cover finite configuration spaces. Hence, these techniques are not applicable to *infinite* configuration spaces as applying CIT would result in an infinite number of configurations. However, we consider *infinite* configuration spaces in case of PTA and CoPTA (see above), both utilizing (possibly) unbounded numeric parameters. Hence, we introduce a sampling approach specifically tailored to product lines having configurable real-time behavior. In particular, we consider boundary behavior of real-time systems in terms of best-case execution times and worst-case execution times (BCET/WCET), respectively [203]. This idea stems from the area of boundary-value testing, having the goal to cover boundary values for each variable [165, 81]. The rationale behind this is the fact that problems often occur at these boundary values. For instance, this approach may reveal off-by-one timing errors [6], where boundary-value behavior w.r.t. delays is faulty. Therefore, we have the goal to adapt sampling strategies to product lines with a (potentially) infinite number of configurations in general, and we instantiate these product lines with CoPTA (i.e., our novel modeling formalism as described above). Note, that our sampling approach is sound (i.e., the generated sample covers BCET/WCET behavior) but incomplete (i.e., the algorithm for generating the sample does, in general, not terminate). In addition to sampling, we also introduce *family-based* (black-box and white-box) analysis techniques which we directly apply on CoPTA as explained in the following.

TEST-CASE GENERATION Considering (black-box) test-case generation in the context of product lines, the goal is to systematically reuse test cases among

different configurations by symbolically accumulating configuration-specific information during test-case generation [53]. More specifically, each test case should be accompanied by a *presence condition* [58] (symbolically) describing the set of configurations for which the test case is applicable. Therewith, we can describe complete test suites for feature-based software product lines having real-time behavior. In particular, the goal is to cover each test goal in every variant without deriving all variants.

As described above, in this thesis we introduce a novel modeling formalism enriching FTA by configurable parametric variability (i.e., CoPTA), potentially having an infinite number of variants. Therefore, we adapt an approach for family-based test-case generation [53] to CoPTA. Therewith, we are able to generate a *complete* and *finite* test suite even in case of CoPTA having infinitely many variants. We achieve this by deriving test cases directly from CoPTA. To this end, we adapt the approach of Bürdek et al. [53] who generate test cases for product lines written in C code. As compared to Bürdek et al. who only support Boolean parameters (i.e., features), our approach additionally supports unbounded numeric parameters.

With the approach for test-case generation as described above, we cover each test goal in every variant without having any further requirements for test cases. However, this might result in critical behavior not being tested. For instance, off-by-one timing errors [6], as mentioned for sampling, may be missed. Therefore, we extend our methodology for test-case generation by a novel coverage criterion for BCET/WCET behavior (i.e., similar to the criterion we want to apply for sampling). Therewith, we may investigate time-critical behavior more effectively by requiring two valid test cases (i.e., test cases being applicable to at least one configuration) for each part of our model, one for each the minimum and the maximum delay w.r.t. the given real-time constraints [74]. Similar to our sampling approach, both methods for test-case generation are sound (i.e., we always generate valid test cases) but incomplete (i.e., it might be the case that the test-case generation does not terminate).

TIMED BISIMULATION In terms of white-box analysis techniques, we improve the state of the art in checking timed bisimilarity (and similarity). Here, a basic model for checking bisimilarity (for two given TA) are so-called *Timed Labeled Transition Systems (TLTS)* [99, 100]. Unfortunately, TLTS are, in general, not finite. Therefore, we introduce a different formalism in this thesis for *effectively* checking timed bisimilarity. Therewith, we may apply timed bisimulation to model-driven development such that refined models can be checked against the original model, ensuring that the observable behavior remains unchanged.

It should be noted that there already exist a few approaches for effectively checking TA bisimilarity. However, all of these approaches have some disadvantages as compared to the approach that we introduce in this thesis. For instance, Čerāns [56] defines timed bisimilarity based on region graphs which suffer from state-space explosion, making the approach generally less efficient than approaches based on zone graphs. Furthermore, the approaches by Weise and Lenzkes [201] and Guha et al. [91, 92] are not defined in a way that would directly allow a generalization for checking timed bisimilarity of PTA and CoPTA.

Furthermore, we also consider timed bisimilarity for *product lines* of TA. Here, we are concerned with the model-driven development process of families of similar systems. Specifically, we want to ensure that the behavior of *each variant* remains unchanged without deriving these variants and checking them in a variant-by-variant fashion. Hence, we lift the concepts of TA bisimulation to product lines with configurable parametric real-time constraints. However, it should be noted that timed bisimulation for PTA (and thus also for CoPTA) is, in general, semi-decidable (i.e., the result of the check is correct but the check might not terminate) as already reachability is semi-decidable for PTA [20, 21].

1.2 OUTLINE

The remainder of this thesis is structured as follows. In Chapter 2, we give an overview on the fundamental principles being utilized as a basis throughout the thesis. Furthermore, we present an illustrating example which we use to motivate and explain the various concepts of this thesis. In Chapter 3, we introduce a model for describing product lines with configurable real-time behavior (possibly) having an infinite number of variants, namely CoPTA. Thereupon, we present an approach from sampling product lines having infinite configuration spaces in Chapter 4. In Chapter 5, we introduce a methodology for deriving finite complete test suites from CoPTA, and we extend this methodology by an approach for generating test cases exposing BCET/WCET behavior. In Chapter 6, we introduce a formalism for effectively checking timed bisimilarity, and we lift the notion of timed bisimilarity to PTA and CoPTA. Finally, we conclude the thesis and give an outlook on possible future work in Chapter 7.

2

BACKGROUND AND MOTIVATION

In this chapter, we motivate and introduce the challenges tackled throughout this thesis. In order to illustrate these challenges, we start with a case study from the automation-engineering domain being used in the following as a running example (see Section 2.1). Afterwards, we present preliminaries on basic concepts and notations for modeling discrete-state/continuous-time behavior (see Section 2.2). Thereafter, we give an introduction to different formalisms for modeling software product lines of real-time systems (see Section 2.3). Here, a single model comprises a family of systems having similar discrete-state/continuous-time behavior. Additionally, we give background information on three different techniques for analyzing these time-critical product-line models. First, we establish the notion of sampling where we derive a subset of variants from a product line (see Section 2.4). As a result, we may apply techniques for analyzing individual systems instead of product lines. Second, we give an introduction to model-based coverage-driven test-case generation for real-time systems (see Section 2.5). Third, we are concerned with verification. More specifically, we present preliminaries for the concept of bisimulation where we want to check for two (real-time) systems whether they behave equivalently. In each section of this chapter, we discuss open research challenges for the respective topics subsequently tackled in the main part of this thesis.

2.1 MOTIVATING EXAMPLE

As a running example, we use an extract of the so-called *Pick and Place Unit (PPU)* [122, 196] throughout this thesis. The PPU is a bench-scale demonstrator of an industrial plant and a case study built at the Institute of Automation and Information Systems of the TU München. This demonstrator models a recycling working cell of a fictitious company, transporting tanks between two working positions. Here, tanks are loaded at the first position and transported by a crane to the second position (where recycling takes place). In order to model this plant, the PPU consists of several components for which Figure 2.1 gives an overview. Here, the circled numbers in the figure correspond to the numbers of the following enumeration.

1. The *stack* provides workpieces (representing tanks) and functions as the loading position.
2. The *crane* picks the workpieces provided by the stack, rotates to the ramp, and places the workpieces at the ramp.

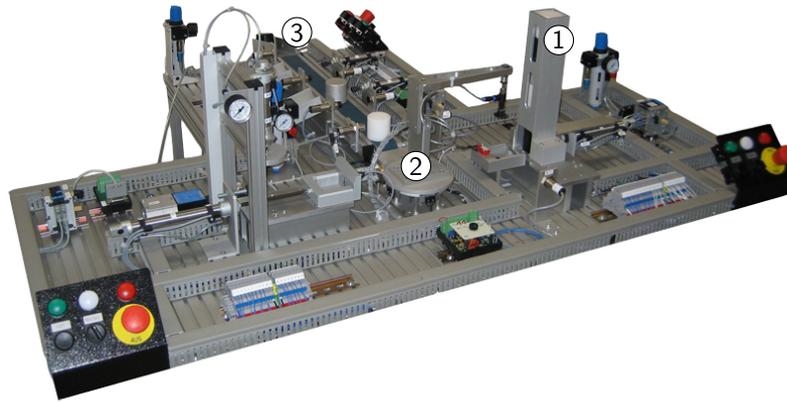


Figure 2.1: Pick and Place Unit¹

3. The *ramp* is used to store the workpieces, representing recycling of the tanks.

Besides functional requirements for the correct processing of workpieces (e.g., the crane may place workpieces only at the ramp), the PPU contains several real-time constraints to ensure safety and maintain a certain level of productivity.

- The allowed time frame for rotating the crane has a lower and an upper bound. Here, the lower bound corresponds to the maximum speed such that the plant is not damaged, and the upper bound limits the maximum idle time for other parts of the PPU.
- The allowed time frames for picking and placing workpieces also have lower and upper bounds to, again, avoid damage and limit the idle time of the demonstrator.
- The PPU contains a safety mechanism to shut down the plant in case of an emergency. When triggered, the emergency systems needs to safely stop the plant within a specific amount of time to avoid further damage.

The initial variant of the PPU is only able to transport workpieces made of black plastic from the stack to the ramp. However, the PPU was improved several times, where some of the improvements will be explained in the following. First, other types of workpieces made of white plastic and metal should be supported. As a result, pressure profiles are added to prevent the more fragile white plastic from breaking apart. Moreover, the ramp may be replaced by a conveyor and three ramps to sort different workpieces by their types. Here, an additional adjustment also makes it possible to mix the sorting of the different types of workpieces on the ramps. Finally, the crane is augmented with a potentiometer, resulting in improved accuracy of crane motions. Here, Figure 2.2 depicts a schematic view on the PPU after having implemented the above-mentioned changes. Again, the circled numbers correspond to stack (1), crane (2), and the original ramp (3).

Furthermore, the *extended Pick and Place Unit (xPPU)* [197] is an extension of the PPU having several improvements in terms of measuring the weight of workpieces,

¹ The author would like to thank Prof. Vogel-Heuser of the Institute of Automation and Information Systems of the TU München for providing this figure. Please note, that the copyright for this figure remains with Prof. Vogel-Heuser.

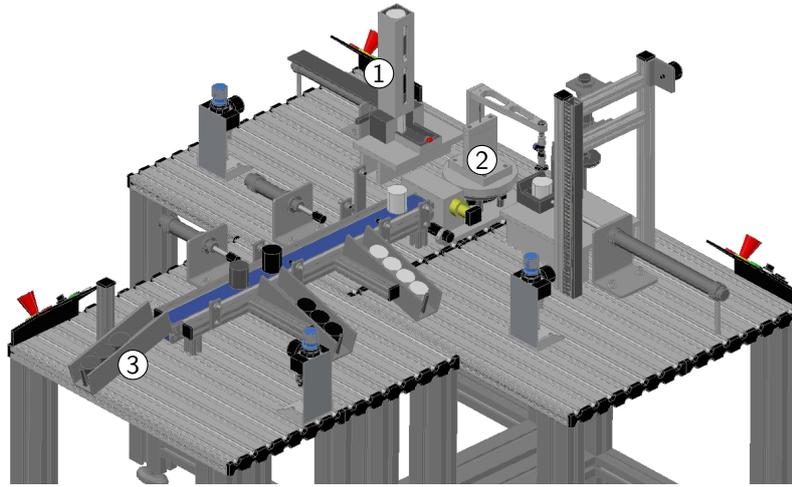


Figure 2.2: Schematic View on the Pick and Place Unit²

a system of conveyors to transport workpieces back to the stack, a safety door to prevent accidents involving humans, measuring energy consumption, and several more additions. However, in this thesis we only consider (an extract of) the original PPU as described above to illustrate the different concepts.

2.2 MODELING REAL-TIME SYSTEMS

In this thesis, we utilize *Timed Automata (TA)* [9, 198] to model the behavior of real-time systems. TA constitute a frequently used formalism to specify discrete-state/continuous-time behavior of time-critical systems. Each TA consists of a finite state-transition graph with states called *locations* and edges called *switches*. Here, switches are labeled by symbols from a finite alphabet Σ of actions. Furthermore, a TA contains a finite set of *clocks* C defined over a numerical *clock domain* \mathbb{T}_C . For instance, we may use $\mathbb{T}_C = \mathbb{R}_+ = \{r \mid r \in \mathbb{R} \wedge r \geq 0\}$ to model *dense-time* behavior. Throughout this thesis we will use $\mathbb{T}_C = \mathbb{N}_0$, modeling *discrete-time* behavior, in all illustrating examples. Formally, we have the following requirements for clock domain \mathbb{T}_C .

- There is an associative operator for addition with neutral element 0.
- There is an operator for subtraction with neutral element 0.
- Elements in \mathbb{T}_C are totally ordered.

Clocks are constantly and synchronously increasing variables over \mathbb{T}_C where each clock c of a set of clocks C may be reset independently. Moreover, we utilize clocks to measure and restrict time intervals corresponding to delays between action occurrences with so-called *clock constraints* φ . A clock constraint comprises a conjunction of inequalities where each inequality compares a clock with a constant

² The author would like to thank Prof. Vogel-Heuser of the Institute of Automation and Information Systems of the TU München for providing this figure. Please note, that the copyright for this figure remains with Prof. Vogel-Heuser.

value of time domain \mathbb{T}_C . Hence, a clock constraint φ is fulfilled if all valuations of all clocks satisfy φ .

Definition 2.1 (Clock Constraint). Let C be a set of clocks defined over clock domain \mathbb{T}_C . The set $\mathcal{B}(C)$ of *clock constraints* φ over C is inductively defined as

$$\varphi := \text{true} \mid c \sim n \mid c - c' \sim n \mid \varphi \wedge \varphi$$

where $\sim \in \{<, \leq, \geq, >\}$, $\{c, c'\} \subseteq C$, and $n \in \mathbb{T}_C$.

In Definition 2.1, we decided to include `true` to explicitly have an annotation for behavior that is allowed at any point in time (although `true` can be expressed by the given grammar). Furthermore, we excluded `false` as this can only be used for forbidden behavior (and `false` is expressible by the given grammar). Instead, forbidden behavior should be expressed by simply not adding respective parts to a model. It should be noted that in the following, we utilize \mathcal{B} also for denoting constraints over other sets, where the type of constraint is indicated by the set (or sets) written in parentheses (e.g., $\mathcal{B}(C)$ denotes clock constraints as C is the set of clocks).

Example 2.1 (Clock Constraint). Consider the real-time constraints of the PPU as described in Section 2.1. Here, the allowed time frame for rotating the crane may have a lower bound of 10 seconds and an upper bound of 15 seconds. Assuming we use clock $c \in C$, this constraint can be expressed by $c \geq 10 \wedge c \leq 15$.

Clock constraints as described in Definition 2.1 neither contain operators for disjunction nor negation to ensure that the resulting constraints always describe convex polyhedra. This is important for practical implementations as operations on clock constraints can be computed more efficiently by utilizing convex polyhedra [38]. Furthermore, all required comparisons are expressible by the given grammar. For instance, $c = 42$ may be expressed by $c \geq 42 \wedge c \leq 42$, and $\neg(c \geq 42)$ may be expressed by $c < 42$. Additionally, $c \geq 42 \vee c \leq 11$ may be expressed by different switches labeled with $c \geq 42$ and $c \leq 11$, respectively. As the latter is not immediately obvious, we formally prove the correctness of this transformation w.r.t. timed bisimilarity later in this thesis (see Proposition 6.2 in Chapter 6). Note, that we may use these operators in our examples to increase readability. Furthermore, we distinguish between two flavors of clock constraints, namely *guards* and *location invariants*.

- Guards denote time intervals in which switches are enabled.
- Location invariants (or simply *invariants*) denote time intervals in which a TA run may reside in a given location.

Without loss of generality, we assume invariants unequal to `true` to be *downward-closed*, i.e., having the form $c < n$ or $c \leq n$ with $c \in C$ and $n \in \mathbb{T}_C$ [38]. We employ this requirement solely to ensure a more efficient analysis of TA. Here, other forms of invariants may be modeled by adding respective guards to incoming switches

of a location. Finally, switches are labeled with a subset $R \subseteq C$ of clocks to be *reset*. Hence, clocks in R are reset to value 0 if the corresponding switch is used.

Definition 2.2 (Timed Automaton). A *timed automaton* (TA) is a tuple $\mathcal{A} = (L, \ell_0, \Sigma, C, I, E)$, where

- L is a finite set of *locations*,
- $\ell_0 \in L$ is the *initial location*,
- Σ is a finite set of *actions* with $L \cap \Sigma = \emptyset$,
- C is a finite set of *clocks* over \mathbb{T}_C such that $C \cap (L \cup \Sigma) = \emptyset$,
- $I : L \rightarrow \mathcal{B}(C)$ is a function assigning *invariants* to locations, and
- $E \subseteq L \times \mathcal{B}(C) \times \Sigma \times 2^C \times L$ is a finite relation defining *switches*.

We use $\ell \xrightarrow{g, \sigma, R} \ell'$ to denote $(\ell, g, \sigma, R, \ell') \in E$, where $\{\ell, \ell'\} \subseteq L$, $g \in \mathcal{B}(C)$, $\sigma \in \Sigma$, and $R \subseteq C$.

It should be noted that we assume *diagonal-free* TA, i.e., TA without *difference constraints* $c - c' \sim n$. We only use difference constraints later on to model the symbolic semantics of TA (see Section 2.5). However, each TA with difference constraints may be transformed into an equivalent diagonal-free TA [51].

Example 2.2 (Timed Automaton of the PPU). Figure 2.3 depicts an example TA for an extract of the PPU, integrating the real-time constraints as described in Section 2.1. Here, locations are depicted by nodes containing the name of the location and an invariant. Locations are connected by switches being labeled with an action, a guard, and a set of clocks (i.e., clocks written in curly brackets) to be reset. Note, that we omit clock constraints equal to true and empty sets of clocks to reset from figures.

The PPU starts in an *initial* position and then *rotates* to the *stack*. This rotation takes at least 10 time units (e.g., seconds), such that the plant is not damaged by moving too fast, and at most 15 seconds to avoid unnecessary idle time of the rest of plant. This requirement is modeled by the guard $x \geq 10 \wedge x \leq 15$. After reaching the *stack*, the PPU *picks* up the workpiece. Here, the first rotation together with picking the workpiece may take up to 30 seconds (due to the invariant of location *stack*). When *picking* up the workpiece, clocks x and y are reset. Thereafter, *rotating* to the *ramp* may again take 10 to 15 seconds. As a result, location *WP picked* (i.e., workpiece picked) must be left after at most 15 seconds as otherwise, the plant is stuck in a deadlock as there is no other way to leave *WP picked*. Additionally, it is not safe to wait in *WP picked* as this location must be left after at most 40 seconds due to the invariant. Again, *rotating* to the *ramp* and *placing* the workpiece may take up to 30 seconds due to invariant $x \leq 30$. After *placing* the workpiece, the PPU is either again *ready* in the *initial* location or may *wait* in location *idle* if there is no additional task to perform. Finally, the plant may initiate an emergency *shut down* at the *stack* and

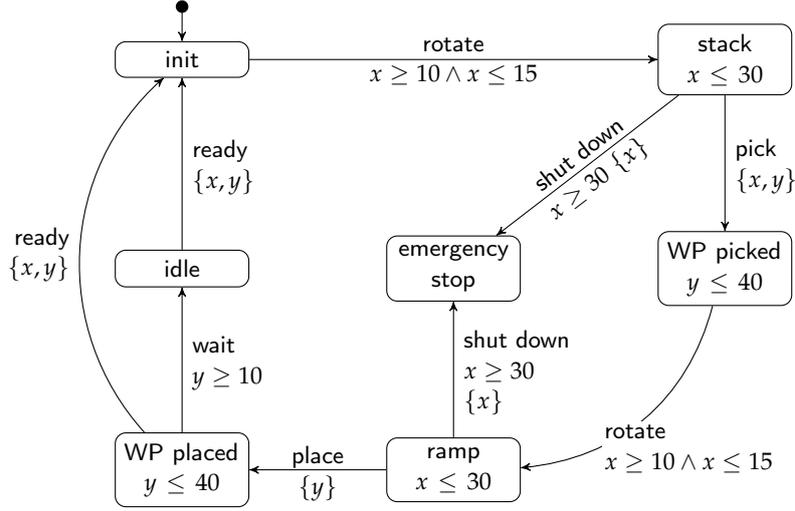


Figure 2.3: TA of an Extract of the PPU with Clocks x and y

the *ramp* if the workpiece cannot be picked up within 30 seconds. Afterwards, location *emergency stop* may not be left such that the PPU needs to be restarted manually by a supervisor.

TA originally had special *acceptance locations* [9] instead of location invariants. Therewith, one may employ Büchi-acceptance semantics on *infinite* runs [9, 101]. However, as we want to apply TA for model-based testing purposes being limited to *finite* runs, we do not consider acceptance locations and instead use *Timed Safety Automata* [101], containing location invariants. Hence, we use *Timed Automata (TA)* to refer to Timed Safety Automata in the remainder of this thesis. Please note, that it is not possible to construct a behaviorally equivalent TA \mathcal{A}' *without* invariants for a TA \mathcal{A} *with* invariants (if we only consider finite runs). Intuitively, we may reside in every location of \mathcal{A}' for an arbitrary amount of time as there are no invariants (even though we may not be able to leave some locations after a given amount of time due to guards of outgoing switches). In contrast, we may *not* reside for an arbitrary amount of time in locations of \mathcal{A} having invariants.

Furthermore, we may extend TA by so-called *internal behavior*. Internal behavior cannot be observed by the environment (or a tester) and is summarized under the symbol $\tau \notin \Sigma$, being annotated as an action for *silent* switches. Here, we use $\Sigma_\tau = \Sigma \cup \{\tau\}$ to denote the set of actions including τ .

Definition 2.3 (TA with Internal Behavior). A *timed automaton with internal behavior* (TA_τ) is a tuple $\mathcal{A}_\tau = (L, \ell_0, \Sigma, C, I, E')$, where

- $(L, \ell_0, \Sigma, C, I, E)$ is a TA,
- $\tau \notin L \cup \Sigma \cup C$ is an internal, unobservable action, and
- $E' \subseteq E \cup (L \times \mathcal{B}(C) \times \{\tau\} \times 2^C \times L)$ is a finite relation defining *switches*.

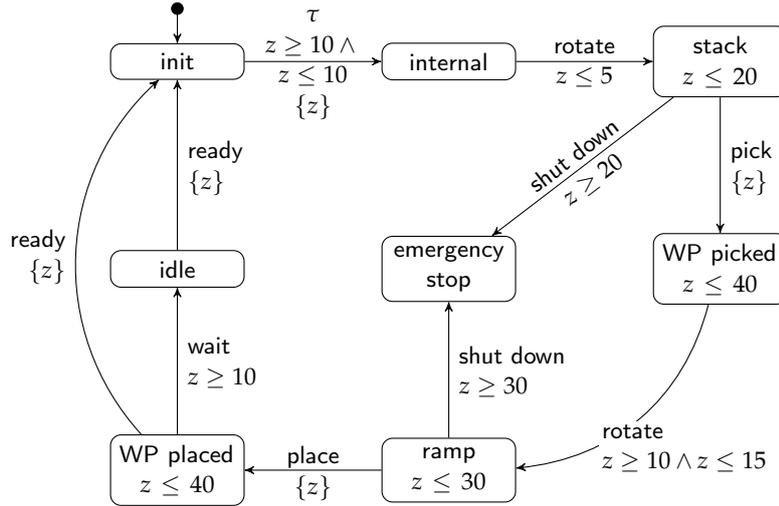


Figure 2.4: TA_τ of an Extract of the PPU

We use $\ell \xrightarrow{g, \sigma, R} \ell'$ to denote $(\ell, g, \sigma, R, \ell') \in E'$, where $\{\ell, \ell'\} \subseteq L$, $g \in \mathcal{B}(C)$, $\sigma \in \Sigma_\tau$, and $R \subseteq C$.

Again, we assume *diagonal-free* TA_τ , i.e., TA_τ without difference constraints $c - c' \sim n$.

Example 2.3 (TA_τ of the PPU). Figure 2.4 gives an example for a TA with internal behavior, being syntactically similar to the TA in Figure 2.3. However, some adjustments have been made to the TA_τ . First, this TA_τ only contains one clock (namely z) instead of two (x and y) as presented in the TA. Furthermore, there is some internal behavior (e.g., a setup) having a clock constraint $z \geq 10 \wedge z \leq 10$ and a reset of z before the first *rotation*. Afterwards, the first *rotation*, the first *shut down*, and location *stack* contain adjusted clock constraints. Finally, the clock resets for both *shut downs* were completely removed in the TA_τ .

So far, we gave an introduction to TA for modeling single time-critical systems with fixed behavior. However, we want to consider product lines of similar systems in this thesis in order to analyze these systems more efficiently by exploiting knowledge about similar behavior. For instance, consider the PPU. Here, real-time constraints for rotating the crane may be adapted to the weight of the workpiece (e.g., lightweight plastic may be transported faster than metal workpieces) while the remaining system behavior is the same for different types of workpieces. Hence, we next present two different formalisms allowing us to integrate sets of similar systems into a single model.

2.3 MODELING PRODUCT LINES OF REAL-TIME SYSTEMS

In this section, we describe extensions to the formalism of TA to incorporate *variability* within a model of a time-critical system. Therewith, some parts of a model may be *configurable* such that only some variants include these parts while

others exclude them. Hence, we subsume a set of similar time-critical systems in one model, resulting in a so-called *software product line (SPL)* [64]. As a result, we avoid variant-by-variant modeling where we have to repeat modeling (or copying) the *core* parts (being part of every variant) over and over again. Instead, an SPL comprises several similar variants having a common core functionality in addition to individual parts. In that way, we are able to construct a specific product for each customer with less effort as compared to variant-by-variant modeling [163]. Moreover, real-time requirements become more and more important in SPL engineering being applied to critical systems (e.g., automation systems, automotive software, and medical devices) [202].

In the remainder of this section, we first give an introduction to software product lines (see Section 2.3.1). Afterwards, we describe two formalism for modeling real-time software product lines, namely *Featured Timed Automata (FTA)* [70] (see Section 2.3.2) and *Parametric Timed Automata (PTA)* [15] (see Section 2.3.3).

2.3.1 Software Product Lines

We start this section with an illustrating example of an SPL.

Example 2.4. The previously introduced PPU may be extended to an SPL. For instance, there may be several different materials for the parts that are moved by the plant. Here, the time constraints for light materials (e.g., *plastic*) may be stricter as these parts can safely be moved faster. In contrast, heavy materials such as metal require stricter constraints as these parts may only be moved at a lower speed to avoid damaging the plant. However, even if these variants move parts with varying speeds, most of the behavior is the same such that we can model these products with a single model where some of the time constraints are configurable. Furthermore, some of the products may not offer an emergency shutdown functionality of the PPU as parts being moved are too light to cause any damage, while the rest of the PPU has the same or similar behavior as products with emergency shutdown. Finally, we may include an automatic reboot if the emergency stop is part of a configuration.

The majority of proposed approaches for modeling product lines utilize additional constructs for behavioral variability [37]. For instance, these constructs may be *parameter-based* and *feature-based*. Here, parameters are used as placeholders for numerical values which may be configured due to the needs of a customer. For instance, the real-time constraints for moving the crane of the PPU may be modeled with parameters such that the constraints may be specifically tailored for different circumstances. As opposed to parameters, a *feature* is “a prominent or distinctive aspect, quality, or characteristic of a software system or systems” [108]. For instance, the functionality for emergency shutdowns of the PPU may be considered as a feature. Based on the notion of features, we employ so-called *feature diagrams* using *featured-oriented domain analysis (FODA) notation* [108]. A feature diagram represents dependencies between features (e.g., some features might exclude each other) and is structured as a tree where the root element is called *root feature*. Additionally, dependencies between *parent features* and *child features* (or groups

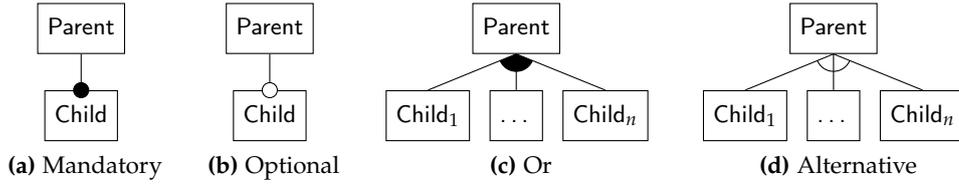


Figure 2.5: Elements in Feature Diagrams

of child features) may occur. The semantics of these dependencies is defined for Boolean logic [28, 52] in terms of *feature constraints* over $\mathbb{B} = \{\text{true}, \text{false}\}$.

Definition 2.4 (Feature Constraint). Let F be a set of features. The set $\mathcal{B}(F)$ of *feature constraints* λ over F is inductively defined as

$$\lambda := \text{true} \mid f \mid \neg\lambda \mid \lambda \wedge \lambda$$

where $f \in F$.

It should be noted that Definition 2.4 only contains operators for negation and conjunction as other operators are expressible by the given grammar. Figure 2.5 gives an overview on the graphical notations of feature diagrams. The first type of dependency is called a *mandatory* feature, meaning that the child feature must be present in the configuration if the parent feature is selected. This translates to the Boolean expression

$$\text{Parent} \Leftrightarrow \text{Child}$$

and is denoted by a black circle at the top of the child feature in feature diagrams (see Figure 2.5a). Additionally, an *optional* feature may be added to the configuration if the parent feature is selected. Here, the corresponding Boolean constraint is

$$\text{Child} \Rightarrow \text{Parent}$$

and it is denoted by a white circle at the top of the child feature (see Figure 2.5b). Furthermore, there are two different groups of sibling features called *or-group* and *alternative group*. An *or-group* below a parent feature denotes that *at least* one of the child features must be included if the parent feature is included. This results in the constraint

$$\text{Parent} \Leftrightarrow (\text{Child}_1 \vee \dots \vee \text{Child}_n)$$

and is denoted by a black arc below the parent feature (see Figure 2.5c). In contrast, an *alternative group* below a parent feature means that *exactly* one of the child features must be included if the parent feature is included, resulting in the constraint

$$\begin{aligned} &(\text{Child}_1 \Leftrightarrow (\text{Parent} \wedge \neg\text{Child}_2 \wedge \dots \wedge \neg\text{Child}_n)) \wedge \\ &\dots \\ &(\text{Child}_n \Leftrightarrow (\text{Parent} \wedge \neg\text{Child}_1 \wedge \dots \wedge \neg\text{Child}_{n-1})) \end{aligned}$$

and being depicted in a feature diagram by an unfilled arc below the parent feature (see Figure 2.5d).

In addition to these four concept, feature diagrams may also contain *cross-tree constraints*, denoting constraints between two arbitrary and hierarchically unrelated features. Examples of cross-tree constraints are *require edges* (denoting constraints of the form $f_1 \Rightarrow f_2$ for features f_1 and f_2) and *exclude edges* (denoting constraints of the form $\neg(f_1 \wedge f_2)$). However, as edges between arbitrary features in the feature diagram may obstruct readability, we simply write these constraints as Boolean expressions below the feature diagram in the following.

Finally, based on feature diagrams, we can construct *feature models* [108]. A feature model $m \in \mathcal{B}(F)$ over a set of features F represents a feature diagram as a feature constraint, following the rules for translation as described above. The semantics of m , denoted by $\llbracket m \rrbracket$, is the set of all *valid* feature configurations Θ . Here, a feature configuration θ sets each feature to either true or false, thus including or excluding the feature. A feature configuration θ is considered valid if it satisfies m .

Definition 2.5 (Feature Model). Let F be a set of features and $m \in \mathcal{B}(F)$ be a *feature model*. The semantics of a feature model is denoted as $\llbracket m \rrbracket = \{c : F \rightarrow \mathbb{B} \mid c \models m\} \subseteq 2^F$, where $c \models m$ is defined recursively as

$$\begin{aligned} c &\models \text{true} \\ c &\models f \Leftrightarrow c(f) = \text{true} \\ c &\models f \wedge f' \Leftrightarrow c \models f \text{ and } c \models f' \\ c &\models \neg f \Leftrightarrow \neg c(f) = \text{true} \end{aligned}$$

with $\{f, f'\} \subseteq F$.

It should be noted that it is possible to construct an arbitrary Boolean constraint with the graphical language of feature diagrams (depending on the basic elements we allow for feature diagrams, such as cross-tree constraints) if we are allowed to introduce additional features for structuring other features in groups [173]. This is similar to the so-called *Tseitin encoding*, where the task is to construct the conjunctive normal form (CNF) for any given Boolean formula such that the CNF is equisatisfiable (i.e., either both formulas are satisfiable or both are not) [189, 162].

Example 2.5 (Feature Model of the PPU). Figure 2.6 depicts the feature model of the (extract of the) PPU product line. Here, the root feature is called *PPU* and has three direct child features. *Workpiece* is a mandatory feature, where we may select to move either *Plastic* or *Metal* but not both at the same time due to the alternative group. Additionally, there are the optional features *Stop* for an emergency stop of the plant and *Resume* to automatically resume after an emergency stop (as opposed to manually restarting the whole plant). Finally, the cross-tree constraint denoted below the feature model restricts products such that *Stop* must be selected if *Resume* is selected (as resuming after an emergency stop only makes sense if the emergency stop is part of the product). Hence, the feature model described by this feature diagram is (equivalent to) $m := \text{PPU} \wedge \text{Workpiece} \wedge ((\text{Plastic} \wedge \neg \text{Metal}) \vee (\neg \text{Plastic} \wedge \text{Metal})) \wedge (\text{Resume} \Rightarrow \text{Stop})$.

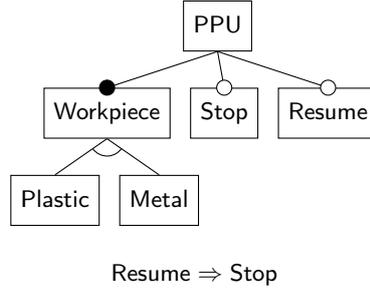


Figure 2.6: Feature Model for the Extract of the PPU

Table 2.1: Valid Configurations of the Feature Diagram Depicted in Figure 2.6

	PPU	Workpiece	Plastic	Metal	Stop	Resume
θ_1	true	true	true	false	true	true
θ_2	true	true	true	false	true	false
θ_3	true	true	true	false	false	false
θ_4	true	true	false	true	true	true
θ_5	true	true	false	true	true	false
θ_6	true	true	false	true	false	false

Furthermore, Table 2.1 gives an overview on the set $\Theta := \llbracket m \rrbracket$ of valid products. Here, each row denotes a feature configuration $\theta_i \in \Theta$, $1 \leq i \leq 6$. Note, that the values for *PPU* and *Workpiece* are always true as these features are root and mandatory, respectively. In addition, it holds that either *Plastic* or *Metal* is set to true as these features exclude each other, and *Resume* may only be true if also *Stop* is true.

2.3.2 Featured Timed Automata

Having defined the notion of features and feature models, we now give an introduction to *Featured Timed Automata (FTA)* [70]. The FTA formalism is based on *Featured Transition Systems (FTS)* [60, 61], and as the name suggests, FTA apply a *feature-based* approach. With features, we may annotate clock constraints and switches. However, it should be noted that we do not annotate locations with feature constraints as we can encode this by simply annotating all incoming and outgoing switches of a location with the respective feature constraint. For instance, rotating the crane of the PPU may have different clock constraints depending on the material of the workpiece (as described before), such that we have specific clock constraints for *Plastic* and *Metal*. Furthermore, we may include or exclude entire switches based on feature annotations. For instance, the switches for shutting down the PPU may be annotated with the feature *Stop*, such that these switches are only included in variants incorporating this feature. Here, we do not only use single features but allow annotations with feature constraints $\lambda \in \mathcal{B}(F)$. Therewith, we may formulate complex restrictions for our behavioral model. Given a feature model m , we assume in the following that $m \wedge \lambda$ is satisfiable (i.e., $\llbracket m \wedge \lambda \rrbracket \neq \emptyset$)

as otherwise we would describe parts of systems that are not present in any valid configuration.

Definition 2.6 (Featured Clock Constraint). Let C be a set of clocks defined over clock domain \mathbb{T}_C , and F be a set of features. The set $\mathcal{B}(C, F)$ of *featured clock constraints* δ is inductively defined as

$$\delta := \text{true} \mid [\lambda]\varphi \mid \delta \wedge \delta$$

where $\lambda \in \mathcal{B}(F)$ and $\varphi \in \mathcal{B}(C)$.

Again, we assume *diagonal-free* FTA, and we only use difference constraints to model the symbolic semantics of FTA. Furthermore, we may write $[\lambda]c \sim n$ (and $[\lambda]c - c' \sim n$) to denote $[\lambda](c \sim n)$ (and $[\lambda](c - c' \sim n)$). Therewith, we now formally define FTA. To this end, we utilize feature constraints in two places of an FTA model. First, featured clock constraints in terms of guards and invariants contain feature constraints (see Definition 2.6). Note, that a featured clock constraint $[\lambda]\varphi$ is removed from configurations $c \notin \llbracket \lambda \rrbracket$ not satisfying feature constraints λ . Hence, a switch of a variant (i.e., TA) may be labeled with true if $[\lambda]\varphi$ is removed. Second, we use function $\eta : E \rightarrow \mathcal{B}(F)$ to annotate switches with feature constraints. Therewith, we control whether a switch is part of a particular configuration.

Definition 2.7 (Featured Timed Automaton). A *featured timed automaton* (FTA) is a tuple $\mathcal{F} = (L, \ell_0, \Sigma, C, F, I, E, m, \eta)$, where

- L is a finite set of *locations*,
- $\ell_0 \in L$ is the *initial location*,
- Σ is a finite set of *actions* with $L \cap \Sigma = \emptyset$,
- C is a finite set of *clocks* over \mathbb{T}_C such that $C \cap (L \cup \Sigma) = \emptyset$,
- F is a finite set of *features* over \mathbb{B} such that $F \cap (C \cup L \cup \Sigma) = \emptyset$,
- $I : L \rightarrow \mathcal{B}(C, F)$ assigns *featured invariants* to locations,
- $E \subseteq L \times \mathcal{B}(C, F) \times \Sigma \times 2^C \times L$ is a finite relation defining *switches*,
- $m \in \mathcal{B}(F)$ is a *feature model* over a set of features F , and
- $\eta : E \rightarrow \mathcal{B}(F)$ assigns *feature constraints* to switches.

We use $\ell \xrightarrow{g, \sigma, R} \ell'$ to denote $(\ell, g, \sigma, R, \ell') \in E$, where $\{\ell, \ell'\} \subseteq L$, $g \in \mathcal{B}(C, F)$, $\sigma \in \Sigma$, and $R \subseteq C$.

For the remainder of this thesis, we assume an FTA $\mathcal{F} = (L, \ell_0, \Sigma, C, F, I, E, m, \eta)$ to be consistent in the sense that $\llbracket m \wedge \lambda \rrbracket \neq \emptyset$ for every feature constraint λ being part of a featured clock constraint $[\lambda]\varphi$ or being annotated to a switch. Here, we recursively define $\llbracket m \wedge \delta \rrbracket$ for a featured clock constraint δ as may have the form $\delta = [\lambda_1]\varphi_1 \wedge \dots \wedge [\lambda_n]\varphi_n$. Specifically, we use $\llbracket m \wedge \delta \rrbracket$ to denote $\llbracket m \wedge \lambda_1 \wedge \dots \wedge \lambda_n \rrbracket$.

Definition 2.8 (Consistent FTA). An FTA $\mathcal{F} = (L, \ell_0, \Sigma, C, F, I, E, m, \eta)$ is *consistent* iff

- $\forall \ell \in L : \llbracket m \wedge I(\ell) \rrbracket \neq \emptyset,$
- $\forall e \in E : \llbracket m \wedge \eta(e) \rrbracket \neq \emptyset,$ and
- $\forall (\ell, g, \sigma, R, \ell) \in E : \llbracket m \wedge g \rrbracket \neq \emptyset,$

where $\llbracket m \wedge \delta \rrbracket$ for featured clock constraint $\delta = [\lambda]\varphi$ is recursively defined as

- $\llbracket m \wedge \lambda \rrbracket$ if $\delta = [\lambda]\varphi$ and
- $\llbracket m \wedge \delta_1 \rrbracket \cap \llbracket m \wedge \delta_2 \rrbracket$ if $\delta = \delta_1 \wedge \delta_2.$

Given an FTA, we derive TA, modeling the behavior of specific variants, by defining an operator for projection [60]. Intuitively, the projection of an FTA is obtained by removing all switches being unavailable in that configuration, replacing featured clock constraints by clock constraints if they are available in that configuration, and discarding all other clock constraints [70]. To this end, a featured clock constraint $[\lambda]\varphi$ is replaced by true for a configuration $c \notin \llbracket \lambda \rrbracket$. Here, the idea is that a switch should not be deactivated if a configuration c does not satisfy feature constraint λ . Instead, there simply is no clock constraint (i.e., $\varphi = \text{true}$).

Definition 2.9 (Projection of an FTA). The *projection* of an FTA $\mathcal{F} = (L, \ell_0, \Sigma, C, F, I, E, m, \eta)$ for configuration $c \in \llbracket m \rrbracket$ is a TA $\mathcal{A} = (L, \ell_0, \Sigma, C, I', E')$, where

- $\forall \ell \in L : I'(\ell) := I(\ell)|_c$ and
- $E' = \{e = (\ell, g|_c, \sigma, R, \ell') \mid e \in E \wedge c \in \llbracket \eta(e) \rrbracket\}$

where the projection of a featured clock constraint δ to a configuration c is recursively defined as

$$\delta|_c = \begin{cases} (\delta_1)|_c \wedge (\delta_2)|_c & \text{if } \delta = \delta_1 \wedge \delta_2, \\ \varphi & \text{if } \delta = [\lambda]\varphi \wedge c \in \llbracket \lambda \rrbracket, \\ \text{true}, & \text{otherwise.} \end{cases}$$

We write $\llbracket \mathcal{F} \rrbracket_F$ to denote the set of projections according to all $c \in \llbracket m \rrbracket$ of \mathcal{F} .

Example 2.6 (Featured Timed Automaton of the PPU). Figure 2.7 depicts an illustrative example of an FTA where we extended the TA depicted in Figure 2.3 to an FTA. Here, we assume the feature diagram in Figure 2.6 to depict the corresponding feature model m . Feature constraints are written in square brackets. Annotations in front of actions denote feature constraints for switches, and annotations in front of clock constraints denote featured clock constraints.

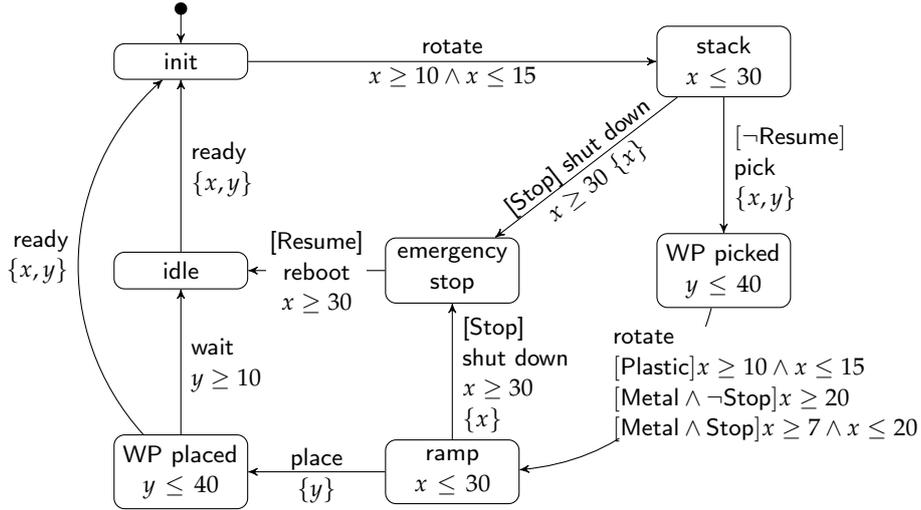


Figure 2.7: FTA for the Extract of the PPU

In this example, both switches for *shutting down* the plant are now annotated with the optional feature *Stop* as not all plants require such a functionality. Furthermore, we introduce a process for *rebooting* after an emergency stop. However, this process is annotated with feature *Resume* (which also disables *picking up* workpieces at the stack) being only intended for testing emergency stops. In normal operation, a supervisor has to manually restart the plant for safety purposes. Finally, *rotating* the crane after picking up the workpiece now has various different clock constraints being dependent on the configuration (e.g., moving workpieces made out of *metal* may be carried out faster if feature *Stop* for emergency shutdowns is selected).

As specified by our feature model m , the FTA comprises six configurations (see Table 2.1). Here, we obtain the TA as depicted in Figure 2.3 by choosing the configuration $\theta_2 = \text{PPU} \wedge \text{Workpiece} \wedge \text{Plastic} \wedge \neg \text{Metal} \wedge \text{Stop} \wedge \neg \text{Resume}$.

Having defined FTA, we are now able to model product lines of systems with time-critical behavior. However, FTA have the disadvantage that we can only specify models with a finite number of configurations. Hence, it is not possible to have a model where some time constraints may be set to very specific values chosen by supervisors for each configuration individually. For instance, a user might want to fine-tune clock constraints for moving workpieces in the PPU according to very specific aspects of the plant (e.g., age of the plant, size of the crane, and size of the workpieces). In order to solve this issue, we require a different TA-based formalism.

2.3.3 Parametric Timed Automata

Parametric Timed Automata (PTA) [15] generalize TA by allowing to use freely configurable *parameters* (in addition to constant values) as boundaries for clocks in *parametric clock constraints*. Then, variant derivation is achieved by replacing each parameter with a constant value, resulting in a TA. Therewith, we are able to set

very specific values for given time constraints (as mentioned before). Here, we apply P to denote a finite set of parameters over *parameter domain* \mathbb{T}_P , where we assume $\mathbb{T}_P = \mathbb{T}_C = \mathbb{N}_0$ for all examples. As a result, a single PTA may comprise an infinite number of variants. Formally, we have the following requirements for parameter domain \mathbb{T}_P (where the first three requirements are the same as imposed for clock domain \mathbb{T}_C).

- There is an associative operator for addition with neutral element 0.
- There is an operator for subtraction with neutral element 0.
- Elements in \mathbb{T}_P are totally ordered.
- There is an associative operator for multiplication with neutral element 1.

In parametric clock constraints, clocks may be compared with *parametric linear terms* (*plt*) having the form $(\sum_{1 \leq i \leq |P|} \alpha_i p_i) + n$ with $\alpha_i \in \mathbb{Z}$, $p_i \in P$, and $n \in \mathbb{T}_C$. Here, we utilize the sum as a shorthand for $\alpha_1 p_1 + \dots + \alpha_m p_m + n$. In particular, $\alpha_i = 0$ means that parameter p_i is *deactivated* in a given parametric linear term.

Definition 2.10 (Parametric Linear Term). Let P be a set of *parameters*. A *parametric linear term* (*plt*) over P is defined as

$$plt_P := \left(\sum_{1 \leq i \leq |P|} \alpha_i p_i \right) + n$$

where $\alpha_i \in \mathbb{Z}$, $p_i \in P$, and $n \in \mathbb{T}_C$.

With parametric linear terms, we can now define parametric clock constraints.

Definition 2.11 (Parametric Clock Constraint). Let C be a set of clocks defined over clock domain \mathbb{T}_C , and P be a set of parameters defined over parameter domain \mathbb{T}_P . The set $\mathcal{B}(C, P)$ of *parametric clock constraints* ϕ is inductively defined as

$$\phi := \text{true} \mid c \sim plt_P \mid c - c' \sim plt \mid \phi \wedge \phi$$

where plt_P is a parametric linear term over P and $\sim \in \{<, \leq, \geq, >\}$.

As before, we assume *diagonal-free* PTA, and we only use difference constraints to model the symbolic semantics of PTA. Note, that we do not exclude parametric clock constraints where $plt < 0$ holds for parameter valuations as we may utilize this to *deactivate* switches. With parameters and parametric clock constraints, we may now formally define PTA.

Definition 2.12 (Parametric Timed Automaton). A *parametric timed automaton* (PTA) is a tuple $\mathcal{P} = (L, \ell_0, \Sigma, C, P, I, E)$, where

- L is a finite set of *locations*,
- $\ell_0 \in L$ is the *initial location*,

- Σ is a finite set of *actions* with $L \cap \Sigma = \emptyset$,
- C is a finite set of *clocks* over \mathbb{T}_C such that $C \cap (L \cup \Sigma) = \emptyset$,
- P is a finite set of *parameters* over \mathbb{T}_P such that $P \cap (L \cup \Sigma \cup C) = \emptyset$,
- $I : L \rightarrow \mathcal{B}(C, P)$ assigns *parametric invariants* to locations, and
- $E \subseteq L \times \mathcal{B}(C, P) \times \Sigma \times 2^C \times L$ is a finite relation defining *switches*.

We use $\ell \xrightarrow{g, \sigma, R} \ell'$ to denote $(\ell, g, \sigma, R, \ell') \in E$, where $\{\ell, \ell'\} \subseteq L$, $g \in \mathcal{B}(C, P)$, $\sigma \in \Sigma$, and $R \subseteq C$.

A TA may be derived from a PTA by applying a so-called *parameter valuation* $\nu : P \rightarrow \mathbb{T}_P$. Here, we replace each occurrence of every parameter by a constant value in \mathbb{T}_P .

Definition 2.13 (Parameter Valuation). Let $\mathcal{P} = (L, \ell_0, \Sigma, C, P, I, E)$ be a PTA. A *parameter valuation* $\nu : P \rightarrow \mathbb{T}_P$ replaces each occurrence of every $p \in P$ in $\mathcal{B}(C, P)$ by the constant values $\nu(p) \in \mathbb{T}_P$. The resulting TA is denoted by $\nu(\mathcal{P})$.

Note, that we use $\llbracket \mathcal{P} \rrbracket_P$ to denote the set of TA being derivable from a PTA \mathcal{P} by applying all possible parameter valuations.

Example 2.7 (Parametric Timed Automaton of the PPU). Figure 2.8 gives an example for the PTA of our PPU extract, containing parameters a and b . Here, parameter a acts as the upper bound for the parametric clock constraints limiting both *rotations*. Furthermore, parameter b limits the maximum amount of time in which the crane is allowed to wait at the *stack* and at the *ramp*. Deriving a variant with values $a = 15$ and $b = 30$ results in the TA as depicted in Figure 2.3. Hence, this PTA and the FTA in Figure 2.7 have *some* variants in common. However, the FTA contains some variants not being part of the PTA (e.g., systems being able to *reboot*) and vice versa (e.g., system with an upper bound of 60 seconds for the *rotation*). Moreover, the PTA in Figure 2.8 illustrates that we may disable some switches in variants by choosing parameters in a clever way. For instance, setting b to a value less than 10 disables the *rotate* switches as the guard then contradicts the invariant of location *stack* and *ramp*, respectively.

With PTA, we are now able to model product lines with unbounded parametric constraints, allowing us to set particular parameters to very specific values for different usage scenarios. However, PTA also have a disadvantage as compared to FTA. More precisely, PTA cannot directly model features and feature constraints that are used in different places of the model. For instance, consider the switches and featured clock constraints containing feature *Stop* in the FTA in Figure 2.7. With PTA, we cannot include these constraints spanning over several switches and clock constraints. Hence, the first research challenge arises.

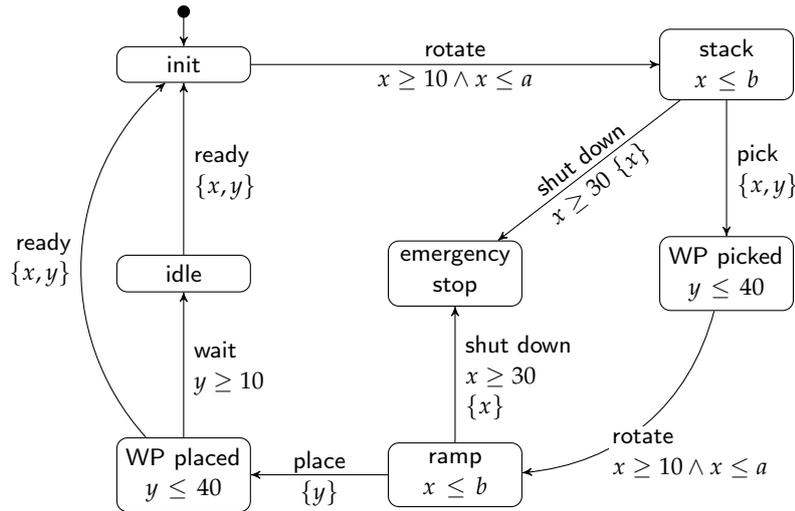


Figure 2.8: PTA for the Extract of the PPU with Parameters $P = \{a, b\}$

Research Challenge 1

Definition of a behavioral modeling formalism for real-time product lines with a potentially infinite number of variants.

This research challenge will be tackled in Chapter 3. As we have now described the foundations for modeling product lines of real-time systems, we proceed with analyzing these systems. In particular, we will describe two approaches for analyzing product lines of time-critical systems. The first approach derives a *meaningful* subset of variants such that analysis results of the subset may be generalized for the product line. With the second approach, we may exploit and reuse similarities of systems to analyze all systems at once [171] instead of applying a system-by-system approach such that we obtain the results (presumably) more efficiently. These *product-line analysis strategies* aim at ensuring particular properties for every valid product w.r.t. a given specification of an SPL [185].

2.4 SAMPLING PRODUCT LINES OF REAL-TIME SYSTEMS

In the previous section, we presented background information on modeling product lines of TA. A naive approach for analyzing product lines would be deriving all variants of a product line and then analyze these variants one by one. However, there are two further approaches applying more sophisticated strategies for analyzing product lines, presumably requiring less effort. First, we may derive a subset of variants (i.e., TA) such that we only analyze these variants. Thereafter, we may generalize the results of analyzing the variants for the product line if the subset of variants is representative. This approach is called *sampling*. Second, we may exploit product-line knowledge to analyze the whole product line at once. The first approach will be explained in this section while the second approach will be the topic of the following section. Note, that analyzing all variants of a product

line is usually infeasible due to the large number of configurations [53]. This is even more problematic in the context of product lines with an infinite number of variants (e.g., PTA) where it is impossible to analyze all variants. Hence, this section gives an introduction for deriving a *representative* subset of variants from a product line. For a comprehensive overview on sampling, we refer the reader to the surveys of Varshosaz et al. [194], Ahmed et al. [5], and Lopez-Herrejon et al. [129].

Here, *combinatorial interaction testing (CIT)* [150] is a mature technique [66, 158, 154, 106, 107, 96, 8] for deriving a relatively small *sample* Θ of representative test configurations $\theta \in \Theta$ from a (usually) large configuration space Ω_Θ , while covering a sufficient number of critical combinations of configuration options. Hence, we achieve coverage of potentially erroneous interactions of all input parameters $\omega_i \in \Omega$ where each parameter ω_i has a set V_i of *parameter values*. As a result, the (*valid*) *combination space* $\Theta_\Omega \subseteq V_1 \times V_2 \times \dots \times V_n$ is a set of vectors of parameter values in V_i such that a *test case* $\theta \in \Theta$ consists of one value for each parameter $\omega_i \in \Omega$ for a complete configuration.

Notation 2.1 (Combinatorial Testing). Let Ω be a finite set of n *parameters* with each parameter $\omega_i \in \Omega$ with $i \in \{1, \dots, n\}$ having a finite set V_i of *parameter values*. We use the following notations:

- $\Theta_\Omega \subseteq V_1 \times V_2 \times \dots \times V_n$ denotes the (*valid*) *combination space*,
- $\theta = (v_1, v_2, \dots, v_n) \in \Theta_\Omega$ denotes a *test case*, and
- $\Theta \subseteq \Theta_\Omega$ denotes a *test suite*.

Note, that we identify a value $v_i \in V_i$ of parameter $\omega_i \in \Omega$ in a test case $\theta = (v_1, v_2, \dots, v_n)$ by its index i (i.e., its position in θ). In this thesis, we apply combinatorial testing to the set of all configurations of a feature model m . Therefore, the set of parameters is $\Omega = F$ (where F is the set of features). Furthermore, we have sets of parameter values $V_i = \mathbb{B}$ as features can only be selected and deselected, respectively. Hence, the valid configuration space $\Omega_\Theta \in 2^\Omega$ consists of Boolean vectors satisfying a set of Boolean constraints, and a (*valid*) test case $\theta \in \Omega$ is a configuration $\theta \in \llbracket m \rrbracket$.

Example 2.8. Consider the feature diagram presented in Figure 2.6 and the corresponding configurations in Table 2.1. Here, the set of parameters $\Omega = \{\text{PPU, Workpiece, Plastic, Metal, Stop, Resume}\}$ consists of features with parameter values $V_i = \mathbb{B}$ for $1 \leq i \leq n$. Moreover, the valid combination space (being restricted by the feature model) is $\Theta_\Omega = \{\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6\}$ (i.e., all configurations in Table 2.1), and each θ_i with $1 \leq i \leq n$ is a test case. Furthermore, every subset of Θ_Ω is, by definition, a proper test suite.

Exhaustive combinatorial testing would require a test suite $\Theta = \Theta_\Omega$ covering all valid combinations of parameter values. This is theoretically possible as all sets of parameter values V_i as well as the number of parameters are finite by definition. However, this is impractical for realistic product lines due to the combinatorial explosion of the size of Θ_Ω in the number of parameters. Hence, in practice

we require heuristics to restrict the number of combinations to specific *value schemas* [176, 150].

Intuitively, a value schema ζ sets *some* parameters to fixed values while leaving the remaining parameters open. Then, the goal is to cover these schemas $\zeta \in Q$ with test cases $\theta \in \Theta_\Omega$, such that for each value schema $\zeta \in Q$ there exists a test case $\theta \in \Theta$ having exactly the parameter-value combination fixed in ζ . Here, we write $\zeta \in \theta$ to denote that value schema ζ is covered by test case θ . Furthermore, a test case θ may cover more than one schema ζ .

Definition 2.14 (Value Schema). Let Θ_Ω be the (valid) combination space of a set Ω of n parameters. A vector

$$\zeta = (v_{\zeta_1}, \dots, v_{\zeta_l}) \in V_{\zeta_1} \times \dots \times V_{\zeta_l}$$

for a subset $\Omega' = \{\omega_{\zeta_1}, \dots, \omega_{\zeta_l}\} \subseteq \Omega$ with $\zeta_i \in \{1, \dots, n\}$, $1 \leq i \leq l \leq n$, is a *value schema* iff there exists a $\theta \in \Theta_\Omega$ that contains for all $\omega_{\zeta_i} \in \Omega'$ with $1 \leq i \leq l$ the value v_{ζ_i} . We write $\zeta \in \theta$ to denote that ζ is covered by θ .

Note, that we write “-” in positions of value schemas if the respective value is not fixed, and we use the value (i.e., true or false) of a parameter, otherwise.

Example 2.9 (Value Schema). Consider the configurations in Table 2.1 as an example, where *PPU* has the first position resulting in index 1, *Workpiece* has index 2 and so on. Then a possible set of value schemas $Q = \{\zeta_1, \zeta_2, \zeta_3\}$ may be given as follows:

- $\zeta_1 = (\text{true}, \text{true}, -, -, -, -)$ such that *PPU* and *Workpiece* should be fixed, whereas
- $\zeta_2 = (-, -, \text{true}, -, -, -)$ denotes that *Plastic* should be fixed, and
- $\zeta_3 = (-, -, \text{false}, \text{true}, -, -)$ such that *Plastic* and *Metal* are deselected and selected, respectively.

Next, a *covering array* (or *sample*) is a test suite Θ specifically derived for covering a set Q of value schemas defined on the valid combination space Θ_Ω . Here, each row denotes a test case $\theta \in \Theta$, and each column is dedicated to a parameter $\omega \in \Omega$.

Definition 2.15 (Covering Array). Let Θ_Ω be the (valid) combination space of a set Ω of n parameters and Q be a set of value schemas. Set $\Theta \subseteq \Theta_\Omega$ is a *covering array* for Q if

$$\forall \zeta \in Q : (\exists \theta \in \Theta : \zeta \in \theta).$$

It should be noted that Θ_Ω itself is always a covering array but it is usually not minimal.

Example 2.10 (Covering Array). Consider again the configurations presented in Table 2.1 and the set of value schemas $Q = \{\zeta_1, \zeta_2, \zeta_3\}$ of Example 2.9. Here,

each row of the table corresponds to a test case, and each column is a feature. Furthermore, the test suite (i.e., covering array) $\Theta = \{\theta_1, \theta_4\}$ is able to cover all three value schemas as θ_1 covers ζ_1 and ζ_2 , and θ_4 covers ζ_3 .

Generally, set Q of value schemas containing critical combinations can be provided in different forms but one of the most frequently used heuristics for automatically selecting a set Q is t -wise CIT with $t \in \{1, \dots, n\}$. Test suite $\Theta \subseteq \Theta_\Omega$ satisfies t -wise coverage if each possible value schema ζ for any subset $\Omega' \subseteq \Omega$ with $t = |\Omega'|$ parameters is covered by at least one test case in Θ .

Definition 2.16 (t -wise CIT). Let Ω be a set of parameters, Θ_Ω be a (valid) combination space over Ω , and $t \in \mathbb{N}$ with $t \leq |\Omega|$. Then, test suite $\Theta \subseteq \Theta_\Omega$ satisfies t -wise coverage if each possible value schema ζ for any subset $\Omega' \subseteq \Omega$ with $t = |\Omega'|$ parameters is covered by at least one test case in Θ .

Intuitively, for 1-wise CIT we have to include at least one test case for each value of each parameter. Here, most works (e.g., [154, 176]) apply $t = 2$, requiring each valid *pair* of parameter values to occur in at least one test case in test suite Θ (hence, the name *pairwise* CIT).

Example 2.11 (t -wise CIT). Concerning the feature model in Figure 2.6 and the corresponding configurations in Table 2.1, we need to include the value true for the features *PPU* and *Workpiece* (as they are root and mandatory, respectively) and the values true and false for all other features for 1-wise coverage. Hence, a test suite $\Theta_1 = \{\theta_1, \theta_6\}$ satisfies 1-wise CIT. For pairwise CIT, we need to cover all pairs of valid parameter values, already resulting in 38 value schemas. For instance, the feature pair $\{\text{Stop}, \text{Resume}\} \subset \Omega$ results in the following three value schemas (where the order of the features corresponds to the order in Table 2.1):

- $\zeta_1 = (-, -, -, -, \text{true}, \text{true}),$
- $\zeta_2 = (-, -, -, -, \text{true}, \text{false}),$ and
- $\zeta_3 = (-, -, -, -, \text{false}, \text{false}).$

In contrast, $(\text{false}, \text{true})$ is not a value schema for $\{\text{Stop}, \text{Resume}\}$ due to the restrictions of the feature model. For pairwise CIT, we additionally have to cover the following 35 value schemas (resulting in an overall number of 38 value schemas):

- | | | |
|--|---|---|
| ▪ $\zeta_4 = (\text{true}, \text{true}, -, -, -, -)$ | ▪ $\zeta_{16} = (-, \text{true}, -, \text{false}, -, -)$ | ▪ $\zeta_{28} = (-, -, \text{true}, -, -, \text{false})$ |
| ▪ $\zeta_5 = (\text{true}, -, \text{true}, -, -, -)$ | ▪ $\zeta_{17} = (-, \text{true}, -, -, \text{true}, -)$ | ▪ $\zeta_{29} = (-, -, \text{false}, -, -, \text{true})$ |
| ▪ $\zeta_6 = (\text{true}, -, \text{false}, -, -, -)$ | ▪ $\zeta_{18} = (-, \text{true}, -, -, \text{false}, -)$ | ▪ $\zeta_{30} = (-, -, \text{false}, -, -, \text{false})$ |
| ▪ $\zeta_7 = (\text{true}, -, -, \text{true}, -, -)$ | ▪ $\zeta_{19} = (-, \text{true}, -, -, -, \text{true})$ | ▪ $\zeta_{31} = (-, -, -, \text{true}, \text{true}, -)$ |
| ▪ $\zeta_8 = (\text{true}, -, -, \text{false}, -, -)$ | ▪ $\zeta_{20} = (-, \text{true}, -, -, -, \text{false})$ | ▪ $\zeta_{32} = (-, -, -, \text{true}, \text{false}, -)$ |
| ▪ $\zeta_9 = (\text{true}, -, -, -, \text{true}, -)$ | ▪ $\zeta_{21} = (-, -, \text{true}, \text{false}, -, -)$ | ▪ $\zeta_{33} = (-, -, -, \text{false}, \text{true}, -)$ |
| ▪ $\zeta_{10} = (\text{true}, -, -, -, \text{false}, -)$ | ▪ $\zeta_{22} = (-, -, \text{false}, \text{true}, -, -)$ | ▪ $\zeta_{34} = (-, -, -, \text{false}, \text{false}, -)$ |
| ▪ $\zeta_{11} = (\text{true}, -, -, -, \text{true})$ | ▪ $\zeta_{23} = (-, -, \text{true}, -, \text{true}, -)$ | ▪ $\zeta_{35} = (-, -, -, \text{true}, -, \text{true})$ |
| ▪ $\zeta_{12} = (\text{true}, -, -, -, \text{false})$ | ▪ $\zeta_{24} = (-, -, \text{true}, -, \text{false}, -)$ | ▪ $\zeta_{36} = (-, -, -, \text{true}, -, \text{false})$ |
| ▪ $\zeta_{13} = (-, \text{true}, \text{true}, -, -, -)$ | ▪ $\zeta_{25} = (-, -, \text{false}, -, \text{true}, -)$ | ▪ $\zeta_{37} = (-, -, -, \text{false}, -, \text{true})$ |
| ▪ $\zeta_{14} = (-, \text{true}, \text{false}, -, -, -)$ | ▪ $\zeta_{26} = (-, -, \text{false}, -, \text{false}, -)$ | ▪ $\zeta_{38} = (-, -, -, \text{false}, -, \text{false})$ |
| ▪ $\zeta_{15} = (-, \text{true}, -, \text{true}, -, -)$ | ▪ $\zeta_{27} = (-, -, \text{true}, -, -, \text{true})$ | |

As a result, a test suite $\Theta_2 = \{\theta_1, \theta_3, \theta_4, \theta_5, \theta_6\}$ satisfies pairwise CIT.

Here, sampling algorithms for t -wise CIT have the goal to generate a preferably small test suite Θ_{min} (w.r.t. the size of Θ_{min}). For instance, test suites Θ_1 and Θ_2 of Example 2.11 are minimal w.r.t. the feature model in Figure 2.6 for 1-wise coverage and pairwise coverage, respectively. However, finding a minimal test suite is, in general, NP-hard as the set cover problem (which is an NP-hard problem [115]) can be reduced to finding a minimal test suite.

Notation 2.2. Let Ω be a set of parameters, Θ_Ω be a combination space over Ω , and $t \in \mathbb{N}$. Then, we use

$$CA(\Omega, \Theta_\Omega, t)$$

to denote the set of all t -wise covering arrays for Θ_Ω over Ω .

A well-known property of CIT is the fact that setting t to a greater or equal value than t' (i.e., $t \geq t'$) always results in $CA(\Omega, \Theta_\Omega, t) \subseteq CA(\Omega, \Theta_\Omega, t')$, meaning that $CA(\Omega, \Theta_\Omega, t)$ has at most as many covering arrays as $CA(\Omega, \Theta_\Omega, t')$ [150]. The reason for this is that increasing t also increases the number of value schemas. Hence, the number of test suites covering all values schemas becomes smaller as more value schemas need to be covered. Note, that this only holds for a non-empty combination space Θ_Ω (i.e., $\Theta_\Omega \neq \emptyset$) as an empty combination space is covered by the empty covering array (such that $t < t'$ would be possible in this case).

Lemma 2.1. Let Ω be a set of parameters, Θ_Ω be a combination space over Ω , and $\{t, t'\} \subseteq \mathbb{N}$. Then

$$CA(\Omega, \Theta_\Omega, t) \subseteq CA(\Omega, \Theta_\Omega, t')$$

iff $t \geq t'$ and $\Theta_\Omega \neq \emptyset$.

Proof. Let Ω be a set of parameters, Θ_Ω be a combination space over Ω , and $\{t, t'\} \subseteq \mathbb{N}$. By definition, it holds that $CA(\Omega, \Theta_\Omega, t) \subseteq CA(\Omega, \Theta_\Omega, t')$ as increasing t also increases the number of value schemas (see Definition 2.16 and Notation 2.2). Hence, the number of test suites covering all values schemas becomes smaller as more value schemas need to be covered. \square

Up to this point, the presented CIT techniques for sampling only cover finite configuration spaces. These techniques are not directly applicable to *infinite* configuration spaces as already 1-wise CIT would result in an infinite number of value schemas. However, we have to consider *infinite* configuration spaces in case of PTA and our novel model incorporating feature variability as well as parametric variability (see Research Challenge 1). Furthermore, we want to consider boundary behavior of real-time systems in terms of best-case execution times and worst-case execution times (BCET/WCET), respectively [203]. This idea stems from the area of boundary-value testing, having the goal to cover boundary values for each variable [165, 81]. The rationale behind this is the fact that problems often occur at these boundary values. However, considering t-wise CIT we cannot guarantee that test cases with BCET/WCET are included in our sample. For instance, test suite $\Theta_1 = \{\theta_1, \theta_6\}$ of Example 2.11, satisfying 1-wise coverage for the FTA in Figure 2.7, does not include the configuration with BCET for location *ramp* although the configuration space is finite (this would require a configuration satisfying $\text{Metal} \wedge \text{Stop}$). Test suite $\Theta_2 = \{\theta_1, \theta_3, \theta_4, \theta_5, \theta_6\}$ includes a configuration with BCET for location *ramp* but Θ_2 is quite inefficient in the sense that it contains five of the six possible configuration. This would result in a much higher effort for analyses after performing the sampling. Therefore, we have the goal to adapt sampling strategies to product lines with a (potentially) infinite number of configurations, and we instantiate these product lines with product lines having configurable parametric real-time constraints (i.e., the result of Research Challenge 1).

Research Challenge 2

Adapting sampling strategies to product lines with a (potentially) infinite number of configurations.

This research challenge will be tackled in Chapter 4. Up to this point, we presented an introduction to basics for modeling and sampling (product lines of) time-critical systems. Next, we tackle the issue of deriving the semantics of TA such that we may apply analysis techniques.

2.5 SEMANTICS AND ANALYSIS OF REAL-TIME SYSTEMS

A critical task in software engineering is the analysis of systems to reason about important properties. As we utilize TA-based models to specify real-time systems, we first give an introduction on modeling the semantics of TA. For instance, consider the TA depicted in Figure 2.3. Here, we cannot directly derive the allowed behavior after reaching location *ramp* if we do not know the values of the clocks (e.g., waiting for 20 seconds is only allowed if x has at most the value 10). Hence, we require different formalisms to model the semantics of TA-based real-time systems. Therefore, we give an introduction to modeling the semantics of real-time systems in this section (see Sections 2.5.1 to 2.5.3). Afterwards, we present basics for model-based coverage-driven testing of TA (see Section 2.5.4).

2.5.1 Semantics of Timed Automata

In order to analyze TA, we model the *operational semantics* in terms of *Timed Labeled Transition Systems (TLTS)* [99, 100]. A TLTS state $\langle \ell, u \rangle$ consists of a location $\ell \in L$ and a *clock valuation* $u \in C \rightarrow \mathbb{T}_C$. Furthermore, TLTS comprise two kinds of transitions. The first kind models passage of time while inactively residing in a location. The second kind models switches between locations due to action occurrences where actions happen instantaneously (i.e., without passage of time). Given a clock valuation u , we use the following standard notations.

- $u + d$ with $d \in \mathbb{T}_C$ denotes the *updated* clock valuation, mapping each clock $c \in C$ to the new value $u(c) + d$.
- $[R \mapsto 0]u$ with $R \subseteq C$ denotes the reset of each clock $c \in R$ to value 0 while the values $u(c')$ of all other clocks $c' \in C \setminus R$ remain unchanged.
- $u \in \varphi$ (with $\varphi \in \mathcal{B}(C)$) denotes that clock valuation u satisfies clock constraint φ .

Here, notation $u \in \varphi$ utilizes an operator usually applied for sets. As this is quite unusual in areas other than TA, we formally define this notation.

Definition 2.17. Let $\varphi \in \mathcal{B}(C)$ be a clock constraint. We write $u \in \varphi$ to denote that clock valuation u satisfies φ where $u \in \varphi$ is recursively defined as

$$\begin{aligned} u \in \text{true} \\ u \in c \sim n &\Leftrightarrow u(c) \sim n \\ u \in c - c' \sim n &\Leftrightarrow u(c) - u(c') \sim n \\ u \in \varphi \wedge \varphi' &\Leftrightarrow u \in \varphi \text{ and } u \in \varphi' \end{aligned}$$

with $\sim \in \{<, \leq, \geq, >\}$, $n \in \mathbb{T}_C$, and $u(c)$ denoting the valuation of clock $c \in C$ in u .

Therewith, we may now formally define TLTS.

Definition 2.18 (Timed Labeled Transition System). The *timed labeled transition system (TLTS)* of a TA $\mathcal{A} = (L, \ell_0, \Sigma, C, I, E)$ is a tuple $(S, s_0, \hat{\Sigma}, \rightarrow)$, where

- $S = L \times (C \rightarrow \mathbb{T}_C)$ is a set of *states*,
- $s_0 = \langle \ell_0, [C \mapsto 0]u \rangle \in S$ is the *initial state*,
- $\hat{\Sigma} = \Sigma \cup \Delta$ is a set of *actions* with $\Delta = \mathbb{T}_C$ and $\Sigma \cap \Delta = \emptyset$, and
- $\rightarrow \subseteq S \times (\hat{\Sigma} \cup \{\tau\}) \times S$ is a set of *strong transitions* being the least relation satisfying the following rules:
 - $\langle \ell, u \rangle \xrightarrow{d} \langle \ell, u + d \rangle$ if $u \in I(\ell)$ and $(u + d) \in I(\ell)$ for $d \in \Delta$, and
 - $\langle \ell, u \rangle \xrightarrow{\sigma} \langle \ell', u' \rangle$ if $u \in I(\ell)$, $u \in g$, $u' = [R \mapsto 0]u$, $u' \in I(\ell')$, and $\sigma \in \Sigma$ for $(\ell, g, \sigma, R, \ell') \in E$.

By $\Rightarrow \subseteq S \times \hat{\Sigma} \times S$, we denote a set of *weak transitions* being the least relation satisfying the following rules, where $\sigma \in \Sigma$ and $d \in \Delta$:

- $s \xrightarrow{\tau^n} s'$ if $s \xrightarrow{\tau} s_1 \xrightarrow{\tau} \dots \xrightarrow{\tau} s_{n-1} \xrightarrow{\tau} s'$ with $n \in \mathbb{N}_0$,
- $s \xrightarrow{\sigma} s'$ if $s \xrightarrow{\tau^m} s_1 \xrightarrow{\sigma} s_2 \xrightarrow{\tau^n} s'$ with $\{m, n\} \subseteq \mathbb{N}_0$,
- $s \xrightarrow{d} s'$ if $s \xrightarrow{d} s'$,
- $s \xrightarrow{0} s'$ if $s \xrightarrow{\tau^n}$ with $n \in \mathbb{N}_0$, and
- $s \xrightarrow{d+d'} s'$ if $s \xrightarrow{d} s'' \xrightarrow{d'} s'$.

By $\llbracket \mathcal{A} \rrbracket_S$, we refer to the TLTS of TA \mathcal{A} . We call \mathcal{A} *deterministic* if it holds for $\llbracket \mathcal{A} \rrbracket_S$ that $\forall s \in S : (\forall s \xrightarrow{\mu} s', s \xrightarrow{\mu'} s'' \in \rightarrow : s' = s'')$ for $\mu \in \hat{\Sigma}$.

We apply the following shorthand notations for TLTS. Here, we use the term *strong* to indicate that a (timed) run or (timed) trace does not contain unobservable τ -steps, and we use *weak* to indicate the possible existence of τ -steps.

Notation 2.3 (TLTS Notations). Let $\llbracket \mathcal{A} \rrbracket_S = (S, s_0, \hat{\Sigma}, \rightarrow)$ be the TLTS of TA $\mathcal{A} = (L, \ell_0, \Sigma, C, I, E)$. We utilize the following notations.

- $s \xrightarrow{\mu} s'$ if $\exists s' \in S : s \xrightarrow{\mu} s'$ and $\mu \in \hat{\Sigma}$,
- $s \xrightarrow{(d, \sigma)} s'$ if $\exists s'' \in S : s \xrightarrow{d} s'' \xrightarrow{\sigma} s'$ with $d \in \Delta$ and $\sigma \in \Sigma$,
- $s_0 \xrightarrow{(d_1, \sigma_1), \dots, (d_n, \sigma_n)} s_n$ if $\exists s_1, \dots, s_{n-1} \in S : s_0 \xrightarrow{(d_1, \sigma_1)} s_1 \xrightarrow{(d_2, \sigma_2)} \dots \xrightarrow{(d_n, \sigma_n)} s_n$ with $\{d_1, \dots, d_n\} \subseteq \Delta$ and $\{\sigma_1, \dots, \sigma_n\} \subseteq \Sigma$, and
- $s_0 \xrightarrow{(d_1, \sigma_1), \dots, (d_n, \sigma_n)}$ if $\exists s_n \in S : s_0 \xrightarrow{(d_1, \sigma_1), \dots, (d_n, \sigma_n)} s_n$ with $\{d_1, \dots, d_n\} \subseteq \Delta$ and $\{\sigma_1, \dots, \sigma_n\} \subseteq \Sigma$.

Furthermore, we utilize the following notions.

- $\rho = s_0 \xrightarrow{(d_1, \sigma_1), \dots, (d_n, \sigma_n)} s_n$ describes a *strong (timed) run* consisting of *strong (timed) steps* $s_i \xrightarrow{(d_{i+1}, \sigma_{i+1})} s_{i+1}$ with $0 \leq i < n$.
- The corresponding *(timed) trace* is $(d_1, \sigma_1), \dots, (d_n, \sigma_n)$.
- The corresponding *untimed trace* is a sequence $\sigma_1 \dots \sigma_n$.
- $\llbracket \mathcal{A} \rrbracket_R$ denotes the set of all runs of the TLTS semantics of \mathcal{A} .
- We obtain *weak (timed) runs* and *weak (timed) steps* by replacing \rightarrow with \Rightarrow .
- Location $\ell \in L$ is *reachable* if $\exists \rho \in \llbracket \mathcal{A} \rrbracket_R : \rho = \langle \ell_0, u_0 \rangle \xrightarrow{\sigma} \langle \ell, u \rangle$.

Furthermore, we recall three essential properties for TLTS semantics [4, 79] derived from a TA, describing determinism for the passage of time. *Time additivity*

describes that we reach the same state when performing a delay $d = d_1 + d_2$ at once as when performing d_1 and d_2 subsequently. By *time reflexivity* we denote that we cannot change the state with a 0-delay. *Time determinism* refers to the fact that performing a delay in a given state always results in reaching the same target state. Obviously, these properties do not hold for the weak TLTS semantics.

Proposition 2.1. Let $(S, s_0, \hat{\Sigma}, \rightarrow)$ be a TLTS derived from a TA.

- (Time Add) $\forall s_1, s_3 \in S, \forall d_1, d_2 \in \Delta : s_1 \xrightarrow{d_1+d_2} s_3 \Leftrightarrow \exists s_2 : s_1 \xrightarrow{d_1} s_2 \xrightarrow{d_2} s_3$
- (Time Reflex) $\forall s_1, s_2 \in S : s_1 \xrightarrow{0} s_2 \Rightarrow s_1 = s_2$
- (Time Determ) $\forall s_1, s_2, s_3 \in S : s_1 \xrightarrow{d} s_2$ and $s_1 \xrightarrow{d} s_3$ then $s_2 = s_3$

Proof. Correctness of Proposition 2.1 directly follows from Definition 2.18. □

Example 2.12 (Timed Labeled Transition System of the PPU). Figure 2.9 gives an example in terms of the TLTS of the TA depicted in Figure 2.3. Here, states are depicted by a location and clock valuations written in brackets. Additionally, transitions are denoted by arrows being labeled with actions and delays.

The initial state consists of location *init* (as this is the initial location of the TA) and all clocks set to 0. The first few states are only able to perform delays as the only outgoing switch of *init* requires $x \geq 10$ (and $x \leq 15$). Note, that we left out several states (e.g., between $\langle \text{init}, x = 2 \wedge y = 2 \rangle$ and $\langle \text{init}, x = 10 \wedge y = 10 \rangle$) and transitions (e.g., the transition labeled with delay 2, 3, 4, ... starting in $\langle \text{init}, x = 0 \wedge y = 0 \rangle$ and subsequent states) for the sake of readability. Between 10 and 15 seconds after starting the system, we are able to *rotate* as the guard of the respective switch is now enabled. Afterwards we may wait for up to 20 seconds to *pick* the workpiece in states containing location *stack*, but we have to leave the *stack* after 20 seconds at the latest due to the invariant. After *picking* the workpiece, we always reach state $\langle \text{WP picked}, x = 0 \wedge y = 0 \rangle$ due to the reset of both clocks. Finally, we may *shut down* the PPU if we reach $\langle \text{stack}, x = 30 \wedge y = 30 \rangle$ (see the topmost state containing location *stack*). Afterwards, x and y have a different value as only x is reset.

Unfortunately, TLTS are only of theoretical interest and not of practical use as the state space is, in general, infinite. Therefore, important properties such as reachability are, in general, not decidable on TLTS [38]. Hence, we require a different formalism for the purpose of test-case generation. For TA, there are two formalisms using symbolic representations of the state space: *region graphs* [13, 10] and *zone graphs* [82]. As region graphs are finite, they are often utilized for proving decidability of particular properties (e.g., language emptiness [10] and timed bisimulation [56]). However, region graphs suffer from state-space explosion as the number of states is exponential in the number of clocks as well as the greatest constant appearing in the clock constraints of a TA [38].

Hence, we use zone graphs instead for a finite representation of TA semantics, having in general a smaller state space as compared to region graphs. A *symbolic*

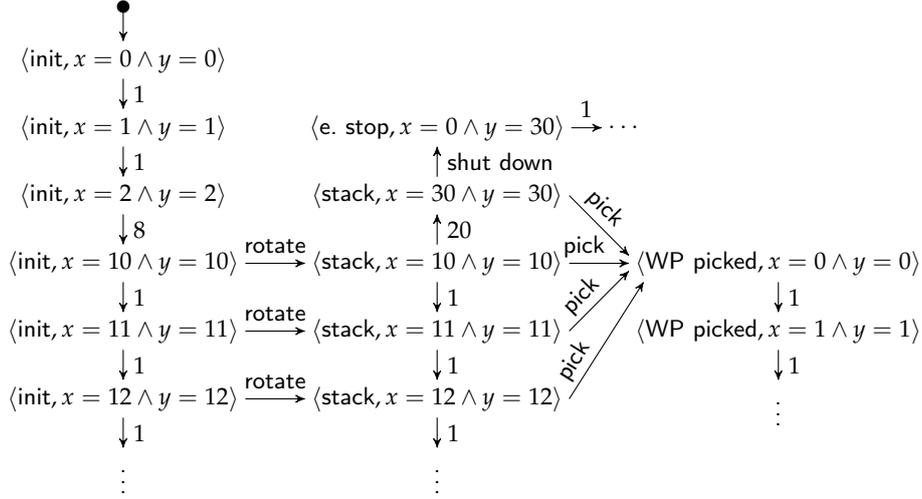


Figure 2.9: TLTS for the Extract of the PPU (see TA Depicted in Figure 2.3)

state $\langle \ell, \varphi \rangle$ of a zone graph of TA \mathcal{A} consists of a location ℓ and a zone $\varphi \in \mathcal{B}(C)$ [82, 36]. Hence, a zone is a clock constraint representing a potentially infinite set D of clock valuations satisfying φ . Here, we assume D to be *closed under entailment*, meaning that φ cannot be strengthened without changing D (resulting in a unique φ for each D).

Example 2.13. For instance, consider the clock constraint

$$\varphi := c \leq 1 \wedge c' \leq 2 \wedge c = c'$$

with clocks $C = \{c, c'\}$ and time domain $\mathbb{T}_C = \mathbb{N}_0$, such that $D = \{c = 0 \wedge c' = 0, c = 1 \wedge c' = 1\}$. It should be noted that $c = 1 \wedge c' = 2 \notin D$ due to $c = c'$. Hence, clock constraint φ can be strengthened to

$$\varphi' := c \leq 1 \wedge c' \leq 1 \wedge c = c'$$

without changing D , such that φ' is closed under entailment.

As usually done in related work (e.g., [38]), we use φ and D interchangeably. Furthermore, the construction of a zone graph is based on two operations.

- D^\uparrow denotes the *future* of zone D . Here, we utilize $u + d$ for $u \in D$ and $d \in \mathbb{T}_C$ to denote that we add d to the valuation $u(c)$ of each clock $c \in C$. This coincides with removing all upper bounds from φ (i.e., replacing all constraints of the form $c < n$ and $c \leq n$ by true for $c \in C$ and $n \in \mathbb{T}_C$).
- $R(D)$ denotes the *reset* of clocks in $R \subseteq C$ on zone D . Intuitively, this coincides with removing all inequalities containing clocks $c \in R$ from φ and conjugating constraints $c = 0$ for each $c \in R$ to φ .

Please note, that we apply logical operator (e.g., conjunction) on the set D as we assume that D always has an underlying constraint φ . We solely use D as it

sometimes allows us to formulate properties and definitions in a more concise way (and as the set D is usually applied for symbolic TA semantics in related work, e.g., [38]).

Definition 2.19 (Zone). Let $\varphi \in \mathcal{B}(C)$ be a clock constraint. In the context of semantics of TA, we call φ a *zone* where D is the set of clock valuations satisfying φ such that

$$u \in D \Leftrightarrow u \in \varphi$$

with $u \in \varphi$ being defined according to Definition 2.17. In the remainder of this thesis, we use φ and D interchangeably. We apply the following operations on zones:

- $D^\uparrow = \{u + d \mid u \in D, d \in \mathbb{T}_C\}$ denotes the *future* of zone D and
- $R(D) = \{[R \mapsto 0]u \mid u \in D\}$ denotes the *reset* of clocks in $R \subseteq C$ on zone D .

Note, that zones also contain difference constraints for each pair of clocks (as indicated in Section 2.2) to keep track of clock differences. This is necessary to not lose information. For instance, consider two clocks x and y . Here, clock constraints $x \leq 10 \wedge y \geq 7$ and $x \leq 10 \wedge y \geq 7 \wedge x - y = 0$ have different sets of solutions, where the latter clock constraint with difference constraint also describes the relation between x and y . Moreover, a reset does not remove difference constraints from φ (as indicated above) but rather updates the difference. Assume, that we reset clock $c \in C$. Then, the minimum difference of c and any other clock $c' \in C$ increases by the minimum value of c allowed by φ , and the maximum difference between c and c' increases by the maximum value of c allowed by φ . Furthermore, recall that we can apply logical operator (e.g., conjunction) on the set D as we assume that D always has an underlying constraint φ . Additionally, each TA has a unique corresponding zone graph as we assume D to be closed under entailment (see above).

Definition 2.20 (Zone Graph). The *zone graph* of TA $\mathcal{A} = (L, \ell_0, \Sigma, C, I, E)$ is a tuple $(Z, z_0, \rightsquigarrow)$, where

- $Z = L \times \mathcal{B}(C)$ is a set of *symbolic states*,
- $z_0 = \langle \ell_0, D_0 \rangle \in Z$ is the *initial state*, and
- $\rightsquigarrow \subseteq Z \times Z$ is a *symbolic transition relation* being the least relation satisfying the following rules:
 - $\langle \ell, D \rangle \rightsquigarrow \langle \ell, D^\uparrow \wedge I(\ell) \rangle$ and
 - $\langle \ell, D \rangle \rightsquigarrow \langle \ell', R(D \wedge g) \wedge I(\ell') \rangle$ if $(\ell, g, \sigma, R, \ell') \in E$.

We refer to the zone-graph semantics of TA \mathcal{A} by $\llbracket \mathcal{A} \rrbracket_Z$.

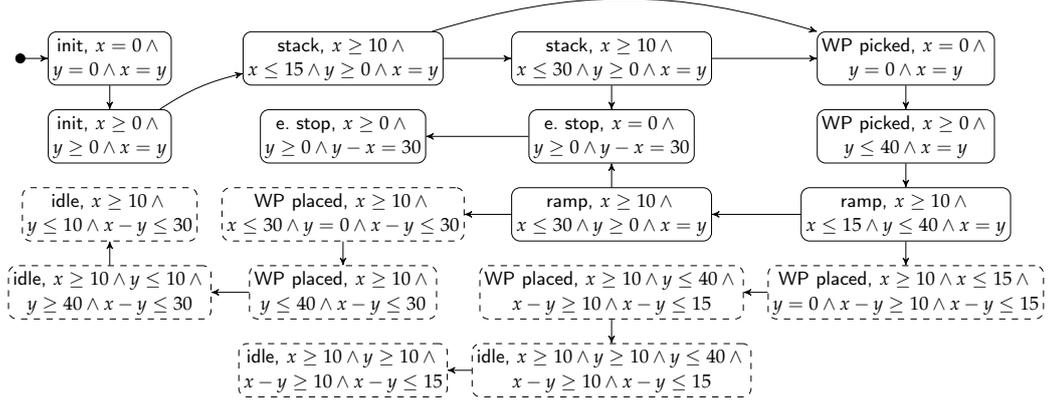


Figure 2.10: Zone Graph for the Extract of the PPU (see TA Depicted in Figure 2.3)

Example 2.14 (Zone Graph of the PPU). Figure 2.10 gives an example for the zone graph of the PPU depicted in Figure 2.3. Here, states consist of a location followed by a zone. Note, that we had to leave out some transitions for the sake of readability. In particular, all symbolic states with a dashed border have an additional outgoing transition to the initial state at the top left.

The initial symbolic state consists of location *init* (as this is the initial location of the corresponding TA) and all clocks set to 0. Then, there is a transition to *init* with $x \geq 0 \wedge y \geq 0$ due to the first clause for the symbolic transition relation (i.e., we apply the future operation, see Definition 2.20). Here, only the second state has a transition to a state comprising location *stack* as the constraint of the initial state (i.e., $x = 0 \wedge y = 0$) does not fulfill the guard of the respective switch in the TA. Hence, the second clause for the symbolic transition relation cannot be applied. In contrast, the symbolic state comprising location *stack* and clock constraint $x \geq 10 \wedge x \leq 15$ has two outgoing transitions as both clauses are applicable. Furthermore, the two symbolic states containing location *ramp* target different states with location *WP placed* as applying the second clause of Definition 2.20 yields different zones. Finally, all states with a dashed border have a transition targeting the initial state as the corresponding switches in the TA reset x and y , resulting in an equal zone. In case the resulting zones would not be equal, we would need to apply loop unrolling until we reach a fixed point in terms of equal zones.

Note, that zone graphs as defined above are, in general, not finite. For instance, consider the TA \mathcal{A} in Figure 2.11a and its zone graph $\llbracket \mathcal{A} \rrbracket_Z$ in Figure 2.11b. \mathcal{A} consists of clocks x and y and two switches where the self-loop of ℓ_0 only resets x . As the invariant only restricts x , this self-loop may be used arbitrarily often. Hence, the difference between x and y grows with each usage of the self-loop, thus resulting in distinct difference constraints and additional states in the zone graph. This can be seen in the extract of $\llbracket \mathcal{A} \rrbracket_Z$ where each row of states contains difference constraints with greater difference, increasing by 2 as the switch labeled with a can only be used after exactly two seconds (due to invariant and guard). Here, the dots on the bottom left side indicate that infinitely many states follow. Please note,

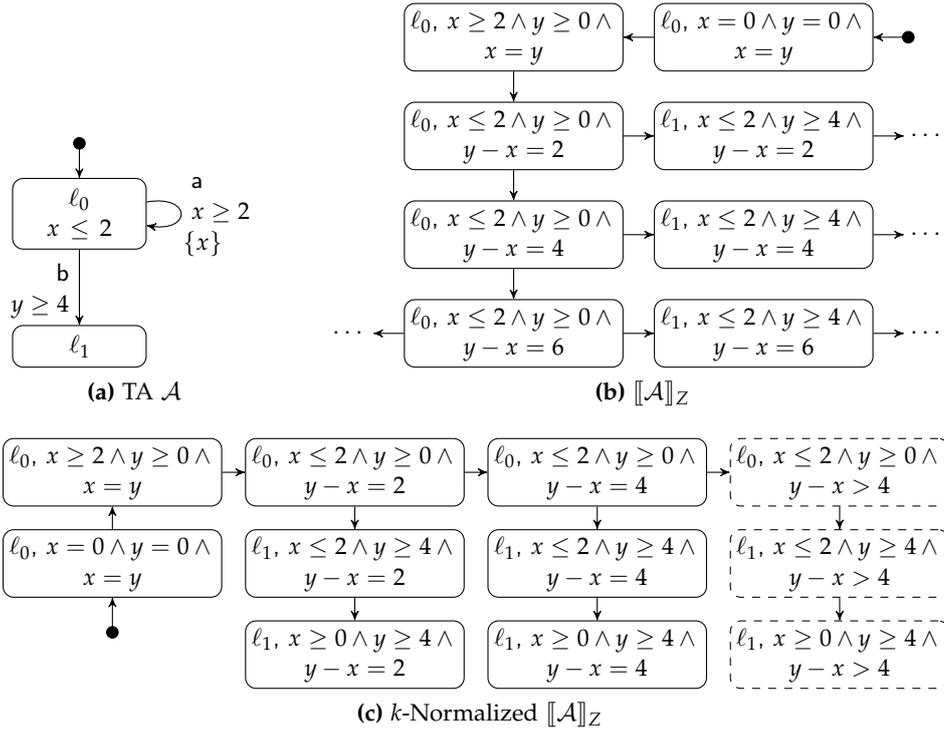


Figure 2.11: Example for k -Normalization

that $\llbracket \mathcal{A} \rrbracket_Z$ in Figure 2.11b does not contain states with $x = 0$ (i.e., states being the result of the reset of the self-loop) to improve readability. Normally, these states are included due to Definition 2.20 between all states on the left side.

To solve the issue of zone graphs with infinite state spaces, we apply so-called *k-normalization* [161, 167]. The basic idea is setting k to the greatest constant appearing in any clock constraint in the TA, resulting in $k = 4$ in \mathcal{A} (due to the guard $y \leq 4$). Then, we replace each difference constraint with a difference greater than k . For our example TA \mathcal{A} , Figure 2.11c shows the k -normalized zone graph. Here, the states with the dashed border contain the k -normalized difference constraint $y - x > 4$ as we know at this point that the difference between x and y is greater than 4. Therewith, we know that we may use the switch of \mathcal{A} labeled with b as we know that the difference is greater than the constant of the guard $y \geq 4$. It should be noted that for the rest of this thesis, we assume k -normalized zone graphs.

2.5.2 Semantics of Featured Timed Automata

For describing the behavior of an FTA \mathcal{F} , we start by considering the operational semantics which can be obtained in two steps. First, we derive the set of all variants $\llbracket \mathcal{F} \rrbracket_F$. Then, we derive TLTS $\llbracket \mathcal{A} \rrbracket_S$ for each $\mathcal{A} \in \llbracket \mathcal{F} \rrbracket_F$.

Example 2.15. Consider the FTA \mathcal{F} depicted in Figure 2.7 together with the feature diagram in Figure 2.5. First we derive the set of all variants $\llbracket \mathcal{F} \rrbracket_F$

corresponding to the configurations Θ in Table 2.1. Second, we derive the TLTS semantics for these configurations. For instance, deriving the semantics for θ_2 results in the TLTS as depicted in Figure 2.9.

The symbolic semantics of an FTA \mathcal{F} may be derived in a similar fashion. Here, we again derive the set of all variants $\llbracket \mathcal{F} \rrbracket_F$. Then, we simply construct a zone graph for each of the variants independently. Alternatively, we exploit the product-line knowledge by integrating feature constraints into the zone graph, resulting in a *featured zone graph* [70]. Intuitively, this can be achieved by extending symbolic states of the zone graph by feature constraints. Then, feature constraints annotated to switches and featured clock constraint are conjugated to the feature constraints of symbolic states. This will be explained in more detail in Chapter 4.

2.5.3 Semantics of Parametric Timed Automata

Similarly to FTA, the operational semantics of PTA \mathcal{P} over P is obtained by applying parameter valuations $\nu : P \rightarrow \mathbb{T}_P$ to \mathcal{P} where $\nu(\mathcal{P})$ denotes the TA resulting from replacing each occurrence of every $p \in P$ within parametric clock constraints of \mathcal{P} by the constant values $\nu(p)$ (see Definition 2.13). Then, we derive the corresponding TLTS semantics $\llbracket \nu(\mathcal{P}) \rrbracket_S$ for the resulting TA $\nu(\mathcal{P})$.

Example 2.16. Consider the PTA \mathcal{P} depicted in Figure 2.8. First we derive the (infinite) set of all variants $\nu(\mathcal{P})$. Second, we can derive the TLTS semantics for these configurations. For instance, deriving the semantics for the variant with $a = 15$ and $b = 30$ results in the TLTS as depicted in Figure 2.9.

Similar to FTA, the symbolic semantics of a PTA \mathcal{P} can be described by first deriving all variants and then constructing the zone graph of each variant. Unfortunately, a PTA comprises (in general) an infinite number of variants in addition to the operational semantics of every variant also being infinite. Hence, we again exploit product-line knowledge by integrating constraints over parameters into the zone graph, resulting in a *parametric zone graph* [103]. Intuitively, this can be achieved by accumulating constraints over parameters whenever we traverse a parametric clock constraint while constructing a parametric zone graph. These constraints are then integrated into symbolic states of the symbolic semantics. This will also be explained in more detail in Chapter 4. Now that we have given an introduction to operational and symbolic semantics of TA, FTA, and PTA, we proceed with an introduction to model-based coverage-driven test-case generation.

2.5.4 Testing Real-Time Systems

One of the critical tasks in software engineering is testing, describing “the process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or component.” [1] As we use TA-based models to specify time-critical systems, we consider *model-based testing* as an approach for systematic automated coverage-based test-case generation, test execution, and evaluation of test results based

on formal modeling languages [190]. A *test case* imitates behavior of the environment of the *implementation under test (IUT)* in terms of sequences of actions (or test inputs) injected into the IUT [190]. We then compare the reactions (or test outputs) of the IUT to the expected reactions. In model-based testing, these test cases are derived from our model (i.e., TA and TA-based formalisms in our case). Additionally, we need to consider the time intervals between actions and reactions as we apply testing on time-critical systems. Hence, an abstract untimed test case derived from a TA corresponds to an *untimed trace*, whereas a concrete test-case execution on an IUT (i.e., a *timed run*) also depends on delays between test steps [54]. Furthermore, we apply a *coverage criterion* (i.e., a criterion describing a particular type of test goals) to determine if we need to add additional test cases to our *test suite* [190] (i.e., the set of test cases). Namely, we use location coverage as a basic coverage criterion where each location needs to be reached by at least one test case.

Example 2.17. Consider the TA depicted in Figure 2.3. Here, we first derive an abstract untimed test case to reach location *WP placed*, being given by the sequence *rotate, pick, rotate, place*. A concrete test-case execution may, for instance, be

- waiting for 12 seconds and *rotating* the crane,
- waiting for 8 seconds and *picking* the workpiece,
- waiting for 14 seconds and *rotating* the crane again, and
- waiting for 9 seconds and *placing* the workpiece.

This test case also covers the locations *stack, WP picked, ramp* and (obviously) *init*. Hence, for a test suite satisfying location coverage, we require additional test cases for *emergency stop* and *idle*.

With TLTS semantics, we are able to derive *timed test cases* for TA as a test case simply corresponds to the delays and actions of a timed run. For instance, consider the test case as described in Example 2.17. The corresponding timed run is $init \xrightarrow{(12, rotate)} stack \xrightarrow{(8, pick)} WP\ picked \xrightarrow{(14, rotate)} ramp \xrightarrow{(9, place)} WP\ placed$. This results in the test case $t = (12, rotate), (8, pick), (14, rotate), (9, place)$.

Considering test cases in the context of product lines, the goal is to systematically reuse test cases among different configurations by symbolically accumulating configuration-specific information during test-case generation [53]. More specifically, each test case should be accompanied by a *presence condition* [58] describing the set of configurations for which the test case is applicable. Therewith, we cover each test goal in every variant. For instance, there might be a test case $t = (12, rotate), (8, pick), (14, rotate), (9, place)$ for the FTA depicted in Figure 2.7. A possible presence condition could be $m \wedge \neg Resume$ (where m denotes the feature model). Here, we require m to ensure that we only consider valid configurations. Hence, t would be applicable to all four configurations where *Resume* is deselected (see Table 2.1).

Presence conditions for PTA may be handled similarly to FTA. For instance, applying test case t to the PTA in Figure 2.8 could be done with presence condition (i.e., a condition over parameters in case of PTA) $a \geq 14 \wedge b \geq 24$. Therefore, t is applicable to all variants with $a \geq 14$ and $b \geq 24$, resulting in an infinite number of configurations covered by a single test case.

At this point, the next research challenge arises. Here, we need to adapt the methodology for family-based test-case generation to our novel model incorporating feature variability as well as parametric variability to model product lines with unbounded parametric real-time constraints (see Research Challenge 1). Hence, we combine the ideas of featured zone graphs and parametric zone graphs. Additionally, we adapt presence conditions of test cases to include both feature constraints and parametric constraints.

Research Challenge 3.1

Family-based test-suite generation for basic coverage criteria of real-time systems with a potentially infinite number of variants.

Up this point, we cover each test goal in every variant, without having any further requirements for test cases. Therefore, we want to extend our test-case generation methodology by a novel coverage criterion systematically considering best-case/worst-case execution time (BCET/WCET) behavior [203]. Therewith, we may investigate time-critical behavior more effectively by requiring two valid test cases (i.e., test cases being applicable to at least one configuration) for each location, one for each the minimum and the maximum delay w.r.t. the given real-time constraints [74]. For instance, this approach may reveal off-by-one timing errors [6] where boundary-value behavior w.r.t. delays is faulty. Therefore, we have to extend our family-based test-case generation methodology (see Research Challenge 3.1), accordingly. This results in the following research challenge.

Research Challenge 3.2

Family-based test-suite generation for enhanced coverage criteria w.r.t. best-case/worst-case execution times of real-time systems with a potentially infinite number of variants.

These two research challenges will be tackled in Chapter 5. Up to this point, we gave an introduction to basics for modeling, testing, and sampling (product lines of) time-critical systems. Next, we tackle the issue of verification time-critical systems in terms of bisimulation. Bisimulation may be applied to check whether two systems have the same behavior.

2.6 BISIMULATION OF REAL-TIME SYSTEMS

In this section, we are concerned with the verification of real-time systems. Here, a central challenge is comparing the behavior of two systems against each other. For instance, this may be applied in the development process of model-driven software

engineering. Here, we have an initial model describing the specified behavior. During the development process, the model is further and further adapted (e.g., there may be restructuring). Throughout this process, we have to ensure that the (time-critical) behavior of our model remains the same as in the original model, such that we do not accidentally introduce faults. Hence, we compare the resulting model of each development step with the original model. This can be achieved with so-called (*timed*) *bisimulation*, where we compare the behavior of the adapted model with the original model step by step for corresponding pairs of states. Moreover, we can check for *similarity* in settings where the task is to ensure that the adapted model contains *at least* or *at most* the behavior of the original model (instead of exactly the same behavior in case of bisimilarity). In fact, we check bisimilarity for TA \mathcal{A} and \mathcal{A}' by checking if \mathcal{A} is similar to \mathcal{A}' , and vice versa.

In addition to this kind of development process, we may apply timed bisimulation also for testing purposes. So far, we have seen the basics for *black-box* testing in Section 2.5 where a test case is a timed trace (i.e., a sequence of delays and actions). Hence, we do not compare behavior of states such that we do not need to know the internal structure of our implementation. In contrast, (timed) bisimulation may be utilized as a means of *white-box* testing where the models of the implementation as well as the specification are available. Therewith, we may compare behavior of states instead of only considering sequences of delays and actions. However, it should be noted that checking timed bisimilarity is more complex than the black-box approach introduced in Section 2.5. In particular, timed bisimulation cannot be checked by applying testing [3].

In the remainder of this section, we present preliminaries for checking (timed) bisimilarity. We start by giving an intuition for *untimed* bisimulation of *Labeled Transition Systems (LTS)* [111].

Definition 2.21 (Labeled Transition System). A *labeled transition system (LTS)* is a tuple $\mathcal{L} = (S, s_0, \Sigma, \rightarrow)$, where

- S is a set of *states*,
- $s_0 \in S$ is the *initial state*,
- Σ is a set of actions with $S \cap \Sigma = \emptyset$, and
- $\rightarrow \subseteq S \times \Sigma \times S$.

We use $s \xrightarrow{\sigma} s'$ to denote $(s, \sigma, s') \in \rightarrow$, where $\{s, s'\} \subseteq S$ and $\sigma \in \Sigma$.

We apply the following shorthand notations for LTS (being similar to the TLTS notations in Notation 2.3).

Notation 2.4 (LTS Notations). Let $\mathcal{L} = (S, s_0, \Sigma, \rightarrow)$ be an LTS. We utilize the following notations.

- $s \xrightarrow{\sigma}$ if $\exists s' \in S : s \xrightarrow{\sigma} s'$ and $\sigma \in \Sigma$,

- $s_0 \xrightarrow{\sigma_1, \dots, \sigma_n} s_n$ if $\exists s_1, \dots, s_{n-1} \in S : s_0 \xrightarrow{\sigma_1} s_1 \xrightarrow{\sigma_2} \dots \xrightarrow{\sigma_n} s_n$ with $\{\sigma_1, \dots, \sigma_n\} \subseteq \Sigma$,
- $s_0 \xrightarrow{\sigma_1, \dots, \sigma_n} s_n$ if $\exists s_n \in S : s_0 \xrightarrow{\sigma_1, \dots, \sigma_n} s_n$ with $\{\sigma_1, \dots, \sigma_n\} \subseteq \Sigma$,
- $s \xrightarrow{\sigma} s'$ if $\exists s_1, s_n \in S : s \xrightarrow{\tau^m} s_1 \xrightarrow{\sigma} s_2 \xrightarrow{\tau^n} s'$ with $\sigma \in \Sigma$ and $\{m, n\} \in \mathbb{N}_0$,
- $s \xrightarrow{\sigma} s'$ if $\exists s' \in S : s \xrightarrow{\sigma} s'$ and $\sigma \in \Sigma$,
- $s_0 \xrightarrow{\sigma_1, \dots, \sigma_n} s_n$ if $\exists s_1, \dots, s_{n-1} \in S : s_0 \xrightarrow{\sigma_1} s_1 \xrightarrow{\sigma_2} \dots \xrightarrow{\sigma_n} s_n$ with $\{\sigma_1, \dots, \sigma_n\} \subseteq \Sigma$, and
- $s_0 \xrightarrow{\sigma_1, \dots, \sigma_n} s_n$ if $\exists s_n \in S : s_0 \xrightarrow{\sigma_1, \dots, \sigma_n} s_n$ with $\{\sigma_1, \dots, \sigma_n\} \subseteq \Sigma$.

Here, $s_0 \xrightarrow{\sigma_1, \dots, \sigma_n} s_n$ describes a *strong path* consisting of *strong steps* $s_i \xrightarrow{\sigma_{i+1}} s_{i+1}$ with $0 \leq i < n$. We obtain *weak paths* and *weak steps* by replacing \rightarrow with \Rightarrow . The corresponding *trace* is $\sigma_1, \dots, \sigma_n$.

Two states of a transition system are considered bisimilar if actions of every outgoing transition of one state can be simulated by the other state (and vice versa) [157]. Furthermore, the states reached after these actions must, again, be bisimilar. Here, two transition systems are bisimilar if their initial states are bisimilar. Furthermore, we differentiate between weak and strong timed bisimulation due to differences between weak and strong steps as caused by internal behavior. Here, weak bisimulation considers weak steps where internal behavior is unobservable. Moreover, LTS \mathcal{L} and LTS \mathcal{L}' are bisimilar if \mathcal{L} simulates \mathcal{L}' and vice versa (i.e., if the simulation relation is symmetrical) [157].

Definition 2.22 (LTS Bisimilarity). Let $\mathcal{L} = (S, s_0, \Sigma, \rightarrow)$ and $\mathcal{L}' = (S', s'_0, \Sigma', \rightarrow')$ be LTS and $\mathcal{R} \subseteq S \times S'$. Then \mathcal{L}' *strongly simulates* \mathcal{L} if it holds for all $(s_1, s'_1) \in \mathcal{R}$ that

$$s_1 \xrightarrow{\mu} s_2 \Rightarrow \left(s'_1 \xrightarrow{\mu} s'_2 \wedge (s_2, s'_2) \in \mathcal{R} \right)$$

where $\mu \in \hat{\Sigma}$, and $(s_0, s'_0) \in \mathcal{R}$. \mathcal{L}' *weakly simulates* \mathcal{L} if it holds for all $(s_1, s'_1) \in \mathcal{R}$ that

$$s_1 \xRightarrow{\mu} s_2 \Rightarrow \left(s'_1 \xRightarrow{\mu} s'_2 \wedge (s_2, s'_2) \in \mathcal{R} \right)$$

and $(s_0, s'_0) \in \mathcal{R}$. We use $\mathcal{L} \sqsubseteq \mathcal{L}'$ to denote that \mathcal{L}' *weakly/strongly simulates* \mathcal{L} . \mathcal{L}' and \mathcal{L} are *weakly/strongly bisimilar*, denoted by $\mathcal{L} \simeq \mathcal{L}'$, iff \mathcal{R} is symmetric.

Example 2.18 (Untimed Bisimulation). Consider the transition systems S_1 and S_2 depicted in Figure 2.12. Both systems comprise the set of traces $\{\epsilon, a, ab, ac\}$ such that they are trace equivalent. Hence, trace equivalence does not consider the branching in s_0 and s_6 , respectively (i.e., the decision for producing b or c is made at different states). This may be problematic as the two LTS seem to behave equal although the decision for performing action b or c is done at different points. For bisimulation, we temporarily relate the initial states as

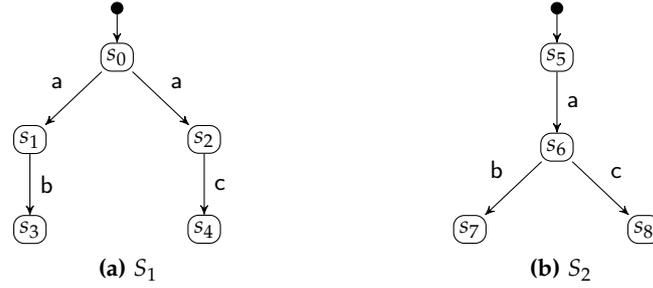


Figure 2.12: Example for Untimed Bisimulation

both have the same behavior in terms of only being able to produce action a , such that $\mathcal{R} = \{(s_0, s_5)\}$. Afterwards, we check if the target states after action a are also in relation. Here, s_6 simulates s_1 and s_2 as s_6 is able to produce b and c . However, neither s_1 nor s_2 can simulate s_6 . Therefore, we cannot add another pair of states to \mathcal{R} . Instead, we remove the pair (s_0, s_5) of initial states from \mathcal{R} such that $\mathcal{R} = \emptyset$ as the states reached after performing action a are not bisimilar. Hence, S_1 and S_2 are not bisimilar.

Next, we consider the notion of *timed bisimulation* [148, 204]. In contrast to untimed bisimilarity, we have to make sure that we additionally check the time spans in which actions are allowed to happen. This is particularly challenging as the allowed time span may depend on the time that preceding actions required to be executed. For instance, consider the TA depicted in Figure 2.4. Here, the allowed span for *picking* up the workpiece depends on how long it took to *rotate to the stack* (i.e., how much time is left until the invariant of location *stack* is violated).

Example 2.19 (Timed Bisimulation). Consider the TA \mathcal{A} depicted in Figure 2.3 as the original model of a development process and the TA $_{\tau}$ \mathcal{A}'_{τ} depicted in Figure 2.4 as a model adapted during development. There are several differences between these TA as pointed out in Example 2.3. For instance, \mathcal{A}'_{τ} only comprises one clock instead of the two clocks of \mathcal{A} , there is a τ -step in \mathcal{A}'_{τ} , and several clock constraints are different. However, \mathcal{A} and \mathcal{A}'_{τ} have similar behavior. When considering weak steps, both systems may initially *rotate* after 10 to 15 seconds and then have to *pick* the workpiece after at most 30 seconds after starting the system. Here, \mathcal{A}'_{τ} *picks* the workpiece after at most 30 seconds as there is a reset of z after exactly 10 seconds when executing the switch labeled with τ . Thereafter, both systems have again 10 to 15 seconds for the next rotation. Additionally, the behavior after both *shut downs* is the same although there is no reset in \mathcal{A}'_{τ} as location *emergency stop* neither has an invariant nor an outgoing transition. Hence, these systems are weak timed bisimilar such that the adapted model is correct w.r.t. the original model (but not strong timed bisimilar due to the τ -step).

With comparisons in terms of timed bisimilarity, we can address the challenge introduced at the beginning of this section: We may compare models having been adapted during the development process w.r.t. the original model, and we

may apply timed bisimulation for white-box testing scenarios where models of the specification as well as the implementation are available. In order to check timed bisimilarity, we consider the semantics of our systems. When using TLTS, we can in fact apply the same strategy as described for untimed bisimulation in the beginning of this section. Here, we consider the label alphabet $\hat{\Sigma}$. Then, we require a symmetrical relation between states such that related states have outgoing transitions with the same behavior and the target states are again bisimilar. As described for LTS bisimulation, TA \mathcal{A}_τ and TA \mathcal{A}'_τ are bisimilar if \mathcal{A}_τ simulates \mathcal{A}'_τ and vice versa. Hence, the TA models are bisimilar if the simulation relation is symmetrical. This holds as a TLTS is just an LTS with an extended label alphabet $\hat{\Sigma}$.

Definition 2.23 (Timed Bisimilarity). Let \mathcal{A}_τ and \mathcal{A}'_τ be TA_τ with $\llbracket \mathcal{A}_\tau \rrbracket_S = (S, s_0, \hat{\Sigma}, \rightarrow)$, $\llbracket \mathcal{A}'_\tau \rrbracket_{S'} = (S', s'_0, \hat{\Sigma}, \rightarrow')$, and $\mathcal{R} \subseteq S \times S'$. Then \mathcal{A}'_τ *strongly timed simulates* \mathcal{A}_τ if it holds for all $(s_1, s'_1) \in \mathcal{R}$ that

$$s_1 \xrightarrow{\mu} s_2 \Rightarrow \left(s'_1 \xrightarrow{\mu} s'_2 \wedge (s_2, s'_2) \in \mathcal{R} \right)$$

where $\mu \in \hat{\Sigma}$ and $(s_0, s'_0) \in \mathcal{R}$. \mathcal{A}'_τ *weakly timed simulates* \mathcal{A}_τ if it holds for all $(s_1, s'_1) \in \mathcal{R}$ that

$$s_1 \xrightarrow{\mu} s_2 \Rightarrow \left(s'_1 \xrightarrow{\mu} s'_2 \wedge (s_2, s'_2) \in \mathcal{R} \right)$$

and $(s_0, s'_0) \in \mathcal{R}$. We use $\mathcal{A}_\tau \sqsubseteq \mathcal{A}'_\tau$ to denote that \mathcal{A}'_τ *weakly/strongly timed simulates* \mathcal{A}_τ . \mathcal{A}'_τ and \mathcal{A}_τ are *weakly/strongly timed bisimilar*, denoted by $\mathcal{A}_\tau \simeq \mathcal{A}'_\tau$, iff \mathcal{R} is symmetric.

Example 2.20 (Timed Bisimilarity of the PPU). The TLTS $\llbracket \mathcal{A} \rrbracket_S$ and $\llbracket \mathcal{A}'_\tau \rrbracket_{S'}$ in Figures 2.9 and 2.13 (corresponding to the TA in Figures 2.3 and 2.4) gives an example for timed bisimilarity. As pointed out in Example 2.19, it holds that $\mathcal{A} \simeq \mathcal{A}'_\tau$ in case of weak timed bisimilarity. This can be checked by applying Definition 2.23. To investigate bisimilarity, we start by assuming $(\langle \text{init}, x = 0 \wedge y = 0 \rangle, \langle \text{init}, z = 0 \rangle) \in \mathcal{R}$. Next, we find for each outgoing transition of both TLTS a matching transition having the same label in the other TLTS. Here, both systems only have one outgoing transition labeled with the same delay. Hence, we relate the subsequent states such that $(\langle \text{init}, x = 1 \wedge y = 1 \rangle, \langle \text{init}, z = 1 \rangle) \in \mathcal{R}$, and we do the same for the next states. Furthermore, $(\langle \text{init}, x = 10 \wedge y = 10 \rangle, \langle \text{internal}, z = 0 \rangle) \in \mathcal{R}$ as $\langle \text{internal}, z = 0 \rangle$ is reached through an unobservable transition labeled with τ and both state have the same action labels on outgoing transitions. Similarly, we relate the subsequent states comprising location *internal*. When doing this for all state pairs of these (infinitely branching) TLTS, we observe that $(\langle \text{init}, x = 0 \wedge y = 0 \rangle, \langle \text{init}, z = 0 \rangle) \in \mathcal{R}$ still holds such that \mathcal{R} is symmetric and $\mathcal{A} \simeq \mathcal{A}'_\tau$ w.r.t. weak timed bisimulation. *Strong* timed bisimulation obviously does not hold as no state of $\llbracket \mathcal{A}_\tau \rrbracket_S$ comprising location *init* has an outgoing transition labeled with *rotate*.

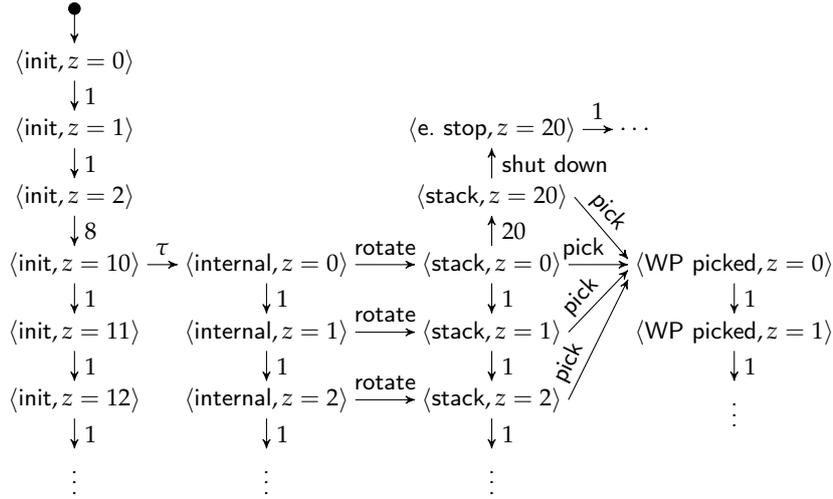


Figure 2.13: TLTS for the Adapted Extract of the PPU (see TA_τ Depicted in Figure 2.4)

Unfortunately, TLTS are, in general, not finite. Therefore, we require a different formalism for effectively checking timed bisimilarity. Therewith, we may apply timed bisimulation to model-driven development such that refined models can be checked against the original model, ensuring that the observable behavior remains unchanged. It should be noted that Čerāns [56] describes a decidable check for timed bisimilarity on TA by utilizing region graphs. However, region graphs suffer from state-space explosion (see Section 2.5). Hence, we want to apply a zone-graph based approach for effectively checking timed bisimilarity.

Research Challenge 4.1

Effectively checking timed bisimilarity of TA.

Furthermore, we also consider timed bisimilarity for *product lines* of TA. Here, we are concerned with the model-driven development process of families of similar systems. Specifically, we want to ensure that the behavior of *each variant* remains unchanged without deriving these variants and checking them in a variant-by-variant fashion. Hence, we lift the concepts of TA bisimulation to product lines with unbounded parametric real-time constraints.

Research Challenge 4.2

Checking timed bisimilarity of product lines with a potentially infinite number of variants.

These two research challenges will be tackled in Chapter 6. Therewith, we conclude the preliminaries. To sum up, we utilized this chapter to introduce the basics for the remainder of this thesis and motivate our research challenges. Namely, we introduced the PPU as our case study and the formalism of timed automata for modeling real-time systems with discrete-state/continuous-time behavior. Thereafter, we presented two approaches for modeling families of time-critical systems

(FTA and PTA). Moreover, we established the notion of sampling to derive a subset of variants from a product line. Furthermore, we gave an introduction to model-based coverage-driven test-case generation for real-time systems and product lines of real-time systems. Finally, we tackled the issue of timed bisimulation. In the next chapter, we introduce a solution for modeling product lines with unbounded parametric real-time constraints (see Research Challenge 1).

3

BEHAVIORAL MODELING OF PRODUCT LINES WITH CONFIGURABLE PARAMETRIC REAL-TIME CONSTRAINTS

In this chapter, we tackle Research Challenge 1 as introduced in the background chapter (see Section 2.3).

Research Challenge 1

Definition of a behavioral modeling formalism for real-time product lines with a potentially infinite number of variants.

So far, approaches for modeling families of similar TA presented in the background chapter (i.e., FTA and PTA) either use Boolean feature variability or unconstrained parametric variability. Therefore, FTA and PTA only comprise one of the following two techniques for expressing behavioral variability.

- FTA incorporate feature constraints allowing us to model dependencies between feature selections and presence, absence, and combinations of switches and clock constraints. Hence, FTA facilitate an explicit mapping between selections of features of a feature model and presence/absence of modeling components of TA. Hence, features guarantee traceability with these mappings.
- PTA incorporate parametric constraints. Hence, we may choose arbitrary values from a potentially infinite parameter domain for time intervals within clock constraints. As a result, we obtain a potentially infinite number of variants.

Table 3.1 gives additional details, summing up the comparison between FTA and PTA made above and in the background chapter (see Section 2.3). Here, we can see that the variability domain of FTA is finite, whereas parameters of PTA may have an infinite domain. Furthermore, FTA are configured by selecting and deselecting features according to the corresponding feature model. As a result, variants (i.e., TA) are derived by a projection where switches and featured clock constraint are removed if they do not satisfy the configuration. As opposed to FTA, PTA are configured by utilizing a parameter valuation where we select a value of the parameter domain \mathbb{T}_p for every parameter. Then, a TA is derived by substituting each parameter by the corresponding value. Additionally, FTA support specifying dependencies between features through feature models, which is not possible for PTA. Moreover, FTA facilitate traceability between artifacts in the problem space and the solution space. This is achieved by linking parts

Table 3.1: Comparison between FTA and PTA

Property	FTA	PTA
Variability domain	B (finite)	T_p (possibly infinite)
Configuration	feature (de-)selection	parameter valuation
Variant derivation	projection	parameter substitution
Dependencies	between features	none
Traceability	features	none
EF-decidability	decidable	semi-decidable

of FTA to the feature model by utilizing constraints over features. Again, this is not supported by PTA. Finally, we consider decidability of reaching given parts of a model (i.e., EF-decidability). Here, reachability for FTA is decidable as we can simply enumerate all variants and utilize zone graphs (see Definition 2.20) for checking reachability. In contrast to FTA, reachability is semi-decidable for PTA [21, 20]. Hence, if the decision procedure terminates, we obtain a correct result. However, the decision procedure does, in general, not terminate.

As a result, the formalisms of FTA and PTA are both useful, depending on the intended purpose of application. Furthermore, the application area for PTA is not covered by FTA (and vice versa), such that there exists no modeling formalism incorporating *both* feature variability and unbounded parametric variability to model product lines with unbounded parametric real-time constraints. In order to achieve this, we utilize feature models extended by attributes to so-called *extended feature models* [109]. In particular, we utilize constraints over unbounded parameters as attributes. As a consequence, a parameter constraint is enforced whenever the respective feature is selected. For instance, consider the following illustrating example for extending the PPU product line as presented in Section 2.3.

Example 3.1. Consider the previously introduced SPL of the PPU (see Section 2.3) where we want to add more precise restrictions. For instance, we would like to specify a fixed lower bound (as before) and a variable upper bound for *rotating* the crane. Here, the upper bound should not be freely configurable but the interval should depend on the selected type of *workpiece* (i.e., *plastic* or *metal*). Additionally, the maximum upper bound for *rotating* metal workpieces should be higher than the maximum upper bound for *plastic* workpieces (as *plastic* is lighter and has a lower risk of damaging the demonstrator). Furthermore, the lower bound for *rebooting* the plant after a *shutdown* should be freely configurable but not be less than 30 seconds.

To satisfy these requirements, we add the parameters a , b , and c to our model, where a and b are utilized for restricting *rotations* and c is applied for *rebooting*. In particular, we require $a \geq 15$ whenever we work on *plastic* pieces such that feature *plastic* includes the attribute $a \geq 15$. Moreover, feature *metal* contains the parameter constraint $a \geq 20 \wedge a \leq 25$ as attribute. As a result,

we may apply parameter a as an upper bound restricting the time interval for the *rotation* of the crane. Additionally, we utilize $c \geq 30$ as an attribute for feature *resume* such that the minimum time for *rebooting* the demonstrator is freely configurable (but takes at least 30 seconds). Therefore, we require Boolean features as well as numeric parameters to fulfill the requirements of this example.

In order to address Research Challenge 1 and to combine Boolean feature variability with unbounded parametric variability, we introduce a novel formalism enriching FTA by configurable parametric variability. We call this formalism *Configurable Parametric Timed Automata (CoPTA)*. Therewith we obtain a model that supports traceability between the problem space and the solution space by utilizing Boolean features while also supporting unbounded numeric variability of parameters. In particular, we achieve this with the following two steps:

1. In the problem space, we utilize constraints over unbounded parameters as attributes of features in extended feature models. Hence, whenever a feature is selected into a set of configurations, the (possibly) unbounded parameters must have a value within an assigned range in all of these configurations.
2. In the solution space, we extend the formalism of FTA, utilizing Boolean parameters in the form of features, by unbounded non-Boolean parameters. Therewith, we may apply behavioral restrictions with constraints over features as well as unbounded parameters.

The remainder of this chapter is structured as follows (see Figure 3.1 for an overview). Here, we already introduced the extension of TA to FTA and PTA in the background chapter. In this chapter, we first extend FTA by constraints over unbounded parameters, which is done in the problem space as well as the solution space (see ① in Figure 3.1).

- In the problem space, we apply extended feature models (see Section 3.1). This is achieved by annotating features (i.e., Boolean parameters) with attributes in terms of constraints over unbounded parameters. Hence, these unbounded parameters have a defined value within a given range whenever a corresponding feature is selected in a configuration.
- In the solution space, we extend FTA to CoPTA (see Section 3.2). In particular, we combine featured clock constraints with parametric clock constraints, allowing us to apply behavioral restrictions with features as well as unbounded parameters.

Second, we introduce a transformation of our novel formalism (see ② in Figure 3.1). In particular, we are concerned with transforming constraints over features into constraints over unbounded parameters such that we may utilize existing results of the PTA theory and an existing model checker for analyses (see Section 3.3). Finally, we give an overview on related work (see Section 3.4) and conclude the chapter (see Section 3.5). The contents of this chapter are based on the following publication:

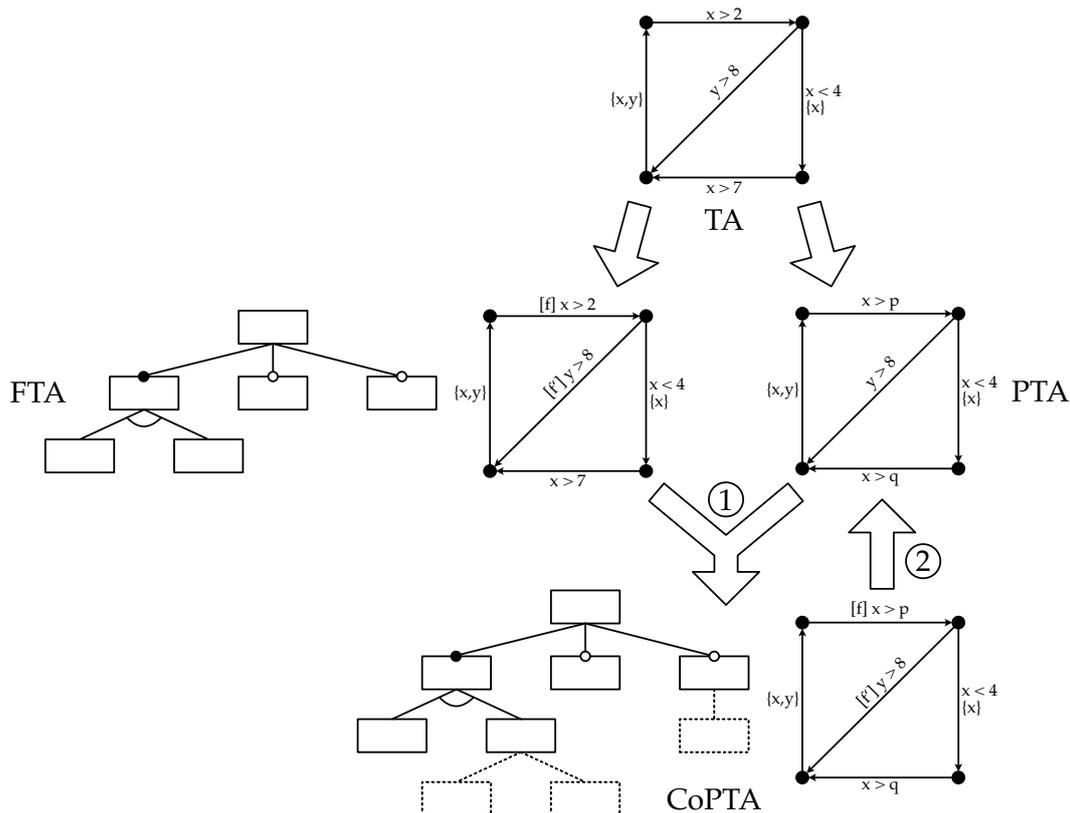


Figure 3.1: Overview on the Contributions Presented in Chapter 3

[133] Lars Luthmann, Andreas Stephan, Johannes Bürdek, and Malte Lochau. Modeling and Testing Product Lines with Unbounded Parametric Real-Time Constraints. In *21st International Systems and Software Product Line Conference (SPLC '17)*, pages 104–113. ACM, 2017. ISBN 978-1-4503-5221-5. doi: 10.1145/3106195.3106204.

3.1 EXTENDED FEATURE MODELS

In this section, we utilize *extended feature models (EFM)* [109] with attributes by means of constraints over unbounded numeric parameters. Therewith, we model dependencies between features and unbounded parameters. We start this section by providing an example for an EFM.

Example 3.2 (Extended Feature Model of the PPU). Figure 3.2 gives an example in terms of the extended feature model m of the PPU as described in Example 3.1. Here, we utilize an extension of feature diagrams (see Section 2.3) as graphical representation of extended feature models. In particular, nodes with solid borders depict features (i.e., Boolean parameters) while dashed borders indicate attributes in terms of constraints over unbounded numeric parameters. Please note, that the feature model without attributes is the same as described in the background chapter (see Figure 2.6).

In the following, the numeric parameter a denotes the range for the upper bound of the duration for *rotating* the crane. Furthermore, the numeric parameter b is utilized for the upper bound of the *rotation* including *picking* and *placing* the workpiece, respectively. Additionally, the numeric parameter c is a lower bound for *rebooting* the demonstrator after an emergency *shutdown*. As described above, the restriction of numeric parameters depends on the feature selection of a configuration. For instance, the duration for *rotating* the crane in case of *plastic* workpieces only has a lower bound of 15 seconds (i.e., $a \geq 15$) while *metal* workpieces require a range of 20 to 25 seconds (i.e., $a \geq 20 \wedge a \leq 25$). Hence, selecting *plastic* results in $a \geq 15$ while selecting *metal* results in $a \geq 20 \wedge a \leq 25$. As a result, we obtain the constraints $\text{Plastic} \Rightarrow (a \geq 15)$ and $\text{Metal} \Rightarrow (a \geq 20 \wedge a \leq 25)$. Moreover, b (for the *rotation* including *picking* and *placing* workpieces) has a range of 20 to 30 seconds for *plastic* and 30 to 40 seconds for *metal*, resulting in the parameter constraints $b \geq 20 \wedge b \leq 30$ and $b \geq 30 \wedge b \leq 40$, respectively. Finally, parameter c has a lower bound of 30 seconds, resulting in the parameter constraint $c \geq 30$.

Furthermore, Table 3.2 presents the set of valid configurations of the extended feature model in Figure 3.2. Here, columns 2 to 7 correspond to the features of the extended feature model while the last three columns correspond to the numeric parameters being used in the example. Moreover, each row $\Theta_i \subseteq \llbracket m \rrbracket$ corresponds to a *subset of configurations* (as opposed to single configurations as used in Example 2.5). This is due to the columns of numeric parameters showing intervals of allowed parameter valuations for the feature configuration in the respective rows. In each interval, the first value denotes the lower bound and the second value denotes the upper bound. Additionally, square brackets denote that the respective value is included in the interval. In contrast, parentheses denote that the respective value is a bound for the interval but the value itself is not included. Hence, ∞ combined with a parenthesis means that a numeric parameter may have an arbitrary, yet finite, value in \mathbb{T}_P . For instance, interval $[15; \infty)$ for a denotes that a has the lower bound 15 but no upper bound. It should be noted that some rows include the interval $[0; \infty)$ for parameter c as c is not part of these configurations such that we may assume a default configuration. As a result, the extended feature model of this example describes an infinite number of configurations as each subset Θ_i with $1 \leq i \leq 6$ describes an infinite number of configurations.

For defining EFM, we utilize *parameter constraints* to compare parametric linear terms (see Definition 2.10) with constants in \mathbb{T}_P . Hence, we utilize parameter constraints as attributes of EFM to restrict the allowed interval of parameters.

Definition 3.1 (Parameter Constraint). Let P be a set of parameters defined over parameter domain \mathbb{T}_P . The set $\mathcal{B}(P)$ of *parameter constraints* ξ is inductively defined as

$$\xi := \text{true} \mid m \sim \text{plt}_P \mid \xi \wedge \xi$$

where $m \in \mathbb{T}_P$, plt_P is a parametric linear term over P , and $\sim \in \{<, \leq, \geq, >\}$.

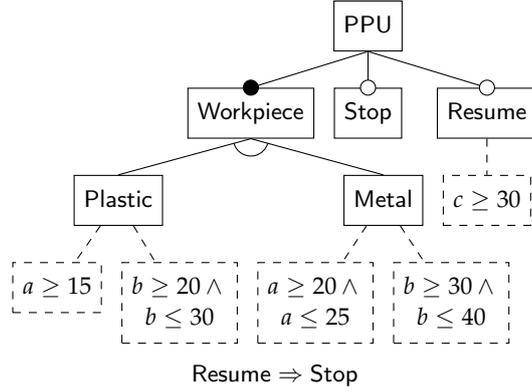


Figure 3.2: Extended Feature Model of the Extract of the PPU

Table 3.2: Valid Configurations of the EFM Depicted in Figure 3.2

	PPU	Workpiece	Plastic	Metal	Stop	Resume	a	b	c
Θ_1	true	true	true	false	true	true	$[15; \infty)$	$[20; 30]$	$[30; \infty)$
Θ_2	true	true	true	false	true	false	$[15; \infty)$	$[20; 30]$	$[0; \infty)$
Θ_3	true	true	true	false	false	false	$[15; \infty)$	$[20; 30]$	$[0; \infty)$
Θ_4	true	true	false	true	true	true	$[20; 25]$	$[30; 40]$	$[30; \infty)$
Θ_5	true	true	false	true	true	false	$[20; 25]$	$[30; 40]$	$[0; \infty)$
Θ_6	true	true	false	true	false	false	$[20; 25]$	$[30; 40]$	$[0; \infty)$

It should be noted that we, again, do not explicitly include operators for equality, inequality, disjunction, and negation as these operators can be semantically emulated with the given grammar as usual (see Section 2.2). However, we may use these operators in the following examples for better readability.

Example 3.3 (Parameter Constraint of the PPU). Consider the extension of the PPU SPL as described in Example 3.1. Here, the PPU SPL contains a parameter a for the upper bound concerning the real-time constraint of the *rotation*. For instance, this bound may be 20 to 25 seconds, such that $a \geq 20 \wedge a \leq 25$. Hence, when deriving a variant of this SPL, a is replaced by a value in $[20; 25]$. As a result, the derived variant has a fixed upper bound for the real-time constraint of the *rotation*.

With parameter constraints, we can now formally define extended feature models as a proper extension of feature models (see Definition 2.5). Here, we generalize the definition of FM to a (restricted) parameter constraint where we instantiate these parameters with Boolean parameters (i.e., features) over \mathbb{B} and unbounded numeric parameters over \mathbb{T}_p . As a result, a constraint comprising solely Boolean parameters corresponds to a feature model (see Definition 2.5). Therewith, we can express the mapping from Boolean parameters to unbounded parameters as indicated in the beginning of this chapter. For instance, assume that $a \geq 20 \wedge a \leq 25$ if feature *metal* is selected. This results in the EFM constraint $\text{Metal} \Rightarrow (a \geq 20 \wedge a \leq 25)$. In this thesis, we restrict these constraints such that a constraint over an unbounded numeric parameter always depends on the selection of a Boolean parameter (i.e., there is an implication between a Boolean parameter and a constraint over

unbounded numeric parameters). Therewith, we ensure that a given part of a model can be traced back to features (i.e., Boolean parameters). Furthermore, unbounded numeric parameters are specifically utilized for configuring clock constraints. We call constraints describing EFM *configuration constraints*.

Definition 3.2 (Extended Feature Model). Let $P = P_N \cup P_F$ with $P_N \cap P_F = \emptyset$ be a set of parameters where P_N is defined over \mathbb{T}_P and P_F over \mathbb{B} . The set $\mathcal{B}(P_F, P_N)$ of *configuration constraints* ξ over P_F and P_N is inductively defined as

$$\xi := \text{true} \mid \xi \wedge \xi \mid \neg \xi \mid p_F \mid p_F \Rightarrow (k \sim \text{plt}_{P_N})$$

where $\sim \in \{<, \leq, \geq, >\}$, $k \in \mathbb{T}_C$, plt_{P_N} is a parametric linear term over P_N , and $p_F \in P_F$. An *extended feature model* (EFM) is a configuration constraint $\xi \in \mathcal{B}(P_F, P_N)$.

Note, that we utilize the implication in $p_F \Rightarrow (n \sim \text{plt}_{P_N})$ as a shorthand for $\neg (p_F \wedge \neg (n \sim \text{plt}_{P_N}))$ to keep Definition 3.2 compact. Moreover, recall that de-selecting p_F in this implication means that a configuration does not need to satisfy parameter constraint $n \sim \text{plt}_{P_N}$. Hence, restrictions described by parameter constraint only need to be satisfied if Boolean parameter p_F is selected in a configuration. Additionally, the grammar in Definition 3.2 allows us to negate the implication such that we can describe constraints of the form

$$\neg (p_F \Rightarrow (k \sim \text{plt}_{P_N})).$$

This can be used to explicitly state forbidden value ranges for numeric parameters in P_N (instead of only describing allowed value ranges for particular selections of Boolean parameters). Furthermore, each parameter constraint always depends on a single feature p_F . However, this is not a restriction as p_F might have child features and other dependencies in terms of cross-tree constraints such that a parameter constraint may depend on a more complex feature constraint. In addition, two different Boolean parameters may have implications to the same numeric parameter such that both restrictions need to be considered. As a result of Definition 3.2, the formal semantics of an extended feature model $\llbracket m \rrbracket$ denotes the set of all configurations satisfying configuration constraint m .

Definition 3.3 (EFM Configuration). Let $m \in \mathcal{B}(P_F, P_N)$ be an extended feature model with $P = P_F \cup P_N$. The set of *EFM configurations* is denoted as $\llbracket m \rrbracket = \{c : P \rightarrow \mathbb{T}_P \cup \mathbb{B} \mid c \models m\}$, where $c \models m$ is defined recursively as

$$\begin{aligned}
c &\models \text{true} \\
c &\models p_F \Leftrightarrow c(p_F) = \text{true} \\
c &\models \zeta \wedge \zeta' \Leftrightarrow c \models \zeta \text{ and } c \models \zeta' \\
c &\models \neg \zeta \Leftrightarrow \neg c(\zeta) = \text{true} \\
c &\models k \sim \left(\sum_{1 \leq i \leq |P_N|} \alpha_i p_i \right) + n \Leftrightarrow k \sim \left(\sum_{1 \leq i \leq |P_N|} \alpha_i \cdot c(p_i) \right) + n
\end{aligned}$$

with $\sim \in \{<, \leq, \geq, >\}$, $p_F \in P_F$, $p_i \in P_N$, and $\left(\sum_{1 \leq i \leq |P_N|} \alpha_i p_i \right) + n$ being a parametric linear term.

Recall, that we utilize the implication in $p_F \Rightarrow (n \sim plt_{P_N})$ as a shorthand for $\neg (p_F \wedge \neg (n \sim plt_{P_N}))$. Hence, we do not explicitly include this implication in Definition 3.3.

Having defined extended feature models, we now proceed to introduce our formalism for behavioral modeling of product lines with unbounded parametric real-time constraints.

3.2 CONFIGURABLE PARAMETRIC TIMED AUTOMATA

In this section, we introduce a TA-based formalism for modeling product lines with unbounded parametric real-time constraints, namely *Configurable Parametric Timed Automata (CoPTA)*. In order to achieve this goal, we combine variability of Boolean parameters from FTA with variability of unbounded numeric parameters from PTA to benefit from properties of both formalisms. Hence, we utilize feature constraints as well as parametric constraints. Again, we start this section by providing an illustrating example.

Example 3.4 (CoPTA of the PPU). Figure 3.3 depicts an example for a CoPTA where we assume that the corresponding extended feature model is depicted in Figure 3.2. Here, we extended the PPU SPL that we previously modeled as an FTA (see Example 2.6). In this figure, we denote the constraint over Boolean parameters of a featured parametric clock constraint by writing them in brackets in front of parametric clock constraints. Please note that we omit Boolean constraints being equal to true for improved readability.

This CoPTA extends the FTA of Example 2.6 in several ways. For instance, the first *rotation* now has a variable numeric parametric upper bound. As *Workpiece* is a mandatory feature in the corresponding feature model, the upper bound in terms of unbounded parameter a is specified by the attribute of feature *Plastic* or *Metal*. Furthermore, the second *rotation* has more complex real-time constraints. Here, the *featured parametric clock constraint (FPCC)* for this second rotation is the same as for the first rotation in case of workpieces made of *plastic*. An FPCC $[\lambda]\phi$ consists of a feature constraint λ (describing the

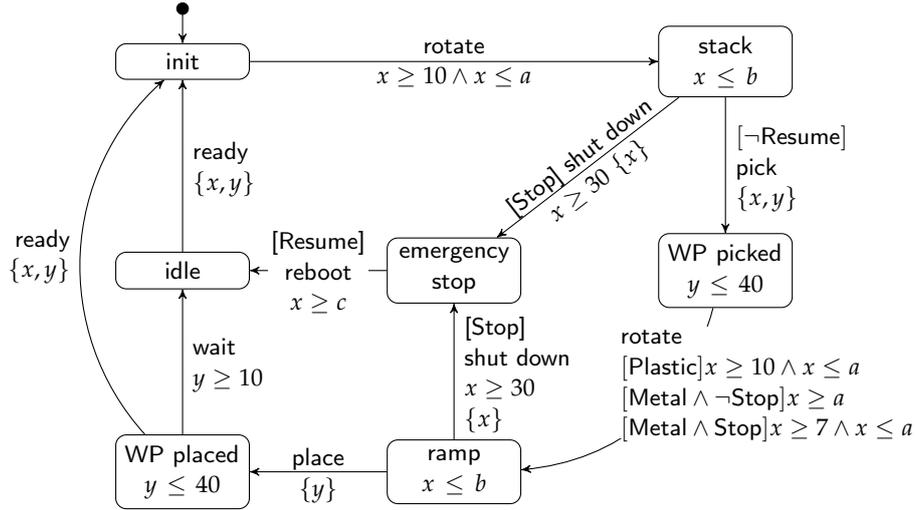


Figure 3.3: CoPTA Model of the Extract of the PPU

configurations in which the FPCC is present) and a parametric clock constraint ϕ . The FPCC for *metal* workpieces depends on the selection or deselection of the emergency *stop*. Including the emergency *stop* results in a reduced lower bound as the demonstrator may be stopped in case of an emergency. Moreover, the invariants of *stack* and *ramp* are now parametric (as already seen in the PTA of the PPU in Example 2.7). Finally, *rebooting* after an emergency *shutdown* now depends on the Boolean parameter *Resume* and unbounded parameter c (where a constraint for c is provided in terms of the attribute of feature *Resume*). When choosing the configuration

$$\text{PPU} \wedge \text{Workpiece} \wedge \text{Stop} \wedge \neg \text{Resume} \wedge \text{Plastic} \wedge \neg \text{Metal} \wedge a = 15 \wedge b = 30 \wedge c = 0$$

we obtain the TA depicted in the background chapter in Figure 2.3 (where we may choose an arbitrary value for c as the switch using c is not part of this configuration). Therefore, a CoPTA may contain variable bounds being replaced by constant (fixed) values when deriving a TA.

SYNTAX OF COPTA Concerning the syntax of CoPTA, we start by introducing *featured parametric clock constraints (FPCC)*, extending parametric clock constraints by constraints over Boolean parameters. Therewith, we can restrict parametric clock constraints to only be present in some variants. This is similar to FTA where we also restrict clock constraints to some variants. Formally, we write $\gamma = [\lambda]\phi$ to denote an FPCC, where $\lambda \in \mathcal{B}(P_F)$ is a parameter constraint over Boolean parameters P_F and $\phi \in \mathcal{B}(C, P_N)$ is a parametric clock constraint. Note, that λ resembles a feature constraint as described in Section 2.3 (hence, the name *featured* parametric clock constraints). Furthermore, an FPCC is defined w.r.t. an extended feature model $m \in \mathcal{B}(P_F, P_N)$ such that we may impose requirements for well-formed FPCC. In particular, we require $\llbracket \lambda \wedge m \rrbracket \neq \emptyset$ (i.e., there exists at least one variant for which the constraint is satisfiable). Otherwise we would describe parts of systems that are

not present in any valid configuration. Note, that we do *not* impose requirements on $\lambda \wedge \phi \wedge m$ as ϕ is a parametric clock constraint where parameters are compared to clocks, such that parameters cannot be explicitly restricted (as opposed to restrictions imposed by EFM m). Moreover, constraint λ and parametric clock constraint ϕ are related by EFM m as all these constraint utilize the same sets of parameters P_F and P_N . Hence, choosing a configuration satisfying λ also affects ϕ as constraints over P_N depend on Boolean parameters in m .

Definition 3.4 (Featured Parametric Clock Constraint). Let C be a set of clocks over \mathbb{T}_C , P_F be a set of Boolean parameters, and P_N be a set of unbounded numeric parameters over \mathbb{T}_P . The set $\mathcal{B}(C, P_F, P_N)$ of *featured parametric clock constraint* (FPCC) is inductively defined as

$$\gamma := \text{true} \mid [\lambda]\phi \mid \gamma \wedge \gamma$$

where $\lambda \in \mathcal{B}(P_F)$ and $\phi \in \mathcal{B}(C, P_N)$. We use $[\lambda]\phi \wedge [\lambda']\phi'$ to denote $[\lambda \wedge \lambda']\phi \wedge \phi'$.

Again, we assume *diagonal-free* CoPTA, and we only use difference constraints to model the symbolic semantics of CoPTA. Moreover, we omit constraints over Boolean parameters λ being equal to true from examples and only depict parametric clock constraints ϕ in this case.

Example 3.5 (Featured Parametric Clock Constraint). Consider the real-time constraints described in Example 3.1 and the extended feature model described in Example 3.2. In particular, we want to describe the featured parametric clock constraint for *rotating* workpieces made of *plastic*. Assuming a fixed lower bound of 10 seconds and a variable upper bound in terms of parameter a for describing this real-time constraint, we can represent such a constraint with $[\text{Plastic}]x \geq 10 \wedge x \leq a$. As noted above, the variable parameter a is replaced by a constant value when deriving a TA.

Having defined extended feature models and featured parametric clock constraints, we now proceed with the formal definition of CoPTA. As compared to FTA, CoPTA additionally contain a set of unbounded numeric parameters (and features are replaced by Boolean parameters). Furthermore, featured clock constraints in guards and invariants are generalized to featured parametric clock constraints, and feature models are generalized to extended feature models to specify the set of valid configurations. Similar to FTA, we utilize constraints over Boolean parameters in two different places of a CoPTA model. First, featured parametric clock constraints $[\lambda]\gamma$ in terms of guards and invariants contain constraints over Boolean parameters (see Definition 3.4). Here, a constraint $[\lambda]\gamma$ is removed from configurations $c \notin \llbracket \lambda \rrbracket$ not satisfying λ . Hence, a switch of a variant (i.e., TA) may be labeled with true if $[\lambda]\gamma$ is removed. Second, we use function η to annotate switches with featured parametric clock constraints. Therewith, we control whether a switch is part of a given configuration.

Definition 3.5 (Configurable Parametric Timed Automaton). A *configurable parametric timed automaton* (CoPTA) is a tuple $\mathcal{C} = (L, \ell_0, \Sigma, C, P_F, P_N, I, E, m, \eta)$, where

- L is a finite set of *locations*,
- $\ell_0 \in L$ is the *initial location*,
- Σ is a finite set of *actions* with $L \cap \Sigma = \emptyset$,
- C is a finite set of *clocks* over \mathbb{T}_C such that $C \cap (L \cup \Sigma) = \emptyset$,
- P_F is a finite set of *Boolean parameters* over \mathbb{B} such that $P_F \cap (C \cup L \cup \Sigma) = \emptyset$,
- P_N is a finite set of *unbounded numeric parameters* over \mathbb{T}_P such that $P_N \cap (C \cup L \cup \Sigma) = \emptyset$,
- $I : L \rightarrow \mathcal{B}(C, P_F, P_N)$ assigns *featured parametric invariants* to locations,
- $m \in \mathcal{B}(P_F, P_N)$ is an *extended feature model* over P_F and P_N ,
- $E \subseteq L \times \mathcal{B}(C, P_F, P_N) \times \Sigma \times 2^C \times L$ is a finite relation defining *switches*, and
- $\eta : E \rightarrow \mathcal{B}(P_F)$ assigns *constraints over Boolean parameters* to switches.

We use $\ell \xrightarrow{g, \sigma, R} \ell'$ to denote $(\ell, g, \sigma, R, \ell') \in E$, where $\{\ell, \ell'\} \subseteq L$, $g \in \mathcal{B}(C, P_F, P_N)$, $\sigma \in \Sigma$, and $R \subseteq C$.

With CoPTA, we now have a model that supports traceability between the problem space and the solution space by utilizing Boolean features while also supporting unbounded numeric variability of parameters. Note that for CoPTA, we apply the same graphical representation for constraints over Boolean parameters as we use for feature constraints in FTA (i.e., we write these constraints in brackets). Next, we introduce the semantics of CoPTA.

SEMANTICS OF COPTA Similar to FTA and PTA, the operational semantics of a CoPTA $\mathcal{C} = (L, \ell_0, \Sigma, C, P_F, P_N, I, E, m, \eta)$ is defined in two steps. First, a configuration $c \in \llbracket m \rrbracket$ is applied to \mathcal{C} . Here, we apply a projection of \mathcal{C} for each $c \in \llbracket m \rrbracket$ such that we obtain a set of PTA (similar to the projection of FTA to TA). In particular, we only consider Boolean parameters and include or exclude the respective parts of \mathcal{C} in the resulting PTA \mathcal{P} . Then, we apply parameter valuations (see Definition 2.13) of parameters in P_N to derive a set of TA from \mathcal{P} . Hence, the result is a TA $\nu(\mathcal{C})$ for configuration (i.e., parameter valuation) ν . Second, the semantics of TA $\nu(\mathcal{C})$ can be derived as described in Definition 2.18 (TLTS) and Definition 2.20 (zone graph).

Definition 3.6 (CoPTA Parameter Valuation). Let $\mathcal{C} = (L, \ell_0, \Sigma, C, P_F, P_N, I, E, m, \eta)$ be a CoPTA, $c \in \llbracket m \rrbracket$ be a configuration of m , and c_F be the configuration

of Boolean parameters P_F in c . The derivation of a TA $\nu(\mathcal{C})$ is achieved in two steps. First, we apply a projection to obtain a PTA $\mathcal{P} = (L, \ell_0, \Sigma, C, P_N, I', E')$ where

- $\forall \ell \in L : I'(\ell) := I(\ell)|_c$ and
- $E' = \{e = (\ell, g|_c, \sigma, R, \ell') \mid e \in E \wedge c \in \llbracket \eta(e) \rrbracket\}$

where the projection of a featured parametric clock constraint γ to a configuration c is recursively defined as

$$\gamma|_c = \begin{cases} (\gamma_1)|_c \wedge (\gamma_2)|_c & \text{if } \gamma = \gamma_1 \wedge \gamma_2, \\ \phi & \text{if } \gamma = [\lambda]\phi \wedge c_F \in \llbracket \lambda \rrbracket, \\ \text{true}, & \text{otherwise.} \end{cases}$$

Second, we apply parameter valuation $\nu(\mathcal{P})$ according to Definition 2.13. We write $\llbracket \mathcal{C} \rrbracket_c$ to denote the set of all resulting TA being derivable from \mathcal{C} .

Up to this point, we defined extended feature models and CoPTA. Next, we give an overview on the transformation of our novel modeling formalism into PTA such that we may reuse existing results of the PTA theory. For instance, we are interested in decidability of some properties (e.g., reachability) in the following chapters. Hence, we can utilize results of PTA instead of providing proofs ourselves.

3.3 TRANSFORMATION OF CONFIGURABLE PARAMETRIC TIMED AUTOMATA INTO EXTENDED PARAMETRIC TIMED AUTOMATA

In this section, we introduce a transformation of CoPTA into PTA with an extended syntax such that we may rely on existing results of the PTA theory. For instance, we consider decidability of some properties (e.g., reachability) in the following chapters. Hence, we can reuse decidability results of PTA [20, 21] instead of directly proving these results ourselves. Additionally, we can use a mature PTA model checker instead of implementing our own prototype for evaluating the approaches presented in Chapters 4 and 5. In particular, we utilize IMITATOR [23, 19, 18], a state-of-the-art model checker for PTA. For these reasons, we explain in the remainder of this section how Boolean parameters of CoPTA (including extended feature models) can be transformed into unbounded parametric variability (i.e., PTA). It should be noted that we do not give details on how to use the model checker for analysis in this section (details on this will be introduced in the respective chapters). Instead, we only show the transformation of CoPTA into PTA. Furthermore, IMITATOR does not provide data structures that are necessary for checking timed bisimilarity. Hence, the presented transformation is not utilized for the evaluation in Chapter 6.

As opposed to plain PTA (see Definition 2.12), IMITATOR supports several extensions of PTA, simplifying the transformation of CoPTA. In particular, constraints in switches and invariants may contain parameter constraints (cf. Definition 3.1) in addition to parametric clock constraints, resulting in so-called *IMITATOR PTA* [22]. In the following, we refer to the resulting formalism as *Parameter-Extended PTA*

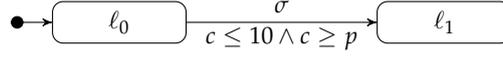


Figure 3.4: Restriction of Parameters by Utilizing Parametric Clock Constraints

(PEPTA). Note, that this extension is not more expressive than PTA as defined in Chapter 2 as parameters can be implicitly restricted with parametric clock constraints. For instance, consider the PTA in Figure 3.4. Here, we need to satisfy the parametric clock constraints $\phi = c \leq 10 \wedge c \geq p$ to reach location ℓ_1 . Hence, we require $p \leq 10$ to reach ℓ_1 (and possible further location after ℓ_1) as otherwise ϕ is not satisfiable. Therewith, we restrict parameter p indirectly by utilizing a parametric clock constraint. Note, that (as described above) locations as well as switches may be annotated with parametric clock constraints as well as parameter constraints (i.e., PEPTA support constraints in $\mathcal{B}(C, P) \times \mathcal{B}(P)$).

Definition 3.7 (Parameter-Extended PTA). A *parameter-extended PTA (PEPTA)* is a tuple $(L, \ell_0, \Sigma, C, P, I, E)$, where

- L is a finite set of *locations*,
- $\ell_0 \in L$ is the *initial location*,
- Σ is a finite set of *actions* with $L \cap \Sigma = \emptyset$,
- C is a finite set of *clocks* over \mathbb{T}_C such that $C \cap (L \cup \Sigma) = \emptyset$,
- P is a finite set of *unbounded numeric parameters* over \mathbb{T}_P such that $P \cap (L \cup \Sigma \cup C) = \emptyset$,
- $I : L \rightarrow \mathcal{B}(C, P) \times \mathcal{B}(P)$ assigns *parametric invariants* to locations, and
- $E \subseteq L \times (\mathcal{B}(C, P) \times \mathcal{B}(P)) \times \Sigma \times 2^C \times L$ is a finite relation defining *switches*.

We use $\ell \xrightarrow{g, \sigma, R} \ell'$ to denote switches $(\ell, g, \sigma, R, \ell') \in E$.

Note, that there are further extension to PTA being supported by IMITATOR (e.g., stopwatches) which, however, are not utilized in our transformation and thus not considered in this section. Furthermore, we slightly adapt parameter valuation (see Definition 2.13) for PEPTA. In particular, a parameter valuation $\nu : P \rightarrow \mathbb{T}_P$ for PEPTA replaces each occurrence of every parameter $p \in P$ in $\mathcal{B}(C, P) \times \mathcal{B}(P)$ by the constant values $\nu(p) \in \mathbb{T}_P$. Here, we use $\llbracket \mathcal{P} \rrbracket_P$ to denote the set of all TA being derivable from a PEPTA \mathcal{P} .

EFM EMBEDDING Having defined PEPTA, we proceed by transforming CoPTA $(L, \ell_0, \Sigma, C, P_F, P_N, I, E, m, \eta)$ into a PEPTA, where we first introduce the embedding of the EFM m into a PEPTA model. In particular, we ensure that in the PEPTA transformation, only variants being valid w.r.t. m can be derived. We achieve this requirement by utilizing the fact that PEPTA allow annotating switches with parameter constraints (see Definition 3.1). In particular, $m \in \mathcal{B}(P_F, P_N)$, being a

configuration constraint, can be transformed into a parameter constraint. Here, we utilize the parameter constraint $p_F = 0$ in the transformation if p_F is negated in m , and $p_F = 1$, otherwise. Constraints over P_N remain unchanged as these constraints already are parameter constraints.

Definition 3.8 (EFM Embedding). Let $m \in \mathcal{B}(P_F, P_N)$ be an extended feature model and $P = P_F \cup P_N$. The *embedding* $\kappa : \mathcal{B}(P_F, P_N) \rightarrow \mathcal{B}(P)$ into a parameter constraint is recursively defined as

$$\begin{aligned}\kappa(\text{true}) &:= \text{true} \\ \kappa(p_F) &:= (p_F = 1) \\ \kappa(\neg p_F) &:= (p_F = 0) \\ \kappa(\xi \wedge \xi') &:= \kappa(\xi) \wedge \kappa(\xi') \\ \kappa(\neg \xi) &:= \neg \kappa(\xi) \\ \kappa(n \sim plt) &:= n \sim plt\end{aligned}$$

with $p_F \in P_F$ and plt being a parametric linear term.

Next, we give an example for the embedding of an EFM.

Example 3.6 (EFM Embedding of the PPU). The EFM m in Figure 3.2 can be described in terms of the following configuration constraint:

$$\begin{aligned}\text{PPU} \wedge \text{Workpiece} \wedge (\text{Plastic} \vee \text{Metal}) \wedge (\neg \text{Plastic} \vee \neg \text{Metal}) \wedge \\ (\neg \text{Resume} \vee \text{Stop}) \wedge (\neg \text{Plastic} \vee (a \geq 15 \wedge b \geq 20 \wedge b \leq 30)) \wedge \\ (\neg \text{Resume} \vee c \geq 30) \wedge (\neg \text{Metal} \vee (a \geq 20 \wedge a \leq 25 \wedge b \geq 30 \wedge b \leq 40)).\end{aligned}$$

Hence, the embedding of $\kappa(m)$ of m is given by

$$\begin{aligned}\text{PPU} = 1 \wedge \text{Workpiece} = 1 \wedge (\text{Plastic} = 1 \vee \text{Metal} = 1) \wedge \\ (\text{Plastic} = 0 \vee \text{Metal} = 0) \wedge (\text{Resume} = 0 \vee \text{Stop} = 1) \wedge \\ (\text{Plastic} = 0 \vee (a \geq 15 \wedge b \geq 20 \wedge b \leq 30)) \wedge (\text{Resume} = 0 \vee c \geq 30) \wedge \\ (\text{Metal} = 0 \vee (a \geq 20 \wedge a \leq 25 \wedge b \geq 30 \wedge b \leq 40)).\end{aligned}$$

Therewith, we obtain the embedding of EFM m by introducing a fresh initial location ℓ'_0 into the PEPTA comprising exactly one outgoing switch. This outgoing switch is annotated with $\kappa(m)$ and targets the original initial location ℓ_0 . As a result, exactly the configurations of the EFM are enforced in the PEPTA transformation. Furthermore, each switch contains, by definition, an action. Here, we annotate the added switch with action $\varepsilon \notin \Sigma$ and assume ε to be unobservable. Specifically, we do *not* annotate the added switch with action τ (see Section 2.2) as this symbol is used to denote internal actions. Finally, the fresh location ℓ'_0 has invariant $c' \leq 0$ for a clock $c' \in C$ such that no time passes before reaching the original initial location ℓ_0 . It should be noted that we add a fresh initial location ℓ'_0 instead of annotating each outgoing switch of the original initial location ℓ_0 with $\kappa(m)$ as in the latter case, the restrictions of EFM m would not be imposed on the location invariant of ℓ_0 .

TRANSFORMATION OF COPTA Finally, we transform the CoPTA model (i.e., the automaton) into a PEPTA model. In particular, we transform featured parametric clock constraints and constraints over Boolean parameters being annotated to switches. This transformation can be achieved in a similar fashion as the embedding of the EFM. For a featured parametric clock constraint, we only transform the constraint over Boolean parameters as the rest of the constraint is a parametric clock constraint (which is already a valid part of a PEPTA model). Here, we utilize the fact that a constraint over Boolean parameters is a configuration constraint $\mathcal{B}(P_F, \emptyset)$ without unbounded numeric parameters. Hence, we can apply the EFM embedding (see Definition 3.8) and conjugate the resulting parameter constraint with the parametric clock constraint. The same procedure can be done for the constraints over Boolean parameter being annotated to switches by function η in CoPTA (see Definition 3.5). Moreover, the fresh location ℓ'_0 has invariant $c' \leq 0$ for a clock c' such that no time passes before reaching the original initial location ℓ_0 .

Definition 3.9 (PEPTA Transformation of CoPTA). Let $\mathcal{C} = (L, \ell_0, \Sigma, C, P_F, P_N, I, E, m, \eta)$ be a CoPTA. The transformation of \mathcal{C} is a PEPTA $\mathcal{P} = (L', \ell'_0, \Sigma, C', P, I', E')$, where

- $L' = L \cup \{\ell'_0\}$ with $\ell'_0 \notin L$ is a finite set of *locations*,
- $\ell'_0 \in L'$ is the *initial location*,
- $C' = C \cup \{c'\}$ with $c' \notin C$ is a finite set of *clocks*,
- $P = P_F \cup P_N$ is a finite set of *unbounded numeric parameters*,
- $I' : L' \rightarrow \mathcal{B}(C, P) \times \mathcal{B}(P)$ assigns *parametric invariants* to locations, and
- $E' = \{(\ell, \kappa[\lambda] \wedge \phi \wedge \kappa(\eta(e)), \sigma, R, \ell') \mid e = (\ell, [\lambda]\phi, \sigma, R, \ell') \in E\} \cup \{(\ell'_0, \kappa(m), \varepsilon, \emptyset, \ell_0)\}$ with $\varepsilon \notin \Sigma$.

The function I' assigning parametric invariants to locations is defined as

$$I'(\ell) = \begin{cases} \kappa(\lambda) \wedge \phi & \text{if } \ell \in L \text{ and } I(\ell) = [\lambda]\phi, \text{ and} \\ c' \leq 0 & \text{if } \ell = \ell'_0. \end{cases}$$

Example 3.7 (Transformation of a CoPTA). Figure 3.5 gives an example for the PEPTA transformation of the CoPTA depicted in Figure 3.3 (including the EFM depicted in Figure 3.2). Here, we initially have to traverse the embedded EFM (see Example 3.6) such that any run through the actual model is a run of a valid configuration. This switch is annotated with the unobservable action ε such that the behavior of the system remains unchanged. The model itself is almost identical to the CoPTA in Figure 3.3 except that there are no constraints over Boolean parameters. Instead, the Boolean variability is transformed in constraints over unbounded numeric parameters as described above. For instance, we require the unbounded numeric parameter *Resume* to have value

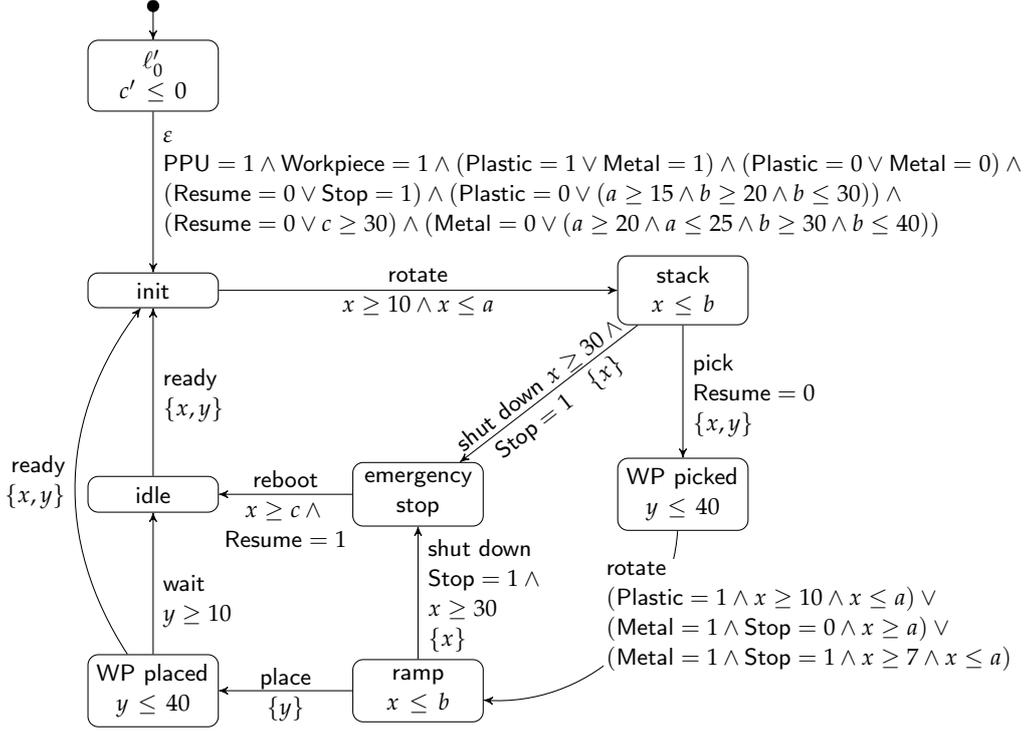


Figure 3.5: Transformation of the CoPTA Depicted in Figure 3.3 Into a PEPTA (Including the EFM depicted in Figure 3.2)

1 in the PEPTA transformation of the switch from *emergency stop* to *idle* instead of requiring the Boolean parameter *Resume* to be selected in the CoPTA for this switch.

Next, we prove correctness of the PEPTA transformation. We achieve this by proving that each TA \mathcal{A}_C being derivable from a CoPTA \mathcal{C} has a bisimilar TA \mathcal{A}_P being derivable from the PEPTA transformation \mathcal{P} (and vice versa).

Theorem 3.1 (Correctness of the PEPTA Transformation). Let \mathcal{C} be a CoPTA and \mathcal{P} be its PEPTA transformation. Then it holds that

1. $\forall \mathcal{A}_C \in \llbracket \mathcal{C} \rrbracket_C : (\exists \mathcal{A}_P \in \llbracket \mathcal{P} \rrbracket_P : \mathcal{A}_C \simeq \mathcal{A}_P)$ and
2. $\forall \mathcal{A}_P \in \llbracket \mathcal{P} \rrbracket_P : (\exists \mathcal{A}_C \in \llbracket \mathcal{C} \rrbracket_C : \mathcal{A}_P \simeq \mathcal{A}_C)$.

Proof. Let $\mathcal{C} = (L, \ell_0, \Sigma, C, P_F, P_N, I, E, m, \eta)$ be a CoPTA and $\mathcal{P} = (L', \ell'_0, \Sigma, C', P, I', E')$ be its PEPTA transformation. Recall, that ε is unobservable. Furthermore, no time passes while traversing the embedded EFM (i.e., the switch $\ell'_0 \xrightarrow{\kappa(m), \varepsilon, \emptyset} \ell_0$) because of the invariant of the fresh initial location. Hence, traversing the EFM embedding is unobservable in all runs of all derivable TA. Additionally, all clocks of the PEPTA transformation have value 0 when reaching the original initial location ℓ_0 . Next, we prove (1) and (2) separately.

1. We prove $\forall \mathcal{A}_C \in \llbracket \mathcal{C} \rrbracket_C : (\exists \mathcal{A}_P \in \llbracket \mathcal{P} \rrbracket_P : \mathcal{A}_C \simeq \mathcal{A}_P)$. Let $\llbracket \mathcal{A}_C \rrbracket_S = (S, s_0, \hat{\Sigma}, \rightarrow)$ and assume that $\llbracket \mathcal{A}_P \rrbracket_{S'} = (S', s'_0, \hat{\Sigma}', \rightarrow')$ is the corresponding bisimilar TA. We show that such an $\mathcal{A}_P \in \llbracket \mathcal{P} \rrbracket_P$ exists for all $\mathcal{A}_C \in \llbracket \mathcal{C} \rrbracket_C$. In particular, we obtain this variant \mathcal{A}_P by using the parameter valuation ν for \mathcal{P} corresponding to configuration $c \in \llbracket m \rrbracket$ of \mathcal{C} . We achieve this by picking values 0 and 1, respectively, for each parameter $p \in P$ of \mathcal{P} corresponding to the respective values of Boolean parameter $p_F \in P_F$ in configuration c of \mathcal{C} . For the remaining numeric parameters, we pick the same values for ν as picked for c . Based on this variant derivation, there exists an $\mathcal{R} \subseteq S \times S'$ such that $s_1 \xrightarrow{h} s_2 \Rightarrow (s'_1 \xrightarrow{h'} s'_2 \wedge (s_2, s'_2) \in \mathcal{R})$ and $(s_0, s'_0) \in \mathcal{R}$, i.e., \mathcal{A}_P timed simulates \mathcal{A}_C . Due to the embedding $\kappa(m)$ of EFM m in \mathcal{P} and parameter valuation ν , we ensure that the corresponding locations are reachable in \mathcal{A}_C and \mathcal{A}_P with the exact same delays. As a result, it also holds that there exists an $\mathcal{R}' \subseteq S' \times S$ such that $s'_1 \xrightarrow{h'} s'_2 \Rightarrow (s_1 \xrightarrow{h} s_2 \wedge (s'_2, s_2) \in \mathcal{R}')$ and $(s'_0, s_0) \in \mathcal{R}'$, i.e., \mathcal{A}_C timed simulates \mathcal{A}_P . Hence, $\forall \mathcal{A}_C \in \llbracket \mathcal{C} \rrbracket_C : (\exists \mathcal{A}_P \in \llbracket \mathcal{P} \rrbracket_P : \mathcal{A}_C \simeq \mathcal{A}_P)$.
2. We prove $\forall \mathcal{A}_P \in \llbracket \mathcal{P} \rrbracket_P : (\exists \mathcal{A}_C \in \llbracket \mathcal{C} \rrbracket_C : \mathcal{A}_P \simeq \mathcal{A}_C)$. Let $\llbracket \mathcal{A}_P \rrbracket_{S'} = (S', s'_0, \hat{\Sigma}', \rightarrow')$ and assume that $\llbracket \mathcal{A}_C \rrbracket_S = (S, s_0, \hat{\Sigma}, \rightarrow)$ is the corresponding bisimilar TA. We show that such an $\mathcal{A}_C \in \llbracket \mathcal{C} \rrbracket_C$ exists for all $\mathcal{A}_P \in \llbracket \mathcal{P} \rrbracket_P$. In particular, we obtain this variant \mathcal{A}_C by using the configuration $c \in \llbracket m \rrbracket$ of \mathcal{C} corresponding to parameter valuation ν of \mathcal{P} . We achieve this by picking values true and false, respectively, for each parameter $p_F \in P_F$ in configuration c of \mathcal{C} corresponding to the value of the respective parameter $p \in P$ of \mathcal{P} . For the remaining numeric parameters, we pick the same values for c as picked for ν . Based on this variant derivation, there exists an $\mathcal{R} \subseteq S \times S'$ such that $s_1 \xrightarrow{h} s_2 \Rightarrow (s'_1 \xrightarrow{h'} s'_2 \wedge (s_2, s'_2) \in \mathcal{R})$ and $(s_0, s'_0) \in \mathcal{R}$, i.e., \mathcal{A}_C timed simulates \mathcal{A}_P . Due to the embedding $\kappa(m)$ of EFM m in \mathcal{P} and parameter valuation ν , we ensure that the corresponding locations are reachable in \mathcal{A}_C and \mathcal{A}_P with the exact same delays. As a result, it also holds that there exists an $\mathcal{R}' \subseteq S' \times S$ such that $s'_1 \xrightarrow{h'} s'_2 \Rightarrow (s_1 \xrightarrow{h} s_2 \wedge (s'_2, s_2) \in \mathcal{R}')$ and $(s'_0, s_0) \in \mathcal{R}'$, i.e., \mathcal{A}_P timed simulates \mathcal{A}_C . Hence, $\forall \mathcal{A}_P \in \llbracket \mathcal{P} \rrbracket_P : (\exists \mathcal{A}_C \in \llbracket \mathcal{C} \rrbracket_C : \mathcal{A}_P \simeq \mathcal{A}_C)$. \square

As CoPTA are basically PTA being extended by Boolean parameters and EFM (as shown by the transformation presented in this section), decidability results of PTA [20, 21] are also applicable to CoPTA. Hence, reachability (i.e., EF-reachability) and other interesting properties are semi-decidable for PTA (and hence, also for CoPTA). Therefore, we obtain a correct result if the decision procedure terminates (but the procedure does, in general, not terminate).

After defining CoPTA and introducing a transformation into PEPTA, we next give an overview on related work for behavioral modeling of product lines with configurable parametric real-time constraints.

3.4 RELATED WORK

In this section, we present an overview on related work concerning modeling of real-time systems. In particular, we first focus on extensions and generalizations of

TA which may be equipped with variability over Boolean and unbounded numeric parameters. Thereafter, we give an overview on other formalism for modeling real-time behavior. Finally, we present related work on modeling variability of real-time behavior.

EXTENSIONS AND GENERALIZATIONS OF TA There are numerous extensions, variations, and generalizations of TA, and most recent works on modeling and automated analysis of time-critical behavior apply TA or extensions of TA [9, 198, 38, 11]. Here, we present an extract of these modeling approaches. For a comprehensive overview on TA-based formalisms, we refer the reader to Waez et al. [198]. First, Alur and Dill [9] introduce TA with multiplication in clock constraints. More precisely, clocks may be multiplied by constant values in guards and invariants. Miller [145] presents TA with irrational numbers as clock domain (i.e., $\mathbb{T}_C = \mathbb{R}_+ \setminus \mathbb{Q}_+$). As a result, interesting properties such as reachability become undecidable. Moreover, Bouyer et al. [46, 47] present *Updatable TA*. Here, clocks may be set to any value instead of just resetting them to 0. In addition to this, clock updates may also depend on values of other clocks. Furthermore, Cassez and Larsen [55] define *Stopwatch Automata*, where the derivative of a clock variable in a location may be either 0 or 1. Hence, a clock is “frozen” if the derivative is 0. In contrast, the derivative of a clock variable is always 1 in classical TA (i.e., clock values always increase with constant speed). Additionally, Behrmann et al. [31] introduce weighted TA, called *Priced Timed Automata* where executing switches and staying in locations is priced (depending on the amount of elapsed time). Next, Alur et al. [12] and Kwiatkowska et al. [117] define TA with probabilities for executing switches. These two ideas are combined by Katoen et al. [110] who present *Priced Probabilistic Timed Automata*. As another concept, Fersman et al. [85] introduce a formalism called *Task Automata*. Here, locations may be annotated with so-called tasks. A task is added to an execution queue when the respective locations is entered (such that multiple instances of a task may be in the queue). Each task has a minimum and maximum execution time and a deadline. Moreover, Maler et al. [143] formalize a game theoretical extension for TA (which is applied for controller synthesis). Finally, Alur et al. [14] define *Hybrid Automata*. This generalization of TA allows updating variables with sets of differential equations. However, none of these approaches support variability in terms of Boolean and/or unbounded numeric parameters, but these formalisms may be extended in a similar fashion as classical TA.

MODELING REAL-TIME BEHAVIOR Besides TA, there are other formalism for modeling real-time behavior. For instance, Merlin and Faber [144], Ramamoorthy and Ho [164], and Wang [199] define real-time extensions of Petri nets. These *Timed Petri nets* are extended with probabilities by Florin et al. [86]. Furthermore, Wang [200] presents a real-time process algebra. Finally, Selic [175] introduces a real-time extension of UML. Again, none of these approaches support feature variability or parametric variability. Thus, these formalism cannot be applied for behavioral modeling of real-time product lines with a potentially infinite number of variants.

MODELING VARIABILITY OF REAL-TIME BEHAVIOR In this paragraph, we present modeling formalisms for variability of real-time systems (besides FTA [70] and PTA [15] which are the basis for our CoPTA formalism). First, Cledou et al. [62] introduce *Interface FTA (IFTA)*, extending FTA by variable interfaces. These interfaces consist of ports for inputs and outputs being utilized for composition such that particular output ports correspond to input ports having the same name. Here, Cledou et al. [63] also present a notion for refinement of IFTA concerning the underlying automaton and feature model. Moreover, Sabouri et al. [169] utilize a modeling language similar to FTA for scheduling analyses of product lines. Kim et al. [112] apply FTA and extend FODA feature models by attributes in terms of *properties*. Therewith, requirements (e.g., deadlock-freedom) on preemptive real-time systems may be expressed. Additionally, Mitsching et al. [147] propose a catalog of design patterns for modeling families of TA variants based on a core model. Furthermore, Čerāns et al. [57] define *Timed Modal Specifications*. Instead of utilizing feature annotations, variability is expressed by modalities of switches. Here, a *mandatory* switch must be included in every variant whereas an *optional* switch may be included or removed during variant derivation. This formalism is enhanced by results for refinement (Bertrand et al. [43]) and reachability and compositionality (King et al. [113]). As a bridge between feature-based variability and modalities, Varshosaz et al. [195, 193] describe the encoding of (untimed) *Featured Transition Systems* [60] into a minimal set of (untimed) *Modal Transition Systems* [118]. This encoding may be extended by real-time constraints. Finally, Fribourg and Kühne [87] define a parameter extension of hybrid automata. Apart from the works mentioned above, TA is the only formalism having been enriched by concepts for modeling variability of real-time systems. Furthermore, none of these approaches have concepts for behavioral modeling of real-time product lines with a potentially infinite number of variants.

3.5 CONCLUSION AND FUTURE WORK

Up until now, there did not exist a formalism for behavioral modeling of real-time product lines with a potentially infinite number of variants. Existing approaches for variant-rich models only comprise one of the following two options for modeling variability. On the one hand, FTA [70] incorporate feature constraints allowing us to model dependencies in terms of presence, absence, and combinations of switches and clock constraints. On the other hand, PTA [15] incorporate unbounded numeric parametric constraints allowing us to choose arbitrary values from a potentially infinite parameter domain for clock constraints, resulting in a potentially infinite number of variants. In this chapter, we tackled Research Challenge 1 and improved this state of the art in two ways.

First, we utilize extended feature models to express dependencies between features (i.e., Boolean parameters) and unbounded numeric parameters. In particular, we use constraints over unbounded parameters as attributes of Boolean parameters in EFM. Hence, an EFM is provided in terms of a parameter constraint. As a result, an EFM may comprise an infinite number of configurations.

Second, we introduced the CoPTA formalism. Compared to FTA, CoPTA utilize extended feature models in terms of constraints over Boolean parameters as well as unbounded numeric parameters instead of classical feature models over features. Furthermore, featured clock constraints (of FTA) are replaced by featured parametric clock constraints. Therewith, we are able to model real-time product lines with a potentially infinite number of variants. Finally, we presented a transformation of CoPTA into an extension of PTA such that we may apply a state-of-the-art PTA model checker for CoPTA analyses in the following chapters.

For future work, we may generalize our formalism to support further non-functional properties. So far, CoPTA only support real-time restrictions. This may be generalized to other properties such as temperature or throughput (e.g., of waterpipes), allowing us to analyze these properties (with the approaches presented in the following chapters). For instance, we may apply our extensions to Hybrid Automata [14] (i.e., we may extend hybrid automata by features and parameters). Moreover, we plan to automatically derive CoPTA from other representations of product lines (e.g., source code) as CoPTA is a novel formalism not yet being used in practice. Therewith, it would be possible to apply our techniques for quality assurance even in cases where CoPTA are not explicitly used. For instance, we could use an existing approach by Liva et al. [125] as a basis, where TA are automatically extract from Java methods. Additionally, we plan to develop a front end specifically tailored to create CoPTA models and to give an overview on analysis results. As a result, we would increase usability of our approaches for modeling and analysis (as presented in Chapters 4 to 6) in practice. Currently, our tool utilizes UPPAAL (i.e., a TA model checker) as a front end. However, UPPAAL was not created to handle features and parameters, such that these parts of CoPTA models are (erroneously) marked as faulty in the UPPAAL user interface.

Having introduced the CoPTA formalism as a basis for behavioral modeling of product lines with configurable parametric real-time constraints, we may now proceed by utilizing this model for analysis purposes in the following chapters.

4

SAMPLING STRATEGIES FOR PRODUCT LINES WITH INFINITE CONFIGURATION SPACES

In this chapter, we tackle Research Challenge 2 as introduced in the background chapter (see Section 2.4).

Research Challenge 2

Adapting sampling strategies to product lines with a (potentially) infinite number of configurations.

Sampling is concerned with deriving a representative subset of variants of a product line. Therewith, we may analyze the resulting subset of variants (presumably requiring less effort) and draw conclusions for the whole product line instead of analyzing *all* variants.

Existing approaches for sampling product lines are mostly limited to finite configuration spaces (e.g., [150, 158, 106]). Here, coverage criteria consider finite sets of test goals. For instance, in case of CoPTA, we could consider location coverage (i.e., each location is contained in at least one variant of the sample) or switch coverage (i.e., each switch is contained in at least one variant of the sample). However, a CoPTA model may comprise an infinite number of variants. Furthermore, to the best of our knowledge there does not exist a coverage criterion specifically tailored to product lines with (configurable parametric) real-time constraints.

In order to sample product lines with infinite configuration spaces, we may consider existing sampling strategies. As a first approach we may consider CIT as introduced in the background chapter. However, CIT is not applicable to infinite configuration spaces as we cannot include *all* possible values of an unbounded numeric parameter (as there may be infinitely many values). Next, we may lift random sampling to infinite configuration spaces. This could be easily done as we can simply pick any number of possible values for each parameter. However, our goal is finding a representative set of configurations w.r.t. a criterion, and we cannot steer a truly random approach into a particular direction.

As a further approach for sampling infinite configuration spaces, we might consider using approaches from *boundary-value testing* [190]. Here, the idea is to explicitly include *boundary* values (i.e., smallest and greatest possible values) as well as non-boundary values of variables. However, this is ineffective for our particular problem as picking boundary values of parameters does not necessarily result in covering boundary-value behavior of time-critical systems. In particular, we cannot simply minimize (or maximize) parameters to derive variants with *best-case/*

worst-case executions times (BCET/WCET) [203] behavior for reaching given locations of a CoPTA. Configurations exhibiting BCET/WCET behavior are especially interesting for product lines with (configurable parametric) real-time constraints as they have a higher chance of exposing faults [203]. For instance, checking behavior with BCET/WCET can reveal *off-by-one timing errors* [6] where boundary behavior w.r.t. delays is faulty. Therefore, we want to derive variants having BCET/WCET behavior such that it is possible to observe this behavior in further analysis steps (e.g., test-case generation as presented in Chapter 5). Here, it should be noted that a finite WCET does not necessarily exist for reaching every test goal in the presence of loops (as we may traverse the loop arbitrarily often before reaching the test goal).

Example 4.1. Consider CoPTA \mathcal{C} of the PPU as depicted in Figure 3.3 with the corresponding EFM m in Figure 3.2. Here, we cannot apply classical sampling strategies (e.g., CIT) as some parameters of our EFM m have an infinite domain. When instead considering random sampling, we might still miss critical configurations exposing BCET/WCET behavior. For instance, consider location *emergency stop* where the BCET for reaching this location is 30 seconds. According to our EFM m , parameter b has a range between 20 and 40, depending on the selected type of *workpiece*. However, we need a value $b \geq 30$ such that *emergency stop* is reachable (due to the invariant of location *stack*), which is not ensured by random sampling.

As an alternative, we might consider approaches from boundary-value testing, including boundary values of variables. However, location *emergency stop* is not reachable at all if we simply minimize all parameters. As indicated above, we require $b \geq 30$ for *emergency stop* to be reachable. Hence, we require additional information from the solution space to derive a meaningful subset of variants of a product line with configurable parametric real-time constraints.

In order to find variants of CoPTA models having runs with BCET/WCET for reaching particular locations, only considering the problem space in terms of the EFM is not sufficient (as illustrated by Example 4.1). Hence, instead of only utilizing a black-box view on product lines by applying sampling based on EFM, we propose a white-box approach also considering solution-space knowledge in terms of CoPTA models. Therewith, we introduce a novel sampling strategy for product lines with infinite configuration spaces, taking into account characteristics of real-time systems (i.e., BCET/WCET behavior). Note, that we are only interested in the execution time for the *first* visit of a test goal. In particular, this is the case for WCET as otherwise the execution time can be arbitrarily great if the test goal is part of a loop.

The remainder of this chapter is structured as follows (see Figure 4.1 for an overview). We start by introducing notions and definitions required for sampling product lines with configurable parametric real-time constraints. Here, we lift the notions from sampling finite configuration spaces (see Section 2.4 in the background chapter) to infinite configuration spaces (see Section 4.1). To this end, we introduce a symbolic adaptation of value schemas. Thereafter, we introduce a symbolic semantics, namely featured parametric zone graphs, for effectively ana-

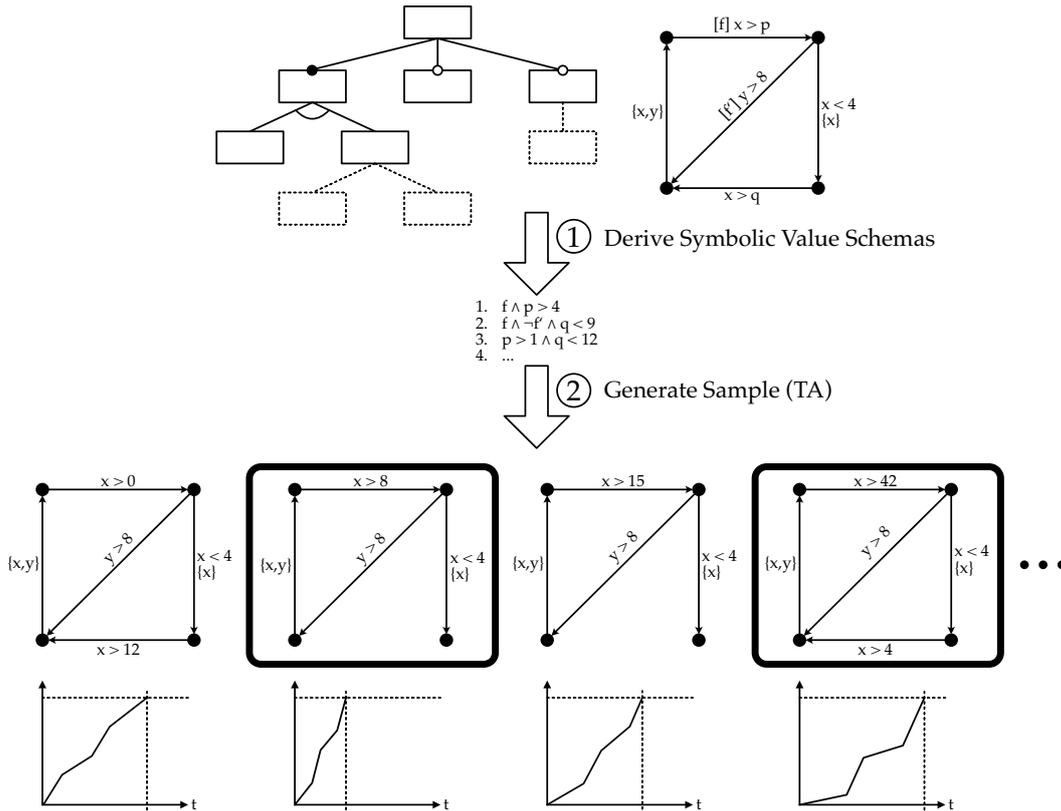


Figure 4.1: Schematic Overview on the Contributions Presented in Chapter 4

lyzing CoPTA models (see Section 4.2). In the subsequent section (see Section 4.3), we introduce a novel coverage criterion, namely *minimum/maximum delay coverage (M/MD coverage)*, explicitly considering BCET/WCET behavior. Furthermore, we utilize featured parametric zone graphs to derive symbolic value schemas for M/MD coverage from CoPTA models (step ① in Figure 4.1). Moreover, we use these symbolic value schemas to select a finite set of configurations (i.e., the resulting sample) such that each symbolic value schema is covered by at least one configuration (step ② in Figure 4.1). Therewith, we derive variants (as indicated by the rectangles around the variants) having BCET/WCET for reaching particular locations (as indicated by the graphs below the variants in Figure 4.1 where the x-axis denotes the time). Finally, we evaluate our novel sampling approach (see Section 4.4), give an overview on related work (see Section 4.5), and conclude this chapter (see Section 4.6). The contents of this chapter are based on the following publications:

[134] Lars Luthmann, Timo Gerecht, and Malte Lochau. Sampling Strategies for Product Lines with Unbounded Parametric Real-Time Constraints. *International Journal on Software Tools for Technology Transfer (STTT)*, 21: 613–633, 2019. ISSN 1433-2787. doi: 10.1007/s10009-019-00532-4.

[133] Lars Luthmann, Andreas Stephan, Johannes Bürdek, and Malte Lochau. Modeling and Testing Product Lines with Unbounded Para-

metric Real-Time Constraints. In *21st International Systems and Software Product Line Conference (SPLC '17)*, pages 104–113. ACM, 2017. ISBN 978-1-4503-5221-5. doi: 10.1145/3106195.3106204.

4.1 SAMPLING PRODUCT LINES WITH INFINITE CONFIGURATION SPACES

So far, we applied sample-based CIT to finite configuration spaces (see Section 2.4 of the background chapter). This technique can be categorized as *black-box* testing as we only consider the problem space in terms of a feature model. Hence, we do not require any knowledge of the solution space in terms of the corresponding CoPTA model and the mapping to features. As a result, sampling becomes an easy-to-use approach for testing. However, this may also lead to several weaknesses in terms of effectiveness. For instance, we might miss critical configurations or derive uninteresting configurations as we only consider the problem space. Thus, we consider *white-box* sampling strategies to have a higher chance of covering these critical configurations.

Furthermore, the sampling approaches for *finite* configuration spaces presented in the background chapter (see Section 2.4) are not directly applicable to *infinite* configuration spaces. Here, already 1-wise CIT results in an infinite amount of value schemas. Hence, we adapt black-box testing strategies to infinite configuration spaces, and we consider white-box testing strategies to ensure coverage of critical configurations.

In the remainder of this section, we first consider black-box sampling strategies for infinite configuration spaces (see Section 4.1.1). In particular, we show potential weaknesses of this approach by utilizing our illustrative example from Chapter 3. Thereafter, we propose an alternative white-box approach for *t*-wise sampling of infinite configuration spaces to counteract the weaknesses of black-box approaches (see Section 4.1.2). Hence, we exploit solution-space knowledge of the system under test for effective sample selection in terms of deriving variants having BCET/WCET for reaching test goals. Therewith, we have a higher chance of revealing some faults (e.g., off-by-one timing errors).

4.1.1 Black-Box Sampling Strategies for Infinite Configuration Spaces

As indicated above, *t*-wise CIT is not directly applicable as soon as there exists at least one parameter $\omega_i \in \Omega$ with an unbounded value domain V_i , such as $V_i = \mathbb{N}_0$. Here, already 1-wise CIT has an infinite set Q of value schemas (i.e., one schema for each $v \in V_i$), resulting in infinitely large covering arrays $\Omega \in CA(\Omega, \Theta_\Omega, t)$. Therefore, the set of values of unbounded parameters has to be bounded to finite subsets such that we may apply existing sampling strategies for finite configuration spaces. A possible solution to this problem may be the introduction of an additional parameter $k \in \mathbb{N}_0$, restricting the number of value schemas that need to be covered. This may be achieved in different ways from which we present two possibilities in the following. To this end, we utilize $[[\mathcal{C}]]_\Theta$ to denote the set of variants of a CoPTA \mathcal{C} corresponding to test cases θ of a covering array Θ .

Notation 4.1. Let $\mathcal{C} = (L, \ell_0, \Sigma, C, P_F, P_N, I, E, m, \eta)$ be a CoPTA and Θ be a covering array with $\theta \in \Theta \Rightarrow \theta \in \llbracket m \rrbracket$. We use $\llbracket \mathcal{C} \rrbracket_{\Theta}$ to denote the set of variants of \mathcal{C} corresponding to test cases $\theta \in \Theta$.

K-RANDOM T-WISE SAMPLING First, we apply parameter $k \in \mathbb{N}$ to define *k-random t-wise sampling*. Here, we choose k different values from the (possibly infinite) set V_i of parameter values for each parameter $\omega_i \in \Omega$ within t -wise value schemas. Therewith, it is possible to cover infinite configuration spaces.

Definition 4.1 (*k-random t-wise Coverage*). Let Ω be a set of parameters, Θ_{Ω} be a combination space over Ω , $k \in \mathbb{N}$, and $t \in \mathbb{N}$. Then, test suite $\Theta \subseteq \Theta_{\Omega}$ satisfies *k-random t-wise coverage* if for each unbounded parameter $\omega_i \in \Omega$, a selection of k different values $v_i \in V_i$ within t -wise value schemas is covered. We use

$$CA_R(\Omega, \Theta_{\Omega}, k, t)$$

to denote the set of all covering arrays satisfying *k-random t-wise coverage*.

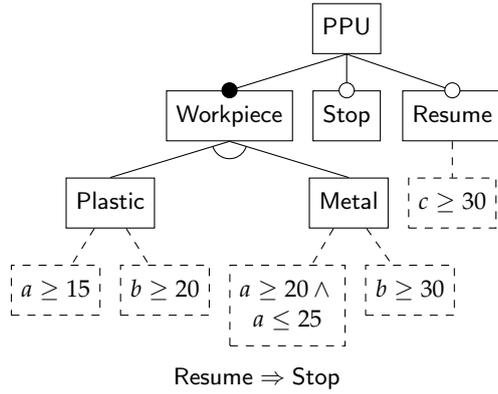
Unfortunately, *k-random t-wise coverage* has a high chance of missing critical configuration (which is an inherent problem of random-based approaches).

Example 4.2 (*k-random t-wise Coverage*). Consider the EFM in Figure 4.2a, describing the (infinite) configuration space of our PPU running example. As compared to the EFM in Chapter 3 (see Figure 3.2), this example restricts parameter b only in terms of a lower bound such that b is unbounded. Furthermore, consider the CoPTA depicted in Figure 3.3 (which we repeat in Figure 4.2b for the convenience of the reader) and the faulty implementation model in Figure 4.2c. This model is faulty as the switch from *stack* to *emergency stop* comprises the clock constraint $x \geq 31$ as opposed to $x \geq 30$ in the original model.

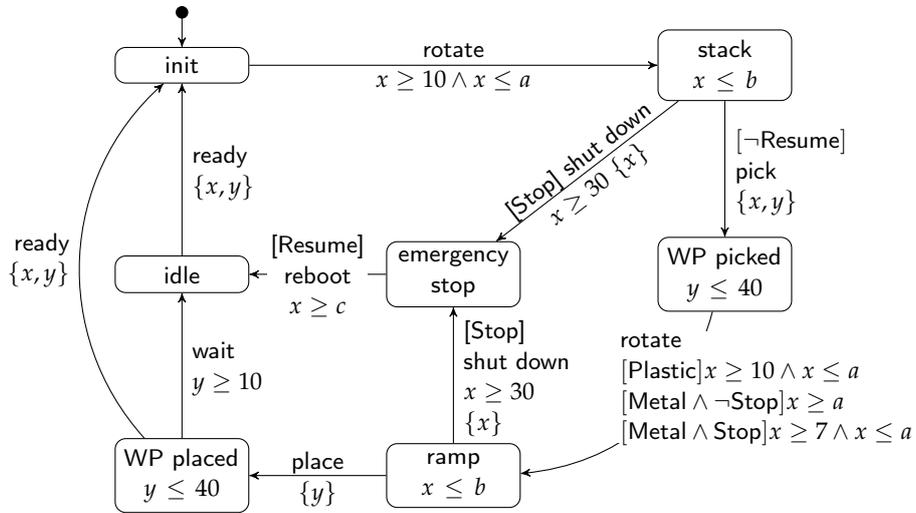
Next, we pick values for k and t , e.g., $k = 2$ and $t = 1$. If the two values for parameter b in our 1-wise value schemas coincidentally include a value $b \geq 30$, then the resulting covering array $\Theta \in CA_R(\Omega, \Theta_{\Omega}, 2, 1)$ would contain a test configuration covering the error of the faulty implementation. However, if we only choose values $b < 30$, we miss this critical case as the faulty switch is not reachable due to the invariant of location *stack* (neither in the original model nor in the faulty implementation model).

The strategy of *k-random t-wise coverage* may be further refined. For instance, we may pick an individual k_i for each unbounded $\omega_i \in \Omega$. However, a random-based selection of test cases might still not be sufficiently effective, especially for erroneous behavior for boundary values of configuration parameters (as described in Example 4.2).

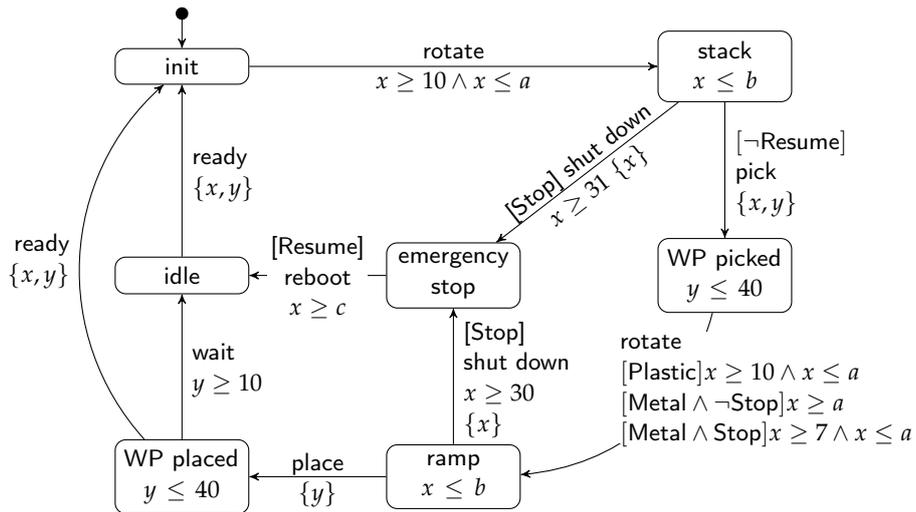
K-BOUNDED T-WISE SAMPLING A further strategy using a value $k \in \mathbb{Z}$ in a more systematic way is *k-bounded t-wise sampling*. Here, we limit the value schemas which have to be covered to the interval $[-k, k]$ of value domain \mathcal{D}_i of V_i (i.e., the



(a) Extended Feature Model of the PPU Extract (Similar to Figure 3.2)



(b) CoPTA Model of the Extract of the PPU (Copy of Figure 3.3)



(c) (Faulty) Implementation Model of the PPU

Figure 4.2: Example for Black-Box Sampling of Infinite Configuration Spaces

intersection of \mathcal{D}_i and $[-k, k]$). As a shorthand, we utilize the interval notation $[k, k']$ to also denote the set of values defined by interval $[k, k']$ such that we may apply set operators (e.g., $[k, k'] \cap \mathbb{N}_0$ denotes the set of natural numbers in interval $[k, k']$).

Notation 4.2. Let $[k, k']$ be an interval of domain \mathcal{D} with $\{k, k'\} \subseteq \mathcal{D}$. We use $[k, k']$ to denote the set $\{k'' \mid k'' \in \mathcal{D} \wedge k'' \geq k \wedge k'' \leq k'\}$.

It should be noted that we limit our considerations in this thesis to value domains \mathbb{N}_0 and \mathbb{B} . Hence, we only include non-negative values in $[-k, k] \cap \mathbb{N}_0$. In general, we require value domain \mathcal{D} to have a total ordering defined on its elements and finiteness of $[-k, k] \cap \mathcal{D}$ for applying k -bounded t -wise sampling.

Definition 4.2 (k -bounded t -wise Coverage). Let Ω be a set of parameters, Θ_Ω be a combination space over Ω , $k \in \mathbb{N}$, and $t \in \mathbb{N}$. Then, test suite $\Theta \subseteq \Theta_\Omega$ satisfies k -bounded t -wise coverage if for each unbounded parameter $\omega_i \in \Omega$, the values $[-k, k] \cap V_i$ within t -wise value schemas are covered. We use

$$CA_B(\Omega, \Theta_\Omega, [-k, k], t)$$

to denote the set of all covering arrays satisfying k -bounded t -wise coverage.

Again, k -bounded t -wise sampling may be further adapted by providing for each unbounded parameter $\omega_i \in \Omega$ an individual value k_i . However, k -bounded t -wise coverage may still miss critical cases, being illustrated by the following example.

Example 4.3 (k -bounded t -wise Coverage). Again, consider the example in Figure 4.2. With k -bounded t -wise sampling, we encounter the same problem as described in Example 4.2 for k -random t -wise sampling. In particular, we have to choose $k \geq 30$ such that we obtain a test configuration covering the error of the faulty implementation. Furthermore, this example illustrates that we might miss critical configurations with black-box approaches. For instance, location *emergency stop* is not even reachable for configurations with $b < 30$.

For the black-box sampling strategies introduced so far, we obtain a similar result as described in the background chapter (see Lemma 2.1). Specifically, consider k -bounded t -wise covering arrays CA_B and k' -random t' -wise covering arrays CA_R . Here, the set of all possible k -bounded t -wise covering arrays CA_B is a subset of the set of all possible k' -random t' -wise covering arrays CA_R in case of $t \geq t'$ and $k \geq k'$. The reasoning behind this result is the following.

- Consider the case of $k = k'$ and $t = t'$. Then, the interval $[-k, k]$ has exactly $2k' + 1$ elements (in case of integers). Hence, the set of all possible k' -random t' -wise covering arrays CA_R includes the set of all possible k -bounded t -wise covering arrays CA_B as we may randomly choose exactly the values in $[-k, k]$.
- Next, consider the values of k and k' . Here, increasing k results in more elements in the interval $[-k, k]$. Hence, increasing the value of k means that there are less possible covering arrays in CA_B , such that we have to require $k \geq k'$.

- Finally, consider the values of t and t' . Here, increasing t (or t') means that more value schemas have to be covered. Hence, increasing t means that there are less possible covering arrays. Therefore, we require $t \geq t'$ for the subset relation to hold.

Furthermore, the assumption only holds for a non-empty combination space Θ_Ω (i.e., $\Theta_\Omega \neq \emptyset$) as an empty combination space is covered by the empty covering array (such that $t < t'$ would be possible in this case).

Lemma 4.1. Let Ω be a set of parameters, Θ_Ω be a combination space over Ω , and $\{k, k', t, t'\} \subseteq \mathbb{N}$. Then

$$CA_B(\Omega, \Theta_\Omega, [-k, k], t) \subseteq CA_R(\Omega, \Theta_\Omega, 2k' + 1, t')$$

iff $t \geq t'$, $k \geq k'$, and $\Theta_\Omega \neq \emptyset$.

Proof. Let Ω be a set of parameters, Θ_Ω be a combination space over Ω , and $\{k, k', t, t'\} \subseteq \mathbb{N}$. Then, it holds by definition that $CA_B(\Omega, \Theta_\Omega, [-k, k], t) \subseteq CA_R(\Omega, \Theta_\Omega, 2k' + 1, t')$ iff $t \geq t'$ and $k \geq k'$ (see Definitions 4.1 and 4.2). Here, $[-k, k]$ contains $2k + 1$ elements such that $k \geq k'$ preserves the subset relation. Increasing the value of t (or t') results in more value schemas that need to be covered such that $t \geq t'$ preserves the subset relation. Finally, the set of all possible k' -random t' -wise covering arrays CA_R includes the set of all possible k -bounded t -wise covering arrays CA_B as we may randomly choose exactly the values in $[-k, k]$. \square

As illustrated above, t -wise CIT may be generalized for sampling infinite configuration spaces (e.g., with k -random t -wise sampling and k -bounded t -wise sampling). Moreover, we might apply more sophisticated strategies for random sampling such as *star discrepancy* [152] to obtain a better distribution of randomly selected values. Additionally, we might use *adversarial machine learning* for sampling as proposed by Temple et al. [184]. Adversarial machine learning exploits knowledge about previously trained classifiers to find configurations in low confidence areas. However, we may still miss critical cases covering behavior with these black-box approaches (see Examples 4.2 and 4.3). Furthermore, it is unclear how to choose a value for k . Hence, we next consider *white-box* sampling strategies taking into account models from the solution space. Here, it should be noted that we could apply techniques for model learning (e.g., as presented by Aichernig et al. [7] for TA, Vaandrager and Midya [191] and Garhewal et al. [89] for *register automata* and Damasceno et al. [77, 78] for product-line models) in a black-box setting such that we can apply white-box strategies based on the learned model.

4.1.2 White-Box Sampling Strategies for Infinite Configuration Spaces

Only considering the problem space (i.e., extended feature models in our case) for sample-based testing may be problematic in terms of the effectiveness of generated test suites as illustrated by Example 4.3. Hence, we utilize information from the solution space (e.g., CoPTA models) to select parameter values and improve effectiveness by applying a white-box sampling approach.

Example 4.4. As described in Example 4.3, location *emergency stop* is only reachable in PPU configurations for which $b \geq 30$ holds (see Figure 4.2b). Furthermore, only configurations with $b \geq 30$ contain the minimum delay for reaching this location. Hence, we cannot simply minimize all parameters to find runs with minimum delay (as the minimum value for b is 20). When we also consider the constraints imposed by the EFM in Figure 4.2a, we obtain the constraint $a \geq 15 \wedge b \geq 30$ for parameters a and b (note, that we omit other parameters for improved readability in this example). As a result, all configurations satisfying $a \geq 15 \wedge b \geq 30$ cover the specified best-case execution time for reaching *emergency stop*. Furthermore, this set of configurations contains infinitely many test cases as neither a nor b is restricted by an upper bound.

The parameter constraint described in Example 4.4 provides solution-space knowledge in addition to the problem-space knowledge from the EFM to guide the selection of test cases in the direction of particularly crucial configurations into a sample for covering minimum/maximum execution delays of product lines with configurable parametric real-time constraints. As a result, a parameter constraint (see Definition 3.1) may subsume a potentially infinite set of value schemas. Hence, we generalize the notion of value schemas (see Definition 2.9) to symbolic representations. Here, we may compare parameters from Ω with other parameters and with values from V . It should be noted that the idea of symbolic value schemas can also be applied in a white-box setting as these schemas have to purpose of (symbolically) describing a set of solutions. We first define the notion of *schema constraints*.

Definition 4.3 (Schema Constraint). Let Θ_Ω be a valid combination space over a set of parameters Ω defined over value domain V . By $\Psi(\Omega)$, we denote the set of *schema constraints* over Ω , where $\psi \in \Psi(\Omega)$ iff

$$\psi := \text{true} \mid \omega \sim \omega' \mid \omega \sim v \mid \psi \wedge \psi$$

with $\sim \in \{<, \leq, \geq, >\}$, $\{\omega, \omega'\} \subseteq \Omega$, and $v \in V$.

Therewith, we next define *symbolic value schemas*, subsuming a potentially infinite set of value schemas.

Definition 4.4 (Symbolic Value Schema). Let Θ_Ω be a valid combination space over a set of parameters Ω defined over value domain V and $\psi \in \Psi(\Omega)$ be a schema constraint.

- By $\llbracket \psi \rrbracket \subseteq \Theta_\Omega$, we denote the set of test cases satisfying constraint $\psi \in \Psi(\Omega)$.
- A constraint $\psi \in \Psi(\Omega)$ is a (valid) *symbolic value schema* w.r.t. Θ_Ω iff $\llbracket \psi \rrbracket \neq \emptyset$.
- By $\Psi(\Theta_\Omega) \subseteq \Psi(\Omega)$, we refer to the set of symbolic value schemas of Θ_Ω .

Example 4.5 (Symbolic Value Schema). Consider the parameter constraint $a \geq 15 \wedge b \geq 30$ from Example 4.4. This constraint is a symbolic value schema with $\Omega = \{a, b\}$ and $V = \mathbb{N}_0$.

Having defined *symbolic* value schemas, we may utilize this representation to characterize (potentially infinite) sets of value schemas (see Definition 2.14) being equivalent w.r.t. a given criterion for CIT sampling of infinite configuration spaces. Furthermore, we may apply symbolic value schemas to include additional solution-space knowledge into our sampling approach. Next, we combine this notion with the concept of *t-wise* sampling, resulting in *t-wise* Ψ -sampling. Here, we not only consider a given set Ψ of symbolic value schemas but also the negations $\neg\psi$ for each $\psi \in \Psi$ (if $\neg\psi$ is a valid symbolic value schema w.r.t. Θ_Ω). For instance, assume that symbolic value schema ψ describes all variants having BCET for reaching ℓ . Then, all variants described by $\neg\psi$ do not exhibit BCET behavior for reaching ℓ . The rationale behind this corresponds to well-established principles known from combinatorial testing based on equivalence classes, where the boundaries as well as a representative from interiors of parameter-value intervals should be covered by test cases. Additionally, this is also done in boundary-value testing, where boundary behavior as well as *normal* behavior should be tested.

Definition 4.5 (*t-wise* Ψ -Sampling). Let Ω be a set of parameters, Θ_Ω be a combination space over Ω , Ψ be a set of symbolic value schemas, and $t \in \mathbb{N}$. We denote

$$\hat{\Psi} = \Psi \cup \{\neg\psi \mid \psi \in \Psi \wedge \neg\psi \in \Psi(\Theta_\Omega)\}.$$

A covering array $\Theta \subseteq \Theta_\Omega$ satisfies *t-wise* Ψ -coverage if $\Theta \cap \llbracket \psi \rrbracket \neq \emptyset$ for each $\psi = (\bigwedge_{\psi' \in \hat{\Psi}} \psi') \in \Psi(\Theta_\Omega)$ with $\hat{\Psi}' \subseteq \hat{\Psi}$ such that $|\hat{\Psi}'| = t$. Furthermore, we use

$$CA_S(\Omega, T_\Omega, \Psi, t)$$

to denote the set of all *t-wise* Ψ -covering arrays.

For *t-wise* Ψ -sampling, the set $\hat{\Psi}$ contains all symbolic value schemas $\psi \in \Psi$ as well as all negations $\neg\psi$ (in case of $\neg\psi$ yielding valid symbolic value schemas w.r.t. Θ_Ω). Thereupon, we have to cover all conjunctions of *t-wise* combinations of symbolic value from $\hat{\Psi}$ yielding valid value schemas to satisfy *t-wise* Ψ -coverage.

Example 4.6 (*t-wise* Ψ -Sampling). Consider the set $\Psi = \{\psi, \psi'\}$ of symbolic value schemas with $\psi = (a \geq 15)$ and $\psi' = (b \geq 30)$. To obtain a 1-wise Ψ -covering array, we cover ψ and ψ' as well as $\hat{\psi} = a < 15$ and $\hat{\psi}' = (b < 30)$. To obtain a pairwise Ψ -covering array, we cover all valid pairwise conjunctions of $\psi, \psi', \hat{\psi}$, and $\hat{\psi}'$. For instance, we have to cover $\psi \wedge \hat{\psi}' = (a \geq 15 \wedge b < 30)$. However, $\psi \wedge \hat{\psi} = (a \geq 30 \wedge a < 30)$ does not have to be covered as this symbolic value schema is not satisfiable.

For *t-wise* Ψ -sampling, we may now adapt the previous result from *k*-random *t-wise* CIT and *k*-bounded *t-wise* coverage (see Lemma 4.1). Specifically, we show that increasing *t* and Ψ results in a smaller set of possible covering arrays CA_S . The

reasoning for this is similar to the previous results. A larger set of symbolic value schemas means that there exist less covering arrays satisfying t -wise Ψ -coverage (hence, $\Psi \supseteq \Psi'$). Furthermore, a larger t increases the number of symbolic value schemas that need to be covered (hence, $t \geq t'$). Moreover, the assumption only holds for a non-empty combination space Θ_Ω (i.e., $\Theta_\Omega \neq \emptyset$) as an empty combination space is covered by the empty covering array (such that $t < t'$ would be possible in this case).

Theorem 4.1. Let Ω be a set of parameters, Θ_Ω be a combination space over Ω , Ψ be a set of symbolic value schemas, and $t \in \mathbb{N}$. Then

$$CA_S(\Omega, \Theta_\Omega, \Psi, t) \subseteq CA_S(\Omega, \Theta_\Omega, \Psi', t')$$

iff $t \geq t'$, $\Psi \supseteq \Psi'$, and $\Theta_\Omega \neq \emptyset$.

Proof. Let Ω be a set of parameters, Θ_Ω be a combination space over Ω , Ψ be a set of symbolic value schemas, and $t \in \mathbb{N}$. Then, it holds by definition that $CA_S(\Omega, \Theta_\Omega, \Psi, t) \subseteq CA_S(\Omega, \Theta_\Omega, \Psi', t')$ (see Definition 4.6). Here, a larger set of symbolic value schemas means that there exist less covering arrays satisfying t -wise Ψ -coverage (hence, $\Psi \supseteq \Psi'$). Furthermore, a larger t increases the number of symbolic value schemas that need to be covered (hence, $t \geq t'$). \square

We conclude this section by presenting an algorithm for computing a t -wise Ψ -covering array for a set of symbolic value schemas. Here, we would like to derive a minimal covering array Θ (in terms of the number of test cases $\theta \in \Theta$). However, this is computationally very expensive as we would have to check satisfiability for each possible combination of symbolic value schemas (to see if there exist a test cases being valid for these combinations). In the worst case, we would have to check $2^{|\Psi|}$ possible combinations. Hence, we do not derive a minimal covering array and instead use a different approach. Here, a naive approach would be to simply derive a test case from a symbolic value schema and then check if this test case is also valid for other schemas. However, this may result in large covering arrays.

Therefore, we utilize a *greedy* algorithm as a tradeoff between a naive and a minimal solution for computing a t -wise Ψ -covering array for a set of symbolic value schemas. To this end, Algorithm 4.1 takes as inputs a set of parameters Ω , the combination space Θ_Ω , a set of symbolic value schemas Ψ , and a value $t \in \mathbb{N}$. It should be noted that the combination space Θ_Ω is represented by a symbolic constraint as Θ_Ω may be infinite. For instance, the combination space for CoPTA is an EFM m . Here, procedure `MAIN` (see lines 1–7) starts by initializing the set Θ of test cases with the empty set (see line 2). Next, we generate the *working set* WS , containing all valid symbolic value schemas for t -wise Ψ -coverage, i.e., all possible t -wise combinations of symbolic value schemas in $\hat{\Psi}$ (line 3). Thereafter, we continuously generate new test cases covering symbolic value schemas from WS until WS is empty (lines 4–7). Specifically, we achieve this by calling `GETTESTCASE` in line 5 by passing WS to this procedure and obtaining a test case θ and an updated *working set* (see description below on how this procedure is implemented). Then, test case θ is added to Θ (line 6) and WS is updated (line 7).

Algorithm 4.1: Greedy-based Sampling Algorithm

Input : parameters Ω , combination space Θ_Ω , symbolic value schemas Ψ , t
Output: covering array Θ

```

1 procedure MAIN
2    $\Theta := \emptyset$ 
3    $WS := \text{GETALLVALUESCHEMAS}(\Omega, \Theta_\Omega, \Psi, t)$ 
4   while  $WS \neq \emptyset$  do
5      $(WS', \theta) := \text{GETTESTCASE}(WS)$ 
6      $\Theta := \Theta \cup \{\theta\}$ 
7      $WS := WS'$ 
8 procedure GETTESTCASE(schemas  $WS$ )
9    $\psi := \text{true}$ 
10  foreach  $\psi' \in WS$  do
11    if  $\llbracket \psi \wedge \psi' \rrbracket \neq \emptyset$  then
12       $\psi := \psi \wedge \psi'$ 
13       $WS := WS \setminus \{\psi'\}$ 
14  return  $(WS, t \in \llbracket \psi \rrbracket)$ 

```

Furthermore, procedure GETTESTCASE (lines 8–14) takes as input a *working set* WS and returns a test case θ covering *some* symbolic value schemas in WS . To achieve this, we first initialize variable ψ with true, serving as the basis for the constraint describing the resulting test case (see line 9). Then, we proceed with the *greedy* part of this algorithm. In the for-each loop (lines 10–13), we conjugate ψ with symbolic value schemas from WS as long as the resulting constraint is satisfiable. This is done by checking for each $\psi' \in WS$ if the conjunction is satisfiable (see line 11). If this is the case, ψ is updated to $\psi \wedge \psi'$ (line 12) and ψ' is removed from WS (see line 13). Finally, GETTESTCASE returns the set of *remaining* uncovered symbolic value schemas WS and a test case satisfying ψ (see line 14). Here, we use an SMT solver to derive a test case satisfying ψ . Note, that this algorithm might not terminate if the satisfiability of symbolic value schemas $\psi \in \Psi$ is undecidable (see lines 11 and 14).

Example 4.7 (Greedy-based Sampling Algorithm). Consider the PPU with CoPTA $\mathcal{C} = (L, \ell_0, \Sigma, C, P_F, P_N, I, E, m, \eta)$ and EFM m depicted in Figures 4.2a and 4.2b, and the three symbolic value schemas

- $\psi_1 := m \wedge \text{Stop} \wedge b \geq 30$,
- $\psi_2 := m \wedge \neg \text{Resume}$, and
- $\psi_3 := m \wedge \neg \text{Resume} \wedge \text{Metal} \wedge \neg \text{Stop} \wedge b \geq 40$.

Here, ψ_1 comprises all variants having BCET for reaching location *emergency stop*, ψ_2 comprises all variants with BCET for reaching *WP picked*, and ψ_3 comprises all variants having WCET for reaching the *ramp* (note, that we show

how to derive these symbolic value schemas later in this chapter). Next, we utilize Algorithm 4.1 to generate a 1-wise Ψ -covering array.

We start by initializing the covering array $\Theta := \emptyset$ and the working set $WS := \{\psi_1, \psi_2, \psi_3\}$ (see lines 2–3). Then, we generate the first test case (line 5). To this end, we set variable ψ to true (line 9) and conjugate a value schema from WS (e.g., ψ_1) to ψ such that $\psi = \psi_1$ (lines 10–12), and we remove ψ_1 from WS (line 13). Thereafter, we check if conjugating ψ with another element from WS is possible (lines 10–11). As we can indeed conjugate the current constraint ψ with ψ_2 , we obtain the intermediate constraint $\psi = \psi_1 \wedge \psi_2$ (line 12). As we cannot conjugate another element from WS to ψ (as $\llbracket \psi \wedge \psi_3 \rrbracket = \emptyset$), we return a concrete test case satisfying $\psi = \psi_1 \wedge \psi_2$ (line 14). For instance, we return test case

$$\begin{aligned} \theta_1 := & \text{PPU} \wedge \text{Workpiece} \wedge \text{Plastic} \wedge \neg \text{Metal} \wedge \text{Stop} \wedge \\ & \neg \text{Resume} \wedge a = 36 \wedge b = 30 \wedge c = 33. \end{aligned}$$

Thereafter, we proceed with Algorithm 4.1 as $WS \neq \emptyset$ such that there are remaining symbolic value schemas that are not yet covered.

To sum up this section, we first adapted t -wise CIT for infinite configurations. However, as black-box approaches might fail to cover critical test cases, we introduced a white-box sampling approach in terms of t -wise Ψ -sampling, being generic in the sense that we solely require a constraint called symbolic value schema, which symbolically describes (potentially infinite) sets of test cases. Based upon this, we apply t -wise Ψ -sampling to product lines with configurable parametric real-time constraints. Specifically, we want to utilize symbolic value schemas in terms of solution-space knowledge obtained from CoPTA models. However, up to this point it is unclear how to obtain these symbolic value schemas. Hence, we next introduce a symbolic semantics for CoPTA such that we can utilize this symbolic semantics afterwards to derive symbolic value schemas.

4.2 SYMBOLIC SEMANTICS OF PRODUCT LINES WITH CONFIGURABLE PARAMETRIC REAL-TIME CONSTRAINTS

As motivated in the previous section, the next step consists of deriving symbolic value schemas from CoPTA models such that we are able to incorporate solution-space knowledge into the our sampling approach. To this end, we introduce a model to describe the semantics of CoPTA in a symbolic fashion (i.e., the *symbolic state space*) in this section. With this model as a basis, we derive symbolic value schemas describing boundary behavior of CoPTA models in the subsequent section. A straightforward solution for modeling the semantics of a CoPTA can be achieved in two steps. First, we derive the set of variants (i.e., TA) of the CoPTA by utilizing parameter valuation (see Definition 3.6). Second, we derive the TLTS semantics for each of the resulting TA (see Definition 2.18). Hence, we analyze the state space for each variant separately.

Example 4.8. Consider the CoPTA in Figure 3.3 for which Table 3.2 represents the set of all possible configurations. For instance, we may choose the variant corresponding to the configuration

$$\text{PPU} \wedge \text{Workpiece} \wedge \text{Stop} \wedge \neg \text{Resume} \wedge \text{Plastic} \wedge \neg \text{Metal} \wedge a = 15 \wedge b = 30 \wedge c = 0$$

which is depicted in Figure 2.3. Then, we can derive the TLTS semantics being depicted in Figure 2.9. Finally, we may utilize this TLTS to analyze the semantics of the respective TA. In order to analyze the CoPTA model, we repeat these steps for each valid configuration.

When analyzing CoPTA as described above, we may encounter two possible issues. First, a CoPTA model may comprise an infinite number of variants, making it impossible to perform a variant-by-variant analysis. However, even a CoPTA model with a finite number of variants may be problematic, as variant-by-variant analyses are usually expensive in case of product lines containing many variants [53]. Furthermore, many analyses are redundant as several parts of a CoPTA model are present in many variants. Second, the TLTS semantics of TA has, in general, an infinite state space (e.g., see Example 2.12). The second problem could be addressed by utilizing zone graphs (see Definition 2.20) instead of TLTS. However, the first problem remains to be solved.

In order to avoid analyzing CoPTA in a variant-by-variant way, we adapt the notion of zone graphs for CoPTA models to represent the CoPTA semantics in a symbolic fashion. Hence, we construct a single *featured parametric zone graph* to reason about all variants of a CoPTA model at once. In particular, this allows us to analyze CoPTA models even in case of an infinite number of variants. However, it should be noted that many interesting properties are semi-decidable for CoPTA (as discussed in the previous chapter in Section 3.3). For this, we lift the notion of zone graphs (see Definition 2.20) to CoPTA. Here, a *symbolic state* $\langle \ell, \gamma \rangle$ of the featured parametric zone graph of CoPTA \mathcal{C} consists of location ℓ and a *featured parametric zone* $\gamma = [\lambda]\phi \in \mathcal{B}(\mathcal{C}, P_F, P_N)$ (i.e., a featured parametric clock constraint). Additionally, we lift the set notation of zone graphs to featured parametric zone graphs such that a featured parametric zone γ represents a (potentially infinite) set \mathcal{D} of clock valuations satisfying γ . Again, we may use γ and \mathcal{D} interchangeably (as done for standard notations of zone graphs). Note, that we apply logical operators (e.g., conjunction) on the set \mathcal{D} as we assume that \mathcal{D} always has an underlying constraint γ . We solely use \mathcal{D} as it sometimes allows us to formulate properties and definitions in a more concise way (and because zones of TA are also usually described in terms of set in related work). In particular, we utilize the following notations and operations for \mathcal{D} .

- D_ν denotes the zone for a valuation $\nu \in \llbracket m \rrbracket$ of a corresponding EFM m . Hence, D_ν describes a zone corresponding to a variant (i.e., TA).
- \mathcal{D}_m denotes the featured parametric zone corresponding to γ . Therefore, \mathcal{D}_m contains for each $\nu \in \llbracket m \rrbracket$ of a corresponding EFM m the set D_ν .
- \mathcal{D}^\uparrow denotes the *future* of \mathcal{D} . This coincides with removing all upper bounds from γ (for all $\nu \in \llbracket m \rrbracket$).

- $R(\mathcal{D})$ denotes the *reset* of clocks $R \subseteq C$ on featured parametric zone \mathcal{D} . Intuitively, this coincides with removing all inequalities containing clocks $c \in R$ from γ and conjugating constraints $c = 0$ for each $c \in R$ to γ .

Definition 4.6 (Featured Parametric Zone). Let $\gamma \in \mathcal{B}(C, P_F, P_N)$ be a featured parametric clock constraint of CoPTA $\mathcal{C} = (L, \ell_0, \Sigma, C, P_F, P_N, I, E, m, \eta)$. For describing the semantics of \mathcal{C} , we call γ a *featured parametric zone*. We utilize the following notations:

- $D_v = \{u \mid u \in v(\gamma)\}$ denotes the zone of γ for $v \in \llbracket m \rrbracket$,
- $\mathcal{D}_m = \{u \mid u \in D_v, v \in \llbracket m \rrbracket\}$ denotes the featured parametric zone γ ,
- $\mathcal{D}^\dagger = \{u + d \mid u \in \mathcal{D}, d \in \mathbb{T}_C\}$ denotes the *future* of featured parametric zone \mathcal{D} , and
- $R(\mathcal{D}) = \{[R \mapsto 0]u \mid u \in \mathcal{D}\}$ denotes the *reset* of clocks in set R on featured parametric zone \mathcal{D} .

We use \mathcal{D} and γ interchangeably such that $u \in \mathcal{D} \Leftrightarrow u \in \gamma$.

Similar to zone graphs of TA, featured parametric zones also contain difference constraints for each pair of clocks to keep track of clock differences. Furthermore, a reset does not remove difference constraints from γ but instead updates the difference. Concerning practical applicability, featured parametric zones can be implemented in a similar way as zones (of TA zone graphs). When considering constraint γ describing set \mathcal{D} , a featured parametric zone contains comparisons of clocks with parameters (in addition to comparisons with constants in clock domain \mathbb{T}_C). Here, the operators for reset and future can be applied as usual (i.e., setting a clock to zero and remove the upper bound of a constraint, respectively), such that clock differences (which are updated by resets) may also include parameters. Moreover, we “collect” constraints over parameters by conjugating new constraints to the current constraint of the previous featured parametric zone.

Having defined featured parametric zones, we proceed with describing featured parametric zone graphs. As indicated above, each symbolic state $z = \langle \ell, \mathcal{D} \rangle$ consists of a location ℓ and a featured parametric zone \mathcal{D} . Here, the zone γ_0 (or \mathcal{D}_0) of the initial state is a constraint where every clock is set to value 0 (i.e., $\forall c \in C : c = 0$). Furthermore, the constraint is conjugated with EFM m . Therewith, we ensure that we only consider behavior of valid variants while exploring the state space of a CoPTA. Finally, *symbolic transitions* of featured parametric zone graphs correspond to switches of a CoPTA such that the target of a symbolic transition contains all constraints imposed by the respective switch.

Definition 4.7 (Featured Parametric Zone Graph). Let $\mathcal{C} = (L, \ell_0, \Sigma, C, P_F, P_N, I, E, m, \eta)$ be a CoPTA. The *featured parametric zone graph* of \mathcal{C} is a tuple $(Z, z_0, \rightsquigarrow)$, where

- $Z = L \times \mathcal{B}(C, P_F, P_N)$ is a set of *symbolic states*,

- $z_0 = \langle \ell_0, \mathcal{D}_0 \rangle$ is the *initial state*, and
- $\rightsquigarrow \subseteq Z \times Z$ is a *symbolic transition relation* being the least relation satisfying the following rule: $\langle \ell, \mathcal{D} \rangle \rightsquigarrow \langle \ell', \mathcal{D}' \rangle$ if $e = (\ell, g, \sigma, R, \ell') \in E$ and

$$\mathcal{D}' = R \left(\mathcal{D}^\uparrow \wedge g \wedge \eta(e) \wedge I(\ell') \right) \wedge I(\ell').$$

We refer to the zone-graph semantics of CoPTA \mathcal{C} by $\llbracket \mathcal{C} \rrbracket_Z$.

With featured parametric zone graphs, we can now describe decision procedures for interesting CoPTA properties (e.g., reachability and derivation of symbolic value schemas). However, decidability of reachability (i.e., EF-reachability) and other interesting properties are semi-decidable for CoPTA. Hence, we obtain a correct result if the decision procedure terminates (but the procedure does, in general, not terminate).

Example 4.9 (Featured Parametric Zone Graph). Figure 4.3 gives an example for an extract of the featured parametric zone graph of the CoPTA depicted in Figure 3.3. Note, that we repeat the CoPTA model as well as the corresponding EFM (see Figure 3.2) in Figure 4.3 for the convenience of the reader. Furthermore, the EFM corresponds to the configuration constraint m :

$$\begin{aligned} & \text{PPU} \wedge \text{Workpiece} \wedge (\text{Plastic} \vee \text{Metal}) \wedge (\neg \text{Plastic} \vee \neg \text{Metal}) \wedge \\ & (\text{Resume} \Rightarrow \text{Stop}) \wedge (\text{Plastic} \Rightarrow (a \geq 15 \wedge b \geq 20 \wedge b \leq 30)) \wedge \\ & (\text{Resume} \Rightarrow c \geq 30) \wedge (\text{Metal} \Rightarrow (a \geq 20 \wedge a \leq 25 \wedge b \geq 30 \wedge b \leq 40)). \end{aligned}$$

The featured parametric zone graph (see Figure 4.3b) starts with an initial state consisting of the initial location of the CoPTA and all clocks set to 0. Additionally, m describes the EFM as mentioned above. Here, m needs to be included into the featured parametric zone as otherwise we may reach states not satisfying m . Next, we reach a state with location *stack* as the CoPTA contains a corresponding switch. Here, we require $x \geq 10 \wedge x \leq a$ due to the clock constraint of the switch from *init* to *stack*. Moreover, we require $x \leq b$ because of the invariant of location *stack*. When reaching the symbolic state comprising location *emergency stop*, the Boolean parameter *Stop* must have value true as otherwise this location would not be reachable. In addition to this requirement, we have $y - x \geq 30$ as x is reset. When reaching the state comprising *WP picked*, we require $\neg \text{Resume}$ due to the constraint of the respective switch. It should be noted that this restriction remains unchanged for all subsequent symbolic state as these subsequent state are also only reachable through this path in case of $\neg \text{Resume}$.

Thereafter, we have three different symbolic states for location *ramp* as there are three different mutually exclusive featured parametric clock constraints. Hence, there does not exist a single state satisfying all of these constraints. Note, that there is no fourth state for *ramp* corresponding to a variant where

all three clock constraints are deselected as either *Plastic* or *Metal* has to be selected according to EFM m .

Finally note, that we omitted redundant parameter constraints to keep the featured parametric zone graph in Figure 4.3c as readable and concise as possible. For instance, we have to require $a \geq 10$ in the symbolic state comprising location *stack*. Otherwise, this location would not be reachable due to the clock constraint $x \geq 10 \wedge x \leq a$ of the switch from *init* to *stack*. However, EFM m already requires $a \geq 15$ such that $a \geq 10$ is redundant.

In this section, we introduced a model for describing the semantics of CoPTA in a symbolic fashion in terms of featured parametric zone graphs. Therewith, we proceed by utilizing this symbolic semantics for t -wise Ψ -sampling of product lines with configurable parametric real-time constraints. In particular, we utilize featured parametric zone graphs to derive symbolic value schemas.

4.3 SAMPLING PRODUCT LINES WITH CONFIGURABLE PARAMETRIC REAL-TIME CONSTRAINTS

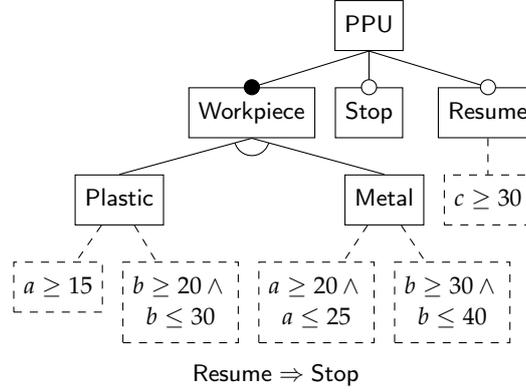
In the previous section, we introduced a model to describe the semantics of CoPTA in a symbolic fashion in terms of featured parametric zone graphs. In this section, we utilize this symbolic semantics for deriving symbolic value schemas describing boundary behavior of CoPTA models. Therewith, we can apply t -wise Ψ -sampling as described in Section 4.1.2. In particular, we are interested in sampling variants which contain runs having the *best-case execution times (BCET)* and *worst-case execution times (WCET)* [203], respectively, for reaching each location of a CoPTA. As a result, we want to achieve *minimum/maximum delay coverage (M/MD coverage)*. For M/MD coverage, the set of test goals consists of the set L of all locations. A covering array Θ satisfies M/MD coverage if it contains at least one configuration for each $\ell \in L$ such that the corresponding variant comprises a run having overall BCET (or WCET) for reaching ℓ (i.e., there exists no other variant having a smaller BCET or greater WCET). Here, it should be noted that a WCET may not necessarily exist for every target location ℓ in the presence of loops (where ℓ is not located in the loop). Hence, a covering array Θ does not need to contain a test case reaching ℓ with WCET in this case. Additionally, a single $\theta \in \Theta$ may contain BCET (or WCET) for more than one location.

Example 4.10. Consider CoPTA \mathcal{C} in Figure 4.3b with EFM m in Figure 4.3a. Here, the BCET for reaching location *ramp* is 17 seconds. This can be achieved in all variants satisfying

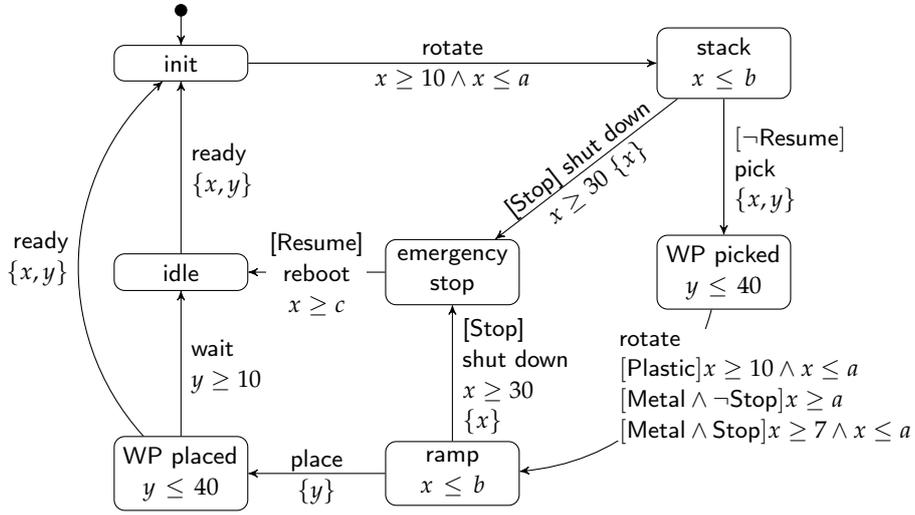
$$m \wedge \neg \text{Resume} \wedge \text{Metal} \wedge \text{Stop}$$

as we require $\neg \text{Resume}$ to reach *WP picked* and $\text{Metal} \wedge \text{Stop}$ for the fastest possible *rotation*. Furthermore, the WCET for reaching the *ramp* is 80 seconds, being achievable in all variants satisfying

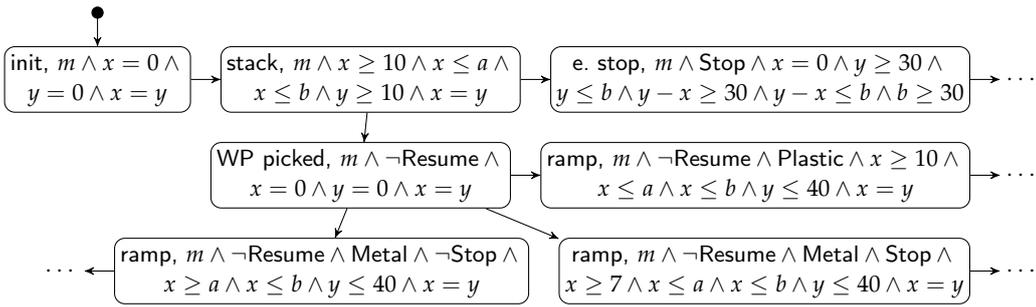
$$m \wedge \neg \text{Resume} \wedge \text{Metal} \wedge \neg \text{Stop} \wedge b = 40$$



(a) Extended Feature Model of the PPU Extract (Copy of Figure 3.2)



(b) CoPTA Model of the Extract of the PPU (Copy of Figure 3.3)



(c) Extract of the Featured Parametric Zone Graph of the CoPTA in Figure 4.3b

Figure 4.3: Example for a Featured Parametric Zone Graph

where we require $b = 40$ (greatest possible value for b according to m) so that we can wait as long as possible in location *stack* and $\text{Metal} \wedge \neg \text{Stop}$ to have no upper bound for *rotating* the crane. In order to satisfy M/MD coverage, we need to find variants for each location ℓ having BCET (and WCET) for reaching ℓ .

In the following section, we formally define BCET and WCET.

4.3.1 Best-Case/Worst-Case Execution Times for TA Locations

In many cases (e.g., time-critical systems), boundary behavior (e.g., BCET/WCET) is interesting as these corner cases often are especially error prone [190]. Recall, that sampling procedures not considering the solution space have a smaller chance of including variants with BCET/WCET (e.g., see Examples 4.2 and 4.3) as we cannot simply minimize or maximize parameters used in an EFM to find these variants. Formally, the execution time of a run ρ of a TA is the sum of delays of each timed step.

Definition 4.8 (Execution Time). Let $\rho = s_0 \xrightarrow{(d_1, \sigma_1), \dots, (d_n, \sigma_n)} s_n$ be a finite timed run of a TA. The *execution time* of ρ is defined as $d(\rho) = d_1 + \dots + d_n$.

Here, a run ρ_{ℓ_n} in $\llbracket \mathcal{A} \rrbracket_R$ (i.e., in the set of runs of TA \mathcal{A}) has *best-case* (or *worst-case*) execution time for reaching ℓ_n if there exists no other run ρ'_{ℓ_n} requiring less (or more) time for reaching the target location. Furthermore, we are only interested in the execution time $d(\rho_{\ell_n})$ for the *first* visit of target location ℓ_n . In particular, this is the case for WCET as otherwise the execution time can be arbitrarily long if the TA contains a loop, allowing us to visit the target over and over again.

Definition 4.9 (Best-Case/Worst-Case Execution Time). Let $\rho_{\ell_n} \in \llbracket \mathcal{A} \rrbracket_R$ be a finite timed run of TA $\mathcal{A} = (L, \ell_0, \Sigma, C, I, E)$ with $\rho_{\ell_n} = \langle \ell_0, u_0 \rangle \xrightarrow{(d_1, \sigma_1), \dots, (d_n, \sigma_n)} \langle \ell_n, u_n \rangle$ such that $\ell_n \notin \{\ell_0, \dots, \ell_{n-1}\}$. Run ρ_{ℓ_n} has the *best-case execution time* for reaching ℓ_n iff

$$\forall \rho'_{\ell_n} = \langle \ell_0, u_0 \rangle \xrightarrow{(d'_1, \sigma'_1), \dots, (d'_n, \sigma'_n)} \langle \ell_n, u'_n \rangle \in \llbracket \mathcal{A} \rrbracket_R : d(\rho_{\ell_n}) \leq d(\rho'_{\ell_n}).$$

Run ρ_{ℓ_n} has the *worst-case execution time* for reaching ℓ_n iff

$$\forall \rho'_{\ell_n} = \langle \ell_0, u_0 \rangle \xrightarrow{(d'_1, \sigma'_1), \dots, (d'_n, \sigma'_n)} \langle \ell_n, u'_n \rangle \in \llbracket \mathcal{A} \rrbracket_R : d(\rho_{\ell_n}) \geq d(\rho'_{\ell_n}).$$

Note, that there is not necessarily a unique run for BCET and WCET as different runs may have the same execution time.

Example 4.11 (Best-Case/Worst-Case Execution Time). Consider CoPTA \mathcal{C} in Figure 4.3b with EFM m in Figure 4.3a as described in Example 4.10.

Furthermore, consider a TA $\mathcal{A} \in \llbracket \mathcal{C} \rrbracket_{\mathcal{C}}$ with a configuration satisfying $m \wedge \neg \text{Resume} \wedge \text{Metal} \wedge \text{Stop}$. Here, \mathcal{A} contains a run $\rho_1 \in \llbracket \mathcal{A} \rrbracket_R$ with

$$\rho_1 = \langle \text{init}, x = 0, y = 0 \rangle \xrightarrow{(10, \text{rotate}), (0, \text{pick}), (7, \text{rotate})} \langle \text{ramp}, x = 7, y = 7 \rangle$$

such that $d(\rho_1) = 10 + 0 + 7 = 17$. This run has the BCET for reaching the *ramp* as there exists no other $\rho'_1 \in \llbracket \mathcal{A} \rrbracket_R$ with $d(\rho'_1) < d(\rho_1)$. Furthermore, this variant \mathcal{A} has the BCET for reaching the *ramp* as there is no $\mathcal{A}' \in \llbracket \mathcal{C} \rrbracket_{\mathcal{C}}$ with a faster run reaching location *ramp*. Next, consider an $\mathcal{A}' \in \llbracket \mathcal{C} \rrbracket_{\mathcal{C}}$ with a configuration satisfying $m \wedge \neg \text{Resume} \wedge \text{Metal} \wedge \neg \text{Stop} \wedge b = 40$. Here, \mathcal{A}' contains a run $\rho_2 \in \llbracket \mathcal{A}' \rrbracket_R$ with

$$\rho_2 = \langle \text{init}, x = 0, y = 0 \rangle \xrightarrow{(10, \text{rotate}), (30, \text{pick}), (40, \text{rotate})} \langle \text{ramp}, x = 40, y = 40 \rangle$$

such that $d(\rho_2) = 10 + 30 + 40 = 80$. As a result, \mathcal{A}' is a variant having the WCET for reaching the *ramp*.

Therewith, we proceed with considering M/MD coverage.

4.3.2 Minimum/Maximum Delay Coverage

As described above, our goal is to achieve *minimum/maximum delay coverage* (M/MD coverage). Here, a covering array Θ satisfies M/MD coverage if for each location $\ell \in L$ of a CoPTA \mathcal{C} , there exists a test case $\theta \in \Theta$ such that the TA corresponding to θ has the overall BCET for reaching location ℓ (i.e., there does not exist another variant of \mathcal{C} in which location ℓ can be reached faster). Moreover, there exists a test case $\theta' \in \Theta$ such that the TA corresponding to θ' has the overall WCET for reaching ℓ (i.e., there does not exist another variant of \mathcal{C} in which location ℓ can be reached slower). Hence, M/MD coverage implies location coverage. Having defined BCET and WCET, we now formally define M/MD coverage, where we first define *minimum-delay coverage*.

Definition 4.10 (Minimum-Delay Coverage). Let $\mathcal{C} = (L, \ell_0, \Sigma, C, P_F, P_N, I, E, m, \eta)$ be a CoPTA, Θ be a covering array, and $\llbracket \mathcal{C} \rrbracket_{\Theta}$ be the set of variants of \mathcal{C} corresponding to test cases $\theta \in \Theta$. Θ satisfies *minimum-delay coverage* iff for all $\ell \in L$ there exists an $\mathcal{A} \in \llbracket \mathcal{C} \rrbracket_{\Theta}$ such that

$$\exists \rho_{\ell} \in \llbracket \mathcal{A} \rrbracket_R : (\forall \mathcal{A}' \in \llbracket \mathcal{C} \rrbracket_{\mathcal{C}} \setminus \{\mathcal{A}\} : (\forall \rho'_{\ell} \in \llbracket \mathcal{A}' \rrbracket_R : (d(\rho_{\ell}) \leq d(\rho'_{\ell}))))$$

where $\rho_{\ell} = \langle \ell_0, u_0 \rangle \xrightarrow{(d_1, \sigma_1), \dots, (d_n, \sigma_n)} \langle \ell, u_n \rangle$, $\rho'_{\ell} = \langle \ell_0, u_0 \rangle \xrightarrow{(d'_1, \sigma'_1), \dots, (d'_n, \sigma'_n)} \langle \ell, u'_n \rangle$, and $\ell \notin \{\ell_0, \dots, \ell_{n-1}\}$.

Next, we define *maximum-delay coverage*. Here, we have to consider that some locations do not have a finite WCET in which these locations can be reached. In particular, the execution time can be arbitrarily long in case of loops allowing us to visit intermediate locations infinitely often. Furthermore, intermediate locations without invariants allow for arbitrarily long waiting in these locations in some

cases (e.g., if outgoing switches reset clocks and/or do not have guards). In these cases, a covering array Θ does not need to contain a respective test case. Hence, in the following definition for maximum-delay coverage, we first check if there is a finite run having WCET in which locations can be reached.

Definition 4.11 (Maximum-Delay Coverage). Let $\mathcal{C} = (L, \ell_0, \Sigma, C, P_F, P_N, I, E, m, \eta)$ be a CoPTA, Θ be a covering array, and $\llbracket \mathcal{C} \rrbracket_{\Theta}$ be the set of variants of \mathcal{C} corresponding to test cases $\theta \in \Theta$. Θ satisfies *maximum-delay coverage* iff for all $\ell \in L$ having a variant $\mathcal{A}'' \in \llbracket \mathcal{C} \rrbracket_{\Theta}$ with run $\rho''_{\ell} \in \llbracket \mathcal{A}'' \rrbracket_R$ satisfying

$$\forall \mathcal{A}''' \in \llbracket \mathcal{C} \rrbracket_{\Theta} : \forall \rho'''_{\ell} \in \llbracket \mathcal{A}''' \rrbracket_R : d(\rho''_{\ell}) \geq d(\rho'''_{\ell}),$$

there exists an $\mathcal{A} \in \llbracket \mathcal{C} \rrbracket_{\Theta}$ such that

$$\exists \rho_{\ell} \in \llbracket \mathcal{A} \rrbracket_R : (\forall \mathcal{A}' \in \llbracket \mathcal{C} \rrbracket_{\Theta} \setminus \{\mathcal{A}\} : (\forall \rho'_{\ell} \in \llbracket \mathcal{A}' \rrbracket_R : (d(\rho_{\ell}) \geq d(\rho'_{\ell}))))$$

where $\rho_{\ell} = \langle \ell_0, u_0 \rangle \xrightarrow{(d_1, \sigma_1), \dots, (d_n, \sigma_n)} \langle \ell, u_n \rangle$, $\rho'_{\ell} = \langle \ell_0, u_0 \rangle \xrightarrow{(d'_1, \sigma'_1), \dots, (d'_n, \sigma'_n)} \langle \ell, u'_n \rangle$, and $\ell \notin \{\ell_0, \dots, \ell_{n-1}\}$.

Finally, a covering array Θ for CoPTA \mathcal{C} satisfies M/MD coverage if it satisfies minimum-delay coverage and maximum-delay coverage.

Definition 4.12 (M/MD Coverage). Let \mathcal{C} be a CoPTA and Θ be a covering array. Θ satisfies *minimum/maximum delay coverage (M/MD coverage)* iff it satisfies minimum-delay coverage and maximum-delay coverage (see Definitions 4.10 and 4.11).

So far, we can utilize featured parametric zone graphs to describe a symbolic value schema comprising every configuration in which a target location is reachable. This can be achieved by finding a solution for the constraint (i.e., a satisfying assignment of variables of a featured parametric zone) of a symbolic state containing the target location (being illustrated in detail in Section 4.3.3). However, to find configurations with BCET/WCET for reaching the target location, we adapt the corresponding CoPTA model such that the resulting featured parametric zone graph contains the necessary information. In particular, we modify CoPTA models such that finding variants with BCET/WCET can be encoded into an ILP problem. To this end, we introduce fresh variables into CoPTA models which are then minimized or maximized, respectively.

In order to solve this issue, we introduce a *minimum/maximum delay instrumentation (M/MD instrumentation)* for CoPTA. The goal of this instrumentation is finding fastest or slowest runs for reaching particular locations, such that we can derive a covering array Θ satisfying M/MD coverage. This instrumentation introduces a fresh clock χ_{ℓ} and fresh parameters $\pi_{\ell_{\text{MIN}}}$ and $\pi_{\ell_{\text{MAX}}}$ for target location ℓ . These components are then utilized as a basis for an ILP term, where we compute the minimum value of $\pi_{\ell_{\text{MIN}}}$ and the maximum value of $\pi_{\ell_{\text{MAX}}}$, respectively. Furthermore, we use these components only for the invariant of ℓ . In particular, we utilize invariant $\chi_{\ell} \leq \pi_{\ell_{\text{MIN}}}$ for BCET and invariant $\chi_{\ell} \geq \pi_{\ell_{\text{MAX}}}$ for WCET. Then, we find fastest runs for reaching ℓ by finding the minimum value for $\pi_{\ell_{\text{MIN}}}$ such that ℓ is

still reachable. Accordingly, we find the slowest runs for reaching ℓ by finding the maximum value for $\pi_{\ell_{\text{MAX}}}$ such that ℓ is still reachable.

Definition 4.13 (M/MD Instrumentation). Let $\mathcal{C} = (L, \ell_0, \Sigma, C, P_F, P_N, I, E, m, \eta)$ be a CoPTA. The M/MD instrumentation of \mathcal{C} for location $\ell \in L$ is a CoPTA

$$\mathcal{C}' = (L, \ell_0, \Sigma, C \cup \{\chi_\ell\}, P_F, P_N \cup \{\pi_{\ell_{\text{MIN}}}, \pi_{\ell_{\text{MAX}}}\}, I', E, m', \eta)$$

where $\chi_\ell \notin C$ is a fresh clock, $\pi_{\ell_{\text{MIN}}} \notin P_N$ and $\pi_{\ell_{\text{MAX}}} \notin P_N$ are fresh parameters,

- $I'(\ell) = I(\ell) \wedge \chi_\ell \leq \pi_{\ell_{\text{MIN}}}$ (best-case execution-time constraint),
- $I'(\ell) = I(\ell) \wedge \chi_\ell \geq \pi_{\ell_{\text{MAX}}}$ (worst-case execution-time constraint),
- $I'(\ell') = I(\ell')$ if $\ell' \neq \ell$, and
- $m' = m \wedge \pi_{\ell_{\text{MIN}}} \geq 0 \wedge \pi_{\ell_{\text{MAX}}} \geq 0$.

Note, that clock χ_ℓ as well as parameters $\pi_{\ell_{\text{MIN}}}$ and $\pi_{\ell_{\text{MAX}}}$ are only applied for the invariant of location ℓ . As a result, χ_ℓ is never reset.

Example 4.12 (M/MD Instrumentation). Consider the CoPTA in Figure 4.3b with M/MD instrumentation for finding the fastest run to location *ramp*. Hence, the invariant of *ramp* is

$$x \leq b \wedge \chi \leq \pi_{\text{MIN}}$$

instead of $x \leq b$. Here, the featured parametric zone graph in Figure 4.4 depicts the zone graph of the instrumented CoPTA. It should be noted that we again omitted redundant constraints to keep the zone graph readable. As we introduced a fresh clock χ , every symbolic state contains a constraint for χ . Initially, χ is set to zero (as all other clocks), and all clocks have the same value such that $x = y = \chi$. Furthermore, the difference between χ and other clocks (i.e., x and y) increases in case of resets. For instance, when reaching the symbolic state comprising *emergency stop*, we observe that $\chi - x \geq 30 \wedge \chi - x \leq b$ due to the reset of x and the corresponding invariant and guard. Additionally, all symbolic state comprising location *ramp* contain the constraint $\chi \leq \pi_{\text{MIN}}$.

Having introduced the M/MD instrumentation for a CoPTA \mathcal{C} , we show that this instrumentation of \mathcal{C} resulting in CoPTA \mathcal{C}' does not change the semantics. To this end, we formally prove that for each variant (i.e., TA) of \mathcal{C} there exists a bisimilar variant (see Definition 2.23 in the background chapter) of \mathcal{C}' and vice versa.

Lemma 4.2. Let $\mathcal{C} = (L, \ell_0, \Sigma, C, P_F, P_N, I, E, m, \eta)$ be a CoPTA and $\mathcal{C}' = (L, \ell_0, \Sigma, C \cup \{\chi_\ell\}, P_F, P_N \cup \{\pi_{\ell_{\text{MIN}}}, \pi_{\ell_{\text{MAX}}}\}, I', E, m', \eta)$ be the M/MD instrumentation of \mathcal{C}' for location $\ell \in L$. Then it holds that

$$\forall \mathcal{A} \in \llbracket \mathcal{C} \rrbracket_{\mathcal{C}} : (\exists \mathcal{A}' \in \llbracket \mathcal{C}' \rrbracket_{\mathcal{C}} : \mathcal{A} \simeq \mathcal{A}').$$

Proof. Let $\mathcal{C} = (L, \ell_0, \Sigma, C, P_F, P_N, I, E, m, \eta)$ be a CoPTA and $\mathcal{C}' = (L, \ell_0, \Sigma, C \cup \{\chi_\ell\}, P_F, P_N \cup \{\pi_{\ell_{\text{MIN}}}, \pi_{\ell_{\text{MAX}}}\}, I', E, m', \eta)$ be the M/MD instrumentation of \mathcal{C}' for

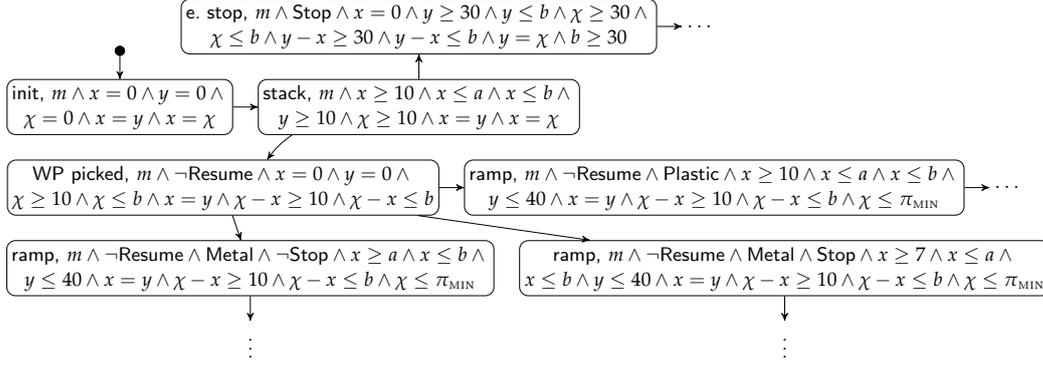


Figure 4.4: Example for a Featured Parametric Zone Graph of a CoPTA with M/MD Instrumentation

location $\ell \in L$. Consider TA $\mathcal{A} \in \llbracket \mathcal{C} \rrbracket_{\mathcal{C}}$ corresponding to configuration $c \in \llbracket m \rrbracket$. In order to find a bisimilar $\mathcal{A}' \in \llbracket \mathcal{C}' \rrbracket_{\mathcal{C}}$ such that $\mathcal{A} \simeq \mathcal{A}'$, we use the same configuration c for parameters in $P_F \cup P_N$ for \mathcal{C}' as done for \mathcal{C} . Furthermore, we choose $\pi_{\ell_{\text{MAX}}} = 0$ as this parameters is used as a lower bound, and we choose an arbitrarily great (but finite) value for $\pi_{\ell_{\text{MIN}}}$ as this parameter is used as an upper bound. Therewith, we obtain a variant $\mathcal{A}' \in \llbracket \mathcal{C}' \rrbracket_{\mathcal{C}}$ such that $\mathcal{A} \simeq \mathcal{A}'$. Hence, it holds that

$$\forall \mathcal{A} \in \llbracket \mathcal{C} \rrbracket_{\mathcal{C}} : (\exists \mathcal{A}' \in \llbracket \mathcal{C}' \rrbracket_{\mathcal{C}} : \mathcal{A} \simeq \mathcal{A}')$$

such that Lemma 4.2 is correct. \square

For the next step, we utilize the M/MD instrumentation to derive symbolic value schemas as a basis for t -wise Ψ -sampling.

4.3.3 Minimum/Maximum Delay Sampling

Having introduced our M/MD instrumentation in the previous section, we utilize featured parametric zone graphs enriched by the instrumentation to derive symbolic value schemas for *minimum/maximum delay sampling (M/MD sampling)*, achieving M/MD coverage for CoPTA models. In particular, we utilize featured parametric zones to derive constraints over Boolean and unbounded numeric parameters describing configurations with minimum and maximum delays for reaching our test goals (i.e., locations). Next, we present an algorithm deriving a set Ψ of symbolic value schemas from a CoPTA model. The set of symbolic value schemas Ψ can then be used as input for Algorithm 4.1 to derive an M/MD covering array.

We utilize Algorithm 4.2 to derive a set of symbolic value schemas Ψ from a CoPTA $\mathcal{C} = (L, \ell_0, \Sigma, C, P_F, P_N, I, E, m, \eta)$. Here, we start by initializing the set of symbolic value schemas with $\Psi := \emptyset$ (see line 2). Thereafter, we generate symbolic value schemas having minimum and maximum delay for each location $\ell \in L$ (lines 3–10), where we start by instrumenting CoPTA \mathcal{C} for the current location ℓ (line 4).

Then, we query a model checker for symbolic value schemas ψ_{MIN}^{ℓ} and ψ_{MAX}^{ℓ} having BCET and WCET, respectively, for reaching ℓ (lines 5–6). For this, the

Algorithm 4.2: Generating Symbolic Value Schemas

Input : CoPTA $\mathcal{C} = (L, \ell_0, \Sigma, C, P_F, P_N, I, E, m, \eta)$
Output: set of symbolic value schemas Ψ

```

1 procedure MAIN
2    $\Psi := \emptyset$ 
3   foreach  $\ell \in L$  do
4      $C' := \text{GETINSTRUMENTATION}(\mathcal{C}, \ell)$ 
5      $\psi_{\text{MIN}}^\ell := \text{GETMINSHEMA}(C', \ell)$ 
6      $\psi_{\text{MAX}}^\ell := \text{GETMAXSCHEMA}(C', \ell)$ 
7     if  $\psi_{\text{MAX}}^\ell \neq \text{false}$  then
8        $\Psi := \Psi \cup \{\psi_{\text{MIN}}^\ell, \psi_{\text{MAX}}^\ell\}$ 
9     else
10       $\Psi := \Psi \cup \{\psi_{\text{MIN}}^\ell\}$ 
11  return  $\Psi$ 

```

model checker searches for configurations of EFM m such that the values of $\pi_{\ell_{\text{MIN}}}$ and $\pi_{\ell_{\text{MAX}}}$ are minimized or maximized, respectively. Here, the model checker starts by finding all symbolic states comprising the target location ℓ in the featured parametric zone graph. Note, that the call of the model checker might not terminate in this case as checking reachability is semi-decidable for CoPTA (see Section 3.3 of the previous chapter). Then, the model checker considers the featured parametric zones of these states and minimizes (or maximizes) the fresh parameter π of the M/MD instrumentation. As a result, the value of π is the minimum (or maximum) delay for reaching ℓ . Thereafter, the model checker replaces π by the minimum (or maximum) value, such that we can utilize the featured parametric zones to derive a symbolic value schema comprising a restriction for each of the original parameters. This is done by eliminating variables in terms of clocks in featured parametric zones such that we obtain configuration constraints (see Definition 3.2).¹ Hence, we derive configuration constraints describing variants with BCET/WCET for reaching our test goals without explicitly finding the actual runs having BCET/WCET. Instead, we minimize (or maximize) parameter π to derive these configuration constraints.

Having derived symbolic value schemas ψ_{MIN}^ℓ and $\psi_{\text{MIN}}^{\ell'}$, we check whether there actually is a finite WCET for reaching location ℓ (this might be the case if there is a loop that can be used infinitely often before reaching ℓ). In particular, we check if $\psi_{\text{MAX}}^\ell \neq \text{false}$ (in which case there is a finite WCET) and add ψ_{MIN}^ℓ and $\psi_{\text{MIN}}^{\ell'}$ to the set of symbolic value schemas in this case (lines 7–8). In case of $\psi_{\text{MAX}}^\ell = \text{false}$ (i.e., there is no finite WCET for reaching ℓ), we only add ψ_{MIN}^ℓ to Ψ (lines 9–10). Finally, we return the set Ψ of symbolic value schemas (line 11).

Example 4.13 (Symbolic Value Schema for M/MD Coverage). Consider the CoPTA in Figure 4.3b containing the M/MD instrumentation for the minimum delay for reaching location *ramp* (see lines 1–4 of Algorithm 4.2). Here, the corresponding featured parametric zone graph is depicted in Figure 4.4 (see

¹ For instance, this can be achieved by applying *Fourier-Motzkin elimination* [174].

Example 4.12 for a detailed description). First, we find the minimum value for π_{MIN} such that *ramp* is still reachable (line 5). Therewith, we have a basis for obtaining the symbolic value schema for variants having the BCET for reaching the *ramp*. In order to find the minimum value for π_{MIN} , we consider all featured parametric zones for symbolic states comprising location *ramp* (i.e., there are three such states in our example). For instance,

$$\begin{aligned} m \wedge \neg\text{Resume} \wedge \text{Metal} \wedge \text{Stop} \wedge x \geq 7 \wedge x \leq a \wedge x \leq b \wedge \\ y \leq 40 \wedge x = y \wedge \chi - x \geq 10 \wedge \chi - x \leq b \wedge \chi \leq \pi_{\text{MIN}} \end{aligned}$$

is the constraint of the bottom right symbolic state right in Figure 4.4. Moreover, we have to consider the featured parametric zones of the two other states of Figure 4.4 comprising location *ramp*. Then, we combine these constraints by disjunction and find the minimum value for π_{MIN} satisfying this resulting constraint (e.g., by using an ILP solver). In this example, we obtain $\pi_{\text{MIN}} = 17$ (satisfying the featured parametric zone of the bottom right symbolic state). Hence, the BCET for reaching the *ramp* is 17 seconds.

Second, we replace π_{MIN} by 17 and eliminate all clock variables of the featured parametric zone. As a result, we obtain the configuration constraint

$$\psi_{\text{MIN}}^{\text{ramp}} := m \wedge \neg\text{Resume} \wedge \text{Metal} \wedge \text{Stop}$$

describing the symbolic value schema for all configurations containing the run with BCET for reaching location *ramp*. Then, we add $\psi_{\text{MIN}}^{\text{ramp}}$ to Ψ (line 8), generate a symbolic schema with WCET for reaching *ramp*, and start another iteration of the loop with the next location (line 3).

In order to achieve M/MD coverage with t -wise Ψ -sampling, we generate symbolic value schemas for each location of a CoPTA as described above. We then use the symbolic value schemas as input for Algorithm 4.1, generating an M/MD covering array. Recall, that boundary-value testing also explicitly considers values that are not boundary cases. Here, the idea is that boundary behavior as well as “normal” behavior should be tested. Therefore, Algorithm 4.1 also adds the negation of all value schemas (if satisfiable) to the set of schemas to be covered. For symbolic value schemas derived from CoPTA, we utilize the naming convention ψ_x^ℓ for $\ell \in L$ and $x \in \{\text{MIN}, \text{MAX}\}$.

Example 4.14 (M/MD Covering Array of the PPU). Considering CoPTA \mathcal{C} of the PPU depicted in Figure 4.3b with EFM m in Figure 4.3a (see page 84), we obtain the following set Ψ of symbolic value schemas when applying M/MD sampling:

- $\psi_{\text{MIN}}^{\text{init}}, \psi_{\text{MAX}}^{\text{init}}, \psi_{\text{MIN}}^{\text{stack}}: m$
- $\psi_{\text{MAX}}^{\text{stack}}: m \wedge \text{Plastic} \wedge a \geq 30 \wedge b \geq 30$
- $\psi_{\text{MIN}}^{\text{WP picked}}: m \wedge \neg\text{Resume}$

- $\psi_{\text{MAX}}^{\text{WP picked}}: m \wedge \neg \text{Resume} \wedge \text{Metal} \wedge b \geq 40$
- $\psi_{\text{MIN}}^{\text{ramp}}, \psi_{\text{MIN}}^{\text{WP placed}}, \psi_{\text{MIN}}^{\text{idle}}: m \wedge \neg \text{Resume} \wedge \text{Metal} \wedge \text{Stop}$
- $\psi_{\text{MAX}}^{\text{ramp}}, \psi_{\text{MAX}}^{\text{WP placed}}: m \wedge \neg \text{Resume} \wedge \text{Metal} \wedge \neg \text{Stop} \wedge b \geq 40$
- $\psi_{\text{MIN}}^{\text{e. stop}}: m \wedge \text{Stop} \wedge b \geq 30$

Note, that this list does not contain the symbolic value schemas $\psi_{\text{MAX}}^{\text{e. stop}}$ and $\psi_{\text{MAX}}^{\text{idle}}$ as the WCET for reaching locations *emergency stop* and *idle* is arbitrarily long. In these cases, there are variants of the PPU where we may use the outer loop arbitrarily often before deciding to reach these locations. Furthermore, some symbolic value schemas only consist of the requirement m as the run with BCET (or WCET) is part of every valid variant of \mathcal{C} . Additionally, this example shows that we cannot simply minimize (or maximize) numeric parameters to achieve M/MD coverage. For instance, $\psi_{\text{MIN}}^{\text{e. stop}}$ requires $b \geq 30$ although EFM m allows the value $b = 20$.

After deriving M/MD-covering symbolic value schemas for all locations of CoPTA \mathcal{C} , we proceed by applying Algorithm 4.1 to obtain a 1-wise Ψ -covering array satisfying M/MD coverage. For instance, this may produce the covering array depicted in Table 4.1. Here, each row (generated from top to bottom) corresponds to a test case, where symbolic value schemas not yet covered by previously generated test cases are written in bold font (i.e., a schema may be covered by multiple test cases). Moreover, symbolic value schemas do not occur in negated form if the negation yields an unsatisfiable constraint w.r.t. EFM m (see Definition 4.4). In every step, the (greedy-based) algorithm conjugates as many compatible symbolic value schemas as possible from the set of uncovered schemas. If no further schema can be added, an arbitrary parameter-value assignment satisfying the resulting constraint is computed (by using an SMT solver) and added as a test case θ to covering array Θ . We repeat this procedure until all symbolic value schemas are covered. As a result, we obtain a covering array $\Theta = \{\theta_1, \theta_2, \theta_3, \theta_4\}$ with four test cases, constituting a 1-wise Ψ -sample for this example. However, Algorithm 4.1 does not necessarily generate a minimal solution as it is a heuristic algorithm.

We conclude this section by proving that using Algorithms 4.1 and 4.2 results in a covering array Θ satisfying M/MD coverage for a CoPTA \mathcal{C} .

Theorem 4.2. Let $\mathcal{C} = (L, \ell_0, \Sigma, C, P_F, P_N, I, E, m, \eta)$ be a CoPTA and Ψ be a set symbolic value schemas obtained by applying Algorithm 4.2 to \mathcal{C} . Then, we get a covering array Θ satisfying M/MD coverage for \mathcal{C} w.r.t. t -wise Ψ -sampling by using Ψ as input for Algorithm 4.1.

Proof. Let $\mathcal{C} = (L, \ell_0, \Sigma, C, P_F, P_N, I, E, m, \eta)$ be a CoPTA. By applying Algorithm 4.2 to \mathcal{C} , we obtain a set Ψ consisting of symbolic value schemas having BCET for reaching all locations (due to line 5) and WCET for reaching all locations being reachable with a finite execution time (due to line 6). Next, we utilize Ψ as input for Algorithm 4.1. Here, we first compute all possible (i.e., satisfiable) t -wise combinations of symbolic value schemas in Ψ to ensure t -wise coverage (line 3),

Table 4.1: Sample Satisfying M/MD Coverage for the PPU (see Figure 4.3b)

	PPU	Workpiece	Plastic	Metal	Stop	Resume	a	b	c	Covered Schemas	Remaining Schemas
θ_1	true	true	true	false	true	false	36	30	33	$\psi_{\min}^{\text{init}}, \psi_{\max}^{\text{init}}, \psi_{\min}^{\text{stack}}, \psi_{\max}^{\text{stack}}$ $\psi_{\min}^{\text{e.stop}}, \psi_{\max}^{\text{e.stop}}, \psi_{\min}^{\text{ramp}}, \psi_{\max}^{\text{ramp}}$ $\psi_{\min}^{\text{idle}}, \psi_{\max}^{\text{idle}}, \psi_{\min}^{\text{stack}}, \psi_{\max}^{\text{stack}}$ $\psi_{\min}^{\text{stack}}, \psi_{\max}^{\text{stack}}, \psi_{\min}^{\text{ramp}}, \psi_{\max}^{\text{ramp}}$ $\psi_{\min}^{\text{ramp}}, \psi_{\max}^{\text{ramp}}, \psi_{\min}^{\text{idle}}, \psi_{\max}^{\text{idle}}$ $\psi_{\min}^{\text{idle}}, \psi_{\max}^{\text{idle}}, \psi_{\min}^{\text{stack}}, \psi_{\max}^{\text{stack}}$	$\psi_{\max}^{\text{ramp}}, \psi_{\min}^{\text{ramp}}, \psi_{\max}^{\text{stack}}, \psi_{\min}^{\text{stack}}$ $\psi_{\max}^{\text{idle}}, \psi_{\min}^{\text{idle}}, \psi_{\max}^{\text{stack}}, \psi_{\min}^{\text{stack}}$ $\psi_{\max}^{\text{stack}}, \psi_{\min}^{\text{stack}}, \psi_{\max}^{\text{ramp}}, \psi_{\min}^{\text{ramp}}$ $\psi_{\max}^{\text{idle}}, \psi_{\min}^{\text{idle}}, \psi_{\max}^{\text{stack}}, \psi_{\min}^{\text{stack}}$
θ_2	true	true	false	true	true	false	23	40	53	$\psi_{\min}^{\text{init}}, \psi_{\max}^{\text{init}}, \psi_{\min}^{\text{stack}}, \psi_{\max}^{\text{stack}}$ $\psi_{\min}^{\text{e.stop}}, \psi_{\max}^{\text{e.stop}}, \psi_{\min}^{\text{ramp}}, \psi_{\max}^{\text{ramp}}$ $\psi_{\min}^{\text{idle}}, \psi_{\max}^{\text{idle}}, \psi_{\min}^{\text{stack}}, \psi_{\max}^{\text{stack}}$ $\psi_{\min}^{\text{stack}}, \psi_{\max}^{\text{stack}}, \psi_{\min}^{\text{ramp}}, \psi_{\max}^{\text{ramp}}$ $\psi_{\min}^{\text{ramp}}, \psi_{\max}^{\text{ramp}}, \psi_{\min}^{\text{idle}}, \psi_{\max}^{\text{idle}}$ $\psi_{\min}^{\text{idle}}, \psi_{\max}^{\text{idle}}, \psi_{\min}^{\text{stack}}, \psi_{\max}^{\text{stack}}$	$\psi_{\max}^{\text{ramp}}, \psi_{\min}^{\text{ramp}}, \psi_{\max}^{\text{stack}}, \psi_{\min}^{\text{stack}}$ $\psi_{\max}^{\text{idle}}, \psi_{\min}^{\text{idle}}, \psi_{\max}^{\text{stack}}, \psi_{\min}^{\text{stack}}$ $\psi_{\max}^{\text{stack}}, \psi_{\min}^{\text{stack}}, \psi_{\max}^{\text{ramp}}, \psi_{\min}^{\text{ramp}}$ $\psi_{\max}^{\text{idle}}, \psi_{\min}^{\text{idle}}, \psi_{\max}^{\text{stack}}, \psi_{\min}^{\text{stack}}$
θ_3	true	true	false	true	false	false	21	40	42	$\psi_{\min}^{\text{init}}, \psi_{\max}^{\text{init}}, \psi_{\min}^{\text{stack}}, \psi_{\max}^{\text{stack}}$ $\psi_{\min}^{\text{e.stop}}, \psi_{\max}^{\text{e.stop}}, \psi_{\min}^{\text{ramp}}, \psi_{\max}^{\text{ramp}}$ $\psi_{\min}^{\text{idle}}, \psi_{\max}^{\text{idle}}, \psi_{\min}^{\text{stack}}, \psi_{\max}^{\text{stack}}$ $\psi_{\min}^{\text{stack}}, \psi_{\max}^{\text{stack}}, \psi_{\min}^{\text{ramp}}, \psi_{\max}^{\text{ramp}}$ $\psi_{\min}^{\text{ramp}}, \psi_{\max}^{\text{ramp}}, \psi_{\min}^{\text{idle}}, \psi_{\max}^{\text{idle}}$ $\psi_{\min}^{\text{idle}}, \psi_{\max}^{\text{idle}}, \psi_{\min}^{\text{stack}}, \psi_{\max}^{\text{stack}}$	$\psi_{\max}^{\text{ramp}}, \psi_{\min}^{\text{ramp}}, \psi_{\max}^{\text{stack}}, \psi_{\min}^{\text{stack}}$ $\psi_{\max}^{\text{idle}}, \psi_{\min}^{\text{idle}}, \psi_{\max}^{\text{stack}}, \psi_{\min}^{\text{stack}}$ $\psi_{\max}^{\text{stack}}, \psi_{\min}^{\text{stack}}, \psi_{\max}^{\text{ramp}}, \psi_{\min}^{\text{ramp}}$ $\psi_{\max}^{\text{idle}}, \psi_{\min}^{\text{idle}}, \psi_{\max}^{\text{stack}}, \psi_{\min}^{\text{stack}}$
θ_4	true	true	true	false	true	true	42	30	38	$\psi_{\min}^{\text{init}}, \psi_{\max}^{\text{init}}, \psi_{\min}^{\text{stack}}, \psi_{\max}^{\text{stack}}$ $\psi_{\min}^{\text{e.stop}}, \psi_{\max}^{\text{e.stop}}, \psi_{\min}^{\text{ramp}}, \psi_{\max}^{\text{ramp}}$ $\psi_{\min}^{\text{idle}}, \psi_{\max}^{\text{idle}}, \psi_{\min}^{\text{stack}}, \psi_{\max}^{\text{stack}}$ $\psi_{\min}^{\text{stack}}, \psi_{\max}^{\text{stack}}, \psi_{\min}^{\text{ramp}}, \psi_{\max}^{\text{ramp}}$ $\psi_{\min}^{\text{ramp}}, \psi_{\max}^{\text{ramp}}, \psi_{\min}^{\text{idle}}, \psi_{\max}^{\text{idle}}$ $\psi_{\min}^{\text{idle}}, \psi_{\max}^{\text{idle}}, \psi_{\min}^{\text{stack}}, \psi_{\max}^{\text{stack}}$	—

and we add each combination to the working set WS . Then, we conjugate symbolic value schemas from WS if the result is satisfiable (lines 9–13), and we derive concrete test cases θ from these conjugated value schemas (lines 5–7) which we add to our covering array Θ . Here, we find a test case for each symbolic value schema in WS (and, by proxy, for each symbolic value schema in the input set Ψ) as we only remove elements from WS if we generated a covering test case θ (see lines 5, 6, 10, and 11). As we generate a test case for each symbolic value schema in WS (line 10) and set Ψ used as a basis for WS has symbolic value schemas with BCET/WCET for each location $\ell \in L$, the resulting covering array Θ satisfies M/MD coverage. \square

To summarize this section, we started by formally defining BCET/WCET for reaching locations of a CoPTA \mathcal{C} . Thereafter, we introduced the novel notion of M/MD coverage. In particular, a covering array Θ satisfies M/MD if for each $\ell \in L$ of \mathcal{C} , there exists a $\theta \in \Theta$ such that the TA corresponding to θ has the overall BCET (or WCET) for reaching ℓ . To achieve M/MD coverage, we introduce an M/MD instrumentation for CoPTA, adding an additional invariant to target locations. This invariant is utilized for reducing M/MD coverage to a reachability check and an optimization problem. Finally, we used featured parametric zone graphs (together with the M/MD instrumentation) to derive symbolic value schemas, serving as input for Algorithm 4.1 (see Section 4.1). Having introduced the concepts needed for sampling product lines with configurable parametric real-time constraints, we proceed by providing implementation details and an evaluation of our approach in the following section.

4.4 EXPERIMENTAL EVALUATION

In this section, we present evaluation results of the approach presented in this chapter, and we give an overview on the corresponding implementation. Here, we investigate the computational effort required for t -wise Ψ -sampling based on M/MD coverage of CoPTA models. Additionally, we are interested in the sample sizes for different values of t . Moreover, we are not able to compare our approach to others due to the lack of similar approaches. It should be noted that effectiveness of M/MD sampling could be evaluated by measuring the number of (real-world and artificial) faults that are contained in a sample. However, we leave this as an open issue for future work. We consider the following research questions.

RESEARCH QUESTIONS We consider two research questions regarding applicability and scalability of M/MD sampling w.r.t. to the size (in terms of the number of locations) of CoPTA models. Here, we are especially interested in the computational effort in terms of CPU time required for generating a covering array satisfying M/MD coverage. Furthermore, we consider the sample size for t -wise Ψ -sampling for different values of t . The sample size is interesting as, for instance, there could be a setting where each additional test configuration requires its own hardware setup. In cases like this, the computational effort for sampling would save the cost for additional hardware and the time needed for the setup of the hardware.

Table 4.2: Subject Systems for the Experimental Evaluation

Subject System	# Boolean Parameters	# Numeric Parameters	# Locations	# Switches
PPU (Figures 3.2, 3.3)	6	3	7	10
TGC	3	0	12	18
PPU ₀	10	5	7	13
PPU ₃	10	5	12	23
PPU ₅	10	5	13	28
PPU ₆	11	5	20	43
PPU ₈	10	5	13	28
PPU ₉	10	5	13	28
BusyBox_dpkg	5	32	77	90
Vim_insecure_flag	7	16	37	55
Vim_varp	8	12	29	41
Vim_gui_base_height	7	14	25	35
synth_calculate	9	17	48	72
synth_method_triggered	9	14	51	69
synth_rand_int	8	16	39	48

- **RQ1 (Computational Effort):** How does the size (i.e., the number of locations) of a CoPTA model impact the computational effort for t -wise Ψ -sample generation regarding M/MD coverage?
- **RQ2 (Sample Size):** How does the size (i.e., the number of locations) of a CoPTA model impact the sample size required for t -wise Ψ -coverage regarding M/MD coverage?

TOOL SUPPORT We implemented our sampling approach for M/MD coverage as described in this chapter based on IMITATOR [18, 19, 23]. As IMITATOR is a model-checker for PTA, we utilize the PEPTA transformation described in Chapter 3 to apply M/MD sampling to CoPTA. For checking satisfiability of parametric clock constraints, IMITATOR relies on an internal constraint solver based on the Parma Polyhedra Library [25]. For computing optimal parameter values in terms of minimum/maximum delays for M/MD coverage, we utilize the built-in ILP solver of IMITATOR (which itself again relies on the Parma Polyhedra Library). For computing satisfiability outside of IMITATOR (e.g., for the greedy part of Algorithm 4.1), we apply the Z3 SMT solver [80]. Note, that IMITATOR supports PTA with parameter domain $\mathbb{T}_P = \mathbb{Q}_+$. Hence, the algorithms utilized by IMITATOR (and thus also our sampling procedure) are inherently incomplete as most semantic properties are undecidable for PTA using parameter domain \mathbb{Q}_+ . As a result, those cases may lead to timeouts (caused by non-termination) of analysis runs. However, we have not encountered non-termination during our experimental evaluation.

SUBJECT SYSTEMS For the experiments, we use 15 different CoPTA models, where we created most of these models for the sake of this evaluation. As the CoPTA formalism is a novel formalism introduced by us, these models are the only CoPTA models currently available and have, in our opinion, reasonable sizes and complexity. Table 4.2 gives an overview on these models.

- The *Pick and Place Unit (PPU)* (first line in Table 4.2) is a simplified version of the PPU which we use as a running example throughout this thesis (see Figures 3.2 and 3.3).
- The *Train-Gate-Controller (TGC)* [15] models a level crossing comprising a railroad-gate controller.
- The models PPU_n with $n \in \{0, 3, 5, 6, 8, 9\}$ are CoPTA models of the actual PPU [122, 196]. Here, index n is the number of the corresponding scenario. It should be noted that we do not evaluate every scenario as some scenarios do not include changes in the corresponding behavioral model (e.g., when introducing an additional color for workpieces). For instance, scenarios 0, 1, and 2 have the exact same behavioral model.

Furthermore, we utilize additional synthetic case studies for evaluating our approach for large-scale models. To obtain these case studies, we generated CoPTA models from product-line source code written in C. However, the CoPTA models that we created this way have the sole purpose of being synthetic large-scale models. The analysis results obtained from these models cannot be directly be generalized for the original source code.

For creating CoPTA models from C code, we first derived the *control-flow automaton (CFA)* [68] corresponding to a method of the source code. Here, each state of the CFA is used as a CoPTA location and each edge of the CFA is used as a CoPTA switch. Furthermore, we annotated the switches with the corresponding line of source code as an action, and we used feature constraints of the source code as feature constraints of switches. Second, we added clock constraints to the resulting CoPTA models. To this end, we executed the source code several times and measured execution times. Then, the execution are used as lower bounds and upper bounds, respectively. Moreover, we utilized parameters in clock constraints whenever there was a call to an external method in the source code. For this approach, we used the following programs as basis. Again, Table 4.2 gives an overview on these models, where the part before the first underscore is the program and the remainder of the name is the method from which we extracted the CoPTA model (e.g., *Vim* has a method called *insecure_flag*).

- *BusyBox*² comprises a set of well-known UNIX command-line tools.
- *Vim*³ is a more recent implementation of the UNIX text editor *Vi*.
- We use *synthetically* generated code of product lines written in C created by Ruland et al. [168]. In particular, this code was generated by mutating existing C code and then merging the resulting set of mutants into an SPL.

EXPERIMENT DESIGN AND MEASUREMENT SETUP We applied our tool implementation to the case studies described above to generate t -wise Ψ -covering arrays for $t \in \{1, 2\}$. We do not include values where $t > 2$ as 2-wise sampling is

² <https://busybox.net/>

³ <https://github.com/vim/vim>

Table 4.3: Overview on the Evaluation Results

Subject System	CPU Time (s) for $t = 1$	CPU Time (s) per location for $t = 1$	CPU Time (s) for $t = 2$	CPU Time (s) per location for $t = 2$	# Test Cases for $t = 1$	# Test Cases for $t = 2$
PPU (Figures 3.2, 3.3)	208.7	29.8	222.2	31.7	3	6
TGC	52.8	4.4	56.3	4.7	2	2
PPU ₀	296.7	42.4	298.8	42.7	2	3
PPU ₃	546.1	45.5	548.3	45.7	3	3
PPU ₅	630.3	48.5	636.3	49.0	4	8
PPU ₆	1495.0	74.8	1502.9	75.1	6	8
PPU ₈	638.1	49.1	638.4	49.1	4	4
PPU ₉	637.1	49.0	637.2	49.0	4	4
BusyBox_dpkg	2075.5	27.0	—	—	4	—
Vim_insecure_flag	3143.8	85.0	—	—	4	—
Vim_varp	2404.5	82.9	—	—	4	—
Vim_gui_base_height	213.2	8.5	219.8	8.8	5	9
synth_calculate	5883.7	122.6	—	—	9	—
synth_method_triggered	4520.2	106.3	—	—	8	—
synth_rand_int	4485.4	115.0	—	—	7	—

in many cases [168, 159, 160] considered to be the best tradeoff between efficiency and effectiveness. However, we do not evaluate effectiveness of M/MD sampling in this thesis (see above), such that we leave t -wise M/MD sampling with $t > 2$ as another open issue for future work. In order to answer RQ1, we measured the CPU time required for generating a sample. For RQ2, we measured the resulting size of the samples (i.e., the number of configurations). Furthermore, we applied IMITATOR version 2.9.3 and Z3 version 4.6.0. We performed all experiments on a machine with Ubuntu 18.04 x64 and 16 GB of RAM running on an Intel Core i7 (4×4.2 GHz) machine.

RESULTS The results of our experimental evaluation are summarized in Table 4.3. Here, the second and the fourth column give an overview on CPU time (in seconds) needed for 1-wise M/MD sampling and 2-wise M/MD sampling, respectively. Furthermore, we normalize the computational effort (see columns three and five) by dividing the CPU time by the number of locations of the respective case study. Note, that we utilize locations for normalizing the results as the number of locations is a good approximation for the size of the symbolic state space (at least for our case studies). Additionally, the last two columns show the number of resulting test cases. Moreover, we depict the results for **RQ1** in Figure 4.5 to visualize a possible correlation between the size of a CoPTA model (in terms of the number of locations) and the CPU time needed to generate the respective sample. Here, the x -axis describes the number of locations of our case studies, whereas the y -axis shows the CPU time (in seconds) for the corresponding subject system.

It should be noted that we only performed 1-wise sampling for most of the synthetically generated subject systems as 2-wise sampling exceeded the available amount of RAM. In particular, combining symbolic value schemas for 2-wise sampling requires a check for satisfiability of the conjugated schemas. Here, the Z3 SMT solver was not able to perform these checks as these conjugated schemas become quite large in case of our synthetically generated case studies.

In case of computational effort (i.e., **RQ1**), we observe that generating a sample for $t = 1$ requires between 52.8s (*TGC*) and 24.9 min (*PPU₆*) for real-world case

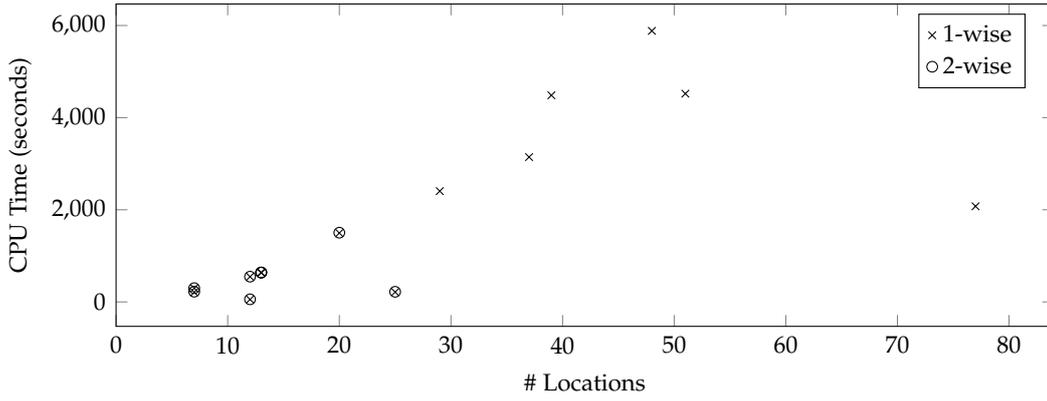


Figure 4.5: Measurement Results for **RQ1** (Computational Effort)

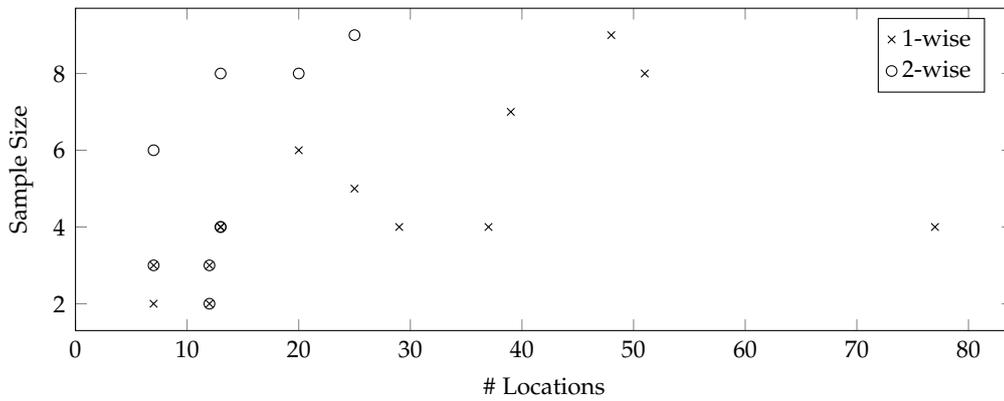


Figure 4.6: Measurement Results for **RQ2** (Sample Size)

studies and 213.2s (*Vim_gui_base_height*) to 1.6h (*synth_calculate*) for our synthetic case studies. Additionally, the CPU time required for $t = 2$ is almost the same as for $t = 1$, as the procedure for deriving symbolic value schemas from CoPTA is identical in both cases. Furthermore, we do not distinguish between the CPU time needed for deriving symbolic value schemas and the CPU time needed for computing the sample. Here, deriving symbolic value schemas takes most of the time whereas computing the sample usually only takes a few seconds (even in case of PPU₆ it still took less than ten seconds).

In case of sample sizes (i.e., **RQ2**), we observe 2 to 6 test cases for $t = 1$ in case of real-world case studies and 4 to 9 test cases for $t = 1$ in case of synthetic case studies. Additionally, using $t = 2$ results in sample sizes of 2 to 8 test cases for our real-world systems. Finally, the only synthetic case study for which we were able to perform 2-wise sampling (i.e., *Vim_gui_base_height*) requires 9 test cases. Similar to **RQ1**, we depict the results for **RQ2** in Figure 4.6, where the x-axis describes the number of locations of our case studies and the y-axis shows the corresponding sample size.

DISCUSSION Concerning **RQ1**, we see a computational effort of at most 1.6h in our experiments even in case of large synthetically generated models. Depending

on the setting for subsequent analyses, this may be an acceptable effort. As described above, there could be a setting where each additional test configuration requires its own hardware setup. In cases like this, the computational effort for sampling would save the cost for additional hardware and the time needed for the setup of the hardware. Furthermore, we see a correlation between the size of CoPTA models (in terms of the number of locations) and the CPU time required for computing the respective samples. Here, we only have two outliers, namely *Vim_gui_base_height* and *BusyBox_dpkg*. These outliers are a result of the internal structures of these models (e.g., there is less branching as compared to the other case studies).

Additionally, we observe that the required CPU time per location grows with the size (i.e., the number of locations) of the CoPTA models in most cases. This is expected as a larger state space means that a larger number of constraints need to be solved. However, this is not always the case. For instance, the CPU time per location for *Vim_gui_base_height* is quite small although this case study comprises 25 locations. Hence, the internal structure (e.g., number of branches and switches which can only be used after traversing other switches) also influences the computational effort. Moreover, we expect that a machine with additional RAM (which is rather cheap nowadays) is able to compute a 2-wise sample also for our larger synthetic models.

Concerning **RQ2**, the difference of sample sizes is relatively small in most cases between $t = 1$ and $t = 2$. This indicates that many runs with minimum/maximum delay for reaching different locations are common to many configurations. Furthermore, we do not observe a clear correlation between the sample size and the size of the CoPTA models (as depicted in Figure 4.6).

THREATS TO VALIDITY A threat to internal validity might be the coverage criterion that we consider in this thesis. However, sampling of real-time systems is still an emerging field of research, such that there are no recent criteria for product lines of TA mentioned in literature (except for covering basic components such as locations). Moreover, M/MD is a coverage criterion specifically designed for CoPTA. In contrast, t -wise Ψ -coverage is a rather general coverage criterion for product lines. Here, we expect similar results when combining t -wise Ψ -coverage with further criteria for selecting a set of symbolic value schemas from product-line models.

Concerning threats to external validity, we do not compare t -wise Ψ -sampling with other approaches. However, to the best of our knowledge, there are no similar techniques for product lines with configurable parametric real-time constraints. Moreover, we only utilize a small number of real-world subject systems. However, these models have, in our opinion, reasonable size and complexity. In particular, *TGC* and the different scenarios of the *PPU* represent real-world models having similar size and complexity as models which are frequently used for evaluating TA-based analysis techniques (e.g., see [32, 98, 30, 119, 29, 45, 182]). However, we plan to consider a greater number of real-world case studies for future work to obtain more meaningful results.

4.5 RELATED WORK

In this section, we give an overview on related work concerning sampling of product lines with configurable parametric real-time constraints. In particular, we start by providing an overview on the optimization of real-time behavior. Thereafter, we focus on sampling of finite and infinite configuration spaces. Finally, we present a list of tools that are interesting in the context of this chapter.

OPTIMIZATION OF REAL-TIME BEHAVIOR The first solution to the M/MD problem of TA was introduced by Courcoubetis and Yannakakis [74]. Here, the authors utilize region graphs to find fastest and slowest runs. Region graphs describe TA semantics in a finite model but are (on average) much larger than zone graphs. Furthermore, Niebert et al. [151] introduce additional algorithms for the minimum/maximum delay problem for TA. Similar to the approach presented in this thesis, Panek et al. [156] apply linear optimization to solve this problem. As compared to these approaches, Abdeddaïm et al. [2] utilize a game-theoretic approach for tackling the minimum/maximum delay problem. Moreover, Alur et al. [16] and Behrmann et al. [31, 33] generalize the underlying optimization problem to weighted TA (i.e., Priced Timed Automata [31]). However, all of these approaches are only defined for single variants (i.e., TA) instead of product lines. Furthermore, the goal of these approaches only is finding fastest and slowest runs but the results are not used for further steps (e.g., sampling).

Another closely related work in terms of optimization of real-time behavior is presented by André et al. [24] (which was developed simultaneously to and independently of our work in [134]). Here, the authors introduced an algorithm to find runs with minimum/maximum delays for PTA. However, the output is again not utilized for further steps such as sampling. Finally, Fahrenberg and Legay [83] describe a general approach for optimization of non-functional quantitative properties for weighted FTS (i.e., untimed product lines). Specifically, the authors show reachability for energy properties (i.e., minimum energy consumption). However, the authors do not optimize real-time behavior.

SAMPLING FINITE CONFIGURATION SPACES In this paragraph, we give an overview on related work for sampling finite configuration spaces. However, we only discuss some interesting approaches for this area due to the large number of available approaches. For a comprehensive overview, we refer the reader to the surveys of Varshosaz et al. [194], Ahmed et al. [5], and Lopez-Herrejon et al. [129].

For sampling finite configuration spaces, Cohen et al. [66] were among the first to apply ideas of CIT to sample-based testing of product lines. Furthermore, Cmyrev and Reissing [65], Perrouin et al. [158], Haslinger et al. [96], and Johansen et al. [106, 107] (among others) apply sampling based on feature models to generate covering arrays (mostly) for pairwise CIT coverage in black-box settings. Moreover, Al-Hajjaji et al. [8] combine several approaches for generating covering to obtain an adjustable sampling procedure. In addition to these approaches, sampling strategies are often concerned with prioritizing subsets of configurations to speed up the process of sampling and improve the quality of test cases. For instance, Oster

et al. [154] use domain knowledge in terms of interaction-coverage information on a behavioral specification to prioritize a subset of products before applying sampling based on feature models. Another approach is introduced by Baller et al. [27] who employ test-coverage criteria to prioritize selection and ordering of configurations under test. Additionally, Reuling et al. [166] apply domain knowledge to generate mutants of feature models by considering common faults. Then, sampling strategies can be evaluated by measuring the rate of detected mutants. Ruland et al. [168] apply a similar approach where mutants are created for product lines of C code.

Concerning non-functional properties, Kolesnikov et al. [114] sample configurations falling into particular non-functional categories (e.g., high memory consumption) using predictors to guide the sampling process. Moreover, Siegmund et al. [177] utilize learning techniques to predict non-functional feature interactions from performance values measured for a given sample (e.g., combining two features may result in especially low or high energy consumption). Finally, Sarkar et al. [170] apply performance measurements to samples to predict performance of other configurations, and Oh et al. [153] try to sample optimally performing configurations for a given workload.

However, to the best of our knowledge, there does not exist a sampling strategy for infinite configuration spaces as presented in this thesis (besides random-based methods such as star discrepancy [152]). Furthermore, we are not aware of any sampling strategies involving coverage criteria based on real-time constraints. Though, strategies from boundary-value testing [190] may be applied as a black-box approach for sampling infinite configuration spaces. Here, the idea is to pick *boundary* values (i.e., minimum and maximum values) for parameters to cover the behavior in case of boundary values. It should be noted that boundary-value testing strategies cannot be applied for achieving M/MD coverage as parameters cannot simply be minimized or maximized to obtain fastest and slowest runs (see Section 4.1). Additionally, the approach for adversarial machine learning presented by Temple et al. [184] could be generalized for infinite configuration spaces.

TOOLS In terms of tools, the UPPAAL tool suite [39, 34] can be utilized to find runs in TA having minimum (or maximum) delays for reaching particular locations. Furthermore, IMITATOR [18, 19, 23] provides similar functionality for PTA. However, none of these tools uses the resulting runs for further steps (e.g., sampling). Additionally, Cordy et al. [71, 72, 73] develop the ProVeLines tool suite. Therewith, it is possible to model and efficiently analyze time-critical product lines in terms of FTA [70]. However, this tool does not include a means to find fastest or slowest runs. Moreover, to the best of our knowledge, there is no tool available for sampling product lines with real-time constraints comprising a potentially infinite number of variants.

4.6 CONCLUSION AND FUTURE WORK

In this chapter, we tackled Research Challenge 2. Up to this point, there did not exist approaches for sampling infinite configuration spaces besides boundary-value

testing, which cannot be applied to find variants with fastest and slowest runs (see Section 4.1). Furthermore, there did not exist sampling strategies involving coverage criteria based on real-time constraints. Established approaches mostly utilize CIT and apply their techniques to feature models with finite configuration spaces in case of product lines. In this chapter, we improved the state of the art in two ways.

First, we introduced a novel coverage criterion called M/MD coverage. Here, a covering array is required to include for every test goal (i.e., CoPTA location) a variant comprising a run with BCET (and WCET) for reaching this test goal. Second, we introduced a technique for achieving M/MD coverage for a given CoPTA. For this, we first defined featured parametric zone graphs to describe the symbolic semantics of a CoPTA, extending zone graphs of TA by constraints over Boolean parameters as well as unbounded numeric parameters. Thereafter, we presented an M/MD instrumentation such that we are able to derive symbolic value schemas from CoPTA models by utilizing featured parametric zone graphs. In particular, we utilize our M/MD instrumentation to reduce the problem of finding a covering array to an optimization problem and reachability checks on the featured parametric zone graph.

Moreover, our experimental evaluation obtained by applying our prototypical tool implementation to our case studies reveals practical applicability of M/MD sampling. Here, our tool can also be applied to larger models with a reasonable computational effort. However, we were not able to compare our approach to other approaches as, to the best of our knowledge there do not exist similar approaches.

For future work, we may apply our M/MD instrumentation for further coverage criteria. As a result, we would obtain different sets of variants, potentially improving the results of subsequent analyses (e.g., by revealing further faults). For instance, we may require a covering array to contain a test case for reaching some locations *via other intermediate locations*. Furthermore, we are interested in finding configurations having fastest and slowest runs *within a given time frame*. In order to achieve this goal, we may utilize an additional instrumentation where the time frame is added as invariant to the target location. Then, we can apply M/MD coverage as introduced in this chapter. Additionally, we could apply M/MD coverage to switches instead of locations, such that every switch is reached with BCET/WCET. Moreover, the idea of our M/MD instrumentation can be used for other non-functional properties as long as there is a variable that can be optimized (i.e., minimized or maximized).

Furthermore, we plan to extend our evaluation by incorporating a greater number of case studies and considering larger real-world models. Here, we would also like to evaluate effectiveness of M/MD sampling by measuring the number of (real-world and artificial) faults that are contained in a sample. Additionally, we plan to consider further measure for the size of case studies (e.g., number of states with more than one outgoing switch) as our measurement results did not correlate with the number of locations in all cases. Finally, we would like to compare our approach for computing runs with minimum (or maximum) delay in an evaluation with the approach presented by André et al. [24] for PTA.

5

TEST-CASE GENERATION FOR PRODUCT LINES WITH CONFIGURABLE PARAMETRIC REAL-TIME CONSTRAINTS

In this chapter, we tackle Research Challenges 3.1 and 3.2 as introduced in the background chapter (see Section 2.5).

Research Challenge 3.1

Family-based test-suite generation for basic coverage criteria of real-time systems with a potentially infinite number of variants.

Research Challenge 3.2

Family-based test-suite generation for enhanced coverage criteria w.r.t. best-case/worst-case execution times of real-time systems with a potentially infinite number of variants.

In particular, we are concerned with generating a *complete* test suite w.r.t. a given coverage criterion. When considering product lines, the goal is to cover all test goals *in every variant* in which the test goals are reachable. A straightforward approach to achieve this goal is deriving all variants and then generating test cases for all test goals in a variant-by-variant fashion. However, this can become quite expensive in case of product lines comprising a lot of variants [53]. Moreover, the variant-by-variant approach may result in many redundant test cases if some test goals are part of many variants (i.e., a higher number of similarities results in a higher number of redundant test cases). Furthermore, variant-by-variant test-suite generation is infeasible for product lines with configurable parametric real-time constraints having an infinite number of variants (as introduced in Chapter 3).

Example 5.1. Consider the CoPTA model \mathcal{C} and the EFM m of the PPU SPL as presented in Chapter 3 (see Figures 3.2 and 3.3). For the sake of this example, assume a slightly adapted EFM

$$m' := m \wedge (\text{Plastic} \Rightarrow a \leq 30) \wedge (\text{PPU} \Rightarrow c \leq 40)$$

such that $\llbracket m' \rrbracket$ only contains a finite number of configurations. Even in case of this small example (in the number of features), EFM m' already describes 18,000 configurations. Hence, complete variant-by-variant test-suite generation is expensive for this adapted example. Moreover, using the original EFM m ,

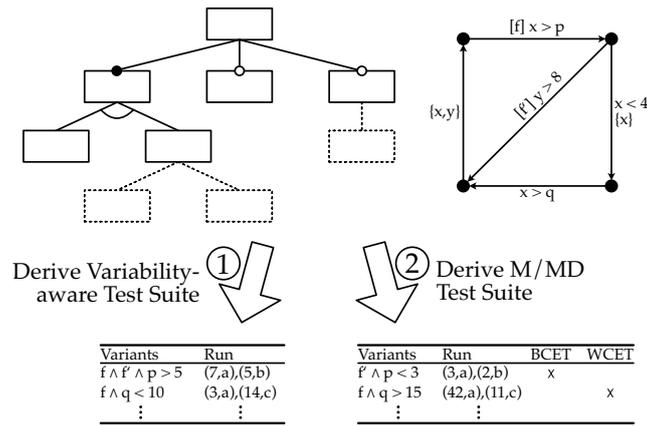


Figure 5.1: Schematic Overview on the Contributions Presented in Chapter 5

complete variant-by-variant test-suite generation is even impossible due to the infinite number of variants.

The goal of this chapter is to describe a method for generating finite complete test suites *for each variant* in a family-based way by exploiting the CoPTA representation of a product line. Therewith, we can check for which variants a test case is valid. Furthermore, we consider test cases having overall BCET/WCET (i.e., as compared to all valid test cases for a CoPTA) for reaching our test goals. As a result, we have a higher chance of revealing faults [203]. For instance, test cases having BCET/WCET may reveal off-by-one timing errors [6] where boundary-value behavior w.r.t. delays is faulty. However, note that the CoPTA-based approaches introduced in this chapter are incomplete. Hence, generating test cases based on CoPTA models may not terminate, but if the generation terminates, then the result is a finite complete test suite.

Figure 5.1 gives an overview on the remainder of this chapter which is structured as follows. We start by recapitulating basic notions of model-based test-case generation from TA, and we present an algorithm for test-suite generation satisfying complete location coverage (see Section 5.1). Thereafter, we lift the notion of location coverage to product lines with configurable parametric real-time constraints, possibly comprising an infinite number of variants (see Section 5.2). In particular, we consider an (incomplete) approach for generating a finite complete test suite (even in case of infinitely many variants) such that every reachable location is covered by at least one test case *in every variant* where the location is reachable (step ① in Figure 5.1). Afterwards, we generate test cases having overall BCET/WCET (as compared to other test cases) for reaching our test goals such that we have a higher chance of revealing faults (see Section 5.3). To this end, we adapt notions of M/MD coverage (as presented for sampling in Chapter 4) to test-case generation (step ② in Figure 5.1). Finally, we evaluate the approaches presented in this chapter (see Section 5.4), give an overview on related work (see Section 5.5), and conclude this chapter (see Section 5.6). The contents of this chapter are based on the following two publications:

[133] Lars Luthmann, Andreas Stephan, Johannes Bürdek, and Malte Lochau. Modeling and Testing Product Lines with Unbounded Para-

metric Real-Time Constraints. In *21st International Systems and Software Product Line Conference (SPLC '17)*, pages 104–113. ACM, 2017. ISBN 978-1-4503-5221-5. doi: 10.1145/3106195.3106204.

[135] Lars Luthmann, Timo Gerech, Andreas Stephan, Johannes Bürdek, and Malte Lochau. Minimum/maximum delay testing of product lines with unbounded parametric real-time constraints. *Journal of Systems and Software (JSS)*, 149: 535–553, 2019. ISSN 0164-1212. doi: 10.1016/j.jss.2018.12.028.

5.1 TEST-CASE GENERATION FROM TIMED AUTOMATA

In this section, we recall the notions of test-case generation and test-suite generation for single discrete-state/continuous-time systems specified as TA. Therewith, we generate test cases into a test suite until a required coverage of the test model is achieved or until we cannot find additional test cases. Furthermore, we facilitate reuse of test cases to cover multiple test goals at once (i.e., a single test case can cover multiple test goals). As a result, we are able to thoroughly (w.r.t. a coverage criterion) check for a system whether the tested behavior conforms to the expected behavior.

5.1.1 Test Cases for Timed Automata

We first describe the notions of test cases and test suites for TA. As TA are an extension of classical (untimed) automata, TA test cases extend test cases of untimed automata (i.e., sequences of actions) by delays. In order to describe test cases for TA, we start by deriving an *untimed* test sequence (i.e., a finite sequence of actions) from a TA \mathcal{A} . Then, an *abstract test case* is a run consisting of timed steps (i.e., pairs of delays and actions) as well as information about reached locations (see Notation 2.3). When applied to an IUT, a *concrete test case* is a sequence of delays and actions (i.e., a timed trace) corresponding to the abstract test case and being allowed by TA \mathcal{A} [54]. Hence, the untimed test sequence is enriched by delays between consecutive timed test steps. Therefore, each valid test case for \mathcal{A} corresponds to a timed run of the TA test model (i.e., to a run in $\llbracket \mathcal{A} \rrbracket_R$). It should be noted that we omit information about reached locations from concrete test cases as we may execute test cases on black-box implementations where the internal structure is unknown to the tester.

Furthermore, we use $visited(t)$ to denote the set of locations reached by the run corresponding to t . To this end, we utilize the distinction between abstract test cases and concrete test cases as abstract test cases allow us to track the set of locations being reached by these abstract test cases. In contrast, concrete test cases do not contain information about reached locations as concrete test cases are created such that they can be executed on black-box implementations where the internal structure (e.g., names of locations) is hidden from the tester.

Definition 5.1 (Test Case). Let $\mathcal{A} = (L, \ell_0, \Sigma, C, I, E)$ be a TA. Then, a timed run

$$\langle \ell_0, u_0 \rangle \xrightarrow{(d_1, \sigma_1)} \dots \xrightarrow{(d_n, \sigma_n)} \langle \ell_n, u_n \rangle$$

is an *abstract test case* if $n \in \mathbb{N}_0$ is finite. The corresponding *timed trace* $t \in (\mathbb{T}_C \times \Sigma)^*$ with $t = (d_1, \sigma_1), \dots, (d_n, \sigma_n)$ is a *concrete test case* with *untimed test sequence* $\sigma_1 \cdots \sigma_n$. Concrete test case t is *valid* w.r.t. \mathcal{A} if

$$\exists \rho \in \llbracket \mathcal{A} \rrbracket_R : \rho = \langle \ell_0, u_0 \rangle \xrightarrow{(d_1, \sigma_1), \dots, (d_n, \sigma_n)}.$$

We use $\text{visited}(t) \subseteq L$ to denote the set of locations reached by t such that $\ell \in \text{visited}(t)$ iff

$$\langle \ell_0, u_0 \rangle \xrightarrow{(d_1, \sigma_1)} \langle \ell_1, u_1 \rangle \dots \xrightarrow{(d_n, \sigma_n)} \langle \ell_n, u_n \rangle \in \llbracket \mathcal{A} \rrbracket_R \wedge \ell \in \{\ell_0, \dots, \ell_n\}.$$

In the following, we refer to concrete test cases simply by test cases if clear from the context. Moreover, a finite set of concrete test cases is called a *test suite* [190]. Here, we overload $\text{visited}(T)$ to denote the set of locations reached by any test case $t \in T$.

Definition 5.2 (Test Suite). Let $T \subseteq (\mathbb{T}_C \times \Sigma)^*$ be a finite set of concrete test cases w.r.t. TA $\mathcal{A} = (L, \ell_0, \Sigma, C, I, E)$. In the context of testing, we call T a *test suite*. T is *valid* if every $t \in T$ is valid. We use $\text{visited}(T) \subseteq L$ to denote the set of locations reached by any $t \in T$ such that

$$\text{visited}(T) = \bigcup_{t \in T} \text{visited}(t).$$

It should be noted that test cases have a finite length (and test suites a finite size) as otherwise, test cases could not be practically executed.

Example 5.2 (Test Case). Consider the TA of the PPU presented in the background chapter in Figure 2.3 (which we show again in Figure 5.2 for the convenience of the reader). Furthermore, consider the abstract test sequence *rotate, pick, rotate, place* presented in Example 2.17 for which a concrete test-case execution may, for instance, be

- waiting for 12 seconds and *rotating* the crane,
- waiting for 8 seconds and *picking* the workpiece,
- waiting for 14 seconds and *rotating* the crane again, and
- waiting for 9 seconds and *placing* the workpiece.

Hence, the resulting test case is $t_1 = (12, \text{rotate}), (8, \text{pick}), (14, \text{rotate}), (9, \text{place})$ with corresponding abstract test case (i.e., run)

$$\rho_1 = \text{init} \xrightarrow{(12, \text{rotate})} \text{stack} \xrightarrow{(8, \text{pick})} \text{WP picked} \xrightarrow{(14, \text{rotate})} \text{ramp} \xrightarrow{(9, \text{place})} \text{WP placed}.$$

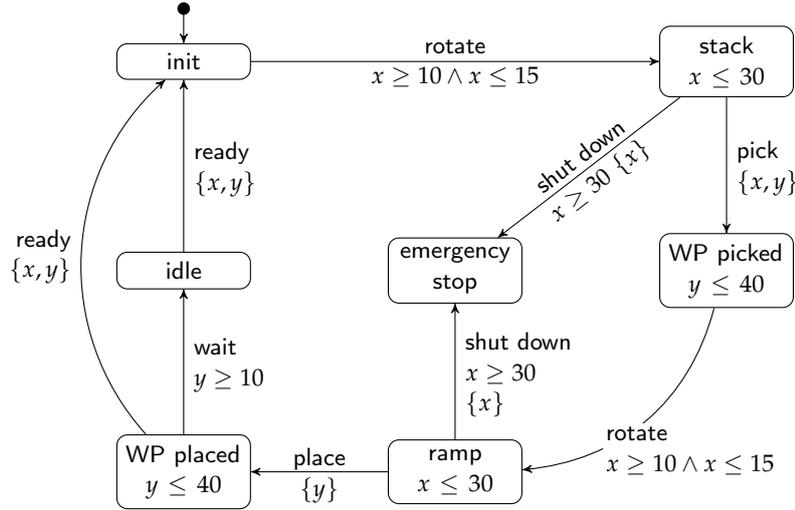


Figure 5.2: TA of an Extract of the PPU with Clocks x and y (Copy of Figure 2.3)

Furthermore, we may add test cases $t_2 = (14, \text{rotate}), (16, \text{shut down})$ and $t_3 = (10, \text{rotate}), (17, \text{pick}), (12, \text{rotate}), (4, \text{place}), (23, \text{wait})$. This results in test suite $T = \{t_1, t_2, t_3\}$ being valid w.r.t. the TA depicted in Figure 5.2.

So far, we have described test cases and test suites. Next, we utilize test cases to detect possible faults in implementations. To this end, we assume that a TA model \mathcal{A} of the implementation is available, such that we can *execute* TA test cases and determine whether test cases *pass* or *fail*. Here, TA \mathcal{A} passes test case t if \mathcal{A} contains a run corresponding to t . Analogously, TA \mathcal{A} fails test case t if \mathcal{A} does not contain a run corresponding to t . Hence, we *inject* actions provided by t after waiting for the respective delay in each test step, and we observe whether the implementation can perform these actions. Furthermore, \mathcal{A} passes test suite T if \mathcal{A} passes all test cases $t \in T$. Finally, \mathcal{A} fails test suite T if \mathcal{A} fails at least one test case $t \in T$.

Definition 5.3 (Test Execution). Let TA \mathcal{A} be a specification, T be a test suite derived from \mathcal{A} , and \mathcal{A}' be the TA of an implementation.

- \mathcal{A}' passes test case $t \in T$, denoted by $\text{pass}_{\mathcal{A}'}(t)$, if $\exists \rho \in \llbracket \mathcal{A}' \rrbracket_R : \rho = \langle \ell_0, u_0 \rangle \xrightarrow{t}$.
- \mathcal{A}' fails test case $t \in T$, denoted by $\text{fail}_{\mathcal{A}'}(t)$, if $\nexists \rho \in \llbracket \mathcal{A}' \rrbracket_R : \rho = \langle \ell_0, u_0 \rangle \xrightarrow{t}$.
- \mathcal{A}' passes test suite T , denoted by $\text{pass}_{\mathcal{A}'}(T)$, if $\forall t \in T : \text{pass}_{\mathcal{A}'}(t)$.
- \mathcal{A}' fails test suite T , denoted by $\text{fail}_{\mathcal{A}'}(T)$, if $\exists t \in T : \text{fail}_{\mathcal{A}'}(t)$.

It should be noted that the definition of *pass* and *fail* could be further extended in case of a distinction between inputs and outputs. In particular, we could compare the output behavior after a given input. However, in this thesis we do not distinguish between inputs and outputs such that our notions of *pass* and *fail* solely rely on runs.

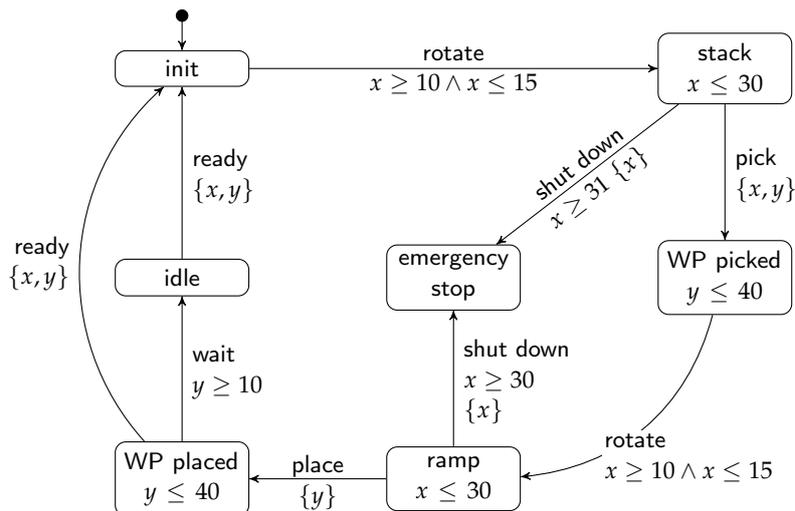


Figure 5.3: (Faulty) Implementation of the TA Depicted in Figure 5.2

Example 5.3 (Test Execution). Consider TA \mathcal{A} depicted in Figure 5.2 as our specification, TA \mathcal{A}' depicted in Figure 5.3 as (faulty) implementation, and test suite T derived from \mathcal{A} as described in Example 5.2. Here, \mathcal{A}' is faulty w.r.t. the specification as the guard of the switch labeled with *shut down* is incorrect (i.e., $x \geq 31$ instead of $x \geq 30$). When executing test case $t_1 \in T$, \mathcal{A}' passes t_1 as this test case does not traverse the faulty part of the implementation, and the same holds for test case t_3 . However, the implementation \mathcal{A}' fails test case $t_2 \in T$ as this test traverses the switch labeled with *shut down*, and clock x has a value of 30 such that the (faulty) guard is violated. As a result, \mathcal{A}' also fails test suite T as at least one of the test cases in T failed.

Next, we consider a coverage criterion for TA such that we have a metric to determine whether test suite T contains a sufficient number of test cases.

5.1.2 Location Coverage for Timed Automata

When generating test cases, we apply a metric to determine whether the current test suite T contains a sufficient number of test cases or whether we have to generate further test cases. This metric is a coverage criterion [190], such that we can stop adding test cases if the criterion is fulfilled (or fulfilled to a particular degree). Note, that coverage criteria also have further use cases such as evaluating existing test suites. In case of TA, a basic criterion is *location coverage* [116]. However, the techniques presented in this chapter can be applied with other criteria (e.g., switch coverage). In particular, location coverage describes the ratio between locations reached by at least one test case $t \in T$ and the number of all (reachable) locations. Compared to well-known coverage criteria for code (e.g., C code), location coverage is similar to C_0 coverage [180]. C_0 coverage describes statement coverage for code, such that every basic element of a program is covered by at least one test case. Hence, location coverage transfers the idea of C_0 coverage to TA.

Note, that a single TA test case might cover multiple locations. Hence, the size of a complete test suite T might be smaller than the number of locations in L . Moreover, some locations of a TA model may not be *reachable* (see Notation 2.3). Hence, we use the subset $L' \subseteq L$ of reachable locations as the set of test goals to compute location coverage.

Definition 5.4 (Location Coverage). Let $\mathcal{A} = (L, \ell_0, \Sigma, C, I, E)$ be a TA, $L' \subseteq L$ be the set of reachable locations, and T be a test suite of \mathcal{A} . Then, *location coverage* denotes the ratio

$$\text{coverage}(T, L') = \frac{|\text{visited}(T)|}{|L'|}.$$

Please note, that a test suite T satisfying location coverage for TA \mathcal{A} is not unambiguous (i.e., in general, there exist several different test suites satisfying complete location coverage). Moreover, location coverage does not necessarily result in coverage of other basic components (i.e., switches) of a TA. For instance, it might be the case that a test suite satisfying complete location coverage does not cover all switches. As a result, it might be necessary to consider other coverage criteria (e.g., switch coverage [54]), depending on the use case. Again, this is similar to metrics for code coverage where, for instance, guidelines for avionics software development require more sophisticated coverage criteria [180].

Furthermore, test suite T satisfies *complete* location coverage if for every (reachable) location $\ell \in L$, there is a test case in T such that the corresponding run reaches location ℓ . Additionally, test suite T is *minimal* if T is complete and there does not exist a smaller complete test suite T' . Next, we consider a generic definition for complete and minimal test suites (i.e., for a set of test goals G and a test suite T w.r.t. G) as these notions are the same for all coverage criteria defined in this chapter.

Definition 5.5. Let G be a set of test goals and T be a test suite w.r.t. G . Then, T satisfies *complete* coverage if $\text{coverage}(T, G) = 1$. T is *minimal* if $\text{coverage}(T, G) = 1$ and there does not exist a test suite T' such that

$$\text{coverage}(T', G) = 1 \wedge |T'| < |T|.$$

Alternatively, the notion of minimal test suites could also take into account further criteria such as the length of the test cases (i.e., the number of test steps) or the number of test cases for test suite having a coverage of less than 1. For test suites, we use coverage as a metric for *effectiveness*, such that a complete test suite is the most effective test suite. Furthermore, the *efficiency* of a test suite T is the size $|T|$ of the test suite. As a result, an empty test suite could be considered as most efficient. However, to have a meaningful efficiency criterion, we consider *minimal* test suites as most efficient.

Example 5.4 (Location Coverage). Consider, again, the TA of the PPU depicted in Figure 5.2 and test suite $T = \{t_1, t_2, t_3\}$ presented in Example 5.2. Here, t_1 covers the locations *init*, *stack*, *WP picked*, *ramp*, and *WP placed*. Furthermore, t_2

covers *init*, *stack*, and *emergency stop*. Finally, test case t_3 covers *idle* in addition the locations already covered by t_1 . Hence, test suite T satisfies complete location coverage.

Note, that a test suite $T' = \{t_2, t_3\}$ also satisfies complete location coverage as the locations covered by t_1 are subsumed by t_3 , such that test suite T is not minimal. Moreover, T as well as T' do not cover all basic components of the TA although both of them satisfy complete location coverage. For instance, neither of the switches labeled with action *ready* are reached by test cases of T or T' . This could be solved by applying a different coverage criterion (e.g., switch coverage).

Having defined test suites and location coverage for TA, we proceed by providing an algorithm for generation test suites satisfying location coverage.

5.1.3 Generating Test Suites for Complete Location Coverage on TA

So far, we described and defined test cases and test suites (satisfying complete location coverage) for TA. Next, we consider a means for deriving such a test suite from a specification given as a TA model. Here, we utilize a reachability check [44] of a TA model checker (e.g., UPPAAL [39, 34]) to generate abstract test cases. In particular, the (partial) function GETTC of our algorithm denotes the usage of the reachability check. Function GETTC receives as input a location ℓ and returns an abstract test case for reaching ℓ .

Definition 5.6. Let $\mathcal{A} = (L, \ell_0, \Sigma, C, I, E)$ be a TA. Then, $\text{GETTC} : L \rightarrow \llbracket \mathcal{A} \rrbracket_R$ returns a timed run $\rho \in \llbracket \mathcal{A} \rrbracket_R$ for reaching locations $\ell \in L$ such that

$$\text{GETTC}(\ell) = \langle \ell_0, u_0 \rangle \xrightarrow{(d_1, \sigma_1), \dots, (d_n, \sigma_n)} \langle \ell, u_n \rangle.$$

Algorithm 5.1 presents a basic procedure for generating a complete test suite satisfying location coverage for any given input TA $\mathcal{A} = (L, \ell_0, \Sigma, C, I, E)$. Please note, that the algorithm is adapted from Cardell-Oliver and Glover [54] who use a similar technique to achieve switch coverage. Furthermore, this algorithm generates a complete, but not necessarily minimal test suite.

Next, we describe Algorithm 5.1 line by line. Here, we start by initializing test suite T with the empty set (see line 2) and the set of test goals G containing the set of locations L (line 3). Then, we iterate over G (without any particular ordering) until all locations are covered (lines 4–10). In particular, we utilize a TA model checker to generate an abstract test case ρ (i.e., a timed run) targeting an uncovered test goal $\ell_n \in G$ (line 5), where test case t corresponds to the delays and actions of the run. Recall, that an abstract test case contains information about reached locations (see Notation 2.3). Moreover, checking reachability for TA locations results in a precise witness for reaching the test goal (i.e., there are no false negatives or false positives). Next, we add test case t to test suite T (line 7) and remove all locations visited by the abstract test case ρ from the set of test goals G (line 8) as all these locations are also covered by t . Finally, we return test suite T (line 11).

Algorithm 5.1: Test-Suite Generation Satisfying Location Coverage for TA
 (Adapted from Cardell-Oliver and Glover [54])

Input : TA $\mathcal{A} = (L, \ell_0, \Sigma, C, I, E)$
Output: test suite T

```

1 procedure MAIN
2    $T := \emptyset$ 
3    $G := L$ 
4   while  $G \neq \emptyset$  do
5      $\rho = \langle \ell_0, u_0 \rangle \xrightarrow{t = (d_1, \sigma_1), \dots, (d_n, \sigma_n)} \langle \ell_n, u_n \rangle := \text{GETTC}(\ell_n \in G)$ 
6     if  $t \neq \epsilon$  then
7        $T := T \cup \{t\}$ 
8        $G := G \setminus \{\ell_0, \dots, \ell_n\}$ 
9     else
10       $G := G \setminus \{\ell_n\}$ 
11  return  $T$ 

```

Please note, that Algorithm 5.1 may have non-deterministic behavior, such that the resulting test suite T may vary (although T always satisfies complete location coverage). In particular, we choose an arbitrary test goal $\ell_n \in G$ as input for the model checker (line 5). Hence, executing Algorithm 5.1 multiple times may result in different test suites. Furthermore note, that we check whether locations are reachable. To this end, we check in line 6 whether the model checker generated a non-empty run (i.e., a run where ℓ_n is reached). In case of an empty run ϵ (i.e., ℓ_n is unreachable), we only remove ℓ_n from the set of test goals G (lines 9–10).

Example 5.5 (Test-Suite Generation for TA). Consider, again, the TA of the PPU depicted in Figure 5.2. Here, we utilize Algorithm 5.1 to generate a test suite T satisfying location coverage. Furthermore, Table 5.1 gives an overview on the development of test suite T and the set of remaining uncovered test goals G for each iteration of the algorithm. Initially, we have $T = \emptyset$ and $G = L$ (see lines 2–3). Then, we pick a location in the set of remaining uncovered test goals G as input for the model checker. For instance, a reachability query for *WP placed* (line 5) might return run

$$\rho_1 = \text{init} \xrightarrow{(12, \text{rotate})} \text{stack} \xrightarrow{(8, \text{pick})} \text{WP picked} \xrightarrow{(14, \text{rotate})} \text{ramp} \xrightarrow{(9, \text{place})} \text{WP placed}$$

with corresponding test case $t_1 = (12, \text{rotate}), (8, \text{pick}), (14, \text{rotate}), (9, \text{place})$ as presented in Example 5.2. Hence, we add t_1 to T and remove all covered locations from G such that $T = \{t_1\}$ and $G = \{\text{idle}, \text{emergency stop}\}$ (lines 7–8). As $G \neq \emptyset$, we continue with a reachability query for location *emergency stop* (line 5). Here, we might obtain test case t_2 as described in Example 5.2. Hence, we have test suite $T = \{t_1, t_2\}$ and remaining test goals $G = \{\text{idle}\}$ (lines 7–8). Thereafter, we use the remaining location in G as input for the model checker to obtain test case t_3 from Example 5.2 (line 5), resulting in

Table 5.1: Development of Test Suite T and Remaining Uncovered Test Goals G in Example 5.5

Iteration	Test Suite T	Remaining Uncovered Test Goals G
initial	\emptyset	{init, stack, e. stop, WP picked, ramp, WP placed, idle}
1	$\{t_1\}$	{idle, e. stop}
2	$\{t_1, t_2\}$	{idle}
3	$\{t_1, t_2, t_3\}$	\emptyset

$T = \{t_1, t_2, t_3\}$ and $G = \emptyset$ (lines 7–8). As there are no remaining test goals, we return test suite T satisfying location coverage as result of Algorithm 5.1. It should be noted that we might obtain a smaller complete test suite $T' = \{t_2, t_3\}$ if we start with location *idle* as input for the model checker in line 5.

We conclude this section by showing that Algorithm 5.1 does, in fact, generate a complete test suite satisfying location coverage for a given TA \mathcal{A} . Note, that $\text{coverage}(T) = 1$ for test suite T is also possible in the presence of unreachable locations as these locations are excluded when we calculate location coverage (see Definition 5.4).

Lemma 5.1. Let $\mathcal{A} = (L, \ell_0, \Sigma, C, I, E)$ be a TA, $L' \subseteq L$ be the set of reachable locations, and T be the test suite obtained by applying Algorithm 5.1 to \mathcal{A} . Then, $\text{coverage}(T, L') = 1$.

Proof. Let $\mathcal{A} = (L, \ell_0, \Sigma, C, I, E)$ be a TA, $L' \subseteq L$ be the set of reachable locations, and T be the test suite obtained by applying Algorithm 5.1 to \mathcal{A} . We prove Lemma 5.1 by induction.

Induction basis: Initially, test suite $T = \emptyset$ (line 2) covers the set of test goals $G = \emptyset$ (line 3).

Induction step: If test suite T covers the set of test goals G , then $T \cup \{t\}$ covers $G \cup G'$, where G' is the set of test goals covered by t . In Algorithm 5.1, we generate a new test case t in each iteration of the while-loop (lines 4–10) if the test goal under consideration is reachable (line 5), where t covers at least the current target location ℓ_n (see properties of partial function GETTC in Definition 5.6). In particular, we add t to test suite T (line 7) and remove exactly the test goals G' covered by t from the set of uncovered test goals (line 8). Finally, there may be unreachable test goals $\ell_n \in L$. However, if ℓ_n is unreachable, ℓ_n is removed from the set of uncovered test goals (lines 9–10). We repeat these steps until $G = L$ (line 4), such that Lemma 5.1 is correct and $\text{coverage}(T, L') = 1$. \square

In this section, we described the notions of test cases and test suites for TA. Furthermore, we considered a coverage criterion, namely location coverage, as an example to measure effectiveness of test suites. Finally, we described a basic algorithm to generate a test suite satisfying location coverage for a TA model. In the next section, we lift the notions of test cases and test suites to variant-rich systems. Here, a straightforward strategy [185] for generating test suites for variant-rich systems is deriving all variants and then generate test suites for variants separately (as described in this section). However, generating a *complete* test suite with a

variant-by-variant approach is impossible in case of product lines with configurable parametric real-time constraints, (potentially) having an infinite number of variants. Hence, we adapt an approach for generating test cases directly from the product-line representation [53, 185] to generate complete test suites for product lines with unbounded parametric real-time constraints.

5.2 FAMILY-BASED TEST-CASE GENERATION FROM PRODUCT LINES WITH CONFIGURABLE PARAMETRIC REAL-TIME BEHAVIOR

In this section, we adapt an approach for family-based test-case generation [53] to product lines with configurable parametric real-time constraints specified as CoPTA models. Therewith, we are able to generate a complete finite test suite even in case of CoPTA having infinitely many variants. We achieve this goal by deriving test cases directly from CoPTA models. Here, we adapt the approach of Bürdek et al. [53] who generate test cases for product lines written in C code. As compared to Bürdek et al. who only support Boolean parameters (i.e., features), our approach additionally supports numeric parameters such that we can generate complete finite test suite even in case of CoPTA having infinitely many variants.

A first approach to derive a test suite for a CoPTA model is deriving all variants of a CoPTA and then perform variant-by-variant test-case generation. However, this is only possible in case of bounded CoPTA having a finite number of variants. Nonetheless, variant-by-variant test-case generation is inefficient even for CoPTA comprising a small number of variants as many parts of systems are present in multiple variants, resulting in redundant test cases [53]. Moreover, variant-by-variant test-case generation is impossible when considering CoPTA having an infinite number of variants. In particular, performing variant-by-variant test-case generation is sound (i.e., the resulting test suite is valid) but incomplete (i.e., not all test goals are covered) in this case.

Example 5.6. Consider the CoPTA model \mathcal{C} and the EFM m of the PPU presented in Chapter 3, which we repeat in Figure 5.4 for the convenience of the reader. Furthermore, assume an adapted EFM

$$m' := m \wedge (\text{Plastic} \Rightarrow a \leq 30) \wedge (\text{PPU} \Rightarrow c \leq 40)$$

such that $\llbracket m' \rrbracket$ contains a finite number of configurations. Even in case of this small example (in the number of features), $\llbracket m' \rrbracket$ already describes 18,000 configurations. Hence, deriving all variants and generating complete test suites in a variant-by-variant fashion results in hundreds of test case being generated for a single location. For instance, we would generate many test cases just to cover location *stack* in every variant although a test case $t = (10, \text{rotate})$ suffices to cover *stack* in *every* variant. This results in a high computational effort even for this small example and would be infeasible for larger real-world systems. Moreover, using the original EFM m as depicted in Figure 5.4a, it is even impossible to derive all variants as the set of configurations $\llbracket m \rrbracket$ is infinite.

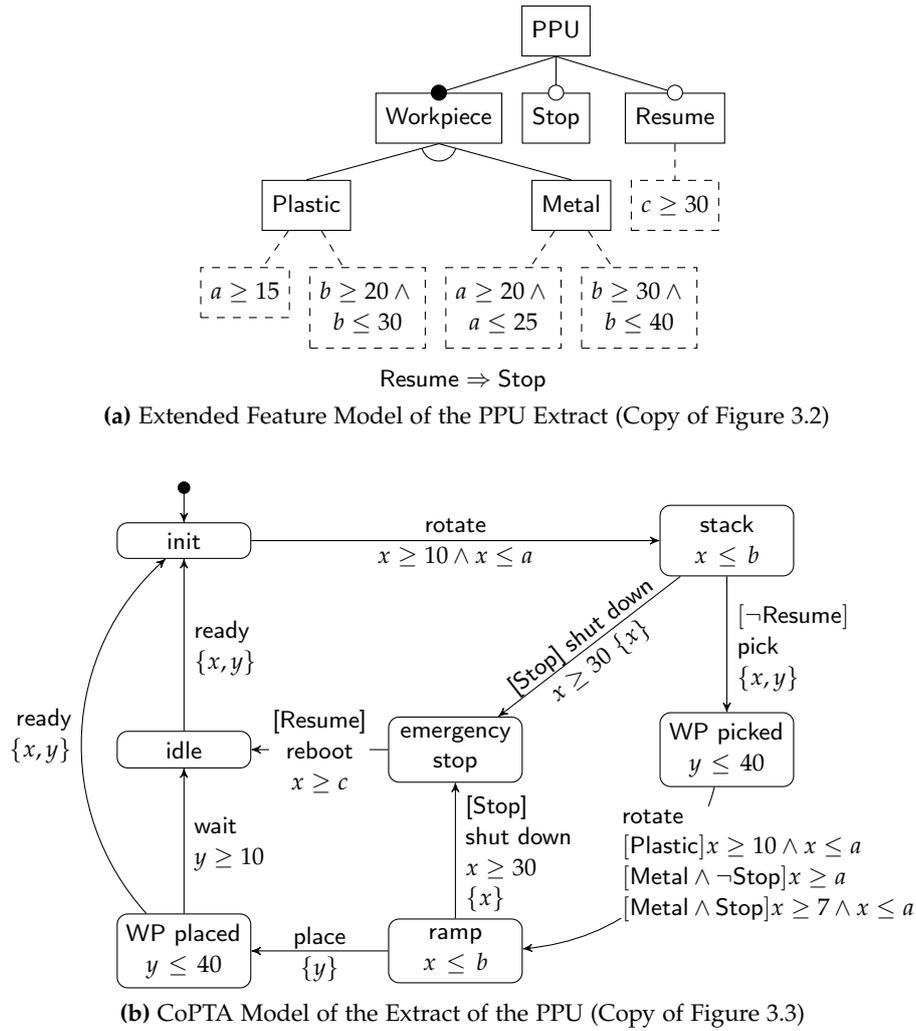


Figure 5.4: Example for a CoPTA of the PPU

Hence, we introduce an approach for generating test cases being applicable to cover test goals in a (potentially) infinite number of variants. To this end, we utilize CoPTA annotations in terms of constraints over Boolean parameters and numeric parameters as constraints (so-called *presence conditions* [58]) for test cases. For instance, the initial switch of CoPTA \mathcal{C} in Figure 5.4 (being annotated with action *rotate*) contains the guard $x \geq 10 \wedge x \leq a$. As a result, a step (13, *rotate*) is valid for all configurations satisfying $m \wedge a \geq 13$ where m is the EFM (i.e., all configurations of m as parameter a has at least value 15).

Another approach to derive a test case for a CoPTA model is simply ignoring the variability (i.e., Boolean and numeric parameters), such that models are treated as TA. Therewith, we are able to apply the notions introduced in the previous section. Furthermore, test cases being generated as done for TA may be valid test cases for multiple variants of a CoPTA model. However, it may also be the case that test cases are not valid for all variants. Hence, when deriving a variant of a CoPTA model, we have to check for every test case if it is valid for the current variant.

Example 5.7. Again, consider the CoPTA model \mathcal{C} and the EFM m of the PPU in Figure 5.4. When we ignore the variability of the CoPTA model and treat the model as a TA, we can, for instance, derive a test case

$$t = (10, \text{rotate}).$$

In fact, test case t suffices to cover locations *init* and *stack* in *every* variant of CoPTA \mathcal{C} . Furthermore, we could derive a test case

$$t' = (12, \text{rotate}), (3, \text{pick}), (7, \text{rotate}).$$

However, t' is only valid for configurations satisfying $m \wedge \text{Metal} \wedge \text{stop}$ (due to the last timed step of t'). Finally, we might derive a test case

$$t'' = (27, \text{rotate}), (4, \text{pick}), (7, \text{rotate})$$

when considering the CoPTA model as a TA. Unfortunately, test case t'' is not valid for any configuration described by EFM m as t'' is only valid for configurations satisfying $\text{Metal} \wedge \text{Stop} \wedge a \geq 27 \wedge b \geq 31$. However, it holds that $\llbracket m \wedge \text{Metal} \wedge \text{Stop} \wedge a \geq 27 \wedge b \geq 31 \rrbracket = \emptyset$, such that there does not exist a valid configuration.

As described by Example 5.7, test-case generation could be done for CoPTA as described in the previous section by ignoring the variability (i.e., Boolean and numeric parameters) of CoPTA models and treating CoPTA models as TA. However, this approach has some drawbacks. First, it might be the case that test cases being generated with this approach are not valid for all variants. As a result, we have to check for all test cases if these test cases are valid for variants that we are interested in. Moreover, we might generate test cases being invalid for all variants, wasting time and resources during test-case generation. Finally, we cannot guarantee that a test suite satisfies complete location coverage as we do not know which test cases are valid for a variant before deriving this variant.

Due to the drawbacks of the approaches described above (i.e., variant-by-variant test-case generation and treating CoPTA models as TA), we introduce an approach for generating a finite test suite (w.r.t. a coverage criterion) for CoPTA models (by deriving test cases directly from the CoPTA model) in this section. To this end, we adapt the notion of test cases such that a *variability-aware test case* additionally contains a *presence condition*, symbolically describing the set of configurations for which the test case is valid. However, note that the CoPTA-based approach introduced in this section is also incomplete. In particular, we utilize reachability queries to generate test cases, which are already incomplete for PTA [20, 21]. Hence, generating test cases based on CoPTA models may not terminate, but if the generation terminates, then the result is a finite complete test suite.

5.2.1 Variability-aware Test Cases for CoPTA

As motivated by Example 5.7, a single test case may cover a test goal in several variants. In order to describe the (possibly infinite) set of variants for which a



Figure 5.5: Example for Necessity of Parametric Delay Constraints for Finite Variability-Aware Test Suites

test case is valid, we adapt our notion of test cases to also include a so-called *presence condition* [58]. As mentioned before, this is similar to the approach of Bürdek et al. [53] who generate test cases for product lines written in C code. However, Bürdek et al. only support Boolean parameters (i.e., features) in presence conditions, whereas our approach additionally supports numeric parameters such that we can generate complete finite test suite even in case of CoPTA having infinitely many variants.

In our case, a presence condition $\xi \in \mathcal{B}(P_F, P_N)$ is a configuration constraint symbolically describing a set $\llbracket \xi \rrbracket$ of configurations for which a test case is valid. Here, $\llbracket \xi \rrbracket$ may be infinite such that a single test case may cover a test goal in an infinite number of variants. As a presence condition can describe an infinite set of configurations, we are able to derive a complete finite test suite even in case of product lines having an infinite number of variants (e.g., as the CoPTA model of the PPU described in Example 5.7). Furthermore, we use presence condition $\xi = m$ (for an EFM m) if a test case is valid for all variants.

However, before we formally define the notion of *variability-aware test case* for CoPTA, we generalize the notion of timed steps. In particular, we introduce *parametric delay constraints* to describe a set of possible delays (as compared to a specific constant delay in Definition 5.1). This is necessary as using specific delays instead of parametric delay constraints is not sufficient to achieve complete family-based location coverage. Instead, *complete* family-based location may require infinite test suites when only considering specific delays, which we illustrate with the following example.

Example 5.8. Figure 5.5 depicts an example showing that it might not be possible to obtain a finite complete test suite when only considering specific delays instead of parametric delay constraints. Assume, that the corresponding EFM is $m := \text{true} \Rightarrow p \geq 0$. First, we consider CoPTA \mathcal{C} in Figure 5.5a, having unbounded parameter p as *upper* bound of the guard. Here, a single test case $t = (0, \sigma)$ with presence condition $\xi := m$ satisfies location coverage for every variant of \mathcal{C} as p has at least value 0. Second, we consider CoPTA \mathcal{C}' in Figure 5.5b, having unbounded parameter p as *lower* bound. Here, we cannot find a test case with a specific delay being valid for all variants. For instance, a test case $t' = (4711, \sigma)$ is valid for all variants satisfying presence condition $\xi' := p \leq 4711$. However, we cannot find a value to cover *all* variants if parameter domain \mathbb{T}_p is unbounded (e.g., $\mathbb{T}_p = \mathbb{N}_0$).

As a result of the example above, we utilize a constraint describing a set of possible delays instead of a specific delay. Hence, for CoPTA \mathcal{C}' in Figure 5.5b we obtain, for instance, a test case $t'' = (d \geq p, \sigma)$ with presence condition $\zeta'' := m$, where the value of delay d is greater or equal to the value of parameter p when deriving a TA test case. In general, a parametric delay constraint compares a delay $d \in \Delta$ with a parametric linear term (where $\Delta = \mathbb{T}_C$). Moreover, a delay d can be compared to another delay $d' \in \Delta$. Therewith, we denote that values for delay d depend on a previous delay d' (e.g., two timed steps have a combined delay of 30 seconds, where the first delay may take 10 to 15 seconds).

Definition 5.7 (Parametric Delay Constraint). Let $\Delta = \mathbb{T}_C$ be the delay domain and P_N be a set of parameters defined over parameter domains \mathbb{T}_P . The set $\mathcal{B}(\Delta)$ of *parametric delay constraints* ω is inductively defined as

$$\omega := \text{true} \mid \omega \wedge \omega' \mid d \sim \text{plt}_{P_N} \mid d - d' \sim \text{plt}_{P_N}$$

where $\{d, d'\} \subseteq \Delta$ are delays, plt_{P_N} is a parametric linear term over P_N , and $\sim \in \{<, \leq, \geq, >\}$.

The formal semantics of a parametric delay constraint ω is denoted as $\llbracket \omega \rrbracket$. Hence, $\llbracket \omega \rrbracket$ denotes the set of all delays satisfying parametric delay constraint ω .

Definition 5.8 (Delay-Constraint Evaluation). Let $\omega \in \mathcal{B}(\Delta)$ be a parametric delay constraint and $m \in \mathcal{B}(P_F, P_N)$ be an EFM with $P = P_F \cup P_N$. The set of *delay configurations* w.r.t. a configuration $c \in \llbracket m \rrbracket$ is denoted as $\llbracket \omega \rrbracket = \{\iota : (\Delta \cup P_N) \rightarrow \mathbb{T}_C \mid \iota \in \omega\}$, where $\iota \in \omega$ is defined recursively as

$$\iota \in \text{true}$$

$$\iota \in \omega \wedge \omega' \Leftrightarrow \iota \in \omega \text{ and } \iota \in \omega'$$

$$\iota \in d \sim \left(\sum_{1 \leq i \leq |P_N|} \alpha_i p_i \right) + n \Leftrightarrow \iota(d) \sim \left(\sum_{1 \leq i \leq |P_N|} \alpha_i \cdot c(p_i) \right) + n$$

$$\iota \in d - d' \sim \left(\sum_{1 \leq i \leq |P_N|} \alpha_i p_i \right) + n \Leftrightarrow \iota(d) - \iota(d') \sim \left(\sum_{1 \leq i \leq |P_N|} \alpha_i \cdot c(p_i) \right) + n$$

with $\sim \in \{<, \leq, \geq, >\}$, $\{d, d'\} \subseteq \Delta$, $p_i \in P_N$, and $\left(\sum_{1 \leq i \leq |P_N|} \alpha_i p_i \right) + n$ being a parametric linear term. Parametric delay constraints $\{\omega, \omega'\} \subseteq \mathcal{B}(\Delta)$ are *equivalent* if $\llbracket \omega \rrbracket = \llbracket \omega' \rrbracket$.

Please note, that we may write a delay instead of a parametric delay constraint if the constraint only has a single valid solution. For instance, we may write $(10, \sigma)$ instead of $(d \geq 10 \wedge d \leq 10, \sigma)$. Furthermore, we may omit to write an explicit conjugation of a parametric delay constraint with an EFM m if this is clear from the context.

Example 5.9 (Parametric Delay Constraint). Consider the CoPTA of the PPU depicted in Figure 5.4. Here, a possible parametric delay constraint for initially *rotating* the crane is given by

$$\omega_1 := d \leq a \wedge d \leq b \wedge d \geq 10$$

due to the guard $x \geq 10 \wedge x \leq a$ of the switch and the invariant $x \leq b$ of location *stack*. Moreover, a possible parametric delay constraint for subsequently *picking* the workpiece is

$$\omega_2 := d' \leq b - d$$

such that this delay depends on the previous delay (as these two switches may take at most b seconds due to the invariant of location *stack*). An alternative parametric delay constraint for the first switch is $\omega'_1 := d = 10$, denoting a delay being valid in all variants of the CoPTA.

Having defined parametric delay constraints, we are now able to formally define *variability-aware test cases* for product lines with configurable real-time behavior. Therewith, we can describe a complete test suite (w.r.t. a coverage criterion) which is finite even in case of CoPTA with an infinite number of variants. In order to achieve a finite complete test suite, each variability-aware test case t consists of two parts. First, the presence condition ξ symbolically denotes the (possibly infinite number of) configurations for which a variability-aware test case is valid (i.e., all configurations $c \in \llbracket \xi \rrbracket$). Second, the actual test case consists of a (finite) sequence of pairs of parametric delay constraints and actions. Moreover, a variability-aware test case $t = [\xi](\omega_1, \sigma_1), \dots, (\omega_n, \sigma_n)$ is *valid* if it holds that every TA \mathcal{A} corresponding to a configuration $c \in \llbracket \xi \rrbracket$ contains a run $\langle \ell_0, u_0 \rangle \xrightarrow{(d_1, \sigma_1), \dots, (d_n, \sigma_n)}$ such that $d_i \in \omega_i$ with $1 \leq i \leq n$. Hence, we can derive a valid TA test case for each variant $c \in \llbracket \xi \rrbracket$.

Definition 5.9 (Variability-aware Test Case). Let $\mathcal{C} = (L, \ell_0, \Sigma, C, P_F, P_N, I, E, m, \eta)$ be a CoPTA. Then, a finite *featured parametric timed trace*

$$t \in \mathcal{B}(P_F, P_N) \times (\mathcal{B}(\Delta) \times \Sigma)^*$$

with $t = [\xi](\omega_1, \sigma_1), \dots, (\omega_n, \sigma_n)$ and $\emptyset \subset \llbracket \xi \rrbracket \subseteq \llbracket m \rrbracket$ is a *variability-aware test case* where $\xi \in \mathcal{B}(P_F, P_N)$ is called a *presence condition*. Variability-aware test case t is *valid* w.r.t. \mathcal{C} iff each TA $\mathcal{A} \in \llbracket \mathcal{C} \rrbracket_c$ corresponding to a configuration $c \in \llbracket \xi \rrbracket$ contains a run

$$\langle \ell_0, u_0 \rangle \xrightarrow{(d_1, \sigma_1), \dots, (d_n, \sigma_n)}$$

such that $d_i \in \omega_i$ with $1 \leq i \leq n$.

It should be noted that we do not distinguish abstract from concrete variability-aware test cases (as done for TA test cases) as we do not execute these test cases. Instead, only TA test cases derived from these variability-aware test cases are executed. Furthermore, as a result of using parametric delay constraints instead of constant values, we have to derive a fixed delay from each constraint corresponding to a configuration to obtain a TA test case. This is different to the family-based approach for test-case generation used by Bürdek et al. [53], where each test case

can directly be applied to all configurations described by the presence condition of the test case. Moreover, a finite set of variability-aware test cases is called a *variability-aware test suite*.

Definition 5.10 (Variability-aware Test Suite). Let $T \subseteq \mathcal{B}(P_F, P_N) \times (\mathcal{B}(\Delta) \times \Sigma)^*$ be a finite set of variability-aware test cases w.r.t. CoPTA $\mathcal{C} = (L, \ell_0, \Sigma, C, P_F, P_N, I, E, m, \eta)$. In the context of testing, we call T a *variability-aware test suite*. T is *valid* if every $t \in T$ is valid.

It should be noted that we may refer to variability-aware test cases and variability-aware test suites simply as test cases and test suites if this is clear from the context. Furthermore, we ensure in Definition 5.9 that each parametric delay constraint ω_i (and the set of parametric delay constraints of a test case) is satisfiable w.r.t. the presence condition ζ of a test case (i.e., each ω is satisfiable for every configuration $c \in \llbracket \zeta \rrbracket$). We achieve this in Definition 5.9 by requiring that there exists a delay $d_i \in \omega_i$ for each TA $\mathcal{A} \in \llbracket \mathcal{C} \rrbracket_c$ corresponding to a configuration $c \in \llbracket \zeta \rrbracket$. Finally, a test suite for a TA \mathcal{A} corresponding to configuration $c \in \llbracket m \rrbracket$ is derived by checking for each test case if the configuration c satisfies the presence condition ζ of the test case (i.e., $c \in \llbracket \zeta \rrbracket$). If this is the case, every parameter of the test case is replaced by the respective value of the configuration, and each parametric delay constraint is replaced by a valid delay (see Definition 5.8).

Example 5.10 (Variability-aware Test Case). Consider again the CoPTA of the PPU depicted in Figure 5.4. Here, a variability-aware test suite $T = \{t_1, t_2, t_3\}$ may consist of the following variability-aware test cases.

- $t_1 = [m \wedge \neg \text{Resume} \wedge \text{Plastic}](d \leq a \wedge d \leq b \wedge d \geq 10, \text{rotate}), (d' \leq b - d, \text{pick}), (14, \text{rotate}), (9, \text{place})$
- $t_2 = [m \wedge \text{Stop} \wedge b \geq 30](14, \text{rotate}), (16, \text{shut down})$
- $t_3 = [m \wedge \neg \text{Resume} \wedge \text{Plastic}](10, \text{rotate}), (d \leq b - 10, \text{pick}), (12, \text{rotate}), (4, \text{place}), (23, \text{wait})$

For instance, test case t_1 has variable parametric delay constraints for the first two steps as already described in Example 5.9 (i.e., the first two steps may take at most b seconds due to the invariant of location *stack*), whereas the remaining two steps have fixed delays. Furthermore, presence condition $\zeta_1 = m \wedge \neg \text{Resume} \wedge \text{Plastic}$ of t_1 denotes that t_1 is a valid test case for all TA corresponding to configurations $c \in \llbracket \zeta_1 \rrbracket$. Hence, t_1 is a valid test case for an infinite number of variants (as, e.g., the unbounded numeric parameter a is not restricted by ζ_1). Moreover, test cases t_2 and t_3 as described above are also valid test cases for an infinite number of variants. Next, consider the TA \mathcal{A} corresponding to the configuration

$$\text{PPU} \wedge \text{Workpiece} \wedge \text{Stop} \wedge \neg \text{Resume} \wedge \text{Plastic} \wedge \neg \text{Metal} \wedge a = 15 \wedge b = 30 \wedge c = 0$$

which is depicted in Figure 5.2. Here, all test cases in test suite T as described above are valid for this configuration. Hence, we derive a test suite for TA \mathcal{A} by removing the presence conditions of the test cases and by replacing all parameters by their concrete values (provided by the configuration). Finally, we compute concrete delays for each parametric delay constraint (e.g., by applying an SMT solver). As a result, we may obtain the TA test suite described in Example 5.2.

The execution of variability-aware test cases is achieved as follows. Given a CoPTA $\mathcal{C} = (L, \ell_0, \Sigma, C, P_F, P_N, I, E, m, \eta)$, we derive a TA \mathcal{A} corresponding to a configuration $c \in \llbracket m \rrbracket$. Then, we check for each variability-aware test case $t = \llbracket \zeta \rrbracket(\omega_1, \sigma_1), \dots, (\omega_n, \sigma_n) \in T$ if the test case is valid for configuration c (i.e., if $c \in \llbracket \zeta \rrbracket$). If this is the case, we can derive a TA test case from variability-aware test case t . For an example describing the derivation of a TA test suite from a variability-aware test suite, we refer the reader to Example 5.10. Furthermore, execution of TA test cases is done as described in Section 5.1.

It should be noted that we do not define a family-based notion of *passing* and *failing* variability-aware test cases. Intuitively, a variability-aware test case passes an execution if for each configuration $c \in \llbracket \zeta \rrbracket$, the corresponding test cases pass for the respective TA model. However, presence condition $\llbracket \zeta \rrbracket$ does, in general, describe an infinite number of configurations. Hence, we cannot practically execute all TA test cases.

Having defined the notions of variability-aware test cases and test suites, we proceed by lifting the notion of location coverage to CoPTA.

5.2.2 Family-based Location Coverage for CoPTA

Similar to TA test suites, a coverage criterion [190] for variability-aware test suites denotes whether a test suite is complete or whether further test cases need to be added. Hence, we generalize the notion of location coverage [116] to *family-based location coverage*. Here, a variability-aware test suite T [53] satisfies complete family-based location coverage for a CoPTA \mathcal{C} if for every location ℓ in every variant $\mathcal{A} \in \llbracket \mathcal{C} \rrbracket$, there exists a variability-aware test case $t \in T$ such that a TA test case corresponding to t reaches location ℓ in \mathcal{A} (i.e., every reachable location of *every variant* of CoPTA \mathcal{C} is covered by at least one test case). Hence, a variability-aware test suite satisfies complete family-based location coverage if every TA test suite satisfies complete location coverage for the corresponding TA. As described before, a single variability-aware test case may cover multiple locations in multiple variants.

Note, that the number of test goals (i.e., TA locations) is, in general, infinite as the number of variants of a CoPTA model may be infinite. Hence, not covering only a single location of a CoPTA means that an infinite number of test goals (i.e., TA locations) is uncovered. As a result, the coverage might equal 0, even though all other locations are covered in every variant. Here, we do not use the set of CoPTA locations as the set of test goal as not every CoPTA location is reachable in each variant. In particular, by only using CoPTA locations, we cannot express that each location of every variant should be covered by at least one test case as we do

not know the set of variants in which a CoPTA location is reachable. Moreover, we do not define family-based location coverage as a fraction (as done for TA in Definition 5.4) as we have to check coverage, in general, for an infinite number of test goals in terms TA locations. Instead, we only define *complete* family-based location coverage, and we leave the definition of a family-based coverage criterion being similar to TA location coverage (i.e., using a fraction) as a challenge for future work.

Definition 5.11 (Family-based Location Coverage). Let $\mathcal{C} = (L, \ell_0, \Sigma, C, P_F, P_N, I, E, m, \eta)$ be a CoPTA and T be a variability-aware test suite of \mathcal{C} . Then, $\text{coverage}(T, L) = 1$ if

$$\forall \mathcal{A} \in \llbracket \mathcal{C} \rrbracket_{\mathcal{C}} : \text{coverage}(T_{\mathcal{A}}, L_{\mathcal{A}}) = 1$$

where $T_{\mathcal{A}}$ is the TA test suite corresponding to \mathcal{A} and $L_{\mathcal{A}}$ is the set of locations being reachable in \mathcal{A} .

Similar to TA test suites, we use family-based coverage as a metric for *effectiveness*, such that a complete test suites is the most effective test suite. However, as the number of test goals may be infinite, an incomplete (non-empty) test suite may have an effectiveness of 0 (as described above). Furthermore, the *efficiency* of a variability-aware test suite T is the size $|T|$ of the test suite. As a result, an empty test suite could be considered as most efficient. However, to have a reasonable efficiency criterion, we consider *minimal* variability-aware test suites as most efficient. Here, it should be noted that even a single variable-aware test case may comprise an infinite number of TA test cases. Nevertheless, we use the size $|T|$ of variability-aware test suite T instead of the number of TA test cases as a metric for efficiency. In particular, we would compare sums of TA test-suite sizes which are, in general, infinite (i.e., computing $\sum_{\mathcal{A} \in \llbracket \mathcal{C} \rrbracket_{\mathcal{C}}} |T_{\mathcal{A}}|$ results, in general, in an infinite value).

Note, that family-based location coverage does not necessarily result in coverage of other basic components (i.e., switches) of a CoPTA. For instance, it might be the case that a variability-aware test suite satisfying complete location coverage does not cover all switches. Therefore, it might be necessary to lift further coverage criteria (e.g., switch coverage [54]) to CoPTA, depending on the use case.

Example 5.11 (Family-based Location Coverage). Consider the CoPTA of the PPU depicted in Figure 5.4 and the variability-aware test suite $T = \{t_1, t_2, t_3\}$ of Example 5.10. Here, test case t_2 already covers location *emergency stop* for all variants in which this location is reachable. Furthermore, test cases t_1 and t_3 together cover locations *stack*, *WP picked*, *ramp*, *WP placed*, and *idle* for all variants satisfying $m \wedge \neg \text{Resume} \wedge \text{Plastic}$. In order to achieve complete family-based location coverage, we add the following two test cases.

- $t_4 = [m \wedge \text{Stop} \wedge \text{Resume} \wedge b \geq 30](13, \text{rotate}), (17, \text{shut down}), (d \geq c, \text{reboot})$

- $t_5 = [m \wedge \neg \text{Resume} \wedge \text{Metal}](10, \text{rotate}), (10, \text{pick}), (d \geq a \wedge d \leq a, \text{rotate}), (5, \text{place}), (17, \text{wait})$

Here, t_4 covers location *idle* in all variants where the Boolean parameter *Resume* is selected. Additionally, we utilize test case t_5 to cover all locations for *Metal* workpieces. Hence, test suite $T' = \{t_1, t_2, t_3, t_4, t_5\}$ satisfies complete family-based location coverage for the CoPTA depicted in Figure 5.4.

Next, we present an algorithm for deriving a variability-aware test suite satisfying complete location coverage for CoPTA.

5.2.3 Generating Test Suites for Complete Family-based Location Coverage on CoPTA

We conclude this section by providing an algorithm for generating a variability-aware test suite satisfying complete family-based location coverage for CoPTA. To this end, Algorithm 5.2 is adapted from Bürdek et al. [53] who generate a test suite for a family-based representation of C code. Here, we apply techniques utilized by Bürdek et al. to extend Algorithm 5.1. Furthermore, we utilize a reachability check of a model checker (similar to TA test case generation as described in Algorithm 5.1) to derive variability-aware test cases. In particular, the (partial) function GETTC of our algorithm denotes the usage of the reachability check. Function GETTC receives as input a location $\ell \in L$ and a configuration constraints (symbolically describing the set of configurations in which ℓ is uncovered) and returns a valid variability-aware test case.

Definition 5.12. Let $\mathcal{C} = (L, \ell_0, \Sigma, C, P_F, P_N, I, E, m, \eta)$ be a CoPTA. Then,

$$\text{GETTC} : L \times \mathcal{B}(P_F, P_N) \rightarrow \mathcal{B}(P_F, P_N) \times (\mathcal{B}(\Delta) \times \Sigma)^*$$

returns a featured parametric timed trace $t = [\xi](\omega_1, \sigma_1), \dots, (\omega_n, \sigma_n)$ such that $\emptyset \subset \llbracket \xi \rrbracket \subseteq \llbracket m \rrbracket$ and each TA $\mathcal{A} \in \llbracket \mathcal{C} \rrbracket_{\mathcal{C}}$ corresponding to a configuration $c \in \llbracket \xi \rrbracket$ contains a run

$$\langle \ell_0, u_0 \rangle \xrightarrow{(d_1, \sigma_1), \dots, (d_n, \sigma_n)}$$

such that $d_i \in \omega_i$ with $1 \leq i \leq n$.

Algorithm 5.2 takes as input a CoPTA \mathcal{C} and returns a variability-aware test suite T satisfying complete family-based location coverage for \mathcal{C} . Similar to TA test-suite generation, we start by initializing our data structures. Here, test suite T is initially empty (see line 2), and cover set CS is set to EFM m for each location $\ell \in L$ (lines 3–4). The cover set symbolically denotes for each location $\ell \in L$ the set of configurations in which ℓ is not yet covered.

In the main loop of Algorithm 5.2, we generate variability-aware test cases until we obtain a test suite satisfying complete location coverage (lines 5–13). To this end, we repeat the following steps for each location $\ell \in L$ until this location is covered in all variants, i.e., until $CS[\ell] = \text{false}$ (lines 6–13). Please note, that the comparison to false (line 6) is not a syntactic check. Instead, we utilize this notation as a shorthand for checking satisfiability. First, we use the current location ℓ and

Algorithm 5.2: Test-Suite Generation Satisfying Complete Location Coverage for CoPTA (Adapted from Bürdek et al. [53])

Input : CoPTA $\mathcal{C} = (L, \ell_0, \Sigma, C, P_F, P_N, I, E, m, \eta)$

Output: family-based test suite T

```

1 procedure MAIN
2    $T := \emptyset$ 
3   foreach  $\ell \in L$  do
4      $CS[\ell] := m$ 
5   foreach  $\ell \in L$  do
6     while  $CS[\ell] \neq \text{false}$  do
7        $(\zeta, \rho = \langle \ell_0, u_0 \rangle \xrightarrow{t = (\omega_1, \sigma_1), \dots, (\omega_n, \sigma_n)} \langle \ell_n, u_n \rangle) := \text{GETTC}(\ell, CS[\ell])$ 
8       if  $\zeta \neq \text{false}$  then
9          $T := T \cup \{(t, \zeta)\}$ 
10        foreach  $\ell_i$  with  $0 \leq i \leq n$  do
11           $CS[\ell_i] := CS[\ell_i] \wedge \neg \zeta$ 
12        else
13           $CS[\ell] := \text{false}$ 
14    return  $T$ 

```

its cover set $CS[\ell]$ as input for a model checker (i.e., partial function GETTC as described in Definition 5.12) and perform a reachability query (line 7). As a result, the model checker returns a test case t for reaching location ℓ and a presence condition ζ , symbolically describing the set of variants for which the test case is valid (see below for an explanation of what the model checker does internally for deriving test cases). Next, we check whether the model checker returned a test case or location ℓ is unreachable for configurations described by cover set $CS[\ell]$ (line 8). If ℓ is unreachable in $CS[\ell]$, we mark ℓ as covered and proceed with the next location (lines 12–13). Here, it should be noted that we utilize $\zeta \neq \text{false}$ (line 8) again as a shorthand for checking satisfiability. Note, that this step is necessary as location ℓ might be unreachable even if $CS[\ell] \neq \text{false}$ (as not all locations are reachable in every variant). Otherwise, we add the generated test case t with presence condition ζ to our test suite T (line 9). Thereafter, we update the cover set for each location being reached by test case t such that we utilize reuse of test cases between variants *and locations* (lines 10–11). In particular, we conjugate the current cover set with the *negation* of presence condition ζ such that subsequent calls of the model checker do not return redundant test cases (line 11). This is called *blocking-clause method* [53] as it blocks a test goal for variants for which this particular test goal is already covered.

Afterwards, we repeat the while-loop (lines 6–13) until the current location ℓ is covered in all variants in which it is reachable. Thereafter, we proceed with the outer for-each-loop (lines 5–13) to generate test cases for further locations. Finally, we return variability-aware test suite T satisfying complete location coverage (line 14).

Internally, the model checker performs a reachability analysis when we call GETTC in line 8. In order to check reachability for location ℓ of CoPTA \mathcal{C} , the model

checker first generates the featured parametric zone graph (see Section 4.2 of the previous chapter). Then, the model checker derives an untimed trace from the zone graph corresponding to path (i.e., a sequence of symbolic transitions) reaching a symbolic state $\langle \ell, \gamma \rangle$, comprising target location ℓ and featured parametric zone γ . Here, we derive the presence condition ζ for our test case by eliminating all clock variables from γ .¹ However, note that checking reachability (and thus, generating test cases) is semi-decidable for CoPTA as featured parametric zone graphs do, in general, not have a finite state space (see Section 4.2). Hence, we obtain a correct and precise result if the decision procedure terminates (but the procedure does, in general, not terminate).

Example 5.12 (Family-based Test-Suite Generation for CoPTA). Consider the CoPTA $\mathcal{C} = (L, \ell_0, \Sigma, C, P_F, P_N, I, E, m, \eta)$ depicted in Figure 5.4. When applying Algorithm 5.2 for family-based test-suite generation for location coverage, we start by initializing test suite $T = \emptyset$ and setting the cover set to EFM m for each location $\ell \in L$ (lines 2–4). Here, Table 5.2 gives an overview on the development of test suite T and the uncovered configurations for every location after each iteration of the algorithm. As a next step, we iterate over all locations (line 5). For instance, we generate a test case for location *emergency stop*. As $\text{CS}[\text{emergency stop}] \neq \text{false}$ (line 6), we call the model checker to generate a test case for this location (line 7). As described above, this comparison to false is a shorthand for checking satisfiability. Internally, the model checker then generates the featured parametric zone graph described in Chapter 4 which we repeat in Figure 5.6 for the convenience of the reader. As a result, we may obtain the test case

$$t_2 = [m \wedge \text{Stop} \wedge b \geq 30](14, \text{rotate}), (16, \text{shut down})$$

as described in Example 5.10. Here, we obtain the presence condition by eliminating the clock variables in the symbolic state comprising location *emergency stop*. As it holds that presence condition $\zeta \neq \text{false}$, we add t_2 to test suite T such that $T = \{t_2\}$ (line 9). Additionally, we update the cover set of all locations visited by test case t_2 (lines 10–11). Hence, we obtain the cover set

$$m \wedge \neg(\text{Stop} \wedge b \geq 30)$$

for locations *emergency stop*, *stack*, and *init* to block the model checker from generating redundant test cases. As $m \wedge \neg(\text{Stop} \wedge b \geq 30) \neq \text{false}$ (line 6), we generate another test case for *emergency stop* by calling the model checker again (line 7). However, as *emergency stop* is not reachable anymore in configurations satisfying $m \wedge \neg(\text{Stop} \wedge b \geq 30)$, we obtain $\zeta = \text{false}$ (line 8) and mark this location as covered in all variants where it is reachable (line 12–13). When continuing with test-case generation, we may obtain variability-aware test suite $T = \{t_1, t_2, t_3, t_4, t_5\}$ satisfying complete family-based location coverage as described in Example 5.11.

¹ For instance, this can be achieved by applying *Fourier-Motzkin elimination* [174].

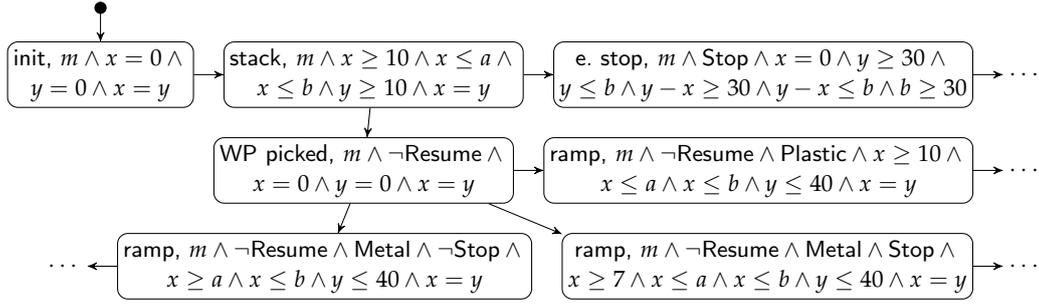


Figure 5.6: Extract of the Featured Parametric Zone Graph of the CoPTA in Figure 5.4 (Copy of Figure 4.3)

We conclude this section by showing that a variability-aware test suite T obtained by applying Algorithm 5.2 to CoPTA \mathcal{C} is sound and complete. In particular, we start by proving soundness, where test suite T is sound if every variability-aware test case $[\xi](\omega_1, \sigma_1), \dots, (\omega_n, \sigma_n) \in T$ corresponding to configuration c (i.e., $c \in \xi$) yields a valid test case for the respective TA.

Theorem 5.1. Let $\mathcal{C} = (L, \ell_0, \Sigma, C, P_F, P_N, I, E, m, \eta)$ be a CoPTA and T be the variability-aware test suite obtained by applying Algorithm 5.2 to \mathcal{C} . Then, for all configurations $c \in \llbracket m \rrbracket$ it holds that

$$\forall [\xi](\omega_1, \sigma_1), \dots, (\omega_n, \sigma_n) \in T : c \in \llbracket \xi \rrbracket \Rightarrow \langle \ell_0, u_0 \rangle \xrightarrow{(d_1, \sigma_1), \dots, (d_n, \sigma_n)} \in \llbracket \mathcal{A} \rrbracket_R$$

where \mathcal{A} is the TA corresponding to c and $d_i \in \omega_i$ with $1 \leq i \leq n$ is a delay corresponding to configuration c .

Proof. Let $\mathcal{C} = (L, \ell_0, \Sigma, C, P_F, P_N, I, E, m, \eta)$ be a CoPTA and T be the variability-aware test suite obtained by applying Algorithm 5.2 to \mathcal{C} . For soundness of test cases $[\xi](\omega_1, \sigma_1), \dots, (\omega_n, \sigma_n) \in T$, we rely on the correctness of the model checker called in line 7 of Algorithm 5.2, which internally computes presence condition ξ as well as featured parametric timed trace $(\omega_1, \sigma_1), \dots, (\omega_n, \sigma_n)$. If implemented correctly to satisfy the requirements we impose on function GETTC (see Definition 5.12), the model checker internally generates a featured parametric zone graph (see Definition 4.7). In particular, the model checker uses the featured parametric zone of the symbolic states as presence condition. As described in Chapter 4, the featured parametric zone only describes configurations in which the current location is reachable with the current trace. Hence, for all configurations $c \in \llbracket m \rrbracket$ it holds that

$$\forall [\xi](\omega_1, \sigma_1), \dots, (\omega_n, \sigma_n) \in T : c \in \llbracket \xi \rrbracket \Rightarrow \langle \ell_0, u_0 \rangle \xrightarrow{(d_1, \sigma_1), \dots, (d_n, \sigma_n)} \in \llbracket \mathcal{A} \rrbracket_R$$

where \mathcal{A} is the TA corresponding to c and $d_i \in \omega_i$ with $1 \leq i \leq n$ is the delay corresponding to configuration c . \square

Next, we prove completeness. In particular, we show that variability-aware test suite T obtained by applying Algorithm 5.2 to CoPTA \mathcal{C} does, in fact, satisfy com-

plete family-based location coverage. To this end, we require $\text{coverage}(T_{\mathcal{A}}, L_{\mathcal{A}}) = 1$ for each test suite $T_{\mathcal{A}}$ and set of locations $L_{\mathcal{A}}$ corresponding to a TA $\mathcal{A} \in \llbracket \mathcal{C} \rrbracket_{\mathcal{C}}$. It should be noted that $\text{coverage}(T) = 1$ for test suite T is also possible in the presence of unreachable locations as these locations are excluded when we calculate location coverage (see Definition 5.4).

Theorem 5.2. Let $\mathcal{C} = (L, \ell_0, \Sigma, C, P_F, P_N, I, E, m, \eta)$ be a CoPTA and T be the variability-aware test suite obtained by applying Algorithm 5.2 to \mathcal{C} . Then,

$$\forall \mathcal{A} \in \llbracket \mathcal{C} \rrbracket_{\mathcal{C}} : \text{coverage}(T_{\mathcal{A}}, L_{\mathcal{A}}) = 1$$

where $T_{\mathcal{A}}$ is the TA test suite corresponding to \mathcal{A} and $L_{\mathcal{A}}$ is the set of locations being reachable in \mathcal{A} .

Proof. Let $\mathcal{C} = (L, \ell_0, \Sigma, C, P_F, P_N, I, E, m, \eta)$ be a CoPTA and T be the variability-aware test suite obtained by applying Algorithm 5.2 to \mathcal{C} . We prove Theorem 5.2 by induction.

Induction basis: Initially, test suite $T = \emptyset$ (line 2) covers the set of test goals $\text{CS}[\ell] = \text{false}$ for each test goal $\ell \in L$ (lines 3–4).

Induction step: After initializing test suite T and the set of uncovered test goals $\text{CS}[\ell]$, the algorithm iterates over all locations $\ell \in L$ (lines 5–13). Here, the algorithm generates variability-aware test cases t for the current location ℓ (lines 6–13). Furthermore, as a result of partial function `GETTC` (see Definition 5.12), the generated test cases t also covers all intermediate locations for all configurations satisfying its presence condition (lines 10–11). Hence, for a test case $t = [\xi](\omega_1, \sigma_1), \dots, (\omega_n, \sigma_n)$ it holds that $T \cup \{t\}$ covers $\text{CS}[\ell] \vee \xi$ for each $\ell \in L$ visited by t . As these steps are repeated until every $\ell \in L$ is covered in all configurations in which it is reachable (line 6), the algorithm returns a variability-aware test suite T , such that $\forall \mathcal{A} \in \llbracket \mathcal{C} \rrbracket_{\mathcal{C}} : \text{coverage}(T_{\mathcal{A}}, L_{\mathcal{A}}) = 1$ where $T_{\mathcal{A}}$ is the TA test suite corresponding to TA \mathcal{A} . \square

In this section, we defined the notions of variability-aware test cases and variability-aware test suites, and we lifted the notion of location coverage to CoPTA. Furthermore, we described an algorithm to derive a variability-aware test suite satisfying complete family-based location coverage for a CoPTA. Therewith, we are now able to effectively generate complete test suites for CoPTA models. We achieve this by augmenting test cases with presence conditions symbolically describing a (possibly infinite) set of variants for which a test case is valid. As we derive these test cases directly from CoPTA models, we can generate a finite test suite even in case of CoPTA comprising an infinite number of variants. In the next section, we utilize the M/MD instrumentation described in Chapter 4 to generate test cases reaching locations with BCET/WCET. Therewith, we generate test cases covering boundary cases, which have a higher chance of revealing faults.

5.3 FAMILY-BASED TEST COVERAGE OF BEST-CASE/WORST-CASE EXECUTION-TIME BEHAVIOR

In this section, we adapt our approach for test-case generation by specifically generating test cases having BCET/WCET for reaching the test goals (i.e., locations). In particular, we are interested in finding two test cases for each CoPTA location, one having the overall BCET for reaching the location and one having the overall WCET for reaching the location. These test cases having minimum or maximum delay for reaching locations have a higher chance of revealing faults (e.g., off-by-one timing errors) [203]. Here, each test case is variability-aware (i.e., contains a presence condition), such that we know to which variants the generated test cases are applicable. Hence, we do not generate test cases with BCET/WCET for each variant of a CoPTA model but only for variants having the overall fastest and slowest runs for reaching a location as compared to all other variants of the given CoPTA model. We refer to this approach by *minimum/maximum delay (M/MD) test-case generation*. Similar to the family-based approach presented in the previous section, M/MD test-case generation based on CoPTA models is incomplete. As described in the previous section, we utilize reachability queries to generate test cases, which are already incomplete for PTA [20, 21]. Hence, generating test cases based on CoPTA models may not terminate, but if the generation terminates, then the result is a finite complete test suite.

Example 5.13. Consider the CoPTA model of the PPU presented in Figure 5.4 and the faulty implementation depicted in Figure 4.2c, which we repeat in Figure 5.7 for the convenience of the reader. Here, the implementation is faulty as the switch labeled with action *shut down* has the guard $x \leq 31$ as opposed to $x \leq 30$ in the specification (i.e., the CoPTA model in Figure 5.4). A possible M/MD test case revealing this fault is the following:

1. Wait for 10 seconds and *rotate* the crane.
2. Wait for another 20 seconds and *shut down* the PPU.

This test case is allowed by the specification but fails in the implementation as we have a clock value $x = 30$ when shutting down the PPU. Furthermore, this test case has the presence condition

$$m \wedge \text{Stop} \wedge b \geq 30$$

as it is valid for all configurations having selected the feature *Stop*. Note, that family-based test-case generation for (complete) location coverage as presented in Section 5.2 does not have any guarantees for generating test cases for boundary behavior. Hence, the fault illustrated in this example would probably not be revealed by applying a variability-aware test suite.

We start by introducing and formally defining the notion of M/MD test cases and M/MD test suites, respectively.

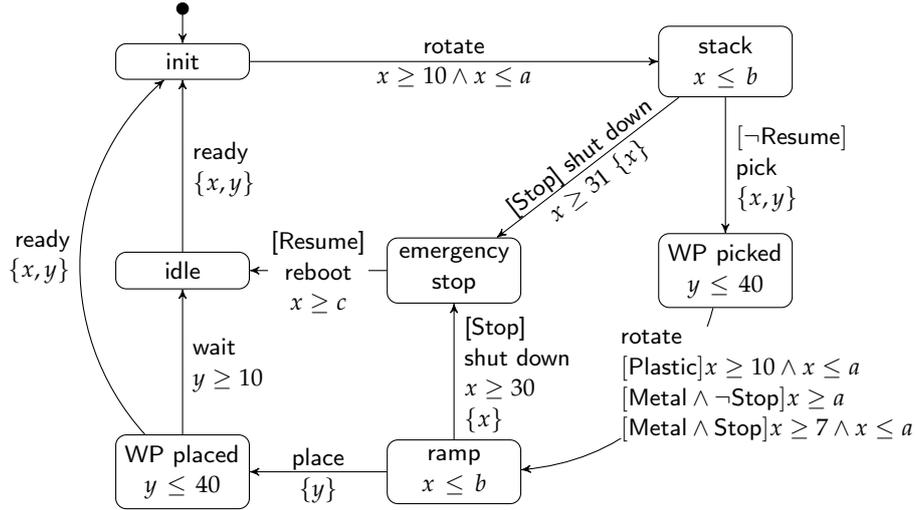


Figure 5.7: Faulty Implementation of the CoPTA depicted in Figure 5.4 (Copy of Figure 4.2c)

5.3.1 M/MD Test Cases for CoPTA

Similar to variability-aware test cases (see Definition 5.9), an M/MD test case consists of a presence condition ζ and a featured parametric timed trace t . Here, the presence condition ζ describes the (possibly infinite) set of configurations in which the M/MD test case is valid. Although we are only interested in finding the overall fastest (or slowest) behavior for reaching each test goal (i.e., location), we still have a presence condition as multiple variants may have the same BCET/WCET for reaching particular test goals. Additionally, the presence condition is needed as a test case may not be valid for all configurations (as described in Section 5.2). Furthermore, featured parametric trace t only has specific delays d instead of variable parametric delay constraints ω as the BCET (or WCET) is a fixed value instead of being variable. Please note, that an M/MD test case as defined below does, in general, not describe a trace reaching a location with BCET (or WCET). Instead, the intention of Definition 5.13 is only to ensure well-formedness of M/MD test cases. Reaching locations with BCET/WCET is ensured by the definition of M/MD coverage as defined later in this section.

Definition 5.13 (M/MD Test Case). Let $\mathcal{C} = (L, \ell_0, \Sigma, C, P_F, P_N, I, E, m, \eta)$ be a CoPTA. Then, a finite featured parametric timed trace

$$t \in \mathcal{B}(P_F, P_N) \times (\mathbb{T}_C \times \Sigma)^*$$

with $t = [\zeta](d_1, \sigma_1), \dots, (d_n, \sigma_n)$ and $\emptyset \subset \llbracket \zeta \rrbracket \subseteq \llbracket m \rrbracket$ is an M/MD test case where $\zeta \in \mathcal{B}(P_F, P_N)$ is called a *presence condition*. M/MD test case t is *valid* w.r.t. \mathcal{C} iff each TA $\mathcal{A} \in \llbracket \mathcal{C} \rrbracket_{\mathcal{C}}$ corresponding to a configuration $c \in \llbracket \zeta \rrbracket$ contains run

$$\langle \ell_0, u_0 \rangle \xrightarrow{(d_1, \sigma_1), \dots, (d_n, \sigma_n)}.$$

It should be noted that we do not distinguish abstract from concrete M/MD test cases (as done for TA test cases) as we do not execute these test cases. Instead, only TA test cases derived from these M/MD test cases are executed. Moreover, a finite set of M/MD test cases is called an *M/MD test suite*.

Definition 5.14 (M/MD Test Suite). Let $T \subseteq \mathcal{B}(P_F, P_N) \times (\mathbb{T}_C \times \Sigma)^*$ be a finite set of M/MD test cases w.r.t. CoPTA \mathcal{C} . In the context of testing, we call T an *M/MD test suite*. T is *valid* if every $t \in T$ is valid.

Note, that we may refer to M/MD test cases and M/MD test suites simply by test cases and test suites if this is clear from the context.

Example 5.14 (M/MD Test Case). Consider the CoPTA model depicted in Figure 5.4 and the test case informally described in Example 5.13. Here, the corresponding M/MD test case is

$$t_1 = [m \wedge \text{Stop} \wedge b \geq 30](10, \text{rotate}), (20, \text{shut down})$$

which reaches locations *stack* and *emergency stop* with BCET as it is impossible to reach *stack* in less than 10 seconds and *emergency stop* in less than 30 seconds. Here, t_1 is valid for all configurations satisfying $m \wedge \text{Stop} \wedge b \geq 30$. An additional M/MD test case is described by

$$t_2 = [m \wedge \text{Plastic} \wedge a \geq 30 \wedge b \geq 30](30, \text{rotate})$$

which has WCET for reaching location *stack*. As a result, $T = \{t_1, t_2\}$ is an M/MD test suite.

Next, we utilize M/MD test cases to achieve M/MD coverage, such that we are able to decide whether an M/MD test suite is complete.

5.3.2 M/MD Coverage for CoPTA

Having defined M/MD test cases, we next adapt the definition of *minimum/maximum delay (M/MD) coverage* as considered for sampling (see Definition 4.12) to M/MD test-case generation. Recall, that for sampling we derive configuration constraints describing variants with BCET/WCET behavior. In contrast, M/MD coverage for test-case generation consists of such constraints *as well as* featured parametric timed traces leading to the test goals *with BCET/WCET*. Hence, a test suite satisfying complete M/MD coverage in the context of this chapter extends the notion of M/MD coverage of sampling by timed runs for reaching the test goals. Here, we define *minimum-delay coverage* and *maximum-delay coverage* separately, and we start by defining minimum-delay coverage.

First, we define $\min(T)$ for an M/MD test suite T . In particular, $\min(T)$ describes the set of locations being reached with minimum delay by M/MD test cases $t \in T$. As a result, it holds that $\ell \in \min(T)$ for CoPTA \mathcal{C} if there exists a test case $t \in T$ such that no run of any variant $\mathcal{A} \in \llbracket \mathcal{C} \rrbracket_{\mathcal{C}}$ reaches location ℓ faster than t .

Definition 5.15. Let $\mathcal{C} = (L, \ell_0, \Sigma, C, P_F, P_N, I, E, m, \eta)$ be a CoPTA and T be an M/MD test suite of \mathcal{C} . We use $\min(T) \subseteq L$ to denote the set of locations $\ell \in L$ being reached with minimum delay. Then, $\ell \in \min(T)$ if there exists a $t = [\xi](d_1, \sigma_1), \dots, (d_n, \sigma_n) \in T$ with $\mathcal{A} \in \llbracket \mathcal{C} \rrbracket_{\mathcal{C}}$ corresponding to ξ such that

$$\exists \rho_\ell \in \llbracket \mathcal{A} \rrbracket_R : (\forall \mathcal{A}' \in \llbracket \mathcal{C} \rrbracket_{\mathcal{C}} : (\forall \rho'_\ell \in \llbracket \mathcal{A}' \rrbracket_R : (d(\rho_\ell) \leq d(\rho'_\ell))))$$

where $\rho_\ell = \langle \ell_0, u_0 \rangle \xrightarrow{(d_1, \sigma_1), \dots, (d_n, \sigma_n)} \langle \ell, u_n \rangle$, $\rho'_\ell = \langle \ell_0, u_0 \rangle \xrightarrow{(d'_1, \sigma'_1), \dots, (d'_n, \sigma'_n)} \langle \ell, u'_n \rangle$.

Having defined the set of locations being reached with minimum delay by test cases in M/MD test suite T , we now proceed by defining minimum-delay coverage. In particular, minimum-delay coverage describes the ratio between locations reached with minimum delay and the number of (reachable) locations of a CoPTA.

Definition 5.16 (Minimum-Delay Coverage). Let $\mathcal{C} = (L, \ell_0, \Sigma, C, P_F, P_N, I, E, m, \eta)$ be a CoPTA, $L' \subseteq L$ be the set of reachable locations, and T be an M/MD test suite of \mathcal{C} . Then, *minimum-delay coverage* denotes the ratio

$$\text{coverage}(T, L') = \frac{|\min(T)|}{|L'|}.$$

Next, we define maximum-delay coverage. However, we have to consider that some locations do not have a finite WCET in which these locations can be reached. Here, the execution time can be arbitrarily long in case of loops allowing us to visit intermediate locations infinitely often. In these cases, a test suite T cannot contain a respective test case. Hence, we use $\text{hasMax}(\mathcal{C})$ to denote the set of locations of CoPTA \mathcal{C} which can be reached with a finite WCET. It should be noted that we only consider the WCET for reaching location ℓ for the first time as ℓ might be reached several times by a test case in the presence of loops (i.e., we require $\ell \notin \{\ell_0, \dots, \ell_{n-1}\}$).

Definition 5.17. Let $\mathcal{C} = (L, \ell_0, \Sigma, C, P_F, P_N, I, E, m, \eta)$ be a CoPTA and T be an M/MD test suite of \mathcal{C} . We use $\text{hasMax}(\mathcal{C}) \subseteq L$ to denote the set of locations $\ell \in L$ having a finite maximum delay for reaching ℓ , such that $\ell \in \text{hasMax}(\mathcal{C})$ if there exists a variant $\mathcal{A} \in \llbracket \mathcal{C} \rrbracket_{\mathcal{C}}$ with run $\rho_\ell \in \llbracket \mathcal{A} \rrbracket$ satisfying

$$\forall \mathcal{A}' \in \llbracket \mathcal{C} \rrbracket_{\mathcal{C}} : (\forall \rho'_\ell \in \llbracket \mathcal{A}' \rrbracket_R : d(\rho_\ell) \geq d(\rho'_\ell))$$

where $\rho_\ell = \langle \ell_0, u_0 \rangle \xrightarrow{(d_1, \sigma_1), \dots, (d_n, \sigma_n)} \langle \ell, u_n \rangle$, $\rho'_\ell = \langle \ell_0, u_0 \rangle \xrightarrow{(d'_1, \sigma'_1), \dots, (d'_n, \sigma'_n)} \langle \ell, u'_n \rangle$, $\ell \notin \{\ell_0, \dots, \ell_{n-1}\}$, and $d(\rho_\ell)$ is a finite value.

Next, we define $\text{max}(T)$ (analogously to $\text{min}(T)$) to denote the set of locations being reached with maximum delay by M/MD test cases $t \in T$.

Definition 5.18. Let $\mathcal{C} = (L, \ell_0, \Sigma, C, P_F, P_N, I, E, m, \eta)$ be a CoPTA and T be an M/MD test suite of \mathcal{C} . We use $\text{max}(T) \subseteq L$ to denote the set of locations $\ell \in L$

being reached with maximum delay. Then, $\ell \in \max(T)$ if $\ell \in \text{hasMax}(\mathcal{C})$ and there exists a $t = [\tilde{\zeta}](d_1, \sigma_1), \dots, (d_n, \sigma_n) \in T$ with $\mathcal{A} \in \llbracket \mathcal{C} \rrbracket_{\mathcal{C}}$ corresponding to $\tilde{\zeta}$ such that

$$\exists \rho_\ell \in \llbracket \mathcal{A} \rrbracket_R : (\forall \mathcal{A}' \in \llbracket \mathcal{C} \rrbracket_{\mathcal{C}} : (\forall \rho'_\ell \in \llbracket \mathcal{A}' \rrbracket_R : (d(\rho_\ell) \geq d(\rho'_\ell))))$$

where $\rho_\ell = \langle \ell_0, u_0 \rangle \xrightarrow{(d_1, \sigma_1), \dots, (d_n, \sigma_n)} \langle \ell, u_n \rangle$, $\rho'_\ell = \langle \ell_0, u_0 \rangle \xrightarrow{(d'_1, \sigma'_1), \dots, (d'_n, \sigma'_n)} \langle \ell, u'_n \rangle$, and $\ell \notin \{\ell_0, \dots, \ell_{n-1}\}$.

Therewith, we define the notion of maximum-delay coverage as the ratio between locations covered with WCET and locations being reachable with a finite WCET.

Definition 5.19 (Maximum-Delay Coverage). Let $\mathcal{C} = (L, \ell_0, \Sigma, C, P_F, P_N, I, E, m, \eta)$ be a CoPTA and T be an M/MD test suite of \mathcal{C} . Then, *maximum-delay coverage* denotes the ratio

$$\text{coverage}(T, \text{hasMax}(\mathcal{C})) = \frac{|\max(T)|}{|\text{hasMax}(\mathcal{C})|}.$$

Finally, we consider the number of covered test goals (i.e., the sum of locations covered with BCET and WCET, respectively) and the number of test goals (i.e., the sum of reachable locations and locations being reachable with a finite WCET) for M/MD coverage.

Definition 5.20 (M/MD Coverage). Let $\mathcal{C} = (L, \ell_0, \Sigma, C, P_F, P_N, I, E, m, \eta)$ be a CoPTA, $L' \subseteq L$ be the set of reachable locations, and T be an M/MD test suite of \mathcal{C} . Then, *M/MD coverage* denotes the ratio

$$\text{coverage}(T, L' \cup \text{hasMax}(\mathcal{C})) = \frac{|\min(T)| + |\max(T)|}{|L'| + |\text{hasMax}(\mathcal{C})|}.$$

Similar to TA test suites, we use the coverage of an M/MD test suite as a metric for effectiveness, such that a complete test suite has the highest effectiveness. Furthermore, the efficiency of a test suite T is given by the size $|T|$ of the test suite. Again, to have a meaningful efficiency criterion, we consider *minimal* test suites as most efficient.

Example 5.15 (M/MD Coverage). Consider the CoPTA of the PPU depicted in Figure 5.4 and test suite $T = \{t_1, t_2\}$ as described in Example 5.14. Here, test cases t_1 and t_2 already cover locations *stack* and *emergency stop* with BCET and *stack* with WCET, respectively. Hence, we add further test cases reaching the remaining locations with BCET and WCET, respectively, to obtain a test suite satisfying M/MD coverage. In particular, test case

$$t_3 = [m \wedge \neg \text{Resume} \wedge \text{Metal} \wedge \text{Stop}](10, \text{rotate}), \\ (0, \text{pick}), (7, \text{rotate}), (0, \text{place}), (10, \text{wait})$$

reaches locations *WP picked*, *ramp*, *WP placed*, and *idle* with BCET. Hence, a test suite $T' = \{t_1, t_3\}$ already satisfies minimum-delay coverage. Moreover, test case

$$t_4 = [m \wedge \neg \text{Resume} \wedge \text{Metal} \wedge \neg \text{Stop} \wedge b \geq 40](17, \text{rotate}), \\ (23, \text{pick}), (40, \text{rotate}), (40, \text{place})$$

reaches the locations *WP picked*, *ramp*, and *WP placed* with WCET. Note, that there does not exist a finite WCET for reaching *emergency stop* and *idle* as it is possible to visit the other locations infinitely often before deciding to use the switches labeled with *shut down* and *reboot*, respectively. Hence, test suite $T'' = \{t_1, t_2, t_3, t_4\}$ satisfies M/MD coverage for the CoPTA model depicted in Figure 5.4. Moreover, note that test case t_4 does not reach location *stack* with WCET although it reaches the subsequent locations with WCET.

The execution of M/MD test cases is achieved in a similar way as described for variability-aware test cases (see Section 5.2). In particular, we choose a configuration $c \in \llbracket \xi \rrbracket$ for test case $t = [\xi](d_1, \sigma_1), \dots, (d_n, \sigma_n)$ and derive the corresponding TA \mathcal{A} . Then, we execute test case t on \mathcal{A} as described for TA (see Section 5.1).

Having defined M/MD test suites and M/MD coverage, we proceed by providing an algorithm for deriving a test suite satisfying complete M/MD coverage for a given CoPTA model.

5.3.3 Generating Test Suites for Complete M/MD Coverage on CoPTA

We conclude this section by providing an algorithm for generating a test suite satisfying complete M/MD coverage for CoPTA models. Similar to the other algorithms that we introduced for test-case generation in the previous sections, we utilize a reachability check of a model checker to derive M/MD test cases. In particular, the (partial) functions `GETMINTC` and `GETMAXTC` of our algorithm denote the usage of reachability checks with BCET and WCET, respectively. Here, function `GETMINTC` receives as input a location $\ell \in L$ and returns a minimum-delay test case t for reaching ℓ (denoted by $\ell \in \min(\{t\})$) as described in Definition 5.15). Accordingly, we define function `GETMAXTC`, where we only consider locations of CoPTA \mathcal{C} being reachable with a finite WCET (as described by `hasMax(C)`).

Definition 5.21. Let $\mathcal{C} = (L, \ell_0, \Sigma, C, P_F, P_N, I, E, m, \eta)$ be a CoPTA. Then,

$$\text{GETMINTC} : L \rightarrow \mathcal{B}(P_F, P_N) \times (\mathbb{T}_C \times \Sigma)^*$$

returns a featured parametric timed trace $t = [\xi](d_1, \sigma_1), \dots, (d_n, \sigma_n)$ for reaching location $\ell \in L$ such that $\ell \in \min(\{t\})$, and

$$\text{GETMAXTC} : \text{hasMax}(\mathcal{C}) \rightarrow \mathcal{B}(P_F, P_N) \times (\mathbb{T}_C \times \Sigma)^*$$

returns a featured parametric timed trace $t' = [\xi'](d'_1, \sigma'_1), \dots, (d'_n, \sigma'_n)$ for reaching location $\ell' \in \text{hasMax}(\mathcal{C})$ such that $\ell' \in \max(\{t'\})$.

Algorithm 5.3: Test-Suite Generation Satisfying Complete M/MD Coverage for CoPTA

Input : CoPTA $\mathcal{C} = (L, \ell_0, \Sigma, C, P_F, P_N, I, E, m, \eta)$
Output: M/MD test suite T

```

1 procedure MAIN
2    $T := \emptyset$ 
3    $G_{min} := L$ 
4    $G_{max} := L$ 
5   while  $G_{min} \neq \emptyset$  do
6      $(\xi, \langle \ell_0, u_0 \rangle \xrightarrow{t = (d_1, \sigma_1), \dots, (d_n, \sigma_n)} \langle \ell_n, u_n \rangle) := \text{GETMINTC}(\ell_n \in G_{min})$ 
7      $T := T \cup \{(t, \xi)\}$ 
8     foreach  $\ell_i \in \{\ell_0, \dots, \ell_n\} \cap G_{min}$  with  $0 \leq i \leq n$  do
9        $(\xi', \langle \ell_0, u_0 \rangle \xrightarrow{t' = (d'_1, \sigma'_1), \dots, (d'_i, \sigma'_i)} \langle \ell_i, u'_i \rangle) := \text{GETMINTC}(\ell_i)$ 
10      if  $d_1 + \dots + d_i \leq d'_1 + \dots + d'_i$  then
11         $G_{min} := G_{min} \setminus \{\ell_i\}$ 
12  while  $G_{max} \neq \emptyset$  do
13     $(\xi, \langle \ell_0, u_0 \rangle \xrightarrow{t = (d_1, \sigma_1), \dots, (d_n, \sigma_n)} \langle \ell_n, u_n \rangle) := \text{GETMAXTC}(\ell_n \in G_{max})$ 
14    if  $\xi \neq \text{false}$  then
15       $T := T \cup \{(t, \xi)\}$ 
16      foreach  $\ell \in \{\ell_0, \dots, \ell_n\} \cap G_{max}$  with  $0 \leq i \leq n$  do
17         $(\xi', \langle \ell_0, u_0 \rangle \xrightarrow{t' = (d'_1, \sigma'_1), \dots, (d'_i, \sigma'_i)} \langle \ell_i, u'_i \rangle) := \text{GETMAXTC}(\ell_i)$ 
18        if  $d_1 + \dots + d_i \geq d'_1 + \dots + d'_i$  then
19           $G_{max} := G_{max} \setminus \{\ell_i\}$ 
20    else
21       $G_{max} := G_{max} \setminus \{\ell_n\}$ 
22  return  $T$ 

```

Algorithm 5.3 adapts techniques from Algorithms 5.1 and 5.2 for M/MD test-suite generation. Here, we start by initializing test suite T (with the empty set) and the sets of test goals G_{min} and G_{max} (with the set of locations L) for minimum-delay coverage and maximum-delay coverage, respectively (see lines 2–4). Thereafter, we proceed with two almost identical while-loops for generating test cases with minimum delay (lines 5–11) and maximum delay (lines 12–21), respectively.

In particular, we generate minimum-delay test cases as follows. As long as the set of test goals G_{min} is not empty (line 5), we utilize a model checker to generate an M/MD test case t having presence condition ξ with minimum delay for a randomly selected location $\ell_n \in G_{min}$ (line 6). Internally, the model checker uses the M/MD instrumentation introduced in Chapter 4 (see Definition 4.13) to find a run with minimum delay reaching location ℓ_n . Here, we introduce a fresh clock χ_ℓ and a fresh parameter $\pi_{\ell_{\text{MIN}}}$ for test goal ℓ . Then, we find fastest runs for reaching ℓ by finding the minimum value for $\pi_{\ell_{\text{MIN}}}$ such that ℓ is still reachable. To this end, we replace $\pi_{\ell_{\text{MIN}}}$ by the respective minimum value and utilize featured parametric zone graphs (see Definition 4.7) as described in Chapter 4 to find a test case having BCET for reaching location ℓ . Afterwards, we add the resulting test case to our test

suite T (see line 7). However, it should be noted that checking reachability (and thus, generating test cases) is semi-decidable for CoPTA as featured parametric zone graphs do, in general, not have a finite state space (see Section 4.2). Hence, we obtain a correct and precise result if the decision procedure terminates (but the procedure does, in general, not terminate).

In contrast to the previous algorithms of this chapter, test case t does not necessarily cover all intermediate locations with minimum delay (e.g., see Example 5.15). Hence, we check for all uncovered locations reached by t (i.e., all locations in $\{\ell_0, \dots, \ell_n\} \cap G_{min}$) whether test case t also has minimum delay for reaching these locations (lines 8–11). To this end, we generate a minimum-delay test case t' for each intermediate location (line 9) and check whether t' has a longer delay (line 10). If this is the case, we remove the intermediate location from the set of test goals G_{min} (line 11). Here, we could optimize Algorithm 5.3 by also directly adding test case t' to our test suite T if it has a smaller delay than test case t . Therewith, we would avoid an additional iteration of the while-loop (lines 5–11). However, we omit this optimization from our algorithm for the sake of readability.

Moreover, the while-loop for generating maximum-delay test case (lines 12–21) is almost identical to the part for generating minimum-delay test cases. The only exception is the following. As there does not always exist a finite maximum delay for reaching a location ℓ (e.g., see Example 5.15), we add an additional check. In particular, the model checker returns a presence condition $\zeta = \text{false}$ if there does not exist a finite maximum delay (line 13). Hence, we only add the generated test case if $\zeta \neq \text{false}$ (line 14) and remove the target location from the set of test goals G_{max} , otherwise (lines 20–21). Finally, we return test suite T satisfying complete location coverage (line 22).

Example 5.16 (M/MD Test-Suite Generation for CoPTA). Consider CoPTA $\mathcal{C} = (L, \ell_0, \Sigma, C, P_F, P_N, I, E, m, \eta)$ of the PPU depicted in Figure 5.4. Here, we utilize Algorithm 5.3 to generate a test suite satisfying complete M/MD coverage for \mathcal{C} . Furthermore, Table 5.3 gives an overview on the development of test suite T and the set of remaining uncovered test goals G_{min} and G_{max} for each iteration of the algorithm. We start by initializing test suite $T := \emptyset$, the set of test goals $G_{min} := L$ for minimum-delay coverage, and the set of test goals $G_{max} := L$ for maximum-delay coverage (see lines 2–4). Next, we generate minimum-delay test cases for all locations, i.e., until $G_{min} = \emptyset$ (lines 5–11). Here, we start by randomly picking a location $\ell_n \in G_{min}$ and querying the model checker for an M/MD test case reaching ℓ_n with BCET (line 6). For instance, querying the model checker for a test case for location *emergency stop* may result in test case

$$t_1 = [m \wedge \text{Stop} \wedge b \geq 30](10, \text{rotate}), (20, \text{shut down})$$

as presented in Example 5.14. Here, test case t_1 covers *emergency stop* as well as *stack* with BCET such that we remove these two locations from the set of test goals G_{min} . Next, we check which locations are covered with BCET by test

Table 5.3: Development of Test Suite T and Remaining Uncovered Test Goals G_{min} and G_{max} in Example 5.16

Iteration	Test Suite T	Remaining Uncovered Test Goals G_{min}	Remaining Uncovered Test Goals G_{max}
initial	\emptyset	{init, stack, e. stop, WP picked, ramp, WP placed, idle}	{init, stack, WP picked, ramp, WP placed}
1	$\{t_1\}$	{WP picked, ramp, WP placed, idle}	{stack, WP picked, ramp, WP placed}
2	$\{t_1, t_2\}$	{WP picked, ramp, WP placed, idle}	{WP picked, ramp, WP placed}
3	$\{t_1, t_2, t_3\}$	\emptyset	{WP picked, ramp, WP placed}
4	$\{t_1, t_2, t_3, t_4\}$	\emptyset	\emptyset

case t_1 (lines 8–11). As it still holds that $G_{min} \neq \emptyset$ (line 5), we generate another test case. For instance, we may generate test case

$$t_3 = [m \wedge \neg \text{Resume} \wedge \text{Metal} \wedge \text{Stop}](10, \text{rotate}), \\ (0, \text{pick}), (7, \text{rotate}), (0, \text{place}), (10, \text{wait})$$

as described in Example 5.20. Therewith, we cover the remaining locations in G_{min} such that the intermediate test suite $T' = \{t_1, t_3\}$ satisfies minimum-delay coverage.

As a next step, we proceed to generate test cases with WCET for reaching the test goals in G_{max} (lines 12–21). The procedure for this is almost identical to the procedure described above for minimum-delay test cases and may result in the test cases t_2 and t_4 described as described in Examples 5.14 and 5.15, respectively. The only difference is that not all locations can be reached with a finite WCET. For instance, there does not exist a finite WCET for reaching *emergency stop* and *idle* as it is possible to visit the other locations infinitely often before deciding to use the switches labeled with *shut down* and *reboot*, respectively. Hence, Algorithm 5.3 does not generate maximum-delay test cases for these locations. For instance, querying the model checker for a test case for location *idle* results in a presence condition $\zeta = \text{false}$ (line 13). Hence, the if-statement evaluates to false (line 14) and we remove location *idle* from the set of test goals G_{max} (lines 20–21). Finally, Algorithm 5.3 returns test suite $T'' = \{t_1, t_2, t_3, t_4\}$ satisfying complete M/MD coverage as described in Example 5.15 (line 22).

We conclude this section by showing that Algorithm 5.3 does, in fact, generate a test suite satisfying complete M/MD coverage for a given CoPTA \mathcal{C} . Note, that we do not prove soundness of M/MD test cases as these test cases are generated in the same way as done for variability-aware test cases in the previous section

(i.e., by utilizing a model checker). Hence, the result presented in Theorem 5.1 also applies to M/MD test cases.

Theorem 5.3. Let $\mathcal{C} = (L, \ell_0, \Sigma, C, P_F, P_N, I, E, m, \eta)$ be a CoPTA, $L' \subseteq L$ be the set of reachable locations, and T be the test suite obtained by applying Algorithm 5.3 to \mathcal{C} . Then, it holds that

$$\text{coverage}(T, L' \cup \text{hasMax}(\mathcal{C})) = 1.$$

Proof. Let $\mathcal{C} = (L, \ell_0, \Sigma, C, P_F, P_N, I, E, m, \eta)$ be a CoPTA, $L' \subseteq L$ be the set of reachable locations, and T be the test suite obtained by applying Algorithm 5.3 to \mathcal{C} . We prove Theorem 5.3 by induction.

Induction basis: Initially, test suite $T = \emptyset$ (line 2) covers the set of test goals $G_{min} = \emptyset$ and $G_{max} = \emptyset$ (line 3).

Induction step: First, the algorithm iterates over all locations $\ell \in G_{min}$ for minimum-delay coverage (lines 5–11). Here, a test case t is generated for the current location (line 6), and intermediate locations are only removed from test goals G_{min} if the current test case t also has the BCET for reaching these locations (lines 8–11). To this end, GETMINTC (see Definition 5.21) guarantees that we reach locations with BCET. Hence, test suite $T \cup \{t\}$ covers $G_{min} \cup G'$, where G' is the set of minimum-delay test goals covered by t . As a result, test suite T satisfies minimum-delay coverage. Furthermore, the algorithm repeats the same procedure for maximum-delay coverage (lines 12–21), such that the algorithm returns an M/MD test suite T where $\text{coverage}(T, L' \cup \text{hasMax}(\mathcal{C})) = 1$ (where GETMAXTC as described in Definition 5.21 guarantees that we reach locations $\ell \in \text{hasMax}(\mathcal{C})$ with WCET). \square

To summarize this section, we started by recalling notions for minimum delays and maximum delays from M/MD sampling, and adapted these notions to M/MD test-case generation. In particular, M/MD test cases consist of a presence and a timed trace. Here, a test suite T for CoPTA \mathcal{C} satisfies complete M/MD coverage if for each location $\ell \in L$, there is a test case $t \in T$ with presence condition ξ and delay d such that every TA corresponding to ξ reaches ℓ in d seconds, and it is impossible to reach ℓ in less (or more) than d seconds in any variant of CoPTA \mathcal{C} . Finally, we presented an algorithm for deriving complete M/MD test suites from CoPTA models. In the next section, we proceed by providing implementation details and an evaluation of the approaches introduced in this chapter.

5.4 EXPERIMENTAL EVALUATION

In this section, we experimentally evaluate the approaches presented in this chapter, and we give an overview on the corresponding tool implementation. In particular, we consider the following two goals for our experiments.

1. We are interested in comparing the computational effort and the size of the resulting test suites of family-based test-suite generation and variant-by-variant test-suite generation. We expect an improved efficiency (in terms

of test-suite size) of the family-based approach as compared to the variant-by-variant strategy as a single family-based test-case usually subsumes several TA test cases. In particular, we consider the union of the test suites of all variants in case of variant-by-variant test-suite generation. Furthermore, we expect that a possible improvement in terms of computational effort depends on the number of variants of the product line. Here, family-based test-case generation is more complex than generating TA test cases, but the family-based approach presumably generates less test cases. It should be noted that we use *bounded* CoPTA (i.e., CoPTA having a finite set of variants) for variant-by-variant test-suite generation.

2. We are interested in experimentally evaluating M/MD test-case generation. As M/MD coverage is a novel coverage criterion, there does not exist a similar tool for comparison with our approach (e.g., in terms of efficiency). Instead, we measure the additional effort for family-based M/MD coverage on unbounded CoPTA models as compared to location coverage on unbounded CoPTA models.

It should be noted that we do not evaluate effectiveness of test-case generation. Effectiveness could be evaluated by measuring the number of (real-world and artificial) faults that are revealed by the generated test cases. However, we leave the evaluation of effectiveness as an open issue for future work. In order to keep the remainder of this section concise and readable, we use the following abbreviations.

- We use **PBP** (**product-by-product**) to refer to the variant-by-variant approach where we derive all TA variants and generate test-cases separately for each model.
- By **FB-U** (**family-based unbounded**), we refer to family-based test-case generation where we use *unbounded* CoPTA models (i.e., CoPTA models with an infinite number of variants).
- We use **FB-MIN** and **FB-MAX** to refer to **minimum-delay** and **maximum-delay** test-case generation, respectively.

RESEARCH QUESTIONS For our evaluation, we refine the two goals mentioned above into three research questions. In our experiments, the main focus is testing efficiency observed in the different experiment settings, where we quantify efficiency in two ways. First, we consider the CPU time (i.e., the computational effort) required for generating test suites satisfying complete test coverage of the CoPTA model as described above. Second, we consider the size of the resulting test suite (i.e., the number of generated test cases) to achieve complete coverage of our test goals. Here, we start by comparing PBP with FB-U (where we restrict CoPTA models to have a finite set of variants in case of PBP).

- **RQ1.1 (Computational Effort):** What is the impact of FB-U on the computational effort for complete test-suite generation as compared to PBP concerning the same coverage criterion?

- **RQ1.2 (Test-Suite Size):** How does FB-U influence the size of the generated test suite as compared to PBP?

Moreover, we evaluate our novel approach for family-based M/MD coverage. Here, the additional ILP-solver calls as well as the checks for reuse of test cases among locations (see Section 5.3) potentially negatively impact test-suite generation in terms of computational effort. In contrast to location coverage, an arbitrary test case may not be sufficient for M/MD coverage of that location as it might not constitute the minimum (or maximum) delay for reaching that location. Conversely, a valid test case for M/MD coverage of a location is not necessarily sufficient to also achieve product-line coverage on that location as the test case might not cover the location in all variants. Hence, we do not compare test-suite sizes for FB-MIN/FB-MAX with FB-U as it appears to be unreasonable.

- **RQ2 (Computational Effort):** What is the impact of FB-MIN/FB-MAX on the computational effort for complete test-suite generation as compared to FB-U?

TOOL SUPPORT For experiments concerning PBP, we utilize reachability checks of the UPPAAL tool suite [39, 34] to generate test cases (i.e., runs) for covering test goals of TA. For unbounded CoPTA models (i.e., FB-U, FB-MIN, and FB-MAX), we utilize the PTA model checker IMITATOR [18, 19, 23] as a basis for a family-based test-suite generator. Hence, we use the PEPTA transformation introduced in Chapter 3 for analyzing CoPTA models. Moreover, we use the built-in ILP solver of IMITATOR for FB-MIN and FB-MAX, as described in the evaluation section of the previous chapter (see Section 4.4). Here, it should be noted that IMITATOR supports PTA with parameter domain $\mathbb{T}_p = \mathbb{Q}_+$. Hence, the integrated algorithms (and thus also our procedure for test-case generation) are inherently incomplete due to the undecidability of almost all semantic properties of this class of PTA. As a result, those cases may lead to timeouts (i.e., non-termination) of analysis runs. Furthermore, IMITATOR does not support delay constraints, such that our tool might not be able to generate a complete yet finite test suite in case of FB-U. However, we have neither encountered non-termination due to undecidability or incomplete test suites during our evaluation.

SUBJECT SYSTEMS Table 5.4 gives an overview on the subject systems that we used for our experimental evaluation described in this chapter. These subject systems are mostly the same as used in the previous chapter. As compared to the overview presented in Table 4.2, Table 5.4 additionally contains the number of variants in cases where we artificially limit our cases studies to a finite number of variants (i.e., for running our experiments for PBP). Furthermore, we did not include the subject systems *BusyBox_dpkg* and *Vim_varp* into this evaluation as the machine we used for our experiments did not contain a sufficient amount of RAM for deriving all variants in case of PBP. In particular, variants are derived by applying the blocking-clause method to the EFM. However, in case of these two subject systems the formulae describing the remaining configurations (which have not yet been derived) became too large (in the number of clauses) to be handled by the Z3 SMT solver.

Table 5.4: Subject Systems for the Experimental Evaluation

Subject System	# Boolean Parameters	# Numeric Parameters	# Locations	# Switches	# Variants
PPU (Figures 3.2, 3.3)	6	3	7	10	636
TGC	3	0	12	18	2
PPU ₀	10	5	7	13	96
PPU ₃	10	5	12	23	96
PPU ₅	10	5	13	28	96
PPU ₆	11	5	20	43	96
PPU ₈	10	5	13	28	96
PPU ₉	10	5	13	28	96
Vim_insecure_flag	7	16	37	55	126
Vim_gui_base_height	7	14	25	35	126
synth_calculate	9	17	48	72	68
synth_method_triggered	9	14	51	69	51
synth_rand_int	8	16	39	48	57

EXPERIMENT DESIGN AND MEASUREMENT SETUP The results from PBP constitute our baseline for **RQ1.1** and **RQ1.2**, and FB-U is our baseline for **RQ2**. As coverage criterion, we first considered location coverage for PBP and FB-U. Thereafter, we applied M/MD coverage for FB-MIN and FB-MAX. For our research questions, we measured the CPU times required for test-suite generation (**RQ1.1**) and the size of the generated test suites (**RQ1.2**) in terms of number of test cases for PBP. Then, we compared these results with the results of FB-U. In case of PBP, we counted the overall number of test cases by summing up the number of test cases for each variant. In particular, we did *not* remove duplicate test cases which were generated for different variants. Here, removing duplicates would resemble a family-based approach as we utilize results from other variants. This would contradict the idea of a *product-by-product* approach. For addressing **RQ2**, we compared the CPU time required for FB-U with FB-MIN and FB-MAX, respectively.

For PBP, we applied UPPAAL in version 4.1.19, and for the remaining approaches (i.e., FB-U, FB-MIN, and FB-MAX), we used IMITATOR in version 2.9.3. We performed all experiments on a machine with Ubuntu 18.04 x64 and 16 GB of RAM running on an Intel Core i7 (4 × 4.2 GHz) machine.

RESULTS The results for **RQ1.1** and **RQ1.2** are summarized in Table 5.5. Here, the second and third column give an overview on the CPU time (i.e., **RQ1.1**) while the remaining columns show the number of resulting test cases (i.e., **RQ1.2**). Based on this data, we observe that PBP is considerably faster in all cases. In particular PBP is 1.3 times (*Vim_gui_base_height*) to 95.6 times (*synth_rand_int*) faster. However, it should be kept in mind that we artificially bounded the configuration spaces to be finite for complete variant-by-variant test-suite generation such that this approach is even applicable. Here, it should be noted that we also applied FB-U to the same bounded CoPTA models that we used for PBP. However, we do not provide results for applying FB-U to bounded CoPTA models as these results are almost the same as applying FB-U to unbounded models for all subject systems. Moreover, we observe that PBP produces a much higher number of test cases

Table 5.5: Results for RQ1.1 and RQ1.2

Subject System	CPU Time (s) for PBP	CPU Time (s) for FB-U	# Test Cases for PBP	# Test Cases for FB-U
PPU (Figures 3.2, 3.3)	120.9	187.8	3500	26
TGC	4.3	20.0	24	24
PPU ₀	20.5	425.1	664	6
PPU ₃	30.3	782.0	796	11
PPU ₅	33.5	914.3	945	13
PPU ₆	55.5	1652.1	1344	19
PPU ₈	35.1	930.9	946	13
PPU ₉	34.3	927.2	934	12
Vim_insecure_flag	124.7	4198.0	1120	184
Vim_gui_base_height	77.0	96.1	1263	92
synth_calculate	84.8	7528.1	1160	51
synth_method_triggered	68.9	5877.7	669	171
synth_rand_int	53.5	5113.8	228	109

Table 5.6: Results for RQ2

Subject System	CPU Time (s) for FB-U	CPU Time (s) for FB-MIN/FB-MAX	# Test Cases for FB-U	# Test Cases for FB-MIN/FB-MAX
PPU (Figures 3.2, 3.3)	187.8	9.5	26	6
TGC	20.0	31.1	24	12
PPU ₀	425.1	97.1	6	2
PPU ₃	782.0	167.9	11	4
PPU ₅	914.3	184.8	13	8
PPU ₆	1652.1	697.4	19	8
PPU ₈	930.9	181.8	13	7
PPU ₉	927.2	180.2	12	8
Vim_insecure_flag	4198.0	550.2	184	48
Vim_gui_base_height	96.1	38.1	92	33
synth_calculate	7528.1	273.4	202	51
synth_method_triggered	5877.7	379.3	171	56
synth_rand_int	5113.8	111.5	109	38

than FB-U for all case studies except for *TGC*. Excluding *TGC*, PBP produces *at least* 228 test cases (*synth_rand_int*), whereas FB-U produces *at most* 184 test cases (*Vim_insecure_flag*).

The results for RQ2 are summarized in Table 5.6. Here, we observe that FB-MIN and FB-MAX require less CPU time than FB-U in all cases except *TGC*. However, FB-MIN and FB-MAX, respectively, only produce a single test case per location, whereas FB-U aims at covering every location for every variant. Hence, we also compare the computational effort required *per test case*. To this end, Figure 5.8 gives an overview on this comparison (where we abbreviated the names of the synthetic case studies for the sake of readability). Here, we observe that the computational effort per test case is smaller for FB-MIN/FB-MAX (white bars) in all cases except *TGC*, *PPU₆*, and *Vim_insecure_flag*.

DISCUSSION First, we discuss the results for RQ1.1 and RQ1.2, where we compare PBP to FB-U. Here, FB-U has a much higher computational effort than

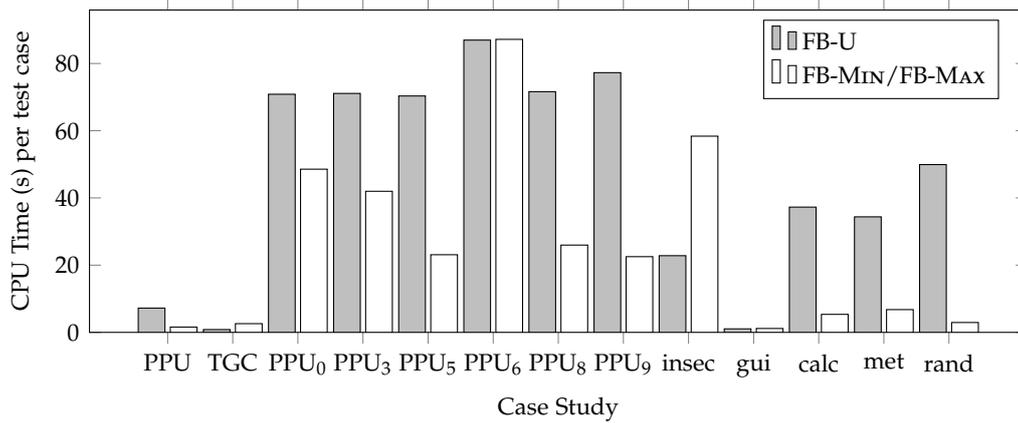


Figure 5.8: Measurement Results for **RQ2**

PBP. However, the difference would naturally become smaller if the number of products would be greater. Furthermore, FB-U produces considerably smaller test suites. We conclude that the choice of the approach for test-case generation (i.e., PBP or FB-U) depends on the setting. On the one hand, PBP could be applied in cases where variants of an SPL are only rarely derived as PBP is much faster in most cases and would still produce a reasonable amount of test cases if the number of derived variants is small. On the other hand, FB-U is useful in cases where we have a sufficient amount of time for a-priori test-case generation, such that we only have to find the applicable test cases when deriving a variant (i.e., checking the presence conditions of the family-based test cases). Finally, PBP is not applicable in scenarios where we want to achieve complete family-based location coverage for unbounded CoPTA models. In general, this is also true for FB-U as family-based test-case generation is semi-decidable. However, we did not encounter non-termination due to semi-decidability of the underlying problem during our evaluation.

Second, we discuss the results for **RQ2**, where we compare FB-MIN/FB-MAX with FB-U. Here, applying FB-MIN/FB-MAX requires less computational effort than FB-U in 12 of 13 cases. This is due to the goal of FB-U where we cover every location *in every variant*. In comparison, FB-MIN and FB-MAX, respectively, have the goal to find one test case per location. Moreover, applying FB-MIN/FB-MAX also requires less computational effort *per test case* than FB-U (in 10 of 13 cases). Here, we assume that the higher average computational effort per test case for FB-U is caused by some test goals being hard to reach. In particular, we observe that (on average) finding further variability-aware test cases for test goals that have already been covered in *some* variants requires more effort than finding variability-aware test cases for completely uncovered test goals (at least for our case studies).

THREATS TO VALIDITY Similar to the experimental evaluation in the previous chapter, a threat to internal validity might arise from the coverage criterion under consideration. However, as testing of time-critical systems is, in general, still an emerging field of research, there are no recent coverage criteria for TA (and generalizations of TA) mentioned in literature (except for location coverage).

Moreover, M/MD coverage is a coverage criterion specifically designed for CoPTA. Nonetheless, we expect similar results for other structural coverage criteria on TA-like models (e.g., switch coverage) as test-suite generation essentially involves consecutive runs for reachability analysis as underlying computational task.

A threat to external validity may be the lack of comparison to other approaches in addition to our own baseline data. However, to the best of our knowledge, no similar techniques (besides FTA model checking) have been developed so far for model-based testing of product lines with configurable parametric real-time constraints. Another external threat to validity might arise from the selection and small number of subject systems. However, these models have, in our opinion, reasonable size and complexity. In particular, *TGC* and the different scenarios of the *PPU* represent real-world models having similar size and complexity as models which are frequently used for evaluating TA-based analysis techniques (e.g., see [32, 98, 30, 119, 29, 45, 182]). However, we plan to consider a greater number of real-world case studies for future work to obtain more meaningful results.

5.5 RELATED WORK

In this section, we give an overview on related work concerning test-case generation for product lines with configurable parametric real-time constraints. To this end, we start by revisiting related work on optimization of real-time behavior. Furthermore, we give an overview on model-based testing for TA and family-based testing of product lines. Finally, we present a list of tools that are interesting in the context of this chapter.

OPTIMIZATION OF REAL-TIME BEHAVIOR The related work for optimization of real-time behavior as presented in Section 4.5 for sampling is also highly relevant here. Hence, we refer the reader to Chapter 4 for this topic. Here, we only discuss one particularly relevant work. André et al. [24] present an algorithm to find runs with minimum/maximum delay for PTA (which was developed simultaneously to and independently of our work in [135]). This is especially relevant in the context of this chapter as an M/MD test cases consists of a presence condition and a run. Hence, the output of the algorithm presented by André et al. can be considered a test case. However, André et al. do not utilize the results of the algorithm for further purposes (e.g., as test cases). Moreover, the approach of André et al. is based on PTA which do not support traceability between a problem space and a solution space (as opposed to our CoPTA formalism utilizing features for this purpose).

MODEL-BASED TESTING OF TA There are many related works concerning model-based testing of TA such that we only present an extract of the related work in this area. Here, Springintveld et al. [181] describe an approach for test-suite derivation in case of black-box conformance testing. Additionally, Krichen and Tripakis [116] provide a framework for on-the-fly testing of black-box real-time systems. Directly related to these two works, Hessel et al. [102] perform black-box

conformance testing of TA using the UPPAAL tool suite [39, 34]. Furthermore, Brandán Briones and Röhl [48] derive test cases from TA models. Moreover, there exist several timed variations of so-called *ioco* testing. In particular, *ioco* as presented by Tretmans [186] is concerned with comparing input/output behavior of an implementation with the corresponding specification. Here, the implementation is allowed to have less output behavior than specified, but at least one of the specified outputs has to be implemented in each state. In case of *timed ioco*, Schmaltz and Tretmans [172] provide a comprehensive overview. For instance, Luthmann et al. [137, 138] consider a variation of *timed ioco* where an implementation must enforce progress (i.e., performing an output action) if this is also the case in the specification. Finally, Göttmann et al. [93] utilize TA to model the real-time behavior in terms of configuration decisions, which could be used as a basis for generating test cases covering these decisions. However, none of these approaches consider product lines of time-critical systems.

FAMILY-BASED TESTING OF PRODUCT LINES As there are many works in the area of family-based testing of product lines, we, again, only present an extract of the related work in this area. For a comprehensive overview, we refer the reader to the surveys of Lee et al. [121], Oster et al. [155], and da Mota Silveira Neto [76]. As a first approach, we can apply sampling such that we derive a representative subset and variants and test these variants. As we thoroughly discussed sampling in the previous chapter, we refer the reader to Section 4.5 for related work on this approach.

Furthermore, there are several works lifting *ioco* (see above) to variant-rich models. Beohar and Mousavi [40, 42] define *ioco* for featured transition systems (FTS) [60] extended by input/output labels such that they obtain a family-based testing formalism for *ioco*. Luthmann et al. [130] also lift *ioco* to a variant-rich formalism but utilize modal interface automata (MIA) [141, 50] instead of FTS. In a MIA model, each transition is either optional (i.e., may be implemented) or mandatory (must be implemented). Therewith, Luthmann et al. only perform one *ioco* check on MIA models instead of checking each variant. Moreover, Luthmann et al. consider operators for (de-)composition [131, 132] and conjunction [139]. As a further approach, Soldani et al. [179, 178] propose to apply *ioco* to management protocols [49], being utilized for managing the communication of enterprise applications. Management protocols comprise variability in the sense that at most a given set of requirements and at least a given set of capabilities (e.g., a web server) need to be implemented.

Furthermore, Bürdek et al. [53] apply family-based test-case generation to C code with preprocessor variability. To this end, Bürdek et al. apply similar techniques as used throughout this chapter (e.g., the blocking-clause method). In fact, Algorithm 5.2 is based on this work (as also indicated in Section 5.2). As opposed to modeling variability with feature annotations or modalities, it is possible to utilize so-called *deltas* [59]. Here, we have a base model and several deltas modeling product lines by describing differences to the base model, where deltas can be linked to particular features. For instance, Lochau et al. [126] utilize this approach for efficient incremental testing of product lines modeled with state ma-

chines. Additionally, Lity et al. [124] make retest decisions for regression testing of delta-oriented product lines. Furthermore, Varshosaz et al. [192] generate test-cases based on finite state machines and utilize deltas for more efficient test-case generation. Moreover, Hafemann Fragal et al. [95] generate test cases for featured finite state machines (FFSM) [94] using complete fault coverage as coverage criterion.

However, to the best of our knowledge, there does not exist an approach for testing product lines with real-time constraints comprising a potentially infinite number of variants. Here, the approach of Cordy and Legay [69] for FTA verification could be utilized as a basis for family-based derivation of test suites in case of *finite* product lines.

TOOLS In terms of tools, JTORX [35] (being based on TORX [187]) can be utilized for automated test-case generation based on TA models. Furthermore, Tretmans and van de Laar [188] recently presented the latest iteration of JTORX, called TORXAKIS. Moreover, the UPPAAL tool suite [39, 34] can be used for generating TA test cases (as described in this section) as well as for finding TA runs having minimum (or maximum) delays for reaching particular locations. In addition to this, there exists an extension of UPPAAL, called UPPAAL TRON [120], for black-box online test-case generation for real-time systems. To this end, UPPAAL TRON implements a variation of timed ioco (see above). In case of FTA, the ProVeLines tool suite [71, 72, 73] could be utilized as a basis for family-based test-case generation. ProVeLines is a model checker for FTA such that the algorithms presented in this chapter could easily be adapted for this tool. Finally, the PTA model checker IMITATOR [18, 19, 23] is used as a basis for the implementation of the approaches presented in this chapter. Here, IMITATOR could also be used to generate test cases for PTA or other formalisms that can be transformed into PTA. However, all of these tools either only consider TA (or similar formalism for single systems) or do not guarantee coverage w.r.t. given criteria. Moreover, to the best of our knowledge, there is no tool available for generating test cases for product lines with real-time constraints comprising a potentially infinite number of variants.

5.6 CONCLUSION AND FUTURE WORK

In this chapter, we tackled Research Challenges 3.1 and 3.2. In particular, there did not exist an approach for generating a finite test suite for a product line comprising infinitely many variants, and thus, also infinitely many test goals. Furthermore, there did not exist strategies for product-line test-suite generation based on real-time constraints in terms of BCET/WCET. Here, established approaches mostly only consider covering basic components of a model (i.e., locations or switches). Hence, in this chapter we improved the state of the art in two ways.

First, we introduced an approach (being sound but incomplete) for generating a *finite* test suite satisfying complete family-based location coverage for product lines with unbounded parametric real-time constraints even in case of a product line with infinitely many variants. In order to achieve this goal, each variability-aware test case comprises a presence condition symbolically describing the (possibly infinite) set of configurations for which the test case is valid. Moreover, we adapted

an existing algorithm to systematically generate a complete variability-aware test-suite for a given CoPTA model. Thereafter, we adapted the concept of M/MD coverage presented in Chapter 4 for M/MD test-case generation. To this end, we utilize the M/MD instrumentation (as introduced in Chapter 4) to find a test case for each location having BCET/WCET for reaching this location. For instance, the generated M/MD test cases allow us to reveal potential off-by-one timing errors.

Finally, our experimental evaluation based on our tool implementation shows applicability of our approach. In particular, our family-based approach for test-case generation has a higher computational effort than product-by-product test-case generation but produce a much smaller number of test cases. However, without our approach for test-case generation, it would not even be possible to generate *complete finite* test suites in this setting.

For future work, there are several open challenges. Here, we may adapt M/MD coverage for further coverage criteria (as also proposed for future work of our sampling approach presented in Chapter 4), potentially revealing additional faults. For instance, we may want to generate test cases reaching particular locations via other intermediate locations with BCET/WCET. Additionally, we are interested in finding runs with minimum and maximum delays within a given time frame. Moreover, we plan to investigate whether it is possible to define family-based location coverage directly based on CoPTA locations instead of counting covered TA locations (as mentioned in Section 5.2.2). In this chapter, we only defined *complete* family based location coverage (see Definition 5.11) to avoid using (possibly) infinite values in fractions. Besides these points, we may extend CoPTA with input/output labels such that (timed) ioco-based testing approaches (see Section 5.5) can be adapted to CoPTA. Furthermore, we only generate test cases for allowed behavior (i.e., behavior described by a CoPTA model) so far. Here, we may also consider generating test cases explicitly describing forbidden behavior. Therewith, we are able to check whether additional (unspecified) behavior was added.

Moreover, it would be interesting to dynamically explore the state space of CoPTA models (i.e., the featured parametric zone graph) instead of picking a location and then querying a model checker for a run reaching this location. Here, a dynamic exploration might faster in finding runs for easily reachable locations (e.g., in case no other parts of a CoPTA model have to be traversed before a particular clock constraint is satisfiable) whereas our approach is probably faster in finding runs for reaching the remaining locations. Depending on evaluation results, we might also consider combining both approaches. In this context, so-called *spinal test suites* [41] may be helpful. Here, the idea is to utilize existing test cases to derive further test cases. To this end, we would take an existing test case and resume test-case generation after the final test step. For instance, a test case consisting of two pairs of delays and actions may be extended by additional steps to traverse further parts of a CoPTA model.

Finally, we plan to extend our evaluation. In particular, we plan to compare our approach for computing runs with minimum (or maximum) delay in an evaluation with the approach presented by André et al. [24] for PTA. Furthermore, we would like to consider a greater number of (real-world) case studies such that we can apply statistical methods. Moreover, we may utilize mutation testing to evaluate

effectiveness (i.e., our capability to reveal errors) of the generated test suites. Here, Aichernig et al. [6] already propose mutation operators for TA. Similar to our evaluation of M/MD sampling, we also plan to consider further measure for the size of case studies (e.g., number of states with more than one outgoing switch) as our measurement results did not correlate with the number of locations in all cases.

6

BISIMILARITY OF PRODUCT LINES WITH CONFIGURABLE PARAMETRIC REAL-TIME CONSTRAINTS

In this chapter, we tackle Research Challenges 4.1 and 4.2 as introduced in the background chapter (see Section 2.6).

Research Challenge 4.1

Effectively checking timed bisimilarity of TA.

Research Challenge 4.2

Checking timed bisimilarity of product lines with a potentially infinite number of variants.

Bisimulation is concerned with comparing the behavior of two systems against each other in a state-by-state fashion. To this end, we are especially interested in *time-critical* behavior as we consider discrete-state/continuous-time models in this thesis. For instance, timed bisimulation may be applied in the development process of model-driven software engineering as depicted in Figure 6.1. Here, we start by creating an initial model describing the specified behavior. During the development process (indicated by the circle), this model is continuously adapted (e.g., by restructuring the model), such that we obtain intermediate or final models. Throughout this process, the intention is to adapt the model without changing the (time-critical) behavior. Hence, we compare the resulting model of each development step with the original model. This can be achieved by applying timed bisimulation (indicated by symbol \simeq in Figure 6.1). To this end, we compare the behavior of the adapted model with the original model for corresponding pairs of states in a state-by-state fashion (which is undecidable when using TLTS as described in the background chapter). Moreover, we can check for *similarity* in settings where the task is to ensure that the adapted model contains *at least* or *at most* the behavior of the original model (instead of exactly the same behavior in case of bisimilarity). In fact, we check bisimilarity for TA \mathcal{A} and \mathcal{A}' by checking if \mathcal{A} is similar to \mathcal{A}' , and vice versa.

Example 6.1. Consider the TA model \mathcal{A} of the PPU as presented in the background chapter (see Figure 2.3). Timed bisimilarity of this model w.r.t. another system (e.g., the adapted TA model \mathcal{A}' of the PPU depicted in Figure 2.4) can (theoretically) be checked by utilizing TLTS derived from these TA \mathcal{A}

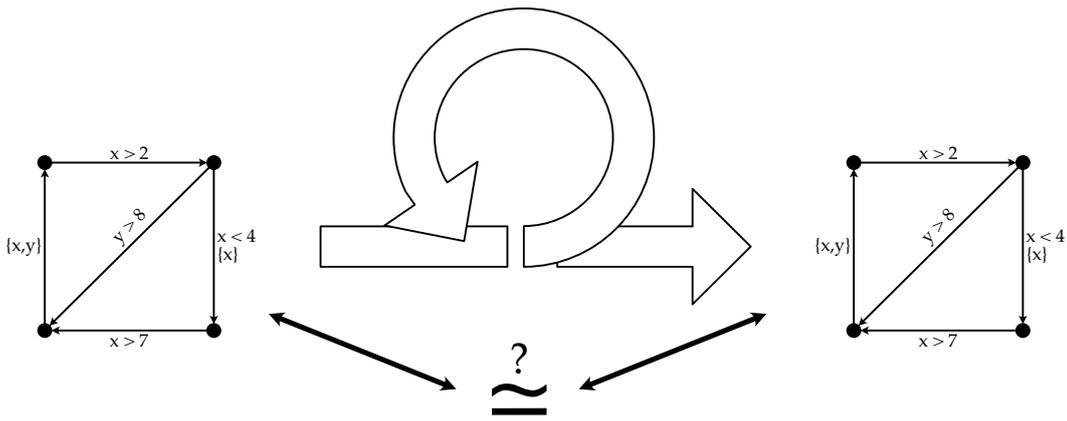


Figure 6.1: Schematic Overview on the Contributions Presented in Chapter 6

and \mathcal{A}' as a basis. For instance, the TLTS of TA \mathcal{A} and \mathcal{A}' are depicted in Figures 2.9 and 2.13, respectively. However, we cannot use these models for checking timed bisimilarity in practice, as these two systems (and TLTS in general) comprise an infinite state space. Hence, in this chapter we introduce a formalism for *effectively* checking timed bisimilarity for two given TA.

In addition to this kind of development process, timed bisimulation can also be applied for testing purposes. So far, we have considered test cases as timed runs (and a presence condition in case of product lines) which can be applied to *black-box* implementations (see Chapter 5). Hence, we do not compare behavior of particular states such that we do not consider the internal structure of our implementation. In contrast, timed bisimulation may be utilized as a means for *white-box* testing, where the models of the implementation as well as the specification are available. Therewith, we may compare the behavior of states instead of only considering sequences of delays and actions. However, it should be noted that checking timed bisimilarity is more complex than the black-box testing approach introduced in Chapter 5.

Finally, we also consider product lines with parametric real-time constraints. In this case, checking timed bisimulation means that we have to check for *each variant* of the adapted model if there exists a variant in the original model such that these variants are bisimilar. Here, variant-by-variant checking of featured parametric timed bisimilarity is possible in case product lines comprising finitely many variants, although this may become quite expensive. Furthermore, applying variant-by-variant checks is impossible in case of CoPTA models comprising an infinite number of variants.

Example 6.2. Consider the CoPTA model \mathcal{C} and the EFM m of the PPU SPL as presented in Chapter 3 (see Figures 3.2 and 3.3). Additionally, assume a slightly adapted EFM

$$m' := m \wedge (\text{Plastic} \Rightarrow a \leq 30) \wedge (\text{PPU} \Rightarrow c \leq 40)$$

such that $\llbracket m' \rrbracket$ only contains a finite number of configurations. As already explained in Example 5.1, EFM m' describes 18,000 valid configurations (even though m' only comprises six features and three parameters). Hence, complete variant-by-variant analysis in terms of timed bisimilarity is practically infeasible for this adapted example. Moreover, using the original EFM m , complete variant-by-variant analysis becomes literally impossible due to the infinite number of variants.

As motivated by Example 6.2, variant-by-variant checking of timed bisimilarity is impossible in case of CoPTA comprising an infinite number of variants. Hence, the goal of this chapter is to also describe a method for symbolically checking timed bisimilarity of product lines with parametric real-time constraints directly on CoPTA models instead of using a variant-by-variant approach.

To this end, we first describe a procedure for *effectively* checking timed bisimilarity of TA (in contrast to the method introduced in Section 2.6 which relies on an infinitely branching TLTS semantics). This approach supports deterministic TA as well as non-deterministic TA. For this thesis, non-determinism is particularly interesting as family-based models such as CoPTA (for which we plan to generalize our approach to) often exhibit a form of *pseudo* non-determinism, where several outgoing switches have the same action label but exclude each other through constraints over Boolean and numeric parameters. It should be noted that there already exist a few approaches for effectively checking TA bisimilarity. However, all of these approaches have some disadvantages as compared to the approach that we introduce in this chapter. For instance, Čerāns [56] defines timed bisimilarity based on region graphs which suffer from state-space explosion, making the approach generally less efficient than approaches based on zone graphs. Furthermore, the approaches by Weise and Lenzkes [201] and Guha et al. [91, 92] are not defined in a way that would directly allow a generalization for checking timed bisimilarity of PTA and CoPTA.

In this chapter, we propose an alternative solution for effectively checking TA bisimilarity, being closer to the concepts of (bi-)simulation equivalence relations on state-transition graphs than the existing approaches (i.e., we enrich symbolic states with additional discriminating information). Moreover, our approach supports a bound for the amount of information accumulated in states, facilitating an adjustable trade-off between efficiency and precision for handling large-scale real-world models. This is not supported by any of the approaches mentioned above. As another advantage, our approach is, to the best of our knowledge, the only approach being implemented in an available tool for effectively checking timed bisimilarity.

After defining TA bisimilarity, we proceed by lifting timed bisimulation based on TLTS to PTA, such that we obtain a semi-symbolic approach for checking timed bisimilarity of PTA. In particular, we consider PTA instead of FTA as PTA facilitate an increased expressiveness by using a-priori unbounded time intervals. For future work, this semi-symbolic approach can then be used as foundation for *effectively* checking bisimilarity for PTA by generalizing our procedure for checking TA bisimilarity. Finally, we lift the notion of timed bisimilarity to CoPTA. However, similar to PTA, we do not introduce a (semi-)decidable approach for checking

bisimilarity of CoPTA. Instead, we give an outlook and leave this challenge as an open issue for future work.

The remainder of this chapter is structured as follows. We start by introducing an approach for effectively checking timed bisimilarity of TA (see Section 6.1). Thereafter, we generalize our approach to PTA, such that we consider unbounded numeric parameters (see Section 6.2). Afterwards, we also include Boolean parameters, such that we introduce timed bisimilarity for CoPTA models (see Section 6.3). Here, it should be noted that we leave the definition of a (semi-)decidable check for (featured) parametric timed bisimulation as an open problem for future work. Thereafter, we present an experimental evaluation for checking TA bisimilarity (see Section 6.4). However, we do *not* evaluate (featured) parametric timed bisimulation as we do not describe a (semi-)decidable check for these approaches in this thesis. Finally, we give an overview on related work (see Section 6.5), and conclude this chapter (see Section 6.6). The contents of this chapter are based on the following publication:

[140] Lars Luthmann, Hendrik Göttmann, Isabelle Bacher, and Malte Lochau. Checking Timed Bisimulation with Bounded Zone-History Graphs. Submitted to *Acta Informatica*, 2020.

[128] Malte Lochau, Lars Luthmann, Hendrik Göttmann, and Isabelle Bacher. Parametric Timed Bisimulation. In *9th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA '20)*, volume 12477 of *LNCS*, pages 55–71. Springer International Publishing, 2020. ISBN 978-3-030-61470-6. doi: 10.1007/978-3-030-61470-6_5.

6.1 CHECKING BISIMULATION FOR TIMED AUTOMATA

In this section, we introduce an approach for *effectively* checking timed bisimilarity for a pair of TA models (as compared to the undecidable check based on TLTS as presented in the background chapter). Similar to the analysis techniques presented for sampling and test-case generation (see Chapters 4 and 5), we utilize a symbolic representation of the state space to perform these checks. In the background chapter, we already described zone graphs, modeling the state space of a TA in a symbolic way (see Definition 2.20). Zone graphs are useful for checking several properties of TA in a decidable way. For instance, we utilize zone graphs to generate test cases (i.e., perform reachability checks) in Chapter 5. However, the information provided by the zone-graph representation of a TA are too imprecise for checking timed bisimilarity [201].

Example 6.3. Consider the TA models and the corresponding zone graphs depicted in Figure 6.2. In TA \mathcal{A} , we are allowed to execute action σ within the first two seconds (due to guard $x \leq 2$), whereas σ can be performed in \mathcal{A}' at any time (as there is no guard). However, the corresponding zone graphs (see Figures 6.2c and 6.2d) are completely identical. This is due to resets *covering*

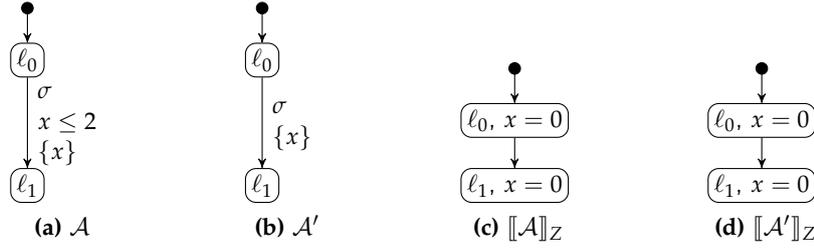


Figure 6.2: Example for the Problem of Checking Timed Bisimilarity with Zone Graphs

other clock constraints in zones. Furthermore, zone graphs do not have any transition labels. Hence, we cannot use plain zone graphs for checking timed bisimilarity.

As illustrated by Example 6.3, we cannot use plain zone graphs for checking timed bisimilarity as the results would potentially be unsound. Hence, we extend zone graphs (in three ways) and use this formalism as a basis for *effectively* checking timed bisimilarity.

1. We explicitly track the effects of resets to avoid the problem which we illustrated in Example 6.3. In particular, we extend states by so-called *histories*, modeling the allowed time frames for reaching the current state from previous states. Therewith, we overcome the issue of resets covering clock constraints in zones.
2. We extend zone-graph transitions by labels (as, e.g., also done by Guha et al. [91]) which are necessary for comparing the behavior of states.
3. We further generalize our extension of zone graphs such that we can also handle TA with non-deterministic behavior.

By utilizing these extensions, we are then able to effectively check timed bisimilarity. In the remainder of this section, we first introduce the notion of histories (see Section 6.1.1). Afterwards, we explain the extension of zone graphs by histories (called *zone-history graphs*) and utilize this extension for checking timed bisimilarity of deterministic TA (see Section 6.1.2). Recall, that a TA is deterministic if there does not exist a state in the corresponding TLTS having two outgoing transitions labeled with the same action (see Definition 2.18). Here, we start by providing a definition potentially resulting in infinite zone-history graphs. Thereafter, we introduce a cut criterion to obtain finite zone-history graphs while preserving the information that are needed for checking timed bisimilarity. Afterwards, we further generalize zone-history graphs to *composite zone-history graphs* such that we can also check timed bisimilarity of non-deterministic TA (see Section 6.1.3). Finally, we introduce a trade-off between precision and scalability, called *bounded zone-history graphs*, such that the check is more efficient but might produce false positives (see Section 6.1.4).

6.1.1 Modeling the History of Real-Time Behavior

As illustrated by Example 6.3, plain zone graphs are not sufficient for checking timed bisimilarity as the result would potentially be unsound. Weise and Lenzkes [201] tackle the issue of insufficient information by considering *good sequences* of zone graphs in an additional check. As opposed to Weise and Lenzkes, we propose an alternative solution being closer to the concepts of (bi-)simulation equivalence relations on state-transition graphs (i.e., we enrich symbolic states with additional discriminating information). In particular, we utilize *histories* to model the allowed time frames for reaching the current state from a previous state. As a result, a history is a sequence of zones (i.e., a sequence of clock constraints).

Example 6.4. Consider again TA \mathcal{A} as depicted in Figure 6.2a. For checking timed bisimilarity, we enrich states of the zone graph $\llbracket \mathcal{A} \rrbracket_Z$ (see Figure 6.2c) by a third component in terms of a history. For instance, the zone-graph state comprising location ℓ_1 is enriched by a history having zone

$$(x = 0 \wedge \chi \leq 2 \wedge \chi - x \leq 2)$$

as its only element. Here, x is the clock used by TA \mathcal{A} and χ is an additional clock describing the allowed time frame for reaching ℓ_1 from the predecessor state comprising location ℓ_0 . Intuitively, we have $\chi \leq 2$ as it may take 0 to 2 seconds for reaching ℓ_1 from ℓ_0 . We explain in detail how we obtain this constraint over χ later in this section.

Considering sequences of zones also allows us to introduce a trade-off between precision and scalability by considering only a prefix or postfix of the sequence of a particular length (which we explain in detail in Section 6.1.4). In order to compare histories, we compare their elements (i.e., zones). To this end, the comparison of zones is based on the notion of *spans* [90]. The span of clock $c \in C$ in zone D describes the interval (lo, up) , where lo is the minimum allowed valuation of c in D and up is the maximum allowed valuation of c in D . Here, we use ∞ to denote open intervals, such that $up = \infty$ means that clock c does not have a maximum allowed valuation. Moreover, we introduce two operators for comparing spans $sp = (lo, up)$ and $sp' = (lo', up')$.

- $sp \preceq sp'$ denotes that sp is contained in sp' , such that $lo \geq lo'$ and $up \leq up'$.
- $sp \leq sp'$ denotes that the length of sp is smaller than the length of sp' , such that $up - lo \leq up' - lo'$.

Definition 6.1 (Span). Let D be a zone and C be a set of clocks with $c \in C$. Then,

$$\text{span}(c, D) = (lo, up) \in \mathbb{T}_C \times (\mathbb{T}_C \cup \{\infty\})$$

is the smallest interval such that $\forall u \in D : u(c) \geq lo \wedge u(c) \leq up$. We use the following operators for spans:

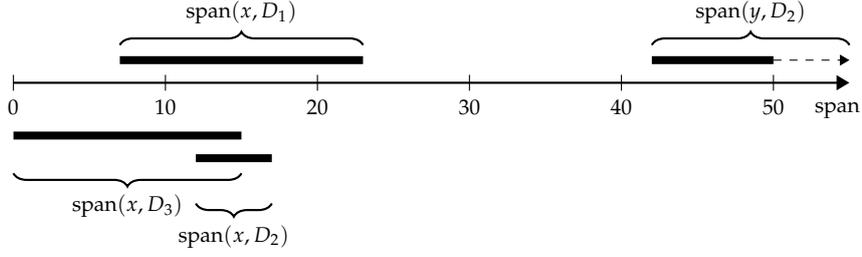


Figure 6.3: Illustration of the Spans of Example 6.5

- $\text{span}(lo, up) = \{n \in \mathbb{T}_C \mid n \geq lo \wedge n \leq up\}$,
- $(lo, up) \preceq (lo', up') \Leftrightarrow lo \geq lo' \wedge up \leq up'$, and
- $(lo, up) \leq (lo', up') \Leftrightarrow up - lo \leq up' - lo'$.

It should be noted that the notion of spans facilitates the comparison of zones (and therewith also occurrences of time-critical action sequences) of TA having different names for locations and clocks.

Example 6.5 (Span). Consider the following zones as an example for which the spans are illustrated in Figure 6.3.

- $D_1 = (x \geq 7 \wedge x \leq 23)$ with $\text{span}(x, D_1) = (7, 23)$,
- $D_2 = (x \geq 12 \wedge x \leq 17 \wedge y \geq 42)$ with $\text{span}(x, D_2) = (12, 17)$ and $\text{span}(y, D_2) = (42, \infty)$, and
- $D_3 = (x \leq 15)$ with $\text{span}(x, D_3) = (0, 15)$.

For instance, it holds that $\text{span}(x, D_2) \preceq \text{span}(x, D_1)$, $\text{span}(x, D_2) \leq \text{span}(x, D_1)$, and $\text{span}(x, D_3) \leq \text{span}(y, D_2)$.

As described above, zone graphs do not contain sufficient information for a sound check of timed bisimilarity. Therefore, we extend zone-graph states $\langle \ell, D \rangle$ by a third component in terms of a *history* $\mathcal{H} \in \mathcal{B}(C \cup \{\chi\})^*$ to triples $\langle \ell, D, \mathcal{H} \rangle$. Here, $\chi \notin C$ is an additional clock that we use to describe the allowed timeframes for reaching certain locations. This is inspired by the notion of *causal histories* as, e.g., proposed for history-preserving event structures [26]. In particular, we utilize history \mathcal{H} to accumulate sequences of zones (i.e., clock constraints) corresponding to the zones of predecessor states.

Example 6.6. Consider again TA \mathcal{A} depicted in Figure 6.2a. Here, the state of the zone-history graph after performing action σ is

$$\langle \ell_1, x = 0, (x = 0 \wedge \chi \leq 2 \wedge \chi - x \leq 2) \rangle$$

where ℓ_1 is the location, $x = 0$ is the zone (as usual for zone graphs), and the remainder of the state is the history (comprising a single element in this case).

As described in Example 6.4, the span of the additional clock χ describes the allowed timeframe for reaching location ℓ_1 from location ℓ_0 .

When adding a state $\langle \ell', D', \mathcal{H}' \rangle$ with predecessor $\langle \ell, D, \mathcal{H} \rangle$, we update \mathcal{H} to \mathcal{H}' according to the update of D leading to D' as described in Definition 2.20 (i.e., we update every element of \mathcal{H} by applying the same update as done for D). Furthermore, we append a new element describing the allowed time interval for the step from $\langle \ell, D, \mathcal{H} \rangle$ to $\langle \ell', D', \mathcal{H}' \rangle$. Hence, the last element of a zone history describes the allowed span for the last step, the second last element describes the allowed span for the last two steps, and so on.

Here, we describe the allowed time interval by introducing a fresh clock $\chi \notin C$ for each new element of a history. Therewith, we measure the respective *spans of histories*, such that we can compare histories by comparing sequences of spans. In particular, clock χ is never reset such that we also correctly measure the elapsed time in the presence of resets. As a result, we can calculate the allowed time frame for reaching the current state from all predecessor states by utilizing the span of χ (as we add a new history element after each transition of the zone-history graph). We use $\mathcal{H} \cdot H$ and $H \cdot \mathcal{H}$, respectively, to denote concatenation of a history element H to a history \mathcal{H} (as first or last element). Additionally, ϵ denotes the *empty sequence* with $\mathcal{H} = \mathcal{H} \cdot \epsilon = \epsilon \cdot \mathcal{H}$.

Definition 6.2 (Zone History). Let $\mathcal{H} \in \mathcal{B}(C \cup \{\chi\})^*$ with $\chi \notin C$ be a *zone history*. The *update* of history \mathcal{H} for a switch $\ell \xrightarrow{g, \sigma, R} \ell'$ leading from zone D to $D' = R(D^\uparrow \wedge g \wedge I(\ell)) \wedge I(\ell')$ is recursively defined by

- $\text{update}(\mathcal{H}, D, D') = R(H^\uparrow \wedge g \wedge I(\ell)) \wedge I(\ell') \cdot \text{update}(\mathcal{H}', D, D')$ if $\mathcal{H} = H \cdot \mathcal{H}'$,
- $\text{update}(\mathcal{H}, D, D') = (R((D \wedge \chi = 0)^\uparrow \wedge g \wedge I(\ell)) \wedge I(\ell'))$ if $\mathcal{H} = \epsilon$.

Example 6.7 (Zone History). Assume, we have an existing symbolic state

$$\langle \ell_1, x \geq 10 \wedge x \leq 15 \wedge x = y, (x \geq 10 \wedge x \leq 15 \wedge x = y \wedge \chi = x) \rangle$$

in a zone-history graph, where ℓ_1 is the location, $D = (x \geq 10 \wedge x \leq 15 \wedge x = y)$ is the zone, and the remainder of the state describes the history \mathcal{H} . Here, Figure 6.4 shows an extract of the corresponding TA model. Next, we update the existing history \mathcal{H} according to the switch in our TA model. Hence, we update the existing element of the history (where clock χ describes the allowed time frame for reaching ℓ_2 from the initial state), and we append a new element (where clock χ describes the allowed time frame for reaching ℓ_2 from ℓ_1). To this end, we apply the first rule of Definition 6.2 for updating the existing history element and the second rule for appending a new element. Here, $D' = (x = 0 \wedge x = y)$ due to the reset of x and y caused by the switch depicted

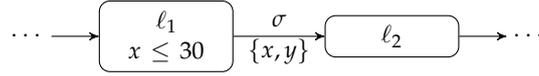


Figure 6.4: TA Extract for Example Describing Zone Histories

in Figure 6.4 (i.e., we update the zone exactly as done for plain zone graphs). As a result, we have

$$\text{update}(\mathcal{H}, D, D') = R(H^\uparrow \wedge \text{true} \wedge x \leq 30) \wedge \text{true}$$

where $H = (x \geq 10 \wedge x \leq 15 \wedge x = y \wedge \chi = x)$ is the existing history element. Applying the zone operators as usual (e.g., see Example 2.14) and simplifying this constraint results in $(x = 0 \wedge x = y \wedge \chi \geq 10 \wedge \chi \leq 30)$. Furthermore, we append a new history element by applying

$$\text{update}(\epsilon, D, D') = R((D \wedge \chi = 0)^\uparrow \wedge \text{true} \wedge x \leq 30) \wedge \text{true}$$

which results in $(x = 0 \wedge x = y \wedge \chi \leq 20)$.

Based on the notions of spans and zone histories, we can now proceed by extending zone graphs with histories for effectively checking timed bisimilarity. To this end, we first introduce zone-history graphs for checking timed bisimilarity of deterministic TA followed by an extension also supporting non-deterministic TA.

6.1.2 Checking Timed Bisimilarity of Deterministic Timed Automata

In the previous section, we defined the notion of zone histories. Based on this concept, we now define the *zone-history graph* $\llbracket \mathcal{A} \rrbracket_H$ of a TA \mathcal{A} for checking timed bisimilarity of deterministic systems. Recall, that a TA is deterministic if there does not exist a state in the corresponding TLTS having two outgoing transitions labeled with the same action (see Definition 2.18). Here, we extend zone graphs with zone histories and transition labels. The initial state $z_0 = \langle \ell_0, D_0, \epsilon \rangle$ of a zone-history graph comprises the initial location ℓ_0 of the corresponding TA, the initial zone D_0 (as defined for zone graphs), and the *empty* history ϵ . Furthermore, a transition $\langle \ell, D, \mathcal{H} \rangle \xrightarrow{\sigma} \langle \ell', D', \mathcal{H}' \rangle$ corresponds to TA switch $\ell \xrightarrow{g, \sigma, R} \ell'$, where the update from zone D to D' is done as usual (see Definition 2.20) and $\mathcal{H}' = \text{update}(\mathcal{H}, D, D')$ is the update of history \mathcal{H} of the predecessor state (see Definition 6.2). In the following, we utilize z_i as a shorthand for state $\langle \ell_i, D_i, \mathcal{H}_i \rangle$ (i.e., every element of the state has the same index i).

Definition 6.3 (Zone-History Graph). Let $\mathcal{A} = (L, \ell_0, \Sigma, C, I, E)$ be a TA and $\chi \notin C$ be a clock. The *zone-history graph* of \mathcal{A} is a tuple $(\mathcal{Z}, z_0, \Sigma, \rightsquigarrow)$, where

- $\mathcal{Z} = L \times \mathcal{B}(C) \times \mathcal{B}(C \cup \{\chi\})^*$ is a set of *symbolic states*,
- $z_0 = \langle \ell_0, D_0, \epsilon \rangle$ is the *initial state*,
- Σ is a set of *actions*, and

- $\rightsquigarrow \subseteq \mathcal{Z} \times \Sigma_\tau \times \mathcal{Z}$ is the least relation satisfying the rule:

$$z \xrightarrow{\mu} z' \text{ if } \ell \xrightarrow{g, \mu, R} \ell', D' = R(D^\uparrow \wedge g \wedge I(\ell)) \wedge I(\ell'),$$
 and $\mathcal{H}' = \text{update}(\mathcal{H}, D, D')$.

We apply Algorithm 6.1 to generate a *finite* zone-history graph from $(\mathcal{Z}, z_0, \Sigma, \rightsquigarrow)$. By $\llbracket \mathcal{A} \rrbracket_H$, we refer to the (finite) zone-history graph of TA \mathcal{A} .

It should be noted that the definition of zone-history graphs only serves as a theoretical baseline as updating the history in every step results in an infinite zone-history graph in the presence of cyclic paths in TA models. However, later in this section we present an algorithm having a cut criterion for pruning possibly infinite zone-history graphs into finite ones while preserving the information being necessary for effectively checking timed bisimilarity (see Algorithm 6.1 on page 162). However, we first give an example of an infinite zone-history graph.

Example 6.8 (Zone-History Graph of the PPU). Consider the (extract of the) zone-history graph depicted in Figure 6.5b corresponding to the TA of the PPU presented in the background chapter (which we repeat in Figure 6.5a for the convenience of the reader). Note, that we omit some redundant constraints in Figure 6.5b for the sake of readability (e.g., $x = 0 \wedge x = y$ is equal to $x = 0 \wedge y = 0 \wedge x = y$). Initially, we start in the symbolic state comprising location *init*, zone $x = 0 \wedge x = y$ (as all clocks are initialized with value 0), and empty history ϵ (as we have not used any transition so far). Then, we add a transition corresponding to the switch labeled with *rotate* in TA \mathcal{A} , such that we reach a new symbolic state comprising location *stack* and zone $x \geq 10 \wedge x \leq 15 \wedge x = y$ (due to the guard of the switch and the fact and the values of all clock always increase at the same rate). Furthermore, we *update* the (empty) zone history of the previous state to

$$(x \geq 10 \wedge x \leq 15 \wedge x = y \wedge \chi = x)$$

where we refer the reader to Example 6.7 for a detailed explanation of updates for zone histories. It holds in this zone history that $\chi \geq 10 \wedge \chi \leq 15$ (due to $\chi = x$). This means that the allowed time frame for reaching location *stack* from *init* is 10 to 15 seconds. Using one of the outgoing transitions of location *stack* results in additional symbolic states where the current history is updated and a new history element (i.e., zone) is appended. For instance, the span of χ in the first history element of the symbolic state comprising location *WP picked* is $(10, 30)$, meaning that it takes at least 10 seconds to reach *WP picked* from *init* and at most 30 seconds. Furthermore, the span of χ in the second element of this zone history is $(0, 20)$, denoting the allowed time span based on the previous location *stack*. In particular, we may immediately leave location *stack* upon arrival (as the switch labeled with *pick* does not have any guard) or wait up to 20 seconds (as $x \geq 10$ when reaching *stack*). Note, that zone-history graph $\llbracket \mathcal{A} \rrbracket_H$ has an infinite state space as TA \mathcal{A} contains cyclic behavior and we extend the zone history in each step by an additional element.

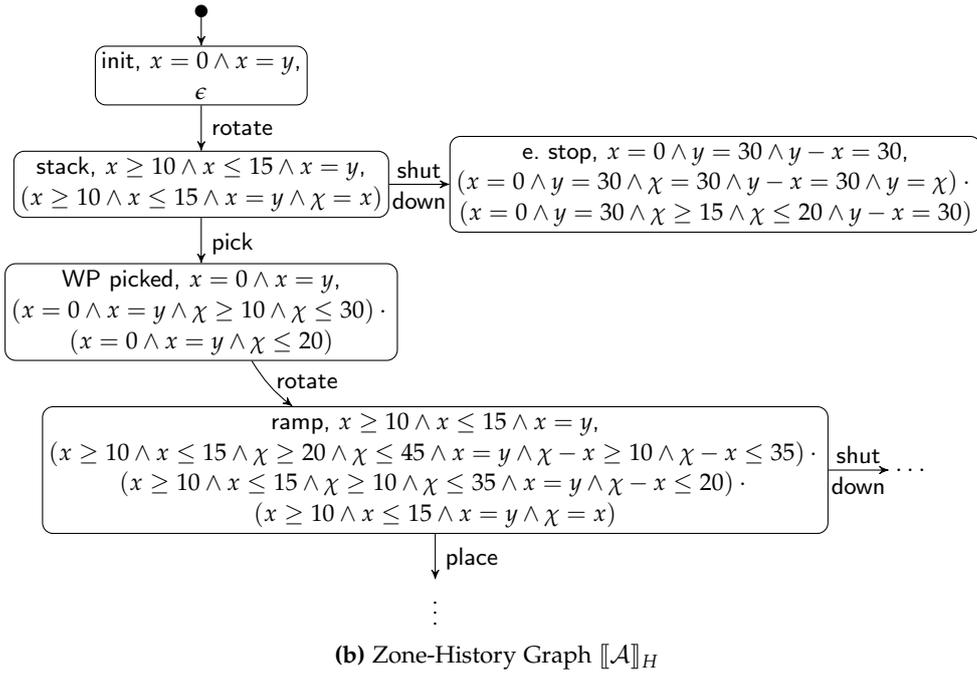
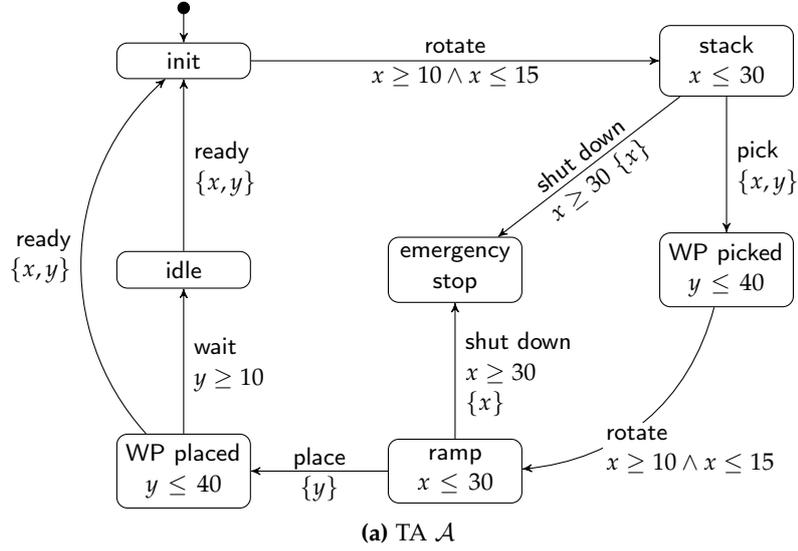


Figure 6.5: Example for the Zone-History Graph of the PPU

Having defined (possibly infinite) zone-history graph, we proceed by introducing Algorithm 6.1 for pruning zone-history graphs. As this pruning can be applied during the construction of zone-history graphs, we therewith can *effectively* check timed bisimilarity. To this end, we first introduce some auxiliary operators being used in Algorithm 6.1. Here, we use $|\mathcal{H}|$ to denote the *length* of history \mathcal{H} (i.e., the number of elements of the sequence) and $\mathcal{H}_{\downarrow k}$ to denote the postfix of length k of \mathcal{H} (or \mathcal{H} if $|\mathcal{H}| \leq k$).

Notation 6.1. Let $\mathcal{H} \in \mathcal{B}(C \cup \{\chi\})^*$ be a zone history with $\mathcal{H} = H_1 \cdots H_n$ and $n \in \mathbb{N}$. We utilize $|\mathcal{H}| = n$ to denote the *length* of \mathcal{H} . We utilize $\mathcal{H}_{\downarrow k}$ to denote the postfix of length k of \mathcal{H} , such that

$$\mathcal{H}_{\downarrow k} = \begin{cases} H_{n-k} \cdots H_n & \text{if } |\mathcal{H}| > k \\ \mathcal{H}, & \text{otherwise.} \end{cases}$$

Example 6.9. Consider the zone history corresponding to state *WP* picked of zone-history graph $\llbracket \mathcal{A} \rrbracket_H$ depicted in Figure 6.5b (see Example 6.8), such that

$$\mathcal{H} = (x = 0 \wedge x = y \wedge \chi \geq 10 \wedge \chi \leq 30) \cdot (x = 0 \wedge x = y \wedge \chi \leq 20).$$

In particular \mathcal{H} consists of two elements, resulting in $|\mathcal{H}| = 2$. Then, it holds that $\mathcal{H}_{\downarrow 3} = \mathcal{H}$ as $|\mathcal{H}| \leq 3$. Furthermore, $\mathcal{H}_{\downarrow 1}$ only considers the postfix of length 1, such that $\mathcal{H}_{\downarrow 1} = (x = 0 \wedge x = y \wedge \chi \leq 20)$.

We utilize these notations for comparing zone histories \mathcal{H} and \mathcal{H}' with $|\mathcal{H}| \neq |\mathcal{H}'|$ (i.e., zone histories having differing lengths). Without loss of generality, assume that $|\mathcal{H}| \leq |\mathcal{H}'|$, such that $k = |\mathcal{H}|$. Then, we compare the the history \mathcal{H} with $\mathcal{H}'_{\downarrow k}$. In particular, we utilize $\mathcal{H} \prec \mathcal{H}'$ and $\mathcal{H} \preceq \mathcal{H}'$ to denote an element-wise comparison of histories such that we compare the i th element of \mathcal{H} with the i th element of $\mathcal{H}'_{\downarrow k}$ (with $1 \leq i \leq k$). To this end, we compare the spans of the additional clock χ (see Definition 6.2). As clock χ is never reset, we therewith compare the spans in which we are allowed to reach the current state from predecessor states. Finally, it should be noted that we utilize the generic symbol $\sqsubseteq \in \{\prec, \preceq\}$ to obtain a more compact definition.

Definition 6.4 (Comparison of Zone Histories). Let $\mathcal{H}, \mathcal{H}' \in \mathcal{B}(C \cup \{\chi\})^*$ with $\chi \notin C$ be *zone histories*. The *comparison* of the *spans* of histories \mathcal{H} and \mathcal{H}' is recursively defined by

- $\mathcal{H} \sqsubseteq \mathcal{H}'$ if $\mathcal{H} = \mathcal{H}' = \epsilon$,
- $\mathcal{H} \sqsubseteq \mathcal{H}' \Leftrightarrow \text{span}(\chi, H) \sqsubseteq \text{span}(\chi, H') \wedge \mathcal{H}'' \sqsubseteq \mathcal{H}'''$ if $|\mathcal{H}| = |\mathcal{H}'| \wedge \mathcal{H} = H \cdot \mathcal{H}'' \wedge \mathcal{H}' = H' \cdot \mathcal{H}'''$, and
- $\mathcal{H} \sqsubseteq \mathcal{H}' \Leftrightarrow \mathcal{H}_{\downarrow k} \sqsubseteq \mathcal{H}'_{\downarrow k}$ if $|\mathcal{H}| \neq |\mathcal{H}'|$ and $k = \min(|\mathcal{H}|, |\mathcal{H}'|)$,

where $\sqsubseteq \in \{\prec, \preceq\}$.

Example 6.10. Consider the following zone histories where we only show clock constraints over clock χ (and omit all other constraints for the sake of readability):

- $\mathcal{H} = (\chi > 1) \cdot (\chi \geq 9) \cdot (\chi > 11 \wedge \chi \leq 47)$
- $\mathcal{H}' = (\chi > 8) \cdot (\chi > 11 \wedge \chi \leq 47)$
- $\mathcal{H}'' = (\chi \geq 23 \wedge \chi \leq 46)$

When comparing these zones, it holds that $\mathcal{H} \preceq \mathcal{H}'$, $\mathcal{H}'' \prec \mathcal{H}$, and $\mathcal{H}'' \prec \mathcal{H}'$. In contrast, $\mathcal{H} \preceq \mathcal{H}''$, $\mathcal{H}' \preceq \mathcal{H}''$, and $\mathcal{H}' \preceq \mathcal{H}$ do not hold.

In addition to the operators for zone histories described in Definition 6.4, we utilize $\mathcal{H} \asymp \mathcal{H}'$ to denote equality of zone histories \mathcal{H} and \mathcal{H}' . In particular, \mathcal{H} and \mathcal{H}' are equal if $\mathcal{H} \preceq \mathcal{H}'$ and $\mathcal{H}' \preceq \mathcal{H}$ (where we, again, only consider the postfix of the longer zone history).

Definition 6.5 (Postfix-Equality of Zone Histories). Let $\{\mathcal{H}, \mathcal{H}'\} \subseteq \mathcal{B}(C \cup \{\chi\})^*$ be zone histories. \mathcal{H} and \mathcal{H}' are equal, denoted by $\mathcal{H} \asymp \mathcal{H}'$, iff $\mathcal{H} \preceq \mathcal{H}'$ and $\mathcal{H}' \preceq \mathcal{H}$.

$\mathcal{H} \asymp_{\circ} \mathcal{H}'$ further *cuts* postfixes in case of periodic zone histories. We will explain the usage of this operator in more detail later on (see description of Algorithm 6.1).

Definition 6.6 (Cut-Equality of Zone Histories). Let $\{\mathcal{H}, \mathcal{H}'\} \subseteq \mathcal{B}(C \cup \{\chi\})^*$ with $\chi \notin C$ be zone histories. The *periodic comparison* of the spans of histories \mathcal{H} and \mathcal{H}' is recursively defined by

- $\mathcal{H} \trianglelefteq_{\circ} \mathcal{H}' \Leftrightarrow \mathcal{H}_{\downarrow k} \trianglelefteq \mathcal{H}'_{\downarrow k}$ if $k = \min(|\mathcal{H}|, |\mathcal{H}'|, |\omega|)$ with $\omega = |\mathcal{H}| - |\mathcal{H}'|$ and
- $\mathcal{H} \asymp_{\circ} \mathcal{H}'$ if $\mathcal{H} \trianglelefteq_{\circ} \mathcal{H}'$ and $\mathcal{H}' \trianglelefteq_{\circ} \mathcal{H}$,

where $\trianglelefteq \in \{\prec, \preceq\}$.

Example 6.11. Consider, again, the zone histories \mathcal{H} , \mathcal{H}' , and \mathcal{H}'' as described in Example 6.10. Here, it does not hold that $\mathcal{H} \asymp \mathcal{H}'$ as $\mathcal{H}' \preceq \mathcal{H}$ does not hold. In contrast, it holds that $\mathcal{H} \asymp_{\circ} \mathcal{H}'$ as this operator only compares the postfix of length $|\mathcal{H}| - |\mathcal{H}'| = 1$ as opposed to $\mathcal{H} \asymp \mathcal{H}'$ considering the postfix of length $\min(|\mathcal{H}|, |\mathcal{H}'|) = 2$. In particular, the second last elements of \mathcal{H} and \mathcal{H}' are differing which is revealed by \asymp but not by \asymp_{\circ} .

Having defined (possibly infinite) zone-history graphs and operators for comparing zone histories, we can now proceed by introducing a procedure for obtaining *finite* zone-history graphs, being the basis for an effective timed-bisimilarity check. To this end, Algorithm 6.1 takes as input a (possibly infinite) zone-history graph $(\mathcal{Z}, z_0, \Sigma, \rightsquigarrow)$ and returns a finite zone-history graph $(\mathcal{Z}', z_0, \Sigma, \rightsquigarrow')$ having equivalent behavior. In particular, the algorithm utilizes the comparison operator

Algorithm 6.1: Generating Finite Zone-History Graphs

Input : zone-history graph $(\mathcal{Z}, z_0, \Sigma, \rightsquigarrow)$
Output : finite zone-history graph $(\mathcal{Z}', z_0, \Sigma, \rightsquigarrow')$

```

1 procedure MAIN
2    $\rightsquigarrow' := \emptyset$ 
3    $\mathcal{Z}' := \{z_0\}$ 
4    $\widehat{\mathcal{Z}} := \{z_0\}$ 
5   while  $\widehat{\mathcal{Z}} \neq \emptyset$  do
6      $z \leftarrow \widehat{\mathcal{Z}}$  // pick and remove element
7     foreach  $z \xrightarrow{\sigma} z'$  do
8       if  $\exists \langle \ell, D, \mathcal{H}'' \rangle \in \mathcal{Z}' : (\mathcal{H} \prec_{\circ} \mathcal{H}'' \wedge \mathcal{H} \neq \mathcal{H}'') \wedge$ 
9          $\exists \langle \ell', D', \mathcal{H}''' \rangle \in \mathcal{Z}' : \mathcal{H}''' \prec_{\circ} \mathcal{H}'$  then
10         $\rightsquigarrow' := \rightsquigarrow' \cup \{z \xrightarrow{\sigma} \langle \ell', D', \mathcal{H}''' \rangle\}$ 
11      else
12         $\rightsquigarrow' := \rightsquigarrow' \cup \{z \xrightarrow{\sigma} z'\}$ 
13         $\mathcal{Z}' := \mathcal{Z}' \cup \{z'\}$ 
14         $\widehat{\mathcal{Z}} := \widehat{\mathcal{Z}} \cup \{z'\}$ 
15    return  $(\mathcal{Z}', z_0, \Sigma, \rightsquigarrow')$ 

```

described in Definition 6.6 (i.e., $\mathcal{H} \prec_{\circ} \mathcal{H}'$) for cutting zone history to stop unrolling of cyclic behavior. The algorithm starts by initializing the transition relation \rightsquigarrow' , the set of states \mathcal{Z}' , and the working set $\widehat{\mathcal{Z}}$ containing states which have not yet been processed (see lines 2–4). Then, we utilize the main loop of Algorithm 6.1 to iterate over $\widehat{\mathcal{Z}}$ until $\widehat{\mathcal{Z}} = \emptyset$ (lines 5–13). In this while-loop, we first pick a state $z \in \widehat{\mathcal{Z}}$ and remove it from $\widehat{\mathcal{Z}}$ (line 6). Then, we iterate over all outgoing transition of z in the original zone-history graph (line 7) and check two conditions for each $z \xrightarrow{\sigma} z'$ (line 8).

1. Does there already exist a state $\langle \ell, D, \mathcal{H}'' \rangle \in \mathcal{Z}'$ satisfying $\mathcal{H} \prec_{\circ} \mathcal{H}''$ and $\mathcal{H} \neq \mathcal{H}''$? Therewith, we check whether we have already reached a state with an equivalent history (w.r.t. \prec_{\circ}) in a previous step. Furthermore, we check $\mathcal{H} \neq \mathcal{H}''$ to explicitly exclude the current state and find a state $z \neq \langle \ell, D, \mathcal{H}'' \rangle$.
2. Does there already exist a state $\langle \ell', D', \mathcal{H}''' \rangle \in \mathcal{Z}'$ satisfying $\mathcal{H}' \prec_{\circ} \mathcal{H}'''$? Therewith, we check whether there also exists a state with a history being equivalent to the target state of the current transition $z \xrightarrow{\sigma} z'$.

For both checks, we utilize operator \prec_{\circ} (see Definition 6.6) for history comparison as this operator only compares the postfix of histories reaching back to the last iteration in case of cyclic behavior (i.e., *cutting* histories in case of regularity). If this is the case, we add a transition from z to the already existing state $\langle \ell', D', \mathcal{H}''' \rangle \in \mathcal{Z}'$ (line 9). As a result, we cut the history and prevent further unrolling of cyclic behavior, such that the zone-history graph becomes finite. If the conditions are not satisfied (lines 10–13), we add the current transition to \rightsquigarrow' (line 11) and the

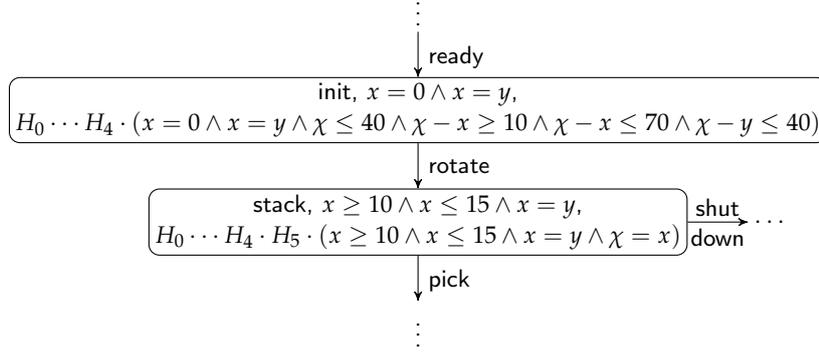


Figure 6.6: Example for Generating the Finite Zone-History Graph of the PPU

target state z' to \mathcal{Z}' (line 12) and working set $\widehat{\mathcal{Z}}$ (line 13). When working set $\widehat{\mathcal{Z}}$ is finally empty, the while-loop terminates and we return the *finite* zone-history graph $(\mathcal{Z}', z_0, \Sigma, \rightsquigarrow')$ (line 14). Intuitively, Algorithm 6.1 always terminates (resulting in a finite zone-history graph) as unrolling loops always eventually results in an identical postfix in the respective zone history (such that zone history are equivalent w.r.t. \simeq_{\circ}).

Example 6.12 (Finite Zone-History Graph of the PPU). Consider again the (infinite) zone history graph $\llbracket \mathcal{A} \rrbracket_H$ of TA \mathcal{A} depicted in Figure 6.5 (as presented in Example 6.8). When returning to the initial location from the location *WP placed* after the first iteration of the cyclic behavior of TA \mathcal{A} , we add a new symbolic state comprising location *init*, zone $x = 0 \wedge x = y$ (i.e., the same as in the initial state), and the history depicted in Figure 6.6. Here, H_0 to H_4 are the history elements obtained in the previous steps. Then, traversing the switch labeled with *rotate* for the seconds time results in a new history element

$$(x \geq 10 \wedge x \leq 15 \wedge x = y \wedge \chi = x)$$

which is identical to the zone history of the first iteration through TA \mathcal{A} and appended to the zone history during the update. In particular, we observe this repeated addition of identical zone-history elements also for the subsequent state. Hence, when we reach line 8 of Algorithm 6.1, we check whether $\mathcal{H} \simeq_{\circ} \mathcal{H}''$ holds. Here, \mathcal{H} is the zone history of the current state, where we just appended the new element. Furthermore, \mathcal{H}'' is the state comprising location *stack*. As $\min(|\mathcal{H}|, |\mathcal{H}''|, |\mathcal{H}| - |\mathcal{H}''|) = 1$, we only compare the (identical) postfix of \mathcal{H} and \mathcal{H}'' of length 1, such that these postfixes are equivalent w.r.t. \simeq_{\circ} . As a result, the condition in line 8 of Algorithm 6.1 is satisfied and we cut the zone history by adding a transition to the already existing symbolic state, thus obtaining a finite zone-history graph.

Note, that in practice we directly generate a finite zone-history graph for a given TA (in contrast to first generating a possibly infinite zone-history graph as described in this section). This can be achieved with on-the-fly checks of zone-history equality w.r.t. \simeq_{\circ} whenever we potentially add a new symbolic state (see line 8 of

Algorithm 6.1). Next, we formally prove that any zone-history graph obtained by applying Algorithm 6.1 is always finite.

Proposition 6.1. Let \mathcal{A} be a TA. Then, $\llbracket \mathcal{A} \rrbracket_H$ is finite.

Proof. Let $\mathcal{A} = (L, \ell_0, \Sigma, C, I, E)$ be a TA and $\llbracket \mathcal{A} \rrbracket_H = (\mathcal{Z}, z_0, \Sigma, \rightsquigarrow)$ be the zone-history graph obtained by applying Algorithm 6.1. Zone graphs $(\mathcal{Z}, z_0, \rightsquigarrow)$ (without histories) are not necessarily finite but it has been shown that an equivalent finite zone graph $(\mathcal{Z}, z_0, \rightsquigarrow_k)$ can be obtained by constructing a k -bounded zone graph with all zones being bound by a maximum global *clock ceiling* k using k -normalization [167, 161] (see Section 2.5.1). Hence, it remains to be shown that when adding histories, k -normalized zone-history graphs $(\mathcal{Z}, z_0, \Sigma, \rightsquigarrow_k)$ remain finite. Here, histories \mathcal{H} are constructed in a way such that \mathcal{H} is eventually cut. In particular, whenever there already exists a state with the same location $\ell \in L$ and an equivalent zone $D \in \mathcal{B}(C)$, we check if $\mathcal{H} \simeq_{\circlearrowleft} \mathcal{H}'$ and do not add a new state in this case (i.e., we add a transition to the existing state $\langle \ell, D, \mathcal{H} \rangle$ instead of adding the new state $\langle \ell, D, \mathcal{H}' \rangle$, see lines 8–9 of Algorithm 6.1). To this end, $\mathcal{H} \simeq_{\circlearrowleft} \mathcal{H}'$ compares the postfix of \mathcal{H} and \mathcal{H}' of length $n = \min(|\mathcal{H}|, |\mathcal{H}'|, |\omega|)$ with $\omega = |\mathcal{H}| - |\mathcal{H}'|$ (see Definition 6.6). As a result, we only compare the newest n elements of a history when unrolling a loop, where n is the number of locations on the loop. Therefore, the history eventually becomes regular as we only compare the postfix of length n and TA are *finite* state-transition graphs (see Definition 2.2). \square

Having defined (finite) zone-history graphs and operators for comparing zone histories, we can proceed by establishing a symbolic notion of (strong and weak) timed bisimilarity based on zone-history graphs. As done for bisimilarity based on TLTS, we relate symbolic states with each other. In particular, state z'_1 (of zone-history graph $\llbracket \mathcal{A}' \rrbracket_H$) simulates state z_1 (of zone-history graph $\llbracket \mathcal{A} \rrbracket_H$) if the following three conditions are satisfied.

1. z'_1 enables *at least* the same actions $\sigma \in \Sigma_{\tau}$. This corresponds to the classical requirement also used for untimed bisimulation (see Definition 2.22).
2. The span for residing in z'_1 is at least as long as the span for residing in z_1 . We achieve this comparison by utilizing a fresh clock χ (for which we check the span) and including the invariant of the respective locations in this check.
3. We check if it holds that $\mathcal{H}_1 \preceq \mathcal{H}'_1$. Therewith, we compare the allowed spans for reaching the current pair of the states from their predecessors.

Definition 6.7 (Symbolic Timed Bisimulation). Let \mathcal{A} and \mathcal{A}' be TA with $\llbracket \mathcal{A} \rrbracket_H = (\mathcal{Z}, z_0, \Sigma, \rightsquigarrow)$, $\llbracket \mathcal{A}' \rrbracket_H = (\mathcal{Z}', z'_0, \Sigma, \rightsquigarrow')$, and $\mathcal{R} \subseteq \mathcal{Z} \times \mathcal{Z}'$. Then \mathcal{A}' *strongly timed simulates* \mathcal{A} if it holds for all $(z_1, z'_1) \in \mathcal{R}$ that

- $z_1 \xrightarrow{\sigma} z_2 \Rightarrow \left(z'_1 \xrightarrow{\sigma} z'_2 \wedge (z_2, z'_2) \in \mathcal{R} \right)$,
- $\text{span}(\chi, (D_1 \wedge \chi = 0)^\uparrow \wedge I(\ell_1)) \leq \text{span}(\chi', (D'_1 \wedge \chi' = 0)^\uparrow \wedge I'(\ell'_1))$, and

$$\blacksquare \mathcal{H}_1 \preceq \mathcal{H}'_1,$$

where $\sigma \in \Sigma$ and $(z_0, z'_0) \in \mathcal{R}$. \mathcal{A}' weakly timed simulates \mathcal{A} if it holds for all $(z_1, z'_1) \in \mathcal{R}$ that

$$\blacksquare z_1 \xrightarrow{\sigma} z_2 \Rightarrow \left(z'_1 \xrightarrow{\sigma} z'_2 \wedge (z_2, z'_2) \in \mathcal{R} \right),$$

$$\blacksquare \text{span}(\chi, (D_1 \wedge \chi = 0)^\uparrow \wedge I(\ell_1)) \leq \text{span}(\chi', (D'_1 \wedge \chi' = 0)^\uparrow \wedge I'(\ell'_1)), \text{ and}$$

$$\blacksquare \mathcal{H}_1 \preceq \mathcal{H}'_1,$$

and $(z_0, z'_0) \in \mathcal{R}$. We use $\mathcal{A} \sqsubseteq \mathcal{A}'$ to denote that \mathcal{A}' weakly/strongly timed simulates \mathcal{A} . \mathcal{A}' and \mathcal{A} are weakly/strongly timed bisimilar, denoted by $\mathcal{A} \simeq \mathcal{A}'$, iff \mathcal{R} is symmetric.

Example 6.13 (Symbolic Timed Bisimulation). Consider the development process of the PPU, where TA \mathcal{A} and zone-history graph $\llbracket \mathcal{A} \rrbracket_H$ as depicted in Figure 6.5 correspond to the original model (see Example 6.8 for a description). Furthermore, TA \mathcal{A}' (as already presented in the background chapter) and zone-history graph $\llbracket \mathcal{A}' \rrbracket_H$ depicted in Figure 6.7 correspond to a refined PPU model obtained after several development steps. Note, that $\llbracket \mathcal{A}' \rrbracket_H$ only shows the weak transition relation where action τ is not displayed. As we want to improve our model during the development process without changing its behavior, we utilize (weak) timed bisimulation to verify equality of these models.

To achieve this goal, we start by relating the initial states of both zone-history graphs. Then, we compare the actions of outgoing transitions for each pair of states (which are equal in each step). Hence, we assume that

$$\mathcal{R} = \{(z_0, z'_0)\}$$

where z_0 and z'_0 are the initial states of $\llbracket \mathcal{A} \rrbracket_H$ and $\llbracket \mathcal{A}' \rrbracket_H$, respectively. Furthermore, we check equality of histories w.r.t. \approx in each step of this process. To this end, we perform an element-by-element comparison of histories by comparing the respective spans of the special clock χ . For instance, we have

$$\mathcal{R} = \{(z_0, z'_0), (z_1, z'_1)\}$$

for the next pair of states (i.e., z_1 and z'_1 correspond to the states comprising location *stack*). Here, we also observe, that these spans are equal (for the state pairs described above and all following state pairs). Hence, it holds that $\mathcal{A} \simeq \mathcal{A}'$ for weak timed bisimilarity. However, it should be noted that \mathcal{A} and \mathcal{A}' are not strongly timed bisimilar due to the τ -step of \mathcal{A}' .

With timed bisimulation defined on TLTS (see Definition 2.23) as well as zone-history graphs (see Definition 6.7), we now have two ways for checking timed (bi-)similarity. If necessary for the context, we use $\mathcal{A} \sqsubseteq_T \mathcal{A}'$ to denote timed simulation based on TLTS and $\mathcal{A} \sqsubseteq_Z \mathcal{A}'$ to denote timed simulation based on zone-history graphs.

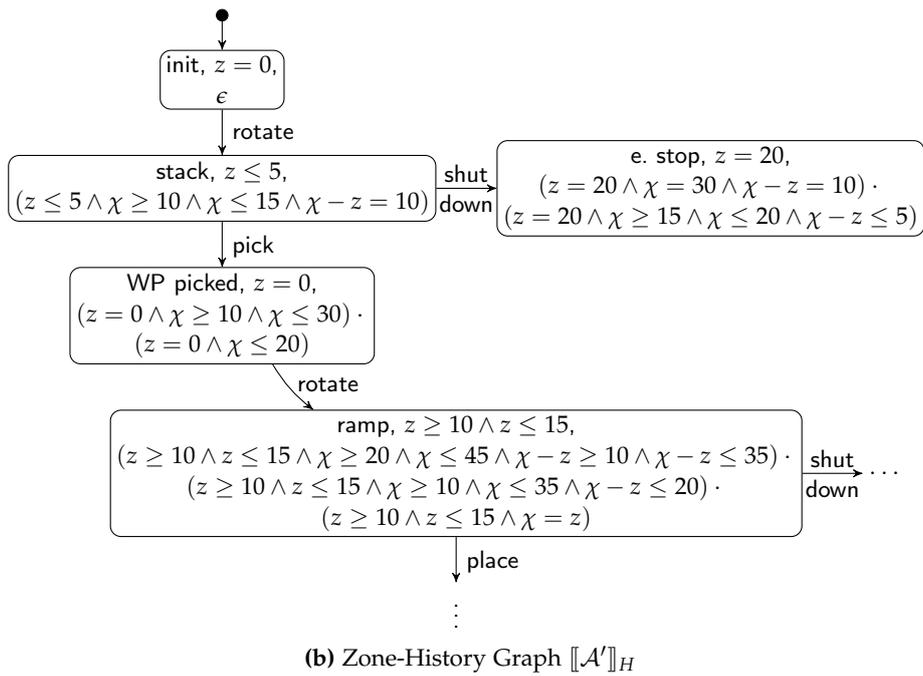
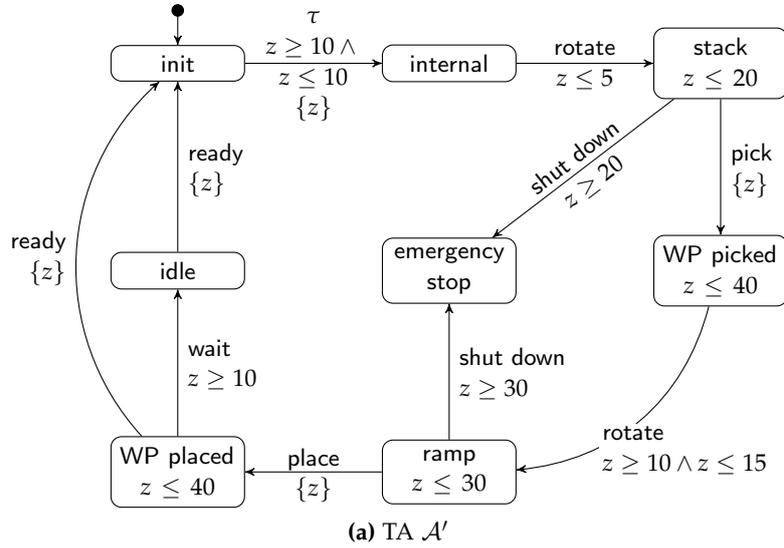


Figure 6.7: Example for Symbolic Timed Bisimulation (with $\llbracket \mathcal{A}' \rrbracket_H$ in Figure 6.7b Displaying the Weak Transition Relation)

Notation 6.2. Let \mathcal{A} and \mathcal{A}' be TA. We use $\mathcal{A} \sqsubseteq_T \mathcal{A}'$ to denote timed simulation according to Definition 2.23 and $\mathcal{A} \sqsubseteq_Z \mathcal{A}'$ to denote timed simulation according to Definition 6.7.

As a next step, we prove correctness of timed bisimulation for deterministic TA based on zone-history graphs w.r.t. timed bisimulation based on TLTS (i.e., both checks always have the same results). Moreover, we also show that checking timed bisimilarity on zone-history graphs is decidable (again, for deterministic TA). Note, that we generalize these results to non-deterministic TA in Section 6.1.3, where we also generalize the notion of zone-history graphs, such that this representation of TA semantics can be utilized for checking non-deterministic models.

Theorem 6.1. Let $\mathcal{A} = (L, \ell_0, \Sigma, C, I, E)$ and $\mathcal{A}' = (L', \ell'_0, \Sigma, C', I', E')$ be deterministic TA. Then, it holds that

1. $\mathcal{A} \sqsubseteq_T \mathcal{A}' \Leftrightarrow \mathcal{A} \sqsubseteq_Z \mathcal{A}'$ and
2. $\mathcal{A} \sqsubseteq_Z \mathcal{A}'$ is decidable.

Proof. Let $\mathcal{A} = (L, \ell_0, \Sigma, C, I, E)$ and $\mathcal{A}' = (L', \ell'_0, \Sigma, C', I', E')$ be deterministic TA. We prove the two parts of Theorem 6.1 separately.

1. We have to consider that clock resets hide clock constraints in the sense that a clock constraint $x \sim n$ is not visible in a zone after x is reset. However, by comparing zone histories \mathcal{H} and \mathcal{H}' , we ensure that the impact of previous clock constraints remains observable by using the fresh clock χ for tracking respective changes to clock differences including those potentially being hidden by subsequent clock resets. Hence, the histories describe exactly the allowed time frames for actions of TA \mathcal{A} and \mathcal{A}' as also described in the TLTS semantics in terms of $\llbracket \mathcal{A} \rrbracket_S$ and $\llbracket \mathcal{A}' \rrbracket_S$. Note, that k -normalization does not impact the bisimilarity check as checking timed bisimilarity relies on the comparison of histories. In particular, loops (being the reason for k -normalization) result in the comparison of the postfix of length $n = \min(|\mathcal{H}|, |\mathcal{H}'|, |\omega|)$ with $\omega = |\mathcal{H}| - |\mathcal{H}'|$ of histories \mathcal{H} and \mathcal{H}' (see Def. 6.2). As a result, we only compare the newest n elements of a history when unrolling a loop, where n is the number of locations on the loop. Hence, the history eventually becomes regular as we only compare the postfix of length n , such that we do not apply any approximation to histories. Therefore, it holds that $\mathcal{A} \sqsubseteq_T \mathcal{A}' \Leftrightarrow \mathcal{A} \sqsubseteq_Z \mathcal{A}'$.
2. As zone-history graphs are finite (see Proposition 6.1), there are finitely many transitions and spans to check (see Def. 6.7). Hence, $\mathcal{A} \sqsubseteq_Z \mathcal{A}'$ is decidable. \square

So far, we have introduced zone-history graphs as a basis for *effectively* checking timed bisimilarity of deterministic TA models. Moreover, we defined a check for timed bisimilarity based on the representation of zone-history graphs, and we proved correctness of this check w.r.t. timed bisimilarity based on TLTS. Next, we further generalize zone-history graphs such that we can utilize timed bisimulation as described in Definition 6.7 also for *non-deterministic* TA models with internal

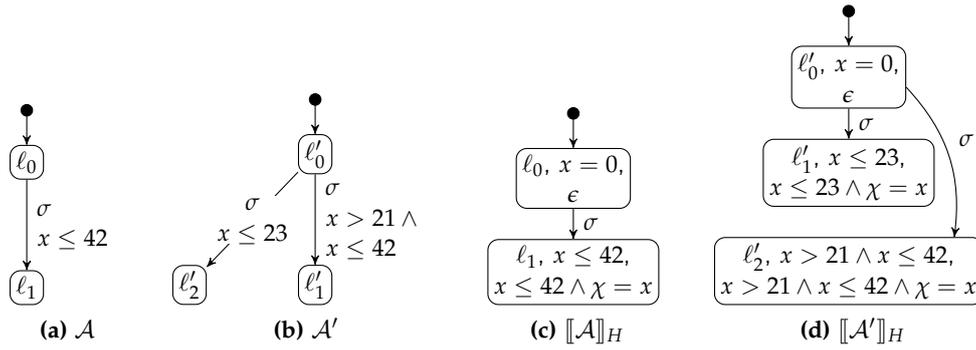


Figure 6.8: Example for the Problem of Checking Timed Bisimilarity of Non-Deterministic TA with Zone-History Graphs

behavior in terms of τ -steps. Intuitively, this is not possible by using zone-history graphs and timed bisimilarity as defined in this section, as this approach does not cover the case where a single switch of a TA \mathcal{A} is simulated by multiple switches of another TA \mathcal{A}' . In particular, we cannot simply combine the switches of \mathcal{A}' as we also have to consider subsequent behavior.

6.1.3 Checking Timed Bisimilarity of Non-Deterministic Timed Automata

In the previous section, we introduced an approach for effectively checking timed bisimilarity based on zone-history graphs for deterministic TA. However, this approach cannot be applied for *non*-deterministic TA, as illustrated by the following example.

Example 6.14. Consider TA \mathcal{A} and \mathcal{A}' together with their corresponding zone-history graphs as depicted in Figure 6.8. Intuitively, \mathcal{A} and \mathcal{A}' are bisimilar as the two switches of \mathcal{A}' simulate exactly the behavior of \mathcal{A} , and vice versa. However, utilizing timed bisimulation as described in Definition 6.7 would yield $\mathcal{A}' \sqsubseteq \mathcal{A}$ but not $\mathcal{A} \sqsubseteq \mathcal{A}'$ as a result, as neither of the transitions of zone-history graph $\llbracket \mathcal{A}' \rrbracket_H$ can simulate the transition of zone-history graph $\llbracket \mathcal{A} \rrbracket_H$ (and timed bisimulation as described in Definition 6.7 does not consider the combined behavior of multiple transitions).

In order to overcome this issue, we generalize the model of zone-history graphs to *composite zone-history graphs*. The basic idea for constructing composite zone-history graphs consists of splitting symbolic states such that the common behavior of the systems under consideration is expressed in the same way for both systems. This is achieved in two steps. First, we construct the zone-history graph $\llbracket \mathcal{A} \times \mathcal{A}' \rrbracket_H$ for the synchronous parallel product $\mathcal{A} \times \mathcal{A}'$ (comprising exactly the shared behavior of TA \mathcal{A} and \mathcal{A}'). Second, the composite zone history graph $\llbracket \mathcal{A} \otimes \mathcal{A}' \rrbracket_H$ (indicated by symbol \otimes instead of \times) of \mathcal{A} w.r.t. \mathcal{A}' extends zone-history graph $\llbracket \mathcal{A} \times \mathcal{A}' \rrbracket_H$ by the behavior being exclusive to \mathcal{A} . Then, we can apply timed bisimulation as described in Definition 6.7 for checking non-deterministic TA. In the following, we first define the synchronous parallel product $\mathcal{A} \times \mathcal{A}'$ as usual.

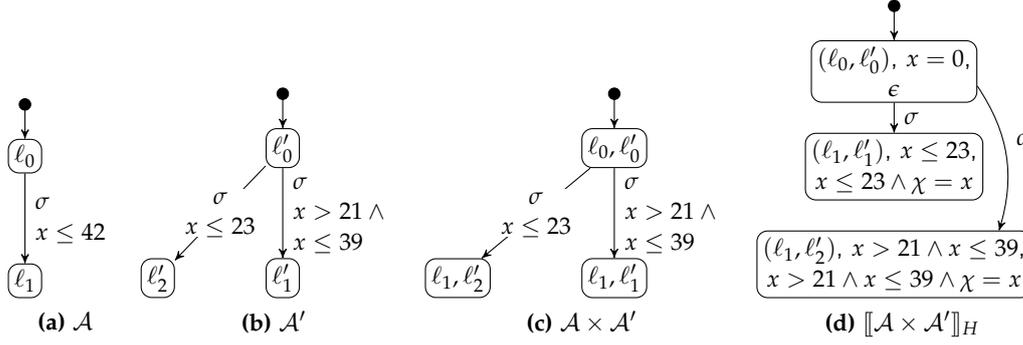


Figure 6.9: Example for the Construction of a Parallel Product

Definition 6.8 (Parallel Product). Let $\mathcal{A} = (L, \ell_0, \Sigma, C, I, E)$ and $\mathcal{A}' = (L', \ell'_0, \Sigma, C', I', E')$ be TA with $C \cap C' = \emptyset$. Then, the *parallel product*

$$\mathcal{A} \times \mathcal{A}' = (L \times L', (\ell_0, \ell'_0), \Sigma, C \cup C', I_{\times}, E_{\times})$$

is a TA with $I_{\times}((\ell, \ell')) = I(\ell) \wedge I(\ell')$ and E_{\times} being the least relation satisfying the following rules:

- (1) $(\ell_1, \ell'_1) \xrightarrow{g \wedge g', \sigma, R \cup R'}_{\times} (\ell_2, \ell'_2) \in E_{\times}$ if $\ell_1 \xrightarrow{g, \sigma, R} \ell_2 \in E \wedge \ell'_1 \xrightarrow{g', \sigma, R'} \ell'_2 \in E'$,
- (2) $(\ell_1, \ell'_1) \xrightarrow{g, \tau, R}_{\times} (\ell_2, \ell'_1) \in E_{\times}$ if $\ell_1 \xrightarrow{g, \tau, R} \ell_2 \in E$, and
- (3) $(\ell_1, \ell'_1) \xrightarrow{g', \tau, R'}_{\times} (\ell_1, \ell'_2) \in E_{\times}$ if $\ell'_1 \xrightarrow{g', \tau, R'} \ell'_2 \in E'$.

Example 6.15 (Parallel Product). Consider TA \mathcal{A} and \mathcal{A}' depicted in Figure 6.9. Here, \mathcal{A} is identical to the TA considered in Example 6.14 and depicted in Figure 6.8, whereas \mathcal{A}' is slightly adapted such that the right switch has an upper bound of 39 seconds. Furthermore, Figure 6.9c depicts the parallel product $\mathcal{A} \times \mathcal{A}'$. To construct this parallel product, we combine every switch of location ℓ_0 of \mathcal{A} with every switch of ℓ'_0 of \mathcal{A}' as all switches are labeled with the same action. Furthermore, we conjugate the guards, for instance resulting in $x \leq 42 \wedge x \leq 23$ which can be simplified to $x \leq 23$. Finally, the zone-history graph $\llbracket \mathcal{A} \times \mathcal{A}' \rrbracket_H$ of $\mathcal{A} \times \mathcal{A}'$ is depicted in Figure 6.9d. Here, we observe that the behavior of $\llbracket \mathcal{A} \times \mathcal{A}' \rrbracket_H$ does not cover action σ in the interval $39 < x \leq 42$, as this is behavior being exclusive to \mathcal{A} (and not being part of \mathcal{A}'). Hence, we need to add further behavior $\llbracket \mathcal{A} \times \mathcal{A}' \rrbracket_H$ before it can be used as a basis for checking timed bisimilarity.

As illustrated by Example 6.15, we need to add further behavior to $\llbracket \mathcal{A} \times \mathcal{A}' \rrbracket_H$ in terms of exclusive behavior of \mathcal{A} for constructing the (generally non-symmetric) composite zone-history graph $\llbracket \mathcal{A} \otimes \mathcal{A}' \rrbracket_H$ of \mathcal{A} w.r.t. \mathcal{A}' . For this purpose, we first introduce some auxiliary definitions and notations to keep the definition of composite zone-history graphs compact and readable. First, we present two auxiliary transition relations, where $\rightsquigarrow_{\times}$ denotes the transition relation of $\llbracket \mathcal{A} \times \mathcal{A}' \rrbracket_H$ and \rightsquigarrow_1 denotes the transition relation of $\llbracket \mathcal{A} \rrbracket_H$.

Notation 6.3. Let \mathcal{A} and \mathcal{A}' be TA with $\llbracket \mathcal{A} \rrbracket_H = (\mathcal{Z}, z_0, \Sigma, \rightsquigarrow)$ and $\llbracket \mathcal{A} \times \mathcal{A}' \rrbracket_H = (\mathcal{Z}', z'_0, \Sigma, \rightsquigarrow')$. Then, the *auxiliary transitions relations* of \mathcal{A} w.r.t. \mathcal{A}' are defined as

- $\rightsquigarrow_1 = \rightsquigarrow$ being the transition relation of $\llbracket \mathcal{A} \rrbracket_H$ and
- $\rightsquigarrow_x = \rightsquigarrow'$ being the transition relation of $\llbracket \mathcal{A} \times \mathcal{A}' \rrbracket_H$.

As described above, we need to check whether the parallel product $\mathcal{A} \times \mathcal{A}'$ already contains all behavior of \mathcal{A} to ensure that the composite zone-history graph $\llbracket \mathcal{A} \otimes \mathcal{A}' \rrbracket_H$ covers all behavior of \mathcal{A} . In order to identify whether the behavior of a switch of TA \mathcal{A} is covered by a *combination* of switches of $\mathcal{A} \times \mathcal{A}'$, we utilize zone-history graphs $\llbracket \mathcal{A} \rrbracket_H$ and $\llbracket \mathcal{A} \times \mathcal{A}' \rrbracket_H$. Here, we *join* histories of $\llbracket \mathcal{A} \times \mathcal{A}' \rrbracket_H$ being reachable with the same action by applying element-by-element disjunction. Therewith, we obtain a single history such that we can check if the respective history of $\llbracket \mathcal{A} \rrbracket_H$ is contained in the *joint* history of $\llbracket \mathcal{A} \times \mathcal{A}' \rrbracket_H$. In particular, function *join* takes as input a set of histories \mathfrak{H} and applies a pairwise join of histories until we obtain a single history covering the spans of all histories in \mathfrak{H} . The join itself is achieved by applying element-by-element disjunction of two histories. In the following definition, we first consider the case where all histories in \mathfrak{H} have the same length.

Definition 6.9 (History Join). Let $\mathfrak{H} \in 2^{\mathcal{B}(C)^*}$ be a set of histories. Function $\text{join} : 2^{\mathcal{B}(C)^*} \rightarrow \mathcal{B}(C)^*$ is recursively defined by

- $\text{join}(\emptyset) = \epsilon$,
- $\text{join}(\{\mathcal{H}\} \cup \mathfrak{H}) = \mathcal{H} \dot{\vee} \text{join}(\mathfrak{H})$,
- $H \dot{\vee} \epsilon = H$, and
- $(H \cdot \mathcal{H}) \dot{\vee} (H' \cdot \mathcal{H}') = (H \dot{\vee} H') \cdot (\mathcal{H} \dot{\vee} \mathcal{H}')$ if $|\mathcal{H}| = |\mathcal{H}'|$.

It should be noted that disjunction can lead to constraints corresponding to non-convex polyhedra (in contrast to clock constraints described in Definition 2.1 being limited to convex polyhedra). Here, comparing non-convex polyhedra is less efficient than comparing convex polyhedra, such that constructing composite zone-history graphs is computationally more expensive than constructing zone-history graphs. However, the non-convex constraints only occur during the construction of composite zone-history graphs where we check if additional states must be added. In contrast, all constraints occurring in the composite zone-history graph itself (i.e., zones and elements of zone histories) are always convex.

Example 6.16 (History Join). Consider again the zone-history graph $\llbracket \mathcal{A} \times \mathcal{A}' \rrbracket_H$ depicted in Figure 6.9d. Here, we apply the join of the two histories being reached with action σ . Hence, we apply

$$\text{join}(\{(x \leq 23 \wedge \chi = x), (x > 21 \wedge x \leq 39 \wedge \chi = x)\}).$$

As both histories only have a single element, we simply connect these elements by disjunction, resulting in

$$(x \leq 23 \wedge \chi = x) \vee (x > 21 \wedge x \leq 39 \wedge \chi = x)$$

which can be simplified to $x \leq 39 \wedge \chi = x$ (describing a convex polyhedron). Furthermore, consider the two histories $\mathcal{H} = (x \leq 2 \wedge \chi = x)$ and $\mathcal{H}' = (x \geq 4 \wedge \chi = x)$ as another example. Here, $\text{join}(\{\mathcal{H}, \mathcal{H}'\})$ results in a non-convex polyhedron as the values $2 < x < 4$ are not valid results.

Next, we generalize Definition 6.16 such that we can also join history \mathcal{H} and \mathcal{H}' with different lengths (i.e., $|\mathcal{H}| \neq |\mathcal{H}'|$). Without loss of generality, assume that $|\mathcal{H}| > |\mathcal{H}'|$. Then, we expand the shorter history \mathcal{H}' to length $k = |\mathcal{H}|$, denoted by $\mathcal{H}'^{\uparrow k}$, by adding elements $\text{false} \in \mathcal{B}(C)$ as prefix until $|\mathcal{H}'| = k$. In particular, we use false as this is the neutral element w.r.t. disjunction. Note, that we expand history \mathcal{H}' by adding a prefix (instead of a postfix) as updating zone histories (see Definition 6.2) appends new elements, such that the last element of a histories is always the newest element. Hence, adding a prefix to the shorter history means that a comparison $\mathcal{H} \preceq \mathcal{H}'$ still compares the newest elements of \mathcal{H} with the newest elements of \mathcal{H}' .

Definition 6.10. Let $\mathcal{H}, \mathcal{H}' \in \mathcal{B}(C \cup \{\chi\})^*$ be zone histories.

- $\mathcal{H} \dot{\vee} \mathcal{H}' \Leftrightarrow \mathcal{H}^{\uparrow k} \dot{\vee} \mathcal{H}'^{\uparrow k}$ if $|\mathcal{H}| \neq |\mathcal{H}'|$ and $k = \max(|\mathcal{H}|, |\mathcal{H}'|)$,
- $\mathcal{H}^{\uparrow k} = \mathcal{H}$ if $|\mathcal{H}| \geq k$, and
- $\mathcal{H}^{\uparrow k} = \text{false} \cdot \mathcal{H}^{\uparrow k-1}$ if $|\mathcal{H}| < k$.

Example 6.17. Consider the following histories as an example.

- $\mathcal{H} = (x \leq 7 \wedge \chi = x) \cdot (x \leq 23 \wedge \chi = x)$ and
- $\mathcal{H}' = (x > 21 \wedge x \leq 39 \wedge \chi = x)$.

Here, we have the lengths $|\mathcal{H}| = 2$ and $|\mathcal{H}'| = 1$, such that \mathcal{H}' is shorter. Applying the join operator yields

$$((x \leq 7 \wedge \chi = x) \vee \text{false}) \cdot ((x \leq 23 \wedge \chi = x) \vee (x > 21 \wedge x \leq 39 \wedge \chi = x))$$

as a result, which can be simplified to $(x \leq 7 \wedge \chi = x) \cdot (x \leq 39 \wedge \chi = x)$.

For a compact definition of composite zone-history graphs, we furthermore define function *histories* which takes as input a symbolic state z , symbolic transition relation \rightsquigarrow , and an action $\sigma \in \Sigma_{\tau}$. Then, this function returns the histories of every symbolic state being reachable from state z with action σ .

Definition 6.11. Let $z \in \mathcal{Z} = L \times \mathcal{B}(C) \times \mathcal{B}(C)^*$ be a symbolic state, $\rightsquigarrow \in \mathcal{Z} \times \Sigma_\tau \times \mathcal{Z}$ be a symbolic transition relation, and $\sigma \in \Sigma_\tau$ be an action. Function

$$\text{histories} : \mathcal{Z} \times (\mathcal{Z} \times \Sigma_\tau \times \mathcal{Z}) \times \Sigma_\tau \rightarrow 2^{\mathcal{B}(C)^*}$$

denotes the set of histories $\mathfrak{H} \in 2^{\mathcal{B}(C)^*}$ being reachable from z with σ , such that $\mathcal{H}' \in \mathfrak{H}$ if $z \xrightarrow{\sigma} \langle \ell', D', \mathcal{H}' \rangle$.

Example 6.18. Consider again zone-history graph $\llbracket \mathcal{A} \times \mathcal{A}' \rrbracket_H$ as depicted in Figure 6.9d where we use z_0 to denote the initial state and \rightsquigarrow to denote the transition relation. Then,

$$\text{histories}(z_0, \rightsquigarrow, \sigma) = \{(x \leq 23 \wedge \chi = x), (x > 21 \wedge x \leq 39 \wedge \chi = x)\}$$

returns the histories of the states comprising locations (ℓ_1, ℓ'_1) and (ℓ_1, ℓ'_2) , respectively.

Finally, we can define *composite zone-history graph* $\llbracket \mathcal{A} \otimes \mathcal{A}' \rrbracket_H$ of TA \mathcal{A} w.r.t. TA \mathcal{A}' . To this end, we utilize the auxiliary definitions and transition relations \rightsquigarrow_1 and \rightsquigarrow_\times (see Notation 6.3) as described above. The transition relation \rightsquigarrow_\otimes of $\llbracket \mathcal{A} \otimes \mathcal{A}' \rrbracket_H$ contains all transitions in \rightsquigarrow_\times (i.e., the transition relation of the zone-history graph $\mathcal{A} \times \mathcal{A}'$) and transitions of \rightsquigarrow_1 in case the allowed spans are not yet covered by \rightsquigarrow_\times . Here, we check whether behavior of \rightsquigarrow_1 is covered by accumulating the respective histories of \rightsquigarrow_\times with function *histories* and *joining* these histories.

Definition 6.12 (Composite Zone-History Graph). Let $\mathcal{A} = (L, \ell_0, \Sigma, C, I, E)$ and $\mathcal{A}' = (L', \ell'_0, \Sigma, C', I', E')$ be TA with $C \cap C' = \emptyset$ and $\chi \notin C \cup C'$, and \rightsquigarrow_1 and \rightsquigarrow_\times be the auxiliary transition relations of \mathcal{A} w.r.t. \mathcal{A}' . The *composite zone-history graph* of \mathcal{A} w.r.t. \mathcal{A}' is a tuple $(\mathcal{Z}, z_0, \Sigma, \rightsquigarrow)$, where

- $\mathcal{Z} = (L \times L') \times \mathcal{B}(C \cup C') \times \mathcal{B}(C \cup C' \cup \{\chi\})^*$ is a set of *symbolic states*,
- $z_0 = \langle (\ell_0, \ell'_0), D_0, \epsilon \rangle$ is the *initial state*,
- Σ is a set of *actions*, and
- $\rightsquigarrow_\otimes \subseteq \mathcal{Z} \times \Sigma_\tau \times \mathcal{Z}$ is the least relation satisfying
 - $z \xrightarrow{\sigma}_\otimes z'$ if $z \xrightarrow{\sigma}_\times z'$ and
 - $z_1 \xrightarrow{\sigma}_\otimes \langle (\ell_2, \ell'_1), D_2, \mathcal{H}_2 \rangle$ if $z_1 \xrightarrow{\sigma}_1 \langle (\ell_2, \ell'_1), D_2, \mathcal{H}_2 \rangle \wedge \text{join}(\mathfrak{H}) \prec \mathcal{H}_2$, where $\mathfrak{H} = \text{histories}(z_1, \rightsquigarrow_\times, \sigma)$.

By $\llbracket \mathcal{A} \otimes \mathcal{A}' \rrbracket_H$, we refer to the composite zone-history graph of \mathcal{A} w.r.t. \mathcal{A}' .

Example 6.19 (Composite Zone-History Graph). Consider TA \mathcal{A} and \mathcal{A}' as depicted in Figure 6.10 (which are the same TA as previously used in Examples 6.15, 6.16, and 6.18). For constructing composite zone-history graph

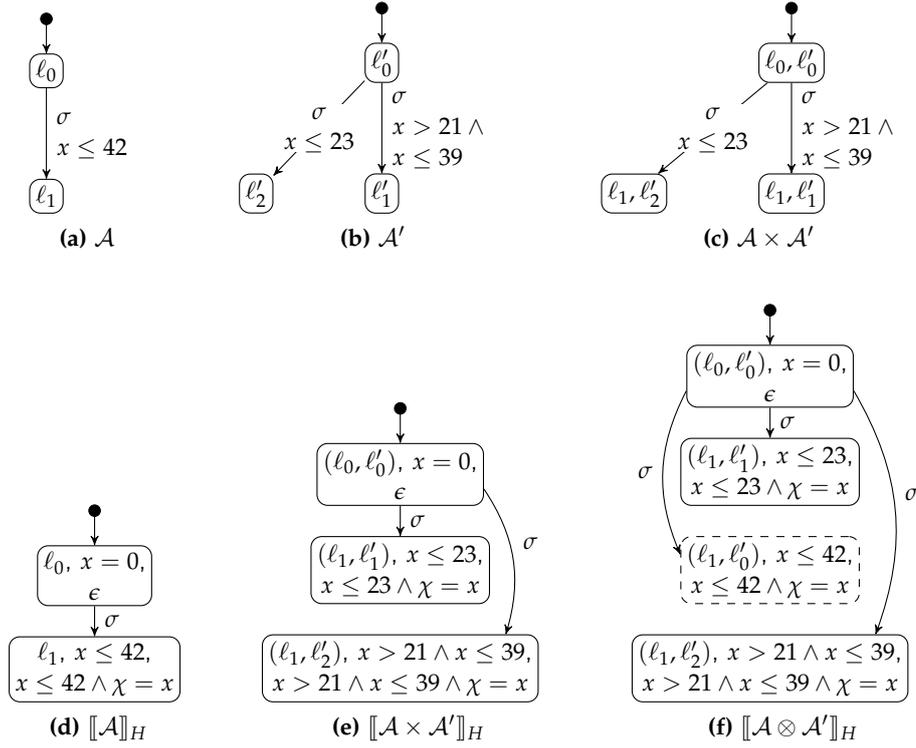


Figure 6.10: Example for the Construction of a Composite Zone-History Graph

$[[\mathcal{A} \otimes \mathcal{A}']]_H$, we first generate the parallel product $\mathcal{A} \times \mathcal{A}'$ (see Figure 6.10c) and the zone history graphs $[[\mathcal{A}]]_H$ and $[[\mathcal{A} \times \mathcal{A}']]_H$ (see Figures 6.10d and 6.10e) as described in the previous examples. Then, we check if all behavior of zone-history graph $[[\mathcal{A}]]_H$ is already contained in $[[\mathcal{A} \times \mathcal{A}']]_H$ by utilizing the functions *history* and *join*. Therewith, we observe that the initial state of $[[\mathcal{A} \times \mathcal{A}']]_H$ does not cover action σ for $39 < x \leq 42$. Hence, we have to add the respective transition and the target state from $[[\mathcal{A}]]_H$ (which we marked with a dashed border in the composite zone-history graph). As a result, we obtain the composite zone-history graph $[[\mathcal{A} \otimes \mathcal{A}']]_H$ of TA \mathcal{A} w.r.t. \mathcal{A}' . Finally, for checking timed bisimilarity of \mathcal{A} and \mathcal{A}' we also have to generate the composite zone-history graph $[[\mathcal{A}' \otimes \mathcal{A}]]_H$ of \mathcal{A}' w.r.t. \mathcal{A} which, in this case, is identical to $[[\mathcal{A} \times \mathcal{A}']]_H$ (see Figure 6.10e).

Next, we can conclude that composite zone-history graphs are finite as these models are derived from proper (finite) zone-history graphs.

Lemma 6.1. Let \mathcal{A} and \mathcal{A}' be a TA. Then, $[[\mathcal{A} \otimes \mathcal{A}']]_H$ is finite.

Proof. Let \mathcal{A} and \mathcal{A}' be a TA. For constructing $[[\mathcal{A} \otimes \mathcal{A}']]_H$, we first construct zone-history graphs $[[\mathcal{A}]]_H$ and $[[\mathcal{A} \times \mathcal{A}']]_H$ for which we have already proven finiteness (see Proposition 6.1). To obtain $[[\mathcal{A} \otimes \mathcal{A}']]_H$, we then add transitions and states of $[[\mathcal{A}]]_H$ to $[[\mathcal{A} \times \mathcal{A}']]_H$ if behavior of \mathcal{A} is uncovered. Hence, $[[\mathcal{A} \otimes \mathcal{A}']]_H$ is finite as also $[[\mathcal{A}]]_H$ and $[[\mathcal{A} \times \mathcal{A}']]_H$ are finite. \square

As suggest in Example 6.19, we can apply timed bisimulation as described in Definition 6.7 also for composite zone-history graphs as these models are, by construction, proper zone-history graphs. Hence, it follows that checking timed bisimilarity on composite zone-history graphs coincides with timed bisimulation on TLTS, and the check is decidable.

Theorem 6.2. Let $\mathcal{A} = (L, \ell_0, \Sigma, C, I, E)$ and $\mathcal{A}' = (L', \ell'_0, \Sigma, C', I', E')$ be TA. Then, it holds that

1. $\mathcal{A} \sqsubseteq_T \mathcal{A}' \Leftrightarrow \mathcal{A} \sqsubseteq_Z \mathcal{A}'$ and
2. $\mathcal{A} \sqsubseteq_Z \mathcal{A}'$ is decidable.

Proof. Let $\mathcal{A} = (L, \ell_0, \Sigma, C, I, E)$ and $\mathcal{A}' = (L', \ell'_0, \Sigma, C', I', E')$ be TA. We prove the two parts of Theorem 6.2 separately.

1. We have already proven that $\mathcal{A} \sqsubseteq_T \mathcal{A}' \Leftrightarrow \mathcal{A} \sqsubseteq_Z \mathcal{A}'$ holds for deterministic TA (see Theorem 6.1). Hence, it remains to be shown that this also holds for non-deterministic TA. Here, $\llbracket \mathcal{A} \times \mathcal{A}' \rrbracket_H$ contains the common behavior of \mathcal{A} and \mathcal{A}' , such that $\llbracket \mathcal{A} \times \mathcal{A}' \rrbracket_H$ and $\llbracket \mathcal{A}' \times \mathcal{A} \rrbracket_H$ are bisimilar (see Theorem 6.1) and even identical (up to location names of symbolic states). Furthermore, we add exclusive behavior of \mathcal{A} and \mathcal{A}' to $\llbracket \mathcal{A} \otimes \mathcal{A}' \rrbracket_H$ and $\llbracket \mathcal{A}' \otimes \mathcal{A} \rrbracket_H$, respectively (see Definition 6.12). Hence, non-bisimilarity can only be caused by exclusive behavior such that $\mathcal{A} \sqsubseteq_T \mathcal{A}' \Leftrightarrow \mathcal{A} \sqsubseteq_Z \mathcal{A}'$.
2. As composite zone-history graphs are finite (see Proposition 6.1), there are finitely many pairs of transitions and spans to check (see Definition 6.7). Hence, checking $\mathcal{A} \sqsubseteq_Z \mathcal{A}'$ is decidable. \square

Having defined timed bisimilarity also for non-deterministic TA, we proceed by formally proving a conjecture we made in the background chapter, where we mentioned that the behavior of a TA *with* clock constraints supporting disjunction can be expressed by a TA *without* clock constraints supporting disjunction. Recall, that we defined the grammar of clock constraints $\varphi \in \mathcal{B}(C)$ as

$$\varphi := \text{true} \mid c \sim n \mid c - c' \sim n \mid \varphi \wedge \varphi$$

where $\sim \in \{<, \leq, \geq, >\}$, $\{c, c'\} \subseteq C$, and $n \in \mathbb{T}_C$ (see Definition 2.1). Given this grammar, we can express a guard $g \vee g'$ with $\{g, g'\} \in \mathcal{B}(C)$ of a switch by using two switches instead, where one has guard g and the other has guard g' .

Example 6.20. Consider TA \mathcal{A} and \mathcal{A}' depicted in Figure 6.11 as an example, where we assume that the grammar for clock constraints also allows disjunction. Here, we can use action σ in \mathcal{A} if clock x has a value between 0 and 11 or a value of at least 47. The exact same behavior is also expressed in TA \mathcal{A}' where we use the left switch for executing σ between 0 and 11 seconds and the right switch if x has a value of at least 47.

Next, we formally prove that we can construct a bisimilar TA without disjunction for any TA with clock constraints allowing disjunction.

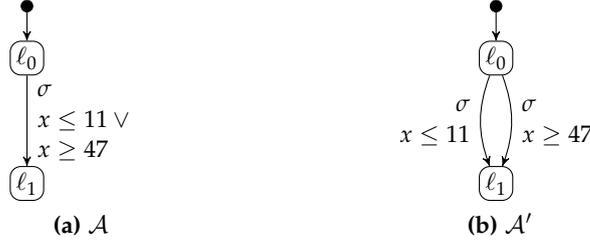


Figure 6.11: Example that Absence of Disjunction does not Reduce Expressiveness

Proposition 6.2. Let $\mathcal{A} = (L, \ell_0, \Sigma, C, I, E)$ and $\mathcal{A}' = (L, \ell_0, \Sigma, C, I, E')$ be TA with $e = (\ell, g \vee g', \sigma, R, \ell') \in E$ and

$$E' = \{e' \mid e' \in E \wedge e' \neq e\} \cup \{(\ell, g, \sigma, R, \ell'), (\ell, g', \sigma, R, \ell')\}.$$

Then, it holds that $\mathcal{A} \simeq \mathcal{A}'$.

Proof. Let $\mathcal{A} = (L, \ell_0, \Sigma, C, I, E)$ and $\mathcal{A}' = (L, \ell_0, \Sigma, C, I, E')$ be TA with $e = (\ell, g \vee g', \sigma, R, \ell') \in E$ and $E' = \{e' \mid e' \in E \wedge e' \neq e\} \cup \{(\ell, g, \sigma, R, \ell'), (\ell, g', \sigma, R, \ell')\}$. Then, $\text{TLTS } \llbracket \mathcal{A} \rrbracket_S$ contains transitions $\langle \ell, u \rangle \xrightarrow{\sigma} \langle \ell', u' \rangle$ if it holds that $u \in g \vee g'$ (among other criteria). Furthermore, $\text{TLTS } \llbracket \mathcal{A} \rrbracket_S$ contains the same transition $\langle \ell, u \rangle \xrightarrow{\sigma} \langle \ell', u' \rangle$ if $u \in g$ or $u \in g'$ (due to the two switches). Hence, it holds that $\mathcal{A} \simeq \mathcal{A}'$. \square

In this section, we have generalized the notion of zone-history graphs to composite zone-history graphs, facilitating an *effective* and precise check of timed bisimilarity also for non-deterministic TA. However, in practice we often encounter large models with many locations, complex clock constraints, and frequent resets, such that zone-history graphs may become quite large. In particular, this may obstruct effective checks of timed bisimilarity by tools. To tackle this issue, we introduce *bounded* zone-history graphs in the next section. These bounds enable a trade-off between precision and scalability.

6.1.4 Bounded Checking of Timed Bisimilarity

So far, we have introduced an effective and precise check for timed bisimilarity of deterministic and non-deterministic TA. Based upon these results, we now extend this approach by an adjustable *bound parameter* $b \in \mathbb{N}_0$. Therewith, we restrict the size of zone histories to at most b elements by pruning a history \mathcal{H} (i.e., removing the prefix) such that we only memorize history $\mathcal{H}_{\downarrow b}$. Furthermore, we utilize $\mathcal{A} \simeq_b \mathcal{A}'$ to denote the b -bounded check of timed bisimilarity (which we apply as described in Definition 6.7). As a result, $\mathcal{A} \simeq_{\infty} \mathcal{A}'$ denotes the unbounded check $\mathcal{A} \simeq_Z \mathcal{A}'$, and $\mathcal{A} \simeq_0 \mathcal{A}'$ checks timed bisimilarity on plain zone graphs. Note, that bound b can also be applied to *composite* zone-history graphs as they are also proper zone-history graphs.

Notation 6.4 (Bounded Zone-History Graph). Let \mathcal{A} and \mathcal{A}' be a TA. By $\llbracket \mathcal{A} \rrbracket_b$, we refer to the b -bounded zone-history graph of TA \mathcal{A} . We use $\mathcal{A} \simeq_b \mathcal{A}'$ to denote that the b -bounded zone-history graphs of \mathcal{A} and \mathcal{A}' are timed bisimilar.

By utilizing these bounds, we can, in general, generate (composite) zone-history graphs faster as we have to update less history elements. Furthermore, we apply, in general, the cut criterion (as described in Algorithm 6.1) earlier as we have shorter zone histories resulting in a smaller number of possibly unequal history prefixes. Finally, also the check for timed bisimilarity itself can be executed faster as shorter zone histories result in fewer comparison of history elements.

Example 6.21 (Bounded Zone-History Graph of the PPU). Consider the zone-history graphs depicted in Figure 6.12 as an example. Here, zone-history graph $\llbracket \mathcal{A} \rrbracket_H$ in Figure 6.12a corresponds to the TA depicted in Figure 6.5a and is the same zone-history graph as discussed in Example 6.8. Furthermore, Figure 6.12b depicts the 1-bounded zone-history graph $\llbracket \mathcal{A} \rrbracket_1$. Here, corresponding symbolic states consist of the same location and zone. Additionally, the zone histories of $\llbracket \mathcal{A} \rrbracket_1$ only comprise the postfix of length 1 of histories in $\llbracket \mathcal{A} \rrbracket_H$.

Next, we prove that there always exists a finite $b \in \mathbb{N}_0$ such that checking timed bisimilarity on bounded zone-history graphs coincides with checking timed bisimilarity on zone-history graphs. Moreover, we show that checking bisimilarity on bounded zone-history graphs can only produce false positives (but no false negatives). We achieve this by proving that bisimilarity on b -bounded zone-history graphs implies bisimilarity on b' -bounded zone-history graphs if $b \geq b'$.

Theorem 6.3. Let \mathcal{A} and \mathcal{A}' be TA, and $\{b, b'\} \subseteq \mathbb{N}_0$. Then, it holds that

1. there exists a finite b such that $\mathcal{A} \simeq_b \mathcal{A}' \Leftrightarrow \mathcal{A} \simeq \mathcal{A}'$ and
2. $\mathcal{A} \simeq_b \mathcal{A}' \Rightarrow \mathcal{A} \simeq_{b'} \mathcal{A}'$ iff $b \geq b'$.

Proof. Let \mathcal{A} and \mathcal{A}' be TA. We prove the two parts of Theorem 6.3 separately.

1. As $\mathcal{A} \simeq \mathcal{A}$ is decidable (see Theorems 6.1 and 6.2) and zone-history graphs as well as composite zone-history graphs are finite (see Proposition 6.1 and Lemma 6.1), zone histories as used for checking timed bisimilarity always have a finite length. Hence, there exists a finite b such that $\mathcal{A} \simeq_b \mathcal{A}' \Leftrightarrow \mathcal{A} \simeq \mathcal{A}'$.
2. When using a bound b' instead of b with $b \geq b'$, then $\mathcal{A} \simeq_{b'} \mathcal{A}'$ checks *at most* the history elements that are also checked in b , such that $\mathcal{A} \simeq_{b'} \mathcal{A}'$ does not check any zone-history elements that are not checked by $\mathcal{A} \simeq_b \mathcal{A}'$. Hence, it holds that $\mathcal{A} \simeq_b \mathcal{A}' \Rightarrow \mathcal{A} \simeq_{b'} \mathcal{A}'$ iff $b \geq b'$. \square

Therewith, we conclude the section about timed bisimulation of (non-)deterministic TA with silent moves. In this section, we introduced (composite) zone-history

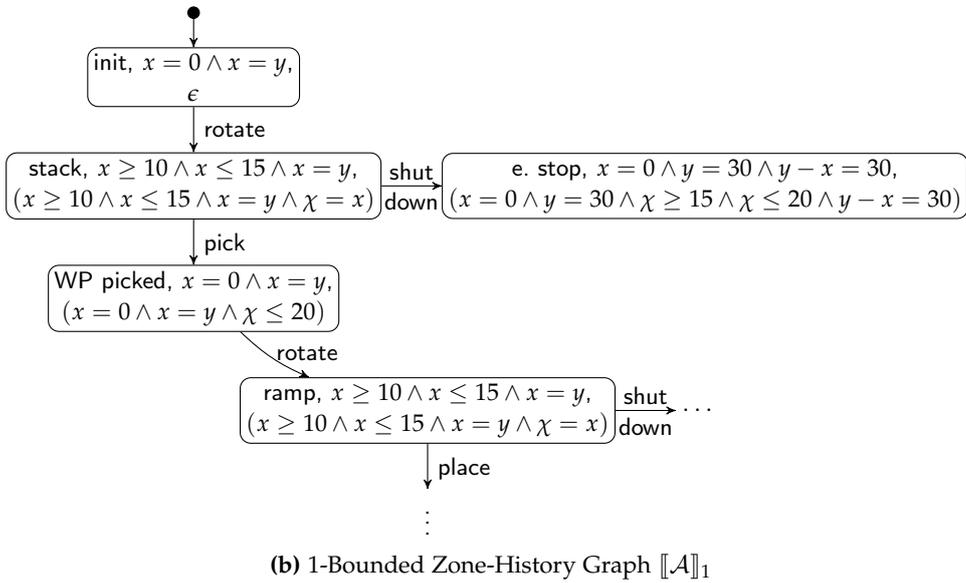
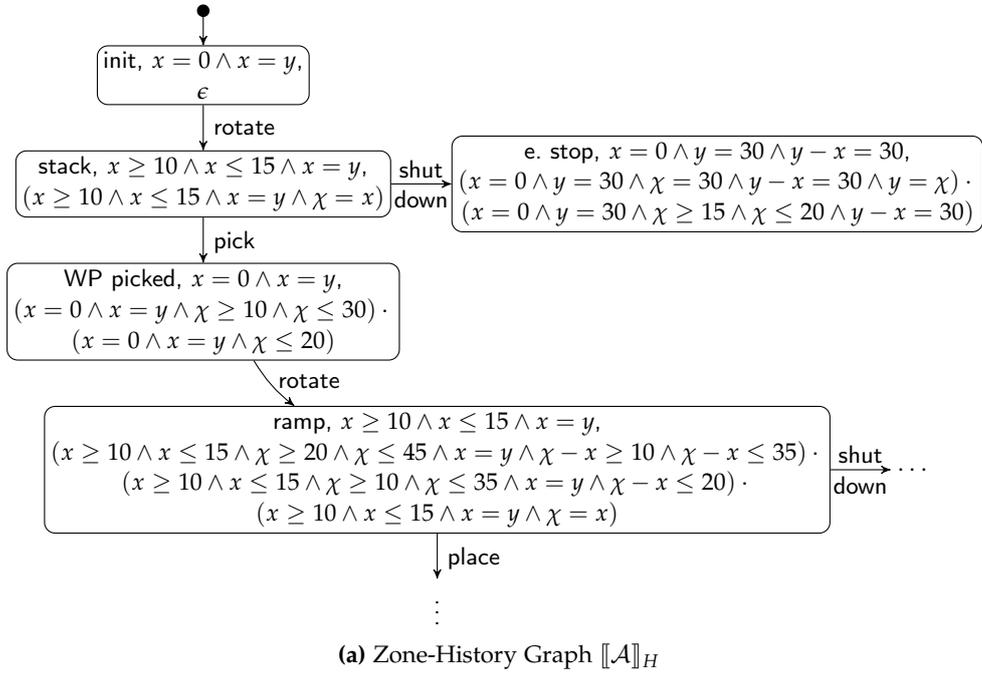


Figure 6.12: Example for the 1-Bounded Zone-History Graph of the PPU (Based on TA \mathcal{A} Depicted in Figure 6.5a)

graphs as a finite characterization of TA semantics expressing exactly the information that are needed for checking timed bisimilarity. Based on this characterization, we defined a decidable check for timed bisimilarity. Additionally, we introduced *bounded* zone-history graphs, facilitating an adjustable trade-off between precision and efficiency. This is especially helpful to handle real-world systems with many locations, complex clock constraints, and frequent resets. Next, we generalize the notion of timed bisimilarity for PTA and CoPTA.

6.2 CHECKING BISIMULATION FOR PARAMETRIC TIMED AUTOMATA

So far, we have defined an effective decision procedure for checking timed bisimilarity of deterministic and non-deterministic TA with silent moves. In this section, we generalize the notion of timed bisimilarity to PTA, which we then extend for timed bisimilarity of CoPTA (see Section 6.3). We first consider timed bisimilarity of PTA as an intermediate step before considering CoPTA as this allows us to concentrate on the problem of independent parameters of a pair of PTA models. In particular, there may be parameters having different names and are used in different clock constraints (i.e., being compared to different constants). Nonetheless, a pair of PTA models may be *parametric timed bisimilar*.

Intuitively, a PTA \mathcal{P} is parametric timed bisimilar to a PTA \mathcal{P}' if for every parameter valuation ν of \mathcal{P} there exists a parameter valuation ν' of (\mathcal{P}') such that $\nu(\mathcal{P}) \simeq \nu'(\mathcal{P}')$ (i.e., the TA variants are bisimilar), and vice versa (which resembles a definition in a product-by-product fashion). We start this section by showing a small collection of interesting PTA fragments and describing their relationship w.r.t. parametric timed (bi-)similarity.

Example 6.22. Figure 6.13 depicts a small collection of pairs of PTA models as an example for parametric timed bisimilarity with clocks $\{c, c'\} \subseteq C$ and parameters $\{p, p', q, q'\} \subseteq P$ (where Figures 6.13a to 6.13f are adapted from Lochau et al. [128]). Here, we have the following relations where, for instance, $\mathcal{P}_1 \sqsubseteq \mathcal{P}_0$ denotes that \mathcal{P}_0 parametric-timed simulates \mathcal{P}_1 .

- For the pair \mathcal{P}_0 and \mathcal{P}_1 it holds that $\mathcal{P}_1 \sqsubseteq \mathcal{P}_0$ and $\mathcal{P}_0 \sqsubseteq \mathcal{P}_1$. In particular, \mathcal{P}_1 uses parameter q for both guards, such that all variants use the same constant for both guards. In contrast, \mathcal{P}_0 also allows variants having two different constants. Here, $\mathcal{P}_1 \sqsubseteq \mathcal{P}_0$ as we obtain identical TA $\nu_0(\mathcal{P}_0)$ and $\nu_1(\mathcal{P}_1)$ by choosing parameter values satisfying $p = p' = q$. Moreover, $\mathcal{P}_0 \sqsubseteq \mathcal{P}_1$ also holds as we can set q to $q = \max(p, p')$. Therewith, a variant $\nu_1(\mathcal{P}_1)$ always simulates the corresponding variant $\nu_0(\mathcal{P}_0)$, as $\nu_1(\mathcal{P}_1)$ has at least the behavior of $\nu_0(\mathcal{P}_0)$.
- \mathcal{P}_2 and \mathcal{P}_3 are incomparable w.r.t. parametric timed bisimulation. \mathcal{P}_2 allows action σ to occur after exactly p seconds and resets c . Then, we have at most 1 second to perform action σ' . Furthermore, exactly p seconds need to elapse between two occurrences of σ . \mathcal{P}_3 waits for exactly q seconds to perform action σ and then waits for at most $q' - q$

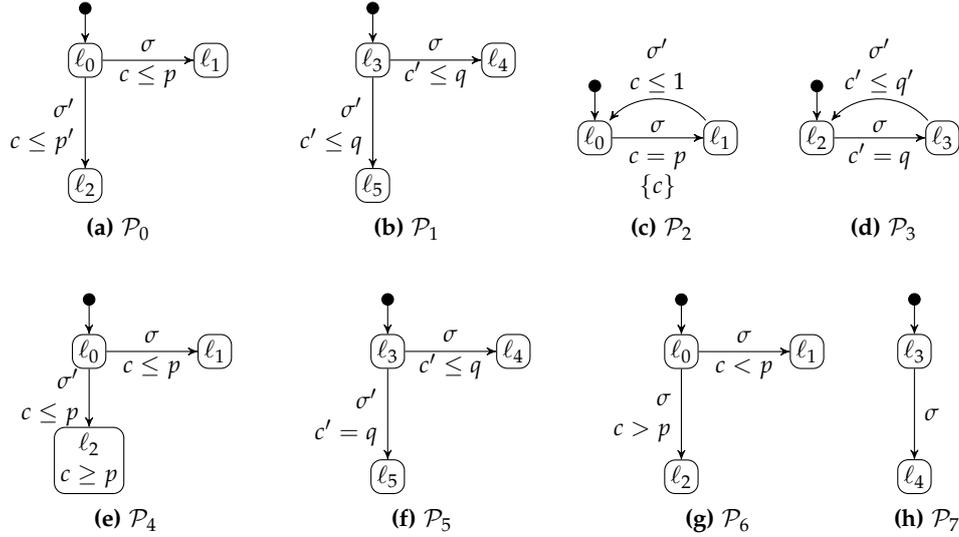


Figure 6.13: Sample PTA Models (Figures 6.13a to 6.13f Adapted from Lochau et al. [128])

seconds to perform σ' . In particular, the execution in \mathcal{P}_3 can get stuck if $q' > q$ (which cannot be simulated by \mathcal{P}_3). Moreover, \mathcal{P}_2 always allows to wait for at most 1 second before performing σ' (which cannot be simulated by \mathcal{P}_2 due to the missing reset of clock c').

- \mathcal{P}_4 and \mathcal{P}_5 are parametric timed bisimilar (i.e., $\mathcal{P}_4 \simeq \mathcal{P}_5$). Here, the combination of guard and invariant for action σ' of PTA \mathcal{P}_4 results in the fact that σ' can only be executed if $c = p$.
- For the pair \mathcal{P}_6 and \mathcal{P}_7 it holds that $\mathcal{P}_6 \sqsubseteq \mathcal{P}_7$ as \mathcal{P}_7 can perform σ at any time whereas \mathcal{P}_6 cannot execute σ if $c = p$.

The remainder of this section is structured as follows. First, we generalize TLTS (modeling TA semantics) to *parametric TLTS (PTLTS)* (see Section 6.2.1) and define parametric timed bisimilarity based on PTLTS. This semi-symbolic representation of PTA semantics has, similar to TLTS, in general an infinite state space, such that checking parametric timed bisimilarity with this approach is undecidable. However, PTLTS can be utilized as a basis for proving correctness of a completely symbolic decision procedure for parametric timed bisimilarity (similar to TLTS for timed bisimilarity, see Theorems 6.1 and 6.2). Here, we leave a completely symbolic representation of PTA semantics (for effectively checking PTA bisimilarity) as an open issue for future work. Second, we introduce a *parameter abstraction* for a subclass of PTA, resulting in TA models. Based on these *parameter-abstracted PTA (PA-PTA)*, we can check parametric timed bisimilarity by utilizing timed bisimulation (as PA-PTA are proper TA). In particular, this check is decidable but imprecise (i.e., there may be false positives but no false negatives).

6.2.1 Parametric Timed Bisimulation

In this section, we formally define parametric timed bisimilarity. To this end, we start by introducing a straightforward definition of parametric timed bisimilarity, where we require a correspondence between *parameter valuations* (as indicated in Example 6.22). However, this definition has some problems regarding practicality, such that we introduce a refined notion of bisimilarity requiring correspondence between *parameters* instead. Furthermore, we introduce a semi-symbolic representation of PTA semantics in terms of parametric TLTS, which we use as a basis for checking parametric timed bisimilarity.

For a first definition of parametric timed bisimilarity, we consider pairs of parameter valuations. In particular, PTA \mathcal{P}' \exists -simulates \mathcal{P} if there *exists* at least one pair of parameter valuations such that $v(\mathcal{P}) \sqsubseteq v'(\mathcal{P}')$ (i.e., there is a simulation relation between the respective TA variants). Furthermore, \mathcal{P}' \forall -simulates \mathcal{P} if *for all* parameter valuations v of \mathcal{P} there exists a simulating variant $v'(\mathcal{P}')$ of \mathcal{P}' . To get a tractable definition of parametric timed bisimilarity, we further include a relation \mathfrak{R} on parameter valuations, explicitly describing pairs of (bi-)similar variants. For defining bisimilarity, we use $\Pi_1(R)$ to denote the projection of first elements of a pairs in relation R . For instance, if $R = \{(a, b), (c, d)\}$, then $\Pi_1(R) = \{a, c\}$.

Notation 6.5. Let $R \subseteq A \times B$ be a relation. Then, $\Pi_1(R) \subseteq A$ denotes the *projection* of elements from A related under R to elements in B .

In the following definition, we utilize the projection described in Notation 6.5 to check which (if any) of the parameter valuations of a PTA \mathcal{P} are simulated by a variant of \mathcal{P}' .

Definition 6.13 (Parametric Timed Bisimulation). Let $\mathcal{P} = (L, \ell_0, \Sigma, C, P, I, E)$ and $\mathcal{P}' = (L', \ell'_0, \Sigma, C', P', I', E')$ be PTA with $C \cap C' = \emptyset$, $P \cap P' = \emptyset$, and $\mathfrak{R} \subseteq (P \rightarrow \mathbb{T}_P) \times (P' \rightarrow \mathbb{T}_{P'})$ such that $(v, v') \in \mathfrak{R} \Leftrightarrow v(\mathcal{P}) \sqsubseteq v'(\mathcal{P}')$. We consider the following cases:

- \mathcal{P}' \exists -simulates \mathcal{P} , denoted by $\mathcal{P} \sqsubseteq_{\exists} \mathcal{P}'$, iff $\Pi_1(\mathfrak{R}) \neq \emptyset$,
- \mathcal{P}' \forall -simulates \mathcal{P} , denoted by $\mathcal{P} \sqsubseteq \mathcal{P}'$, iff $\Pi_1(\mathfrak{R}) = (P \rightarrow \mathbb{T}_P)$,
- \mathcal{P} and \mathcal{P}' are *bisimilar*, denoted by $\mathcal{P} \simeq \mathcal{P}'$, iff \mathfrak{R} is symmetric.

Note, that \forall -similarity implies \exists -similarity, and \exists -simulates is not a preorder (as opposed to \forall -simulates). Moreover, we introduce \exists -similarity for the sake of completeness, but we only consider \forall -similarity (and \forall -bisimilarity) in the remainder of this section.

Example 6.23 (Parametric Timed Bisimulation). Consider again the PTA fragments depicted in Figure 6.13 as described in Example 6.22. Here, it holds that \mathcal{P}_0 \forall -simulates \mathcal{P}_1 (i.e., $\mathcal{P}_1 \sqsubseteq \mathcal{P}_0$), and vice versa. Hence, there is a mutual simulation relation between \mathcal{P}_0 and \mathcal{P}_1 . This is rather unintuitive as \mathcal{P}_0 contains an additional degree of freedom as \mathcal{P}_0 has two independent parameters

p and p' (as opposed to \mathcal{P}_1 which only uses a single parameter). Furthermore, \mathcal{P}_2 \exists -simulates \mathcal{P}_3 (i.e., $\mathcal{P}_3 \sqsubseteq_{\exists} \mathcal{P}_2$). For instance, we can choose parameter valuations $p = q = 0$ and $q' = 1$, such that the corresponding variant of \mathcal{P}_3 is simulated by the variant of \mathcal{P}_2 . Finally, $\mathcal{P}_4 \simeq \mathcal{P}_5$ and $\mathcal{P}_6 \sqsubseteq \mathcal{P}_7$ as described in Example 6.22.

As already indicated in the beginning of this section, parametric timed bisimilarity as described in Definition 6.13 suffers from the problem that we have to find correspondences between *parameter valuations*. However, a PTA model comprises, in general, an infinite number of variants such that we cannot practically check bisimilarity in this way. Furthermore, parametric timed bisimulation as described in Definition 6.13 might lead to rather unintuitive results where PTA models with a different number of independent parameters are bisimilar (see \mathcal{P}_0 and \mathcal{P}_1 in Figure 6.13 and the descriptions in Examples 6.22 and 6.23). In order to define a tractable notion of parametric timed bisimilarity, we define a correspondence between *parameters* instead. To this end, we first lift the notion of TLTS from TA to PTA, such that we have a model for deriving these parameter correspondences. Here, we start by introducing some auxiliary notations before we formally define parametric TLTS.

For a parametric clock constraint $\phi = \phi_c \wedge \phi_p$, we use ϕ_c to refer to the non-parametric clauses of ϕ , and we use ϕ_p to refer to the parametric clauses of ϕ . For instance, assume that we have a constraint $\phi = c > 12 \wedge c \leq p \wedge c' \leq 42$ with clocks $\{c, c'\} \subseteq C$ and parameter $p \in P$. Then, $\phi_c = c > 12 \wedge c' \leq 42$ and $\phi_p = c \leq p$.

Notation 6.6. Let $\phi \in \mathcal{B}(C, P)$ be a parametric clock constraint. Then, we use ϕ_c to refer to the conjunction of non-parametric clauses of ϕ , and we use ϕ_p to refer to the conjunction of parametric clauses of ϕ .

Second, we introduce a notation for replacing clocks in parametric clock constraints by their clock valuations. In particular, $[C/u]\phi$ means that we replace every clock $c \in C$ in parametric clock constraints ϕ by clock value $u(c)$, resulting in a parameter constraint. For instance, $[\{c\}/u]c \leq p \wedge c > q$ with clock $c \in C$, parameters $\{p, q\} \in P$, and clock valuation $u(c) = 12$ results in $p \geq 12 \wedge q < 12$.

Notation 6.7. Let $\phi \in \mathcal{B}(C, P)$ be a parametric clock constraint. Then, $[C/u]\phi \in \mathcal{B}(P)$ denotes the parameter constraint obtained from replacing every occurrence of a clock $c \in C$ in ϕ by clock value $u(c)$.

With these notations, we now formally define the *parametric TLTS semantics* (*PTLTS semantics*) of a PTA \mathcal{P} . We achieve this in two steps.

1. We derive the TLTS semantics from the non-parametric clauses of clock constraints. Here, we use g_c and I_c for guards and invariants, respectively (see Notation 6.6).
2. We extend TLTS states by a third component $\xi \in \mathcal{B}(P)$. This parameter constraint ξ symbolically describes the set of parameter valuations for which the current state is reachable. We obtain ξ by considering the parametric

clauses g_p and I_p of guards and invariants, respectively, and replacing all clocks in g_p and I_p by the clock values $u(c)$ (corresponding to the second component of a TLTS state $\langle \ell, u \rangle$).

Definition 6.14 (Parametric TLTS). Let $\mathcal{P} = (L, \ell_0, \Sigma, C, P, I, E)$ be a PTA. The *parametric timed labeled transition system* (PTLTS) is a tuple $(\bar{S}, \bar{s}_0, \hat{\Sigma}, \rightarrow)$, where

- $\bar{S} = L \times (C \rightarrow \mathbb{T}_C) \times \mathcal{B}(P)$ is a set of *parametric states*,
- $\bar{s}_0 = \langle \ell_0, [C \mapsto 0], \text{true} \rangle \in \bar{S}$ is the *initial state*,
- $\hat{\Sigma} = \Sigma \cup \Delta$ is a set of *actions* with $\Delta = \mathbb{T}_C$ and $\Sigma \cap \Delta = \emptyset$, and
- $\rightarrow \subseteq \bar{S} \times (\hat{\Sigma} \cup \{\tau\}) \times \bar{S}$ is a set of *strong parametric transitions* being the least relation satisfying the following rules:
 - $\langle \ell, u, \xi \rangle \xrightarrow{d} \langle \ell, u + d, \xi \wedge [C/(u + d)]I_p(\ell) \rangle$ if $u \in I_c(\ell)$ and $(u + d) \in I_c(\ell)$ for $d \in \Delta$, and
 - $\langle \ell, u, \xi \rangle \xrightarrow{\sigma} \langle \ell', u', \xi \wedge [C/u]g_p \wedge [C/[R \mapsto 0]u]I_p(\ell') \rangle$ if $u \in g_c$, $u' = [R \mapsto 0]u$, $u' \in I_c(\ell')$, and $\sigma \in \Sigma$ for $(\ell, g, \sigma, R, \ell') \in E$.

By $\llbracket \mathcal{P} \rrbracket_S$, we refer to the PTLTS of PTA \mathcal{P} .

It should be noted that the PTLTS semantics $\llbracket \mathcal{P} \rrbracket_S$ of a PTA \mathcal{P} has, in general, an infinite state space (similar to the TLTS semantics of TA models).

Example 6.24 (Parametric TLTS of the PPU). Consider PTA \mathcal{P} and PTLTS $\llbracket \mathcal{P} \rrbracket_S$ of the PPU as depicted in Figure 6.14 (which is the same model as used in the background chapter, see Figure 2.8). Here, the initial state of $\llbracket \mathcal{P} \rrbracket_S$ consists of the initial location *init* of PTA \mathcal{P} , clock valuation $x = 0 \wedge y = 0$ (as clocks are always initialized with value 0), and parameter constraint true. Then, there are several delay transitions as location *init* does not have an invariant. Additionally, all states comprising location *init* have parameter constraint true as, again, there is no invariant.

When we add transitions corresponding to the switch labeled with action *rotate*, the clock valuations remain unchanged (as there is no reset) but we obtain parameter constraints being unequal to true. In particular, we consider guard $x \geq 10 \wedge x \leq a$ and invariant $x \leq b$ for this step. Here, we replace clock x by its clock valuation $u(x)$, which is also the second component of each PTLTS state.

Note, that $\llbracket \mathcal{P} \rrbracket_S$ does not contain delay transitions between some of the depicted states comprising location *stack* as delays in this particular situation only change the constraint for parameter b as we do not consider the previous guard anymore. For instance, a 1-second delay in state

$$\langle \text{stack}, x = 10 \wedge y = 10, a \geq 10 \wedge b \geq 10 \rangle$$

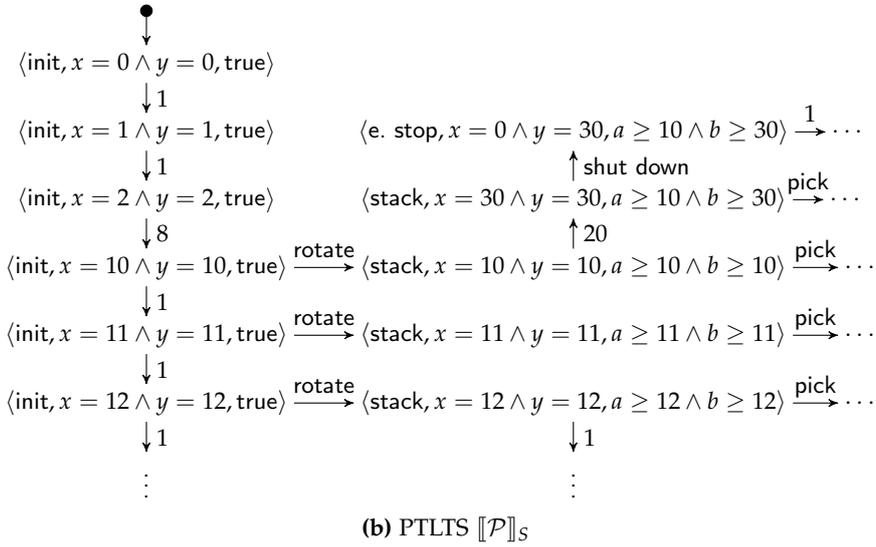
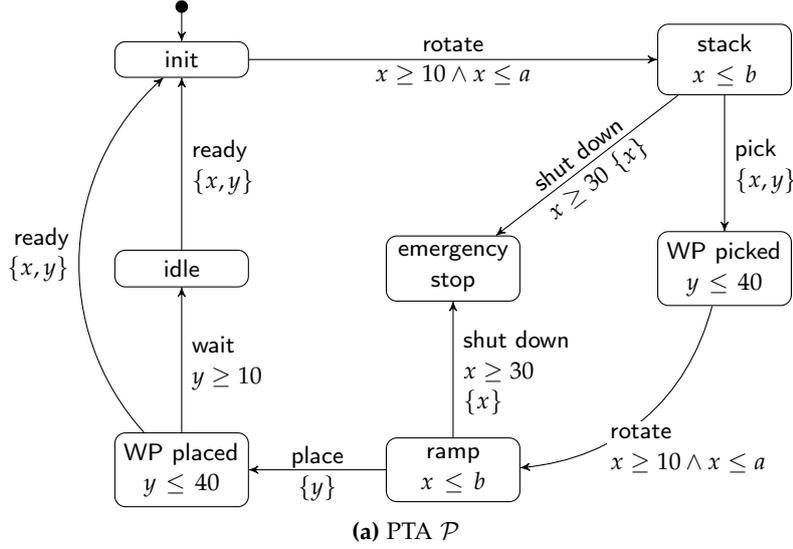


Figure 6.14: Example for the PTLTS of the PPU with Parameters $P = \{a, b\}$

leads to a state comprising parameter constraint $a \geq 10 \wedge b \geq 11$ (i.e., the constraint over parameter a does not change), which is not depicted in Figure 6.14b.

Having defined PTLTS, we can now introduce an alternative definition for PTA bisimulation where we relate parameters instead of parameter valuations. For checking timed bisimilarity of a pair of states of PTA \mathcal{P} and \mathcal{P}' , we check the action labels of the outgoing transitions and bisimilarity of the target states (as also done for timed bisimulation on TLTS). Furthermore, we relate parameters P of PTA \mathcal{P} with parameters P' of PTA \mathcal{P}' . We achieve this by utilizing the parameter constraints ζ and ζ' of PTLTS states (i.e., the third component of each state). In particular, we apply *parameter substitution* Φ and replace parameters $p' \in P'$ used in ζ' by parameters in P (or constant values in \mathbb{T}_P), denoted by $[\Phi]\zeta'$. Then, we check

for similarity by checking if $\zeta \Rightarrow [\Phi]\zeta'$ holds (and vice versa in case of bisimilarity). Hence, we relate parameters by checking if these parameters can be replaced by each without violating the mentioned implication.

Notation 6.8 (Parameter Substitution). Let P and P' be sets of parameters and $\zeta' \in \mathcal{B}(P')$ be a parameter constraint. We use $[\Phi]\zeta' \in \mathcal{B}(P)$ to denote the parameter constraint obtained from apply *parameter substitution* $\Phi : P' \rightarrow P \cup \mathbb{T}_P$ to ζ' .

Therewith, we formally define parametric timed bisimulation on PTLTS.

Definition 6.15 (PTLTS Bisimulation). Let \mathcal{P} and \mathcal{P}' be PTA with $\llbracket \mathcal{P} \rrbracket_S = (\bar{S}, \bar{s}_0, \hat{\Sigma}, \rightarrow)$, $\llbracket \mathcal{P}' \rrbracket_S = (\bar{S}', \bar{s}'_0, \hat{\Sigma}', \rightarrow')$, and $\bar{\mathcal{R}} \subseteq \bar{S} \times \bar{S}'$. Then \mathcal{P}' *strongly parametric-timed simulates* \mathcal{P} if it holds for all $(\bar{s}_1, \bar{s}'_1) \in \bar{\mathcal{R}}$ that

$$\bar{s}_1 \xrightarrow{\mu} \bar{s}_2 \Rightarrow (\bar{s}'_1 \xrightarrow{\mu} \bar{s}'_2 \wedge \zeta_2 \Rightarrow [\Phi]\zeta'_2 \wedge (\bar{s}_2, \bar{s}'_2) \in \bar{\mathcal{R}})$$

where $\mu \in \hat{\Sigma}$ and $(\bar{s}_0, \bar{s}'_0) \in \bar{\mathcal{R}}$. \mathcal{P}' *weakly parametric-timed simulates* \mathcal{P} if it holds for all $(\bar{s}_1, \bar{s}'_1) \in \bar{\mathcal{R}}$ that

$$\bar{s}_1 \xrightarrow{\mu} \bar{s}_2 \Rightarrow (\bar{s}'_1 \xrightarrow{\mu} \bar{s}'_2 \wedge \zeta_2 \Rightarrow [\Phi]\zeta'_2 \wedge (\bar{s}_2, \bar{s}'_2) \in \bar{\mathcal{R}})$$

and $(\bar{s}_0, \bar{s}'_0) \in \bar{\mathcal{R}}$. We use $\mathcal{P} \sqsubseteq_{\Phi} \mathcal{P}'$ to denote that \mathcal{P}' weakly/strongly parametric-timed simulates \mathcal{P} . \mathcal{P} and \mathcal{P}' are *weakly/strongly parametric-timed bisimilar*, denoted by $\mathcal{P} \simeq_{\Phi} \mathcal{P}'$, if there further exists $\Phi' : P \rightarrow P' \cup \mathbb{T}_P$ such that $\bar{\mathcal{R}}$ is symmetric.

Example 6.25 (PTLTS Bisimulation). Consider the PTLTS semantics $\llbracket \mathcal{P}_0 \rrbracket_S$ and $\llbracket \mathcal{P}_1 \rrbracket_S$ depicted in Figure 6.15 corresponding to PTA \mathcal{P}_0 and \mathcal{P}_1 depicted in Figure 6.13. Here, $\llbracket \mathcal{P}_0 \rrbracket_S$ and $\llbracket \mathcal{P}_1 \rrbracket_S$ are parametric-timed bisimilar when we only consider transition labels. Hence, it remains to be checked if there are valid parameter substitutions. For instance, we can use substitution $\Phi(q) = p$ for the states depicted on the right half of $\llbracket \mathcal{P}_0 \rrbracket_S$ and $\llbracket \mathcal{P}_1 \rrbracket_S$, respectively. This results, for instance, in checking

$$p \geq 1 \wedge p' \geq 0 \Rightarrow [\Phi]q \geq 1$$

which means that we check $p \geq 1 \wedge p' \geq 0 \Rightarrow p \geq 1$ (which is true) when substituting q with p . As we can substitute q in every state of $\llbracket \mathcal{P}_1 \rrbracket_S$, it holds that $\mathcal{P}_1 \sqsubseteq_{\Phi} \mathcal{P}_0$. However, $\mathcal{P}_0 \sqsubseteq_{\Phi} \mathcal{P}_1$ does not hold as a single parameter q cannot simulate independent parameters p and q . Hence, PTLTS bisimulation addresses the problem of rather unintuitive results of \forall -bisimulation where PTA models with a different number of independent parameters might still be bisimilar. For instance, it holds that \mathcal{P}_0 and \mathcal{P}_1 are \forall -bisimilar (see Example 6.23), whereas \mathcal{P}_0 and \mathcal{P}_1 are not bisimilar w.r.t. PTLTS bisimulation.

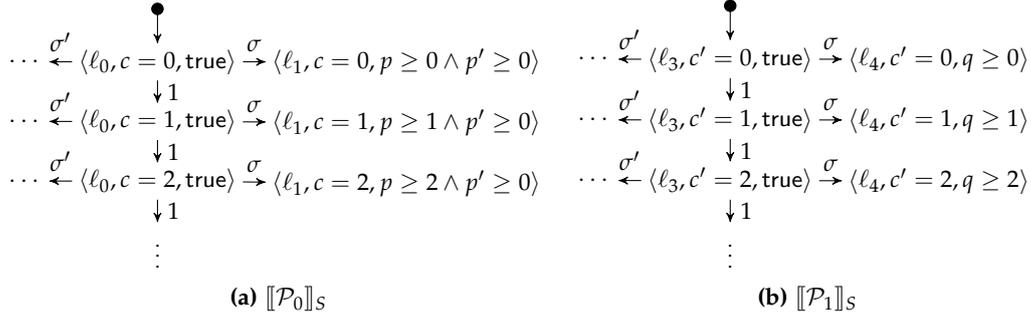


Figure 6.15: PTLTS of PTA \mathcal{P}_0 and \mathcal{P}_1 as Depicted in Figures 6.13a and 6.13b



Figure 6.16: Counterexample for $\mathcal{P} \sqsubseteq \mathcal{P}' \Rightarrow \mathcal{P} \sqsubseteq_{\Phi} \mathcal{P}'$

As already indicated in this section, \forall -bisimilarity (see Definition 6.13) does not coincide with PTLTS bisimilarity (see Definition 6.15). In particular, \forall -bisimilarity requires a correspondence between *parameter valuations*, whereas PTLTS bisimilarity requires a correspondence between *parameters*. Intuitively, \forall -bisimilarity allows to relate arbitrary bisimilar variants of PTA models, which is not possible when applying PTLTS bisimilarity. Hence, PTLTS bisimilarity does not follow from \forall -bisimilarity, as illustrated by the following example.

Example 6.26. Consider PTA \mathcal{P} and \mathcal{P}' as depicted in Figure 6.16. Here, it holds that $\mathcal{P} \sqsubseteq \mathcal{P}'$ (i.e., \mathcal{P}' \forall -simulates \mathcal{P}). However, $\mathcal{P} \not\sqsubseteq_{\Phi} \mathcal{P}'$ (i.e., PTLTS similarity) does not hold as we have to choose either $\Phi(q) = p$ or $\Phi(q) = 12$. In particular, $\Phi(q) = p$ means that similarity of variants does not hold if $p < 12$, and $\Phi(q) = 12$ means that similarity of variants does not hold if $p > 12$.

As illustrated by the previous example, PTLTS (bi-)similarity does not follow from \forall -(bi-)similarity. However, the opposite is true, such that \forall -(bi-)similarity follows from PTLTS (bi-)similarity.

Theorem 6.4. Let \mathcal{P} and \mathcal{P}' be PTA. Then, $\mathcal{P} \sqsubseteq \mathcal{P}'$ if $\mathcal{P} \sqsubseteq_{\Phi} \mathcal{P}'$.

Proof. We prove Theorem 6.4 by using Φ (see Definition 6.15) to derive \mathfrak{R} (see Definition 6.13). Here, Φ provides a correspondence between parameters in \mathcal{P} and \mathcal{P}' such that if there exists a Φ , this Φ directly provides a canonical correspondence between parametric-timed similar valuations ν and ν' (i.e., $\nu(\mathcal{P}) \sqsubseteq \nu'(\mathcal{P}')$). Hence, it holds that $\mathcal{P} \sqsubseteq \mathcal{P}'$ if $\mathcal{P} \sqsubseteq_{\Phi} \mathcal{P}'$. \square

In this section, we formally defined PTLTS to semi-symbolically describe the semantics of PTA models. Furthermore, we introduced a notion of parametric

timed bisimilarity based on PTLTS, where we utilize a parameter substitution and compare the behavior of PTLTS representations in a state-by-state fashion. However, as the PTLTS semantics of a PTA has, in general, an infinite state space, we cannot utilize this notion for *effectively* checking parametric timed bisimilarity. Here, we plan to generalize zone-history graphs to parametric zone-history graphs (by extending states to also include parameter constraints as done for PTLTS), but we leave this as an open issue for future work. However, it should be noted that even a parametric extension of zone-history graphs would only lead to a semi-decidable check of parametric timed bisimilarity as already reachability is only semi-decidable for PTA [20, 21].

As a next step, we consider an interesting subclass of PTA, and we introduce an abstraction for this subclass (resulting in TA models). Based upon this abstraction, we obtain an effective but imprecise check for parametric timed bisimilarity (i.e., the check might produce false positives).

6.2.2 Parameter-Abstracted Timed Bisimulation

In this section, we consider an interesting subclass of PTA called *lower-bound/upper-bound PTA (L/U-PTA)* [103]. L/U-PTA are interesting for parametric timed bisimilarity as we can introduce an abstraction of L/U-PTA resulting in TA models. Therewith, we obtain a decidable but imprecise check for parametric timed bisimilarity (i.e., the results might but false positives) by applying timed bisimulation as defined in Section 6.1 to the abstracted models.

Before we formally define L/U-PTA, we first explain the notion of *lower-bound* and *upper-bound parameters* [103], respectively. Intuitively, a parameter p is a lower-bound parameter in PTA \mathcal{P} if it occurs in a clause of the form $c \geq p$ or $c > p$ of any parametric clock constraint (i.e., guard or invariant). Conversely, parameter p is an upper-bound parameter in PTA \mathcal{P} if it occurs in a clause of the form $c \leq p$ or $c < p$. Note, that a parameter may be a lower-bound parameter and an upper bound parameter at the same time if it occurs in multiple clauses.

Notation 6.9. Let $\mathcal{P} = (L, \ell_0, \Sigma, C, P, I, E)$ be a PTA with parametric linear terms $plt_p := \left(\sum_{1 \leq i \leq |P|} \alpha_i p_i \right) + n$ and $c \in C$ be a clock.

- $p_i \in P$ is a *lower-bound parameter* if it occurs in a $\phi \in \mathcal{B}(C, P)$ of \mathcal{P} in at least one clause of the form $c \geq plt$ or $c > plt$ with $\alpha_i \neq 0$.
- $p_i \in P$ is an *upper-bound parameter* if it occurs in a $\phi \in \mathcal{B}(C, P)$ of \mathcal{P} in at least one clause of the form $c \leq plt$ or $c < plt$ with $\alpha_i \neq 0$.

Example 6.27. Consider, again, the PTA fragments depicted in Figure 6.13 as described in Example 6.22. For instance, parameters p , p' , and q of PTA \mathcal{P}_0 and \mathcal{P}_1 , respectively, are upper-bound parameters. Furthermore, parameter p of PTA \mathcal{P}_2 is both a lower-bound parameter and an upper-bound parameter as $c = p$ is a shorthand for $c \geq p \wedge c \leq p$.

A PTA is a L/U-PTA if every parameter $p \in P$ is *either* a lower-bound parameter or an upper-bound parameter (but not both).

Definition 6.16 (L/U-PTA). Let $\mathcal{P} = (L, \ell_0, \Sigma, C, P, I, E)$ be a PTA with lower-bound parameters $P_L \subseteq P$ and upper bound-parameters $P_U \subseteq P$. Then, \mathcal{P} is a *lower-bound/upper-bound PTA* (L/U-PTA) if $P_L \cap P_U = \emptyset$.

Example 6.28 (L/U-PTA). Consider the PTA fragments depicted in Figure 6.13. For instance, PTA \mathcal{P}_0 and \mathcal{P}_1 are L/U-PTA as p, p' , and q are only upper-bound parameters (see Example 6.27). Furthermore, \mathcal{P}_6 is not a L/U-PTA as p is a lower-bound and an upper-bound parameter. Finally, PTA \mathcal{P}_7 is a L/U-PTA as it does not contain any clock constraints using parameters (or any clock constraints at all).

Next, we utilize L/U-PTA as a basis for parameter abstraction. In particular, the *parameter-abstract PTA* (PA-PTA) $[\mathcal{P}]$ of L/U-PTA \mathcal{P} is the most permissive TA variant of \mathcal{P} . We obtain PA-PTA $[\mathcal{P}]$ by replacing each occurrence of a lower-bound parameter in \mathcal{P} by 0 and replacing each clause containing an upper-bound parameter by true. Here, lower-bound parameters are replaced by 0 (instead of also replacing the respective clause by true) as constraints $c > p$ (with clock $c \in C$ and parameter $p \in P$) do not allow steps with a 0-delay. In contrast, clauses containing upper-bound parameters are replaced by true as this emulates replacing the respective parameters by an arbitrarily great but finite value in \mathbb{T}_p .

Definition 6.17 (PA-PTA). Let \mathcal{P} be a L/U-PTA. The *parameter-abstracted PTA* (PA-PTA) of \mathcal{P} , denoted by $[\mathcal{P}]$, is derived from \mathcal{P} by

- replacing each occurrence of lower-bound parameters p_i by constant 0 and
- replacing each clause with upper-bound parameter p_i by constant true.

Note, that a PA-PTA is a proper TA, such that we can effectively check parametric timed bisimilarity by applying timed bisimulation as described in Definition 6.7. This check might produce false positives but not false negatives. Intuitively, false negatives are not possible as this would require combinations of parametric clock constraints making reachable locations in $[\mathcal{P}]$ unreachable in \mathcal{P} (which is not possible for L/U-PTA [103]).

Example 6.29 (PA-PTA). Consider, again, the PTA fragments depicted in Figure 6.13. As described in Example 6.28, \mathcal{P}_0 and \mathcal{P}_1 are L/U-PTA. Abstracting these PTA models results in identical PA-PTA $[\mathcal{P}_0]$ and $[\mathcal{P}_1]$ (up to names of locations) in which all parametric clock constraints are replaced by true (as all parameters are upper-bound parameters). Checking timed bisimilarity on these PA-PTA results in $[\mathcal{P}_0] \simeq [\mathcal{P}_1]$, which is a false positive (see Example 6.25).

As indicated above, checking parametric timed bisimilarity by utilizing the PA-PTA abstraction and timed bisimulation might result in false positives. However, this check does not produce false negatives, which we prove in the following.

Theorem 6.5. Let \mathcal{P} and \mathcal{P}' be L/U-PTA. Then, $\mathcal{P} \not\sqsubseteq \mathcal{P}'$ if $[\mathcal{P}] \not\sqsubseteq [\mathcal{P}']$.

Proof. Let \mathcal{P} and \mathcal{P}' be L/U-PTA. We prove Theorem 6.5 by contradiction. In particular, $[\mathcal{P}]$ and $[\mathcal{P}']$ constitutes the most permissive parameter valuation. Hence, for contradicting $[\mathcal{P}] \not\sqsubseteq [\mathcal{P}'] \Rightarrow \mathcal{P} \not\sqsubseteq \mathcal{P}'$, we have to find an example with $[\mathcal{P}] \not\sqsubseteq [\mathcal{P}']$ such that when stepping back from PA-PTA to PTA, re-introduced parameters constraints of \mathcal{P} are more restrictive than those of \mathcal{P}' , thus resulting in $\mathcal{P} \sqsubseteq \mathcal{P}'$. This would require combinations of parametric clock constraints making reachable locations in $[\mathcal{P}]$ unreachable in \mathcal{P} . However, in case of L/U-PTA, additional parametric clock constraints can never affect reachability [103]. Hence, it holds that $\mathcal{P} \not\sqsubseteq \mathcal{P}'$ if $[\mathcal{P}] \not\sqsubseteq [\mathcal{P}']$. \square

Therewith, we conclude the section about parametric timed bisimulation. Here, we introduced PTLTS and a check for parametric timed bisimilarity based on the PTLTS semantics of a PTA, relating parameters instead of parameter valuations (i.e., variants) with each other. Additionally, we introduced PA-PTA as an abstraction of L/U-PTA, a subclass of PTA. In particular, we can apply timed bisimulation as PA-PTA are proper TA. As a result, we can effectively check parametric timed bisimilarity which, however, might result in false positives. Next, we further generalize the notion of timed bisimilarity to product lines with configurable parametric real-time constraints in terms of CoPTA.

6.3 CHECKING BISIMULATION FOR PRODUCT LINES WITH CONFIGURABLE PARAMETRIC REAL-TIME CONSTRAINTS

In this section, we introduce the notion of timed bisimilarity for CoPTA and utilize PTA bisimulation as basis. Similar to the previous section, we start by giving a straightforward definition for *featured parametric timed bisimilarity* (in a product-by-product fashion). Hence, CoPTA \mathcal{C} and \mathcal{C}' are bisimilar if every variant of \mathcal{C} is bisimilar to a variant of \mathcal{C}' , and vice versa. Thereafter, we give an outlook for a refined notion of featured parametric timed bisimilarity, where (Boolean and numeric) parameters are related with each other (as also done for PTA). Here, checking bisimulation of CoPTA with an extension of zone-history graphs (or an extension of PTLTS) has the additional challenge that we have to take into account the extended feature model to exclude invalid configurations. This is different from TA and PTA where we only need to consider the models themselves without taking into account further information. However, it should be noted that we leave the detailed description and the definition of this approach as an open issue for future work.

As indicated above, we first define *featured parametric timed bisimilarity* for CoPTA \mathcal{C} and \mathcal{C}' by requiring that every variant of \mathcal{C} is bisimilar to a variant of \mathcal{C}' , and vice versa. Similar to parametric timed bisimulation (see Definition 6.13), we define bisimulation for CoPTA by relating configurations c and c' of \mathcal{C} and \mathcal{C}' , respectively,

in relation \mathfrak{R} , such that the TA variants corresponding to c and c' are bisimilar. As opposed to PTA, a check for bisimulation of CoPTA also has to take into account the EFM instead of only considering the CoPTA model itself. However, we avoid the problem of explicitly considering the EFM during the check for bisimilarity by simply relating configurations to each other in a product-by-product fashion (which is undecidable in case of CoPTA models comprising an infinite number of variants). Moreover, we only introduce \exists -similarity for the sake of completeness.

Definition 6.18 (Featured Parametric Timed Bisimulation). Let $\mathcal{C} = (L, \ell_0, \Sigma, C, P_F, P_N, I, E, m, \eta)$ and $\mathcal{C}' = (L', \ell'_0, \Sigma, C', P'_F, P'_N, I', E', m', \eta')$ be CoPTA with $\mathcal{C} \cap \mathcal{C}' = \emptyset$, $(P_F \cup P_N) \cap (P'_F \cup P'_N) = \emptyset$, and $\mathfrak{R} \subseteq (P_F \cup P_N \rightarrow \mathbb{B} \cup \mathbb{T}_P) \times (P'_F \cup P'_N \rightarrow \mathbb{B} \cup \mathbb{T}_{P'})$ such that $(c, c') \in \mathfrak{R} \Leftrightarrow \mathcal{A} \sqsubseteq \mathcal{A}'$, where $\mathcal{A} \in \llbracket \mathcal{C} \rrbracket_{\mathcal{C}}$ and $\mathcal{A}' \in \llbracket \mathcal{C}' \rrbracket_{\mathcal{C}'}$ are the TA corresponding to configurations $c \in \llbracket m \rrbracket$ and $c' \in \llbracket m' \rrbracket$, respectively. We consider the following cases:

- \mathcal{C}' \exists -simulates \mathcal{C} , denoted by $\mathcal{C} \sqsubseteq_{\exists} \mathcal{C}'$, iff $\Pi_1(\mathfrak{R}) \neq \emptyset$,
- \mathcal{C}' \forall -simulates \mathcal{C} , denoted by $\mathcal{C} \sqsubseteq \mathcal{C}'$, iff $\Pi_1(\mathfrak{R}) = (P_F \cup P_N \rightarrow \mathbb{B} \cup \mathbb{T}_P)$,
- \mathcal{C} and \mathcal{C}' are bisimilar, denoted by $\mathcal{C} \simeq \mathcal{C}'$, iff \mathfrak{R} is symmetric.

Example 6.30 (Featured Parametric Timed Bisimulation). Consider the CoPTA models \mathcal{C} and \mathcal{C}' of the PPU as depicted in Figure 6.17 which both use the same EFM m (see Figure 6.17a). Here, \mathcal{C} (see Figure 6.17b) is the same CoPTA as used in Chapter 3, whereas \mathcal{C}' (see Figure 6.17c) is almost identical and only has a different guard for the switch labeled with action *shut down* (i.e., $x \geq 31$ instead of $x \geq 30$). CoPTA \mathcal{C} and \mathcal{C}' are \exists -bisimilar as, for instance, choosing configuration

$$\text{PPU} \wedge \text{Workpiece} \wedge \neg \text{Stop} \wedge \neg \text{Resume} \wedge \text{Plastic} \wedge \neg \text{Metal} \wedge a = 15 \wedge b = 20 \wedge c = 0$$

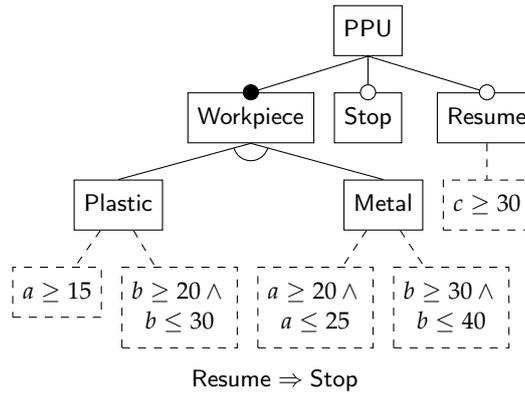
results in identical (and, thus, bisimilar) TA models. However, \mathcal{C} and \mathcal{C}' are not \forall -bisimilar. For instance, TA $\mathcal{A} \in \llbracket \mathcal{C} \rrbracket_{\mathcal{C}}$ corresponding to configuration

$$\text{PPU} \wedge \text{Workpiece} \wedge \text{Stop} \wedge \neg \text{Resume} \wedge \text{Plastic} \wedge \neg \text{Metal} \wedge a = 15 \wedge b = 30 \wedge c = 0$$

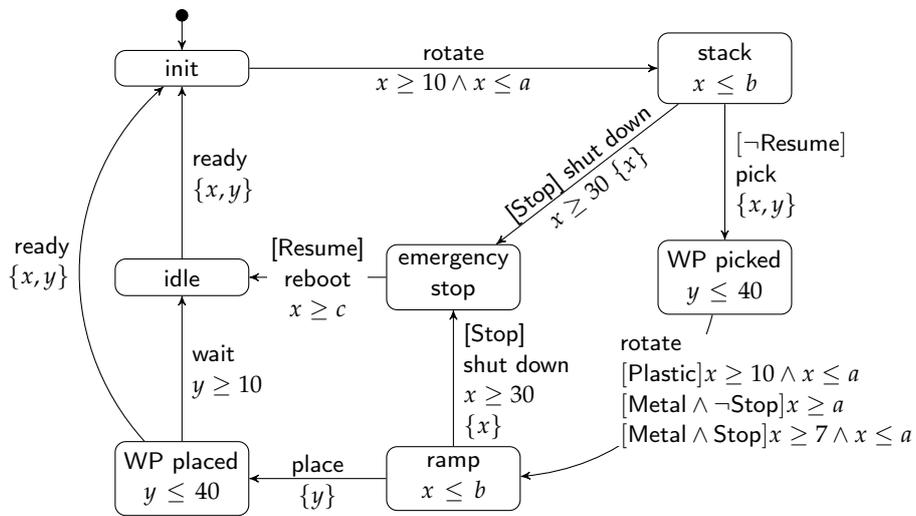
allows the usage of the switch labeled with *shut down* after exactly 30 seconds. However, there does not exist a variant of CoPTA \mathcal{C}' allowing this behavior (due to guard $x \geq 31$).

FUTURE WORK Based on this definition of featured parametric timed bisimilarity, we plan to introduce a completely symbolic decision procedure for checking CoPTA bisimilarity. In order to achieve this goals, we consider the following next steps and challenges.

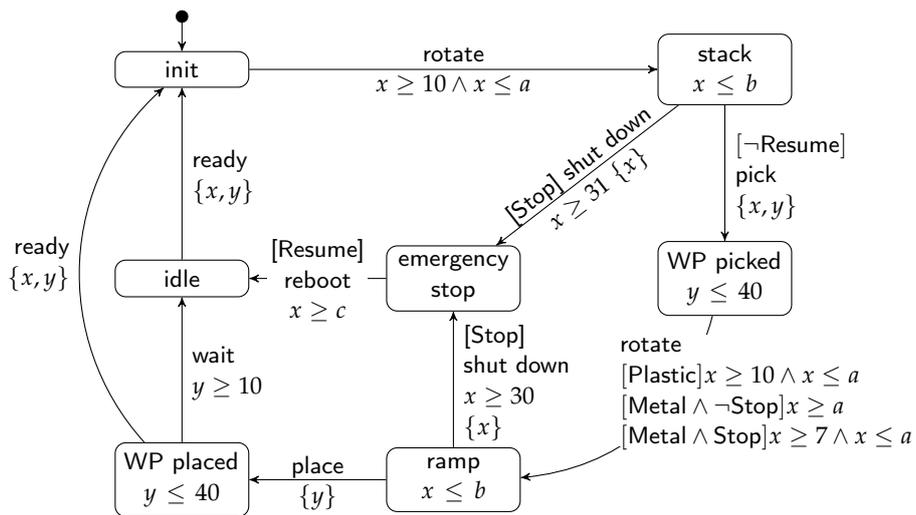
1. As a first step, we plan to introduce a PTLTS-like approach for checking CoPTA bisimilarity. Here, parameter constraints $\zeta \in \mathcal{B}(P)$ (i.e., the third



(a) Extended Feature Model m of the PPU Extract (Copy of Figure 3.2)



(b) CoPTA \mathcal{C} (Copy of Figure 3.3)



(c) CoPTA \mathcal{C}' (Copy of Figure 4.2c)

Figure 6.17: Example for Featured Parametric Timed Bisimulation

component of PTLTS states) must be replaced by configuration constraints $\xi \in \mathcal{B}(P_F, P_N)$, such that also Boolean parameters are included in the check. As compared to PTA, an additional challenge arises from the fact that we also have to include the EFM into the check for bisimilarity (instead of only considering the information which are available in the CoPTA model itself).

2. We plan to introduce a completely symbolic check for CoPTA bisimilarity by extending zone-history graphs to also include configuration constraints. Similar to the PTLTS-like definition, we also have to take into account the EFM to ensure that only valid configurations are considered during the check for bisimilarity. Here, the semi-symbolic PTLTS-like representation of CoPTA semantics can then be utilized as a basis for proving correctness of the completely symbolic decision procedure for CoPTA bisimilarity (as also proposed for PTA).
3. Finally, it might be possible to utilize the PEPTA transformation of CoPTA (see Definition 3.9) for checking featured parametric timed bisimilarity. In particular, we propose to transform a CoPTA \mathcal{C} into its corresponding PEPTA \mathcal{P} . As PEPTA \mathcal{P} is a PTA, this transformation might allow us to check CoPTA bisimilarity by utilizing the approach presented in Section 6.2. However, for applicability of this approach, we have to formally prove that CoPTA bisimilarity follows from bisimilarity of the corresponding PEPTA transformation. We leave the formal description and the proof of this assumption as an open issue for future work.

Therewith, we conclude the conceptual part of this chapter. In this section, we introduced a (generally undecidable) notion of featured parametric timed bisimilarity. Furthermore, we gave an overview on open issues for future work, namely a PTLTS-like definition of CoPTA bisimilarity and a completely symbolic decision procedure for CoPTA bisimilarity (as also proposed for PTA). Next, we experimentally evaluate our proposed check for timed bisimilarity.

6.4 EXPERIMENTAL EVALUATION

In this section, we experimentally evaluate our approach for checking TA bisimilarity (see Section 6.1) by applying our tool implementation to a collection of TA community benchmarks. Here, we do *not* evaluate (featured) parametric timed bisimulation as these checks are defined on the (generally infinite) PTLTS semantics (i.e., we did not introduce a semi-decidable decision procedure for PTA bisimilarity and CoPTA bisimilarity, respectively). We consider the following research questions, and we reuse our evaluation results from Luthmann et al. [140, 136].

RESEARCH QUESTIONS Our tool implementation allows us to investigate the impact of bound b on efficiency and precision for checking timed bisimilarity of deterministic TA as well as non-deterministic TA. Intuitively, we expect that an increased bound b negatively impacts performance but positively influences precision. Furthermore, we expect that on average there exists a value for b marking

an ideal trade-off between computational effort and precision. It should be noted that we do not investigate recall as our approach for checking timed bisimilarity (if implemented correctly) does not produce false negatives (see Theorem 6.3). Additionally, we expect that non-deterministic TA models result in a negative impact on the computational effort for checking TA bisimilarity as these models require the usage of *composite* zone-history graphs. Hence, we consider two research questions for efficiency (i.e., one for each deterministic and non-deterministic TA models). As opposed to efficiency, we only consider a single research question for precision as we expect that bound b has a similar impact for deterministic and non-deterministic TA. In particular, we consider the following research questions.

- **RQ1.1 (Efficiency for Deterministic TA):** How does the value of b impact the *computational effort* of checking timed bisimilarity of deterministic TA?
- **RQ1.2 (Efficiency for Non-Deterministic TA):** How does the value of b impact the *computational effort* of checking timed bisimilarity of non-deterministic TA?
- **RQ2 (Precision):** How does the value of b impact the precision of checking timed bisimilarity?
- **RQ3 (Trade-off):** Which value of b constitutes the best *trade-off between efficiency and precision* for checking timed bisimilarity?

TOOL SUPPORT Our tool implementation uses UPPAAL [39] as a front-end for modeling input TA. In particular, our tool imports TA models being specified in the UPPAAL file format. After the import of TA models, our tool generates (bounded) zone-history graphs as described in this chapter. To this end, our tool implementation supports non-determinism by utilizing *composite* zone-history graphs (see Section 6.1.3), and it supports a configurable bound b (see Section 6.1.4).

For representing convex constraints resulting from clock constraints (i.e., zones and zone histories), our tool utilizes so-called *difference bound matrices (DBM)* [36, 82], which constitute a matrix representation for an efficient comparison (e.g., in terms of intersection) of convex constraints. However, constructing composite zone-history graphs requires the usage of non-convex constraints as described in Section 6.1.3 (e.g., see Example 6.16). Recall, that we construct the union of clock constraints (which may lead to non-convex constraints) for checking whether additional states need to be added. Hence, we apply the ILP solver IBM ILOG CPLEX [104] instead of DBM for performing these particular checks. To this end, we check if we must add an additional state (see last bullet of Definition 6.12) by joining histories and then checking $\text{join}(\mathcal{H}_1) \prec \mathcal{H}_2$ by conjugating elements from \mathcal{H}_2 with the negation of elements from $\text{join}(\mathcal{H}_1)$. Then, we utilize CPLEX to check satisfiability of the resulting (possibly non-convex) constraints. In contrast, all clock constraints occurring in states (i.e., zones and zone histories) of (composite) zone-history graphs are convex, thus allowing us to utilize the DBM theory.

SUBJECT SYSTEMS For evaluating our approach, we use five different TA models from well-established community benchmarks which are frequently used for evaluating analysis techniques based on TA.

Table 6.1: Overview on Subject Systems

	TGC	TGC _τ	GC	GC _τ	CA	CA _τ	RCP	RCP _τ	AVC	AVC _τ
# Locations	14	15	23	24	6	7	10	10	18	19
# Switches	18	20	28	32	13	15	26	28	30	33
# Clocks	1	1	1	1	1	1	2	2	1	1
# Resets	6	6	12	12	1	2	9	9	18	19
# Mutants	26	26	34	34	15	15	26	26	32	32
# Bisimilar Mutants	11	11	15	15	9	9	2	2	1	1
# Internal Transitions	0	1	0	1	0	1	0	1	0	1
# Non-Det. Choices	0	1	0	3	0	1	0	1	0	2

- The *Train-Gate-Controller (TGC)* [15] models a level crossing comprising a railroad-gate controller.
- The *Gear Controller (GC)* [123] is a component of the control system of gear boxes in modern vehicles.
- The *Collision Avoidance (CA)* [105] is a TA representation of a protocol for modeling communication in an Ethernet-like medium.
- The IEEE 1394 *Root Contention Protocol (RCP)* [67] is part of the FireWire bus.
- *Audio/Video Components (AVC)* [97] describe a messaging protocol for communication between AV components.

However, none of these benchmarks (or other widely used benchmarks) includes any non-determinism or internal behavior in terms of τ -steps. Hence, we manually adapted the five subject systems by adding non-determinism and internal behavior, resulting in overall ten case studies. This allows us to also evaluate the impact of non-determinism on computational effort and precision. Table 6.1 gives an overview on the key metrics of our subject systems. Based on these TA models, we experimentally evaluate two settings for checking timed bisimilarity.

1. We check timed bisimilarity of each of our ten subject systems against itself (which should succeed).
2. We *mutate* each of our models to obtain a rich corpus of syntactically slightly different models and check timed bisimilarity of the mutants w.r.t. the original model (which may either succeed or fail).

Checking timed bisimilarity for the first setting is straightforward. For the second setting, we utilize an existing framework for TA mutation operators [6]. Mutation testing is usually applied for evaluating effectiveness of testing techniques. However, mutation testing often has the problem that (undetected) equivalent mutants negatively influence the mutation score (i.e., the number of mutants detected by test cases). In contrast, equivalent mutants are even desirable in our setting as we utilize equivalent and inequivalent mutants for investigating efficiency and precision for positive cases and negative cases, respectively. For our experimental evaluations, we selected two mutation operators presumably having the highest probability to produce slightly different, yet similar mutants.

- The operator *invert resets* flips the set R of resets of a switch (i.e., we reset the set of clocks $C \setminus R$ instead of R).
- The operator *change guards* changes comparison operators in guards (e.g., $c < n$ is replaced by $c > n$).

We applied both operators to all of our ten subject systems, resulting in 268 mutants. Of these 268 mutants, 76 are timed bisimilar to the original models as described in Table 6.1. Checking timed bisimilarity of all of these models against the original models with varying values for bound b (see below) results in an overall number of 2029 runs of our tool implementations. Here, 512 runs should yield positive results (i.e., the input models are bisimilar), whereas 1517 runs should yield negative results. However, it should be noted that we do not have measurement results for every mutant and every value of b as we use a timeout of 30 minutes for each check.

EXPERIMENT DESIGN AND MEASUREMENT SETUP In order to experimentally evaluate the impact of bound b , we execute our experiments with ten different values for b , such that $b \in \{0, 1, 2, 3, 4, 5, 10, 20, 25, 30\}$. In particular, two values for b represent our baselines.

- We consider $b = 0$ (i.e., using plain zone graphs without histories) as the most efficient and least precise instantiation.
- We consider $b = \infty$ as the baseline for the precision of our results.

Moreover, we use a timeout of 30 minutes for generating (composite) zone-history graphs and checking timed bisimilarity, such that we keep the overall runtime of our evaluation realistic. For keeping the overall runtime realistic, we also only check strong timed bisimilarity for case studies *without* τ -steps and weak timed bisimilarity for case studies *with* τ -steps.

For answering **RQ1.1** and **RQ1.2**, we measure the CPU time for generating bounded (composite) zone-history graphs and for checking timed bisimilarity, and we aggregate the results over all mutants for every subject system and every value of b . For **RQ2**, we count the number of false positives and compute precision as follows (where we obtain the number of true positives by using $b = \infty$):

$$\text{precision} = \frac{\# \text{ true positives}}{(\# \text{ true positives}) + (\# \text{ false positives})}$$

Here, we do not evaluate recall as our approach does not produce false negatives (see Theorem 6.3). Moreover, we evaluated all experiments on an Intel Core i7-8700K machine with $6 \times 3.7\text{GHz}$, 4GB RAM, and Windows 10. Our tool is implemented in Java and compiled with AdoptOpenJDK 11.0.6.10.

RESULTS Next, we give an overview on the results of our experimental evaluation that we conducted as described above. Here, Figure 6.18 gives an overview on the measurements for **RQ1.1**, where we measured efficiency of our approach for checking timed bisimilarity of deterministic TA. First, we observe that the time

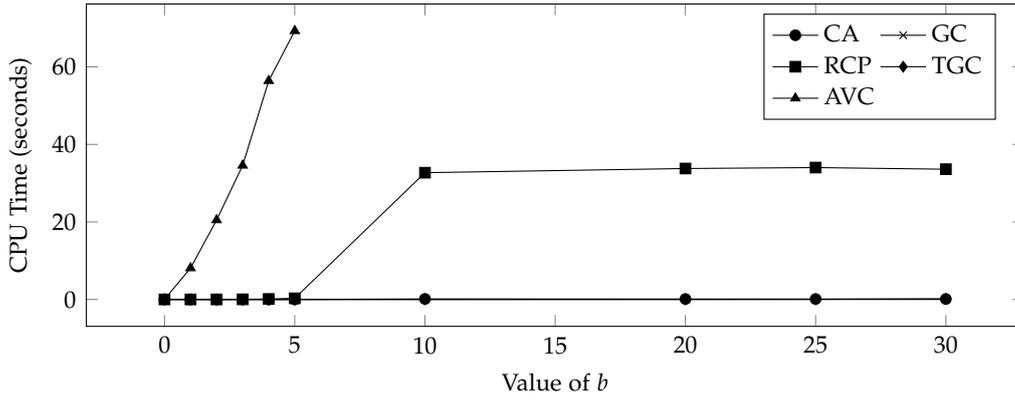


Figure 6.18: Measurement Results for RQ1.1

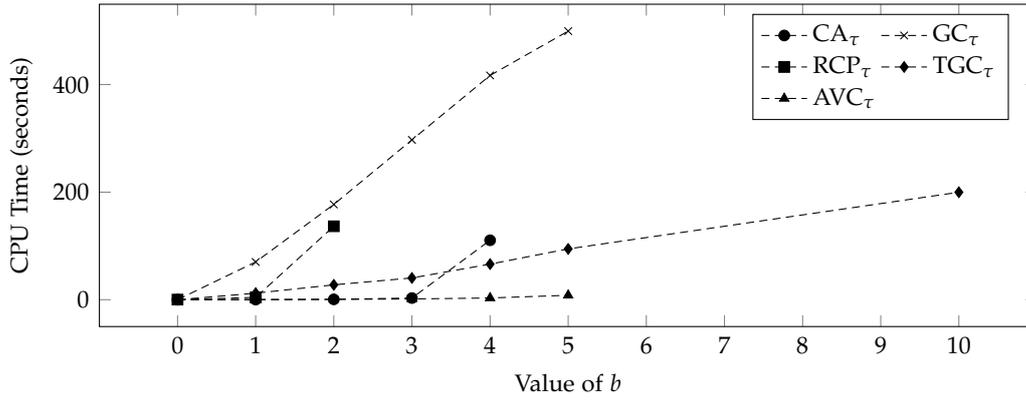


Figure 6.19: Measurement Results for RQ1.2

required for checking bisimilarity is negligible (with at most 74ms), such that we only display the sum of CPU time needed for generating (composite) zone-history graphs and checking timed bisimilarity. Additionally, the overall computational effort in case of CA, GC, and TGC is below one second, whereas the CPU time needed for checking bisimilarity in case of RCP increases to more than 30 seconds for $b \geq 10$. Moreover, AVC is the only deterministic subject systems exceeding our timeout of 30 minutes (for $b \geq 10$).

The measurement results for RQ1.2 (i.e., efficiency for our non-deterministic case studies) are depicted in Figure 6.19. Here, we see that all case studies exceed the timeout of 30 minutes, where RCP $_{\tau}$ already reaches this limit for $b = 3$. Furthermore, the other subject systems reach the timeout for values 4, 5, and 10 for bound b .

The measurement results for RQ2 are depicted in Figure 6.20, where we are concerned with the precision of checking timed bisimilarity for different values of bound b . Additionally, the boxplots in Figure 6.21 summarize the statistical distribution for each value of b . Here, we observe that the median precision for $b = 0$ is 0.54. Moreover, the probability for false positives drastically decreases for $b \geq 3$, where the interquartile range starts at 1. Furthermore, we do not observe any false positives for $b \geq 10$ except for one outlier for $b = 10$. It should be noted that the precision seemingly decreases for $b = 5$. However, as we used a timeout

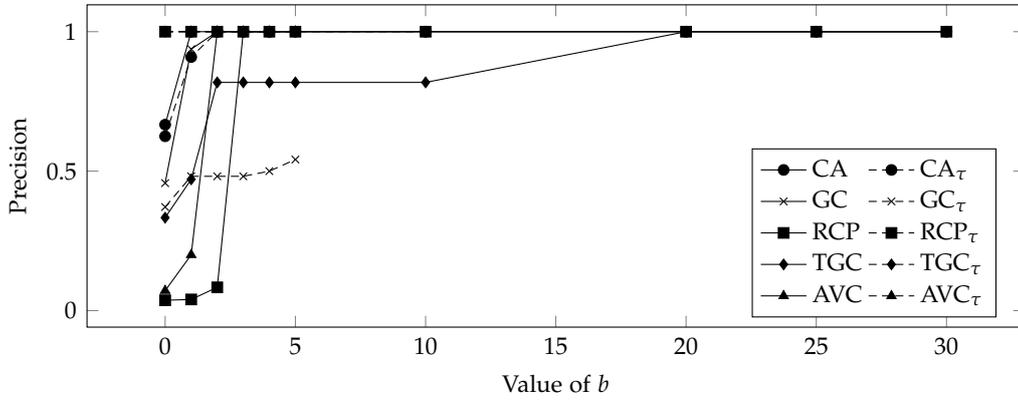


Figure 6.20: Measurement Results for RQ2 (Adapted from Luthmann et al. [140])

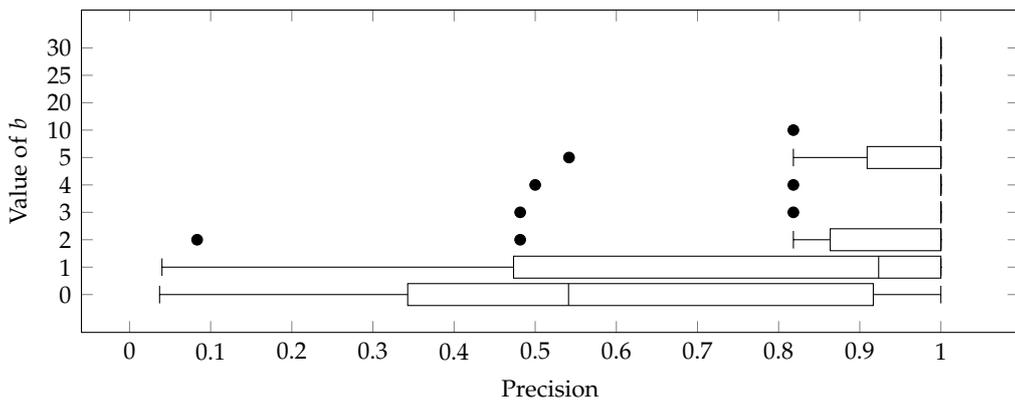


Figure 6.21: Summary of Results for RQ2 (Adapted from Luthmann et al. [140])

of 30 minutes, we consider a smaller number of subject systems for $b = 5$ than for $b \leq 4$. Finally, the presence of non-determinism does not impact the precision (at least for our case studies).

In case of **RQ3**, where we consider the trade-off between efficiency and precision, we observe that the precision drastically increases in case of $b \geq 3$. Additionally, the required CPU time is less than one second for $b \geq 5$ for all deterministic case studies except *AVC*, whereas non-deterministic case studies reach the timeout of 30 minutes already for values $b \leq 10$.

DISCUSSION Next, we discuss the results for **RQ1.1** and **RQ1.2**. First, we observe that our tool implementations performs well for deterministic case studies (except for *AVC*). Here, the computational effort for *AVC* is the result of the internal structure of the TA model (e.g., the branching structure in combination with resets at certain positions) rather than syntactic properties (such as the size of the model in terms of locations). In particular, the size of *GC* and *RCP*, respectively, is similar to *AVC* although the computational effort needed for handling *AVC* is much higher. Second, the results for our non-deterministic subject systems have a much higher computational effort, such that we already reach our timeout of 30 minutes for $b \leq 10$. However, this is expected due to the inherent complexity of checking timed bisimilarity for non-deterministic TA.

For **RQ2**, the precision of 0.54 for $b = 0$ shows the essential necessity of using (composite) zone-history graphs instead of plain zone graphs. Furthermore, we observe that, on average, the length of the zone history should be at least $b = 3$ to avoid most of the false positives. However, a proper recommendation for choosing a value for b does, again, not necessarily depend on syntactic properties (similar to **RQ1**). Instead, the value for b depends on the internal structure of the TA models (e.g., distance between resets and combinations of clocks which are reset).

For **RQ3**, a reasonable trade-off between precision and efficiency seems to be a value of $b = 3$ as using this value avoids all but two false positives in our experiments. Additionally, the required CPU for checking timed bisimilarity is still very low even for our non-deterministic case studies. When we only consider deterministic case studies, we might increase b to 10 as there is only one false positive left for this value, and the computational effort is still low (except for *AVC*).

THREATS TO VALIDITY We discuss internal and external threats to validity, and we start by considering *internal threats*. As described in the background chapter, our approach (and thus, our experimental evaluation) is based on (and limited to) timed safety automata. For instance, we do not consider acceptance locations for checking Büchi-acceptance of infinite runs. Furthermore, many non-trivial TA extension (e.g., see the paragraph on *extensions and generalizations of TA* in Section 3.4) obstruct zone-graph properties (see Definition 2.20), rendering our approach imprecise or even inapplicable. Moreover, the mutation operators that we use to synthetically generate a large number of subject systems for our evaluation only apply small and locally restricted changes (as usual for mutant generation). However, our evaluation results show that these mutations result in equivalent

as well as inequivalent TA models, indicating that mutation is an appropriate approach for our evaluation. Finally, we ensure correctness of our theory by providing proofs, and we thoroughly tested our tool implementation on a large collection of test cases in terms of TA fragments.

We identified as an *external threat* that we do not provide a comparison to other tools. However, to the best of our knowledge, there currently does not exist another tool for effectively checking timed bisimilarity of deterministic or non-deterministic TA. Moreover, we only applied our tool implementation to a small number of case studies. However, the subject systems used for this experimental evaluation represent well-established community benchmarks, have reasonable size and complexity, and are frequently used for evaluating analysis techniques based on TA.

6.5 RELATED WORK

In this section, we give an overview on related work concerning bisimilarity of time-critical systems. Here, the notion of timed bisimulation was first introduced by Moller and Tofts [148] as well as Yi [204], who define bisimilarity on real-time extensions of the *calculus of communicating systems* (CCS) [146]. Similarly, Nicollin and Sifakis [149] define timed bisimulation on the *algebra of timed processes* (ATP). Additionally, Cardell-Oliver and Glover [54] consider an implementation and a specification (both given as a TA) bisimilar if the implementation passes all test cases generated from the specification. However, this approach has several weaknesses. For instance, only TA having a finite TLTS semantics can be checked, and the implementation must be deterministic.

Furthermore, Čerāns [56] introduces the first proof of decidability for checking timed bisimilarity by utilizing region graphs [13, 10] as a finite representation for the check. Weise and Lenzkes [201] improve the work of Čerāns by using so-called FBS graphs as a less space-consuming representation of TA semantics. FBS graphs constitute a variation of zone graphs and also build the basis for zone-history graphs as introduced in Section 6.1. Guha et al. [91, 92] also follow a zone-based approach for bisimilarity-checking on TA as well as the weaker notion of timed prebisimilarity, by employing so-called zone-valuation graphs. Moreover, Tanimoto et al. [183] employ timed bisimulation to check if a given behavioral abstraction preserves time-critical system behavior.

However, none of these approaches support a variable bound for checking bisimilarity more efficiently (in case the occurrence of false positives is acceptable). Additionally, our proposed solution is closer to the concepts of (bi-)simulation equivalence relations on state-transition graphs, where symbolic states are enriched with additional discriminating information. Furthermore, the works describing effective checks of timed bisimilarity by Weise and Lenzkes [201] and Guha et al. [91, 92] miss critical details in terms of replicability of their approaches. Moreover, the approaches by Weise and Lenzkes and Guha et al. are not defined in a way that would directly allow a generalization for checking timed bisimilarity of PTA and CoPTA. In addition to this, there does not (to the best of our knowledge) exist an approach for checking timed bisimilarity of PTA (or CoPTA). Here, almost all works investigate analysis problems concerned with properties of *one* PTA [21, 20].

Finally, CAAL [17] is the only available tool for checking timed bisimilarity that we are aware of. However, this tool utilizes TLTS for the bisimulation check. Hence, CAAL is only applicable to time-critical systems having a finite TLTS semantics.

6.6 CONCLUSION AND FUTURE WORK

In this chapter, we tackled Research Challenges 4.1 and 4.2. In particular, we presented an approach for effectively checking timed bisimilarity for TA with an adjustable trade-off between precision and scalability. Here, it should be noted that there already exist a few approaches for effectively checking TA bisimilarity (without an adjustable trade-off). However, all of these approaches have some disadvantages as compared to the approach that we introduced in this chapter. For instance, Čerāns [56] defines timed bisimilarity based on region graphs which suffer from state-space explosion, making the approach generally less efficient than approaches based on zone graphs. Furthermore, the approaches by Weise and Lenzen [201] and Guha et al. [91, 92] are not defined in a way that would directly allow a generalization for checking timed bisimilarity of PTA and CoPTA. Moreover, there did not exist strategies for checking timed bisimilarity of PTA and product lines with configurable parametric real-time constraints in terms of CoPTA. Hence, in this chapter we improved the state of the art in two ways.

First, we introduced a novel formalism, called bounded (composite) zone-history graphs, for precise, yet scalable checking of timed bisimilarity for non-deterministic TA with silent moves. In particular, we first introduced zone-history graphs for checking timed bisimilarity of deterministic TA. Then, we generalized zone-history graphs to composite zone-history graphs for also checking timed bisimilarity of non-deterministic TA. To this end, we utilize the parallel product of a given pair of TA to ensure that transitions are split in the same way in both composite zone-history graphs in case of non-determinism.

Furthermore, we presented the first definition for checking timed bisimilarity of PTA. Here, we introduced PTLTS (a generalization of TLTS as used for TA) to model PTA semantics, and we defined parametric timed bisimilarity based on PTLTS. Additionally, we investigated the subclass of L/U-PTA, where we can use a parameter abstraction, such that we can utilize our framework for TA bisimilarity to check timed bisimilarity for L/U-PTA (having a possibly imprecise result). Moreover, we also presented the first definition for timed bisimulation of CoPTA.

Finally, our experimental evaluation showed applicability of our approach, and by using bounded zone histories, we can also apply our tool to larger-scale models. Here, the evaluation also revealed that non-determinism has a considerable impact on the computational effort for constructing composite zone-history graphs (as expected).

For future work, we plan to extend our technique to more advanced classes of TA (see the paragraph on *extensions and generalizations of TA* in Section 3.4). This would allow us to cover a wider range of real-time systems which cannot be modeled with classical TA. In case of PTA and CoPTA, we plan to extend zone-history graphs by (Boolean and numeric) parameters such that we have a fully symbolic semantics as a basis for an effective (but incomplete) decision procedure for parametric timed

bisimilarity. Based on these extensions for zone-history graphs, we could then also evaluate these approaches to show practicality of timed bisimulation for (real-time) product lines. Moreover, we are also interested in compositionality properties of (featured parametric) timed bisimulation to tackle the issue of scalability in the formal analysis of (configurable) real-time behavior. Finally, we plan to evaluate our tool implementation for checking TA bisimilarity with a larger number of case studies, where we especially plan to include additional real-world systems.

CONCLUSION AND FUTURE WORK

The goal of this thesis was the development of a framework for the specification and analysis of (safety-critical) software systems with configurable real-time behavior. Therewith, we are now able to model software product lines comprising an infinite number of variants also supporting traceability between the problem space and the solution space. As foundation for our framework, we utilized existing approaches being based on Boolean features, and we extended these approaches by also considering numeric parameters.

In Section 2.1, we introduced the PPU, a bench-scale demonstrator of an industrial plant, as our motivating example which we used to illustrate our research challenges and various concepts throughout this thesis. Moreover, we concluded each section of the background chapter by motivating research challenges addressed in this thesis. In the remainder of the conclusion, we summarize this thesis based on the research challenges (see Section 7.1), and we summarize possible future work and open challenges presented in each chapter (see Section 7.2).

7.1 SUMMARY

Research Challenge 1 had the goal to define a behavioral modeling formalism for real-time product lines with a potentially infinite number of variants. Approaches existing prior to this thesis either incorporated feature constraints in behavioral models allowing us to model dependencies in terms of presence, absence, and combinations of switches and real-time constraints (i.e., FTA [70]) or constraints over unbounded numeric parameters allowing us to choose arbitrary values from a potentially infinite parameter domain (i.e., PTA [15]). Here, the usage of features allows us to trace parts of the behavioral model back to configuration decisions made in the problem space, such that we have traceability between the problem space and the solution space. Furthermore, numeric parameters facilitate modeling variant-rich systems comprising a possibly infinite number of configurations, where real-time constraints may be set to very specific values chosen by engineers for each configuration individually. In Chapter 3, we tackled this challenge and improved the state of the art in two ways.

First, we utilize extended feature models (EFM) to express dependencies between features (i.e., Boolean parameters) and unbounded numeric parameters. In particular, we use constraints over unbounded parameters as attributes of Boolean parameters. Hence, an EFM is given in terms of a parameter constraint. As a result, an EFM may comprise an infinite number of configurations. Second, we introduced the CoPTA formalism. Compared to FTA, CoPTA utilize extended feature models

in terms of constraints over Boolean parameters as well as unbounded numeric parameters instead of classical feature models only defined over features. Furthermore, feature-annotated real-time constraints (of FTA) are generalized to also contain numeric parameters. As a result, we obtain a modeling formalism supporting traceability between the problem space and the solution space also comprising a possibly infinite number of configurations. However, analyzing CoPTA (having an increased expressiveness as compared to FTA) requires solving inequations over numeric parameters in addition to Boolean formulae. Finally, we presented a transformation of CoPTA into an extension of PTA such that we can apply results for PTA to CoPTA (e.g., semi-decidability of reachability).

Research Challenge 2 had the goal to adapt sampling strategies to product lines with a (potentially) infinite number of configurations. Therewith, we derive a representative subset of variants (in terms of real-time behavior) of a product line, such that we only analyze these variants (e.g., for quality assurance) instead of analyzing all variants (which is even impossible in case of CoPTA). Prior to this thesis, there did not exist an approach for sampling infinite configuration spaces besides boundary-value testing, which cannot be applied to find variants with fastest and slowest runs. Furthermore, there did not exist sampling strategies involving coverage criteria based on real-time constraints. Established approaches mostly utilize CIT and apply their techniques to feature models with finite configuration spaces in case of product lines. In Chapter 4, we improved this state of the art in two ways.

First, we introduced a novel coverage criterion called minimum/maximum-delay coverage (M/MD coverage). Here, a covering array is required to include for every test goal (i.e., CoPTA location) a variant comprising a run with BCET (and WCET) for reaching this test goal. Second, we introduced a technique for achieving M/MD coverage for a given CoPTA model. For this, we first defined featured parametric zone graphs to describe the symbolic semantics of a CoPTA, extending zone graphs of TA by constraints over Boolean parameters as well as unbounded numeric parameters. As a result, we obtain a sampling approach specifically tailored to take into account the boundary behavior of product lines in terms of BCET/WCET. This allows us to analyze the real-time behavior of the resulting sample with a higher chance of revealing possible faults concerning boundary behavior. Moreover, our experimental evaluation obtained by applying our prototypical tool implementation to a number of case studies reveals practical applicability of M/MD sampling. Here, our tool can also be applied to larger models with a reasonable computational effort. However, we were not able to compare our approach to other approaches as, to the best of our knowledge there do not exist similar approaches.

Research Challenge 3.1 had the goal to provide an approach for family-based test-suite generation for basic coverage criteria of real-time systems with a potentially infinite number of variants, whereas **Research Challenge 3.2** had the goal to improve the approach of Research Challenge 3.1 such that we can present enhanced coverage criteria w.r.t. best-case/worst-case execution times of real-time systems with a potentially infinite number of variants. Therewith, we can perform quality assurance of CoPTA in terms of generating *complete finite* test suites even in

case of CoPTA having an infinite number of variants. Prior to this work, there did not exist formalisms for these challenges. Instead, established approaches mostly only consider covering basic components of a model (i.e., locations or switches). Hence, in Chapter 5 we improved the state of the art in two ways.

First, we introduced an approach (being sound but incomplete) for generating a *finite* test suite satisfying complete family-based location coverage for product lines with unbounded parametric real-time constraints even in case of a product line with infinitely many variants. In order to achieve this goal, each variability-aware test case comprises a presence condition symbolically describing the (possibly infinite) set of configurations for which the test case is valid. Moreover, we adapted an existing algorithm to systematically generate a complete variability-aware test-suite for a given CoPTA model. Thereafter, we adapted the concept of M/MD coverage presented in Chapter 4 for M/MD test-case generation. For instance, the generated M/MD test cases allow us to reveal potential off-by-one timing errors. Finally, our experimental evaluation based on our tool implementation shows applicability of our approach. In particular, our family-based approach for test-case generation has a higher computational effort than product-by-product test-case generation but produce a much smaller number of test cases. However, without our approach for test-case generation, it would not even be possible to generate *complete finite* test suites in this setting.

Research Challenge 4.1 had the goal to effectively check timed bisimilarity of TA, whereas the goal of **Research Challenge 4.2** was to introduce a basis for checking timed bisimilarity of product lines with a potentially infinite number of variants. As a result, we are able to check whether models being adapted in refinement steps of model-driven engineering still have the same behavior as the original model. In the state of the art before this thesis, there already existed formalisms for effectively checking timed bisimilarity. However, all of these approaches have some disadvantages as compared to the approach that we introduced in Chapter 6. For instance, Čerāns [56] defines timed bisimilarity based on region graphs which suffer from state-space explosion, making the approach generally less efficient than approaches based on zone graphs. Furthermore, the approaches by Weise and Lenzen [201] and Guha et al. [91, 92] are not defined in a way that would directly allow a generalization for checking timed bisimilarity of PTA and CoPTA. Moreover, there did not (to the best of our knowledge) exist an approach for checking timed bisimilarity of PTA and CoPTA. Hence, in Chapter 6 we improved the state of the art in two ways.

In particular, we introduced a novel formalism, called bounded zone-history graphs, for precise, yet scalable checking of timed bisimilarity for non-deterministic TA with silent moves. This allows us to check timed bisimilarity even for large-scale real-world models. Furthermore, we presented the first definition for checking timed bisimilarity of PTA and CoPTA. Additionally, we investigated the subclass of L/U-PTA, where we can use a parameter abstraction, such that we can utilize our framework for TA bisimilarity to check timed bisimilarity for L/U-PTA. As a result, we obtain an effective but imprecise check for parametric timed bisimulation of L/U-PTA. Finally, our experimental evaluation for TA bisimilarity showed applicability of our approach, and by using bounded zone histories, we can

also apply our tool to large-scale models. Here, the evaluation also revealed that non-determinism has a considerable impact on the computational effort for constructing composite zone-history graphs (as expected).

7.2 FUTURE WORK

For future work, there are several open challenges to further improve our framework for specifying and analyzing software systems with configurable real-time behavior.

In terms of **behavioral modeling** of product lines with configurable parametric real-time constraints, we may generalize our formalism to support further non-functional properties. So far, CoPTA only support real-time restrictions. This may be generalized to other properties such as temperature or throughput (e.g., of waterpipes), allowing us to analyze these properties. For instance, we may apply our extensions to Hybrid Automata [14] (i.e., we may extend hybrid automata by features and parameters). Moreover, we plan to automatically derive CoPTA from other representations of product lines (e.g., source code) as CoPTA is a novel formalism not yet being used in practice. Therewith, it would be possible to apply our techniques for quality assurance even in cases where CoPTA are not explicitly used. For instance, we could use an existing approach by Liva et al. [125] as a basis, where TA are automatically extract from Java methods. Additionally, we plan to develop a front end specifically tailored to create CoPTA models and to give an overview on analysis results. This would increase usability of our approaches for modeling and analysis in practice. Currently, our tool utilizes UPPAAL (i.e., a TA model checker) as a front end. However, UPPAAL was not created to handle features and parameters, such that these parts of CoPTA models are (erroneously) marked as faulty in the UPPAAL user interface.

For future work of **sampling strategies** for product lines with infinite configuration spaces, we may adapt M/MD sampling to further coverage criteria. As a result, we would obtain different sets of variants, potentially improving the results of subsequent analyses (e.g., by revealing further faults). For instance, we may require a covering array to contain a test case for reaching locations *via other intermediate locations*. Furthermore, we are interested in finding configurations having fastest and slowest runs *within a particular time frame*. Moreover, the basic idea of M/MD sampling can be used for other non-functional properties as long as there is a variable that can be optimized (i.e., minimized or maximized). Additionally, we plan to extend our evaluation by incorporating a greater number of case studies especially considering larger real-world models. Finally, we would like to compare our approach for computing runs with minimum (or maximum) delay in an evaluation with the approach presented by André et al. [24] for PTA.

For future work of **test-case generation** for product lines with configurable parametric real-time constraints, there are several open challenges. Here, we may adapt M/MD coverage for further coverage criteria (as also proposed for future work of our sampling approach), potentially revealing additional faults. Furthermore, we only generate test cases for allowed behavior (i.e., behavior described by a CoPTA model) so far. Here, we may also consider generating test cases explicitly

describing forbidden behavior. Therewith, we are able to check whether additional (unspecified) behavior was added.

Moreover, it would be interesting to dynamically explore the state space of CoPTA models (i.e., the featured parametric zone graph) instead of picking a location and then querying a model checker for a run reaching this location. Here, a dynamic exploration might be faster in finding runs for easily reachable locations (e.g., in case no other parts of a CoPTA model have to be traversed before a particular clock constraint is satisfiable) whereas our approach is probably faster in finding runs for reaching the remaining locations. Depending on evaluation results, we might also consider combining both approaches. In this context, so-called *spinal test suites* [41] may be helpful. Here, the idea is to utilize existing test cases to derive further test cases. To this end, we would take an existing test case and resume test-case generation after the final test step. For instance, a test case consisting of two pairs of delays and actions may be extended by additional steps to traverse further parts of a CoPTA model. As another point for future work, we plan to extend our evaluation. In particular, we may utilize mutation testing to evaluate effectiveness of the generated test suites. Here, Aichernig et al. [6] already propose mutation operators for TA.

For future work of **checking timed bisimilarity**, we plan to extend our technique to more advanced classes of TA (see the paragraph on *extensions and generalizations of TA* in Section 3.4). This would allow us to cover a wider range of real-time systems which cannot be modeled with classical TA. In case of PTA and CoPTA, we plan to extend zone-history graphs by (Boolean and numeric) parameters such that we have a fully symbolic semantics as a basis for an effective (but incomplete) decision procedure for parametric timed bisimilarity. Based on these extensions for zone-history graphs, we could then also evaluate these approaches to show practicality of timed bisimulation for (real-time) product lines. Finally, we are also interested in compositionality properties of (featured parametric) timed bisimulation to tackle the issue of scalability in the formal analysis of (configurable) real-time behavior.

BIBLIOGRAPHY

- [1] IEEE Standard Glossary of Software Engineering Terminology. *IEEE Std 610.12-1990*, pages 1–84, 1990. doi: 10.1109/IEEESTD.1990.101064. (Cited on pages 2 and 38.)
- [2] Yasmina Abdeddaïm, Eugene Asarin, and Oded Maler. Scheduling with timed automata. *Theoretical Computer Science (TCS)*, 354(2):272–300, 2006. ISSN 0304-3975. doi: 10.1016/j.tcs.2005.11.018. (Cited on page 100.)
- [3] Samson Abramsky. A Domain Equation for Bisimulation. *Information and Computation*, 92(2):161–218, 1991. ISSN 0890-5401. doi: 10.1006/inco.1991.9999. (Cited on page 41.)
- [4] Luca Aceto, Augusto Burgueño, and Kim G. Larsen. Model Checking via Reachability Testing for Timed Automata. In *4th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '98)*, volume 1384 of LNCS, pages 263–280. Springer Berlin Heidelberg, 1998. ISBN 978-3-540-69753-4. doi: 10.1007/BFb0054177. (Cited on page 32.)
- [5] Bestoun S. Ahmed, Kamal Z. Zamli, Wasif Afzal, and Miroslav Bures. Constrained Interaction Testing: A Systematic Literature Study. *IEEE Access*, 5:25706–25730, 2017. ISSN 2169-3536. doi: 10.1109/ACCESS.2017.2771562. (Cited on pages 26 and 100.)
- [6] Bernhard K. Aichernig, Klaus Hörmaier, and Florian Lorber. Debugging with Timed Automata Mutations. In *33rd International Conference on Computer Safety, Reliability, and Security (SAFECOMP '14)*, volume 8666 of LNCS, pages 49–64. Springer International Publishing, 2014. ISBN 978-3-319-10506-2. doi: 10.1007/978-3-319-10506-2_4. (Cited on pages 6, 7, 40, 68, 104, 147, 193, and 205.)
- [7] Bernhard K. Aichernig, Andrea Pferscher, and Martin Tappler. From Passive to Active: Learning Timed Automata Efficiently. In *12th International NASA Formal Methods Symposium (NFM '20)*, volume 12229 of LNCS, pages 1–19. Springer International Publishing, 2020. ISBN 978-3-030-55754-6. doi: 10.1007/978-3-030-55754-6_1. (Cited on page 74.)
- [8] Mustafa Al-Hajjaji, Jens Meinicke, Sebastian Krieter, Reimar Schröter, Thomas Thüm, Thomas Leich, and Gunter Saake. Tool Demo: Testing Configurable Systems with FeatureIDE. In *15th International Conference on Generative Programming: Concepts and Experiences (GPCE '16)*, pages 173–177. ACM, 2016. ISBN 978-1-4503-4446-3. doi: 10.1145/2993236.2993254. (Cited on pages 26 and 100.)
- [9] Rajeev Alur and David L. Dill. Automata for Modeling Real-Time Systems. In *17th International Colloquium on Automata, Languages and Programming (ICALP*

- '90), volume 443 of *LNCS*, pages 322–335. Springer Berlin Heidelberg, 1990. ISBN 978-3-540-47159-2. doi: 10.1007/BFb0032042. (Cited on pages 11, 14, and 64.)
- [10] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994. ISSN 0304-3975. doi: 10.1016/0304-3975(94)90010-8. (Cited on pages 33 and 198.)
- [11] Rajeev Alur and Parthasarathy Madhusudan. *Decision Problems for Timed Automata: A Survey*, volume 3185 of *LNCS*, pages 1–24. Springer Berlin Heidelberg, 2004. ISBN 978-3-540-30080-9. doi: 10.1007/978-3-540-30080-9_1. (Cited on page 64.)
- [12] Rajeev Alur, Costas Courcoubetis, and David L. Dill. Model-checking for Probabilistic Real-time Systems. In *18th International Colloquium on Automata, Languages and Programming (ICALP '90)*, volume 510 of *LNCS*, pages 115–126. Springer Berlin Heidelberg, 1991. ISBN 978-3-540-47516-3. doi: 10.1007/3-540-54233-7_128. (Cited on page 64.)
- [13] Rajeev Alur, Costas Courcoubetis, and David L. Dill. Model-Checking in Dense Real-Time. *Information and Computation (InCo)*, 104(1):2–34, 1993. ISSN 0890-5401. doi: 10.1006/inco.1993.1024. (Cited on pages 33 and 198.)
- [14] Rajeev Alur, Costas Courcoubetis, Thomas A. Henzinger, and Pei-Hsin Ho. Hybrid Automata: An Algorithmic Approach to the Specification and Verification of Hybrid Systems. In *International Hybrid Systems Workshop (HS)*, volume 736 of *LNCS*, pages 209–229. Springer Berlin Heidelberg, 1993. ISBN 978-3-540-48060-0. doi: 10.1007/3-540-57318-6_30. (Cited on pages 64, 66, and 204.)
- [15] Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. Parametric Real-time Reasoning. In *25th Annual Symposium on Theory of Computing (STOC '93)*, pages 592–601. ACM, 1993. ISBN 0-89791-591-7. doi: 10.1145/167088.167242. (Cited on pages 3, 16, 22, 65, 96, 193, and 201.)
- [16] Rajeev Alur, Salvatore La Torre, and George J. Pappas. Optimal Paths in Weighted Timed Automata. In *4th International Workshop on Hybrid Systems: Computation and Control (HSCC '01)*, volume 2034 of *LNCS*, pages 49–62. Springer Berlin Heidelberg, 2001. ISBN 978-3-540-45351-2. doi: 10.1007/3-540-45351-2_8. (Cited on page 100.)
- [17] Jesper R. Andersen, Mathias M. Hansen, and Nicklas Andersen. CAAL 2.0. Technical report, Aalborg University, Department of Computer Science, 2015. URL http://caal.cs.aau.dk/docs/CAAL2_EPG.pdf. (Cited on page 199.)
- [18] Étienne André. IMITATOR: A Tool for Synthesizing Constraints on Timing Bounds of Timed Automata. In *6th International Colloquium on Theoretical Aspects of Computing (ICTAC '09)*, volume 5684 of *LNCS*, pages 336–342. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-03466-4. doi: 10.1007/978-3-642-03466-4_22. (Cited on pages 58, 95, 101, 139, and 145.)

- [19] Étienne André. IMITATOR II: A Tool for Solving the Good Parameters Problem in Timed Automata. In *Proceedings 12th International Workshop on Verification of Infinite-State Systems (INFINITY '10)*, volume 39 of *EPTCS*, pages 91–99. arXiv, 2010. doi: 10.4204/EPTCS.39.7. (Cited on pages 58, 95, 101, 139, and 145.)
- [20] Étienne André. What’s Decidable About Parametric Timed Automata? In *4th International Workshop Formal Techniques for Safety-Critical Systems (FTSCS '15)*, volume 596 of *CCIS*, pages 52–68. Springer International Publishing, 2015. ISBN 978-3-319-29510-7. doi: 10.1007/978-3-319-29510-7_3. (Cited on pages 8, 48, 58, 63, 115, 128, 186, and 198.)
- [21] Étienne André. What’s decidable about parametric timed automata? *International Journal on Software Tools for Technology Transfer (STTT)*, 21(2):203–219, 2019. ISSN 1433-2787. doi: 10.1007/s10009-017-0467-0. (Cited on pages 8, 48, 58, 63, 115, 128, 186, and 198.)
- [22] Étienne André. *IMITATOR User Manual*, 2019. URL <https://github.com/imitator-model-checker/imitator/releases/download/v2.12/IMITATOR-user-manual.pdf>. (Cited on page 58.)
- [23] Étienne André, Laurent Fribourg, Ulrich Kühne, and Romain Soulat. IMITATOR 2.5: A Tool for Analyzing Robustness in Scheduling Problems. In *18th International Symposium on Formal Methods (FM '12)*, volume 7436 of *LNCS*, pages 33–36. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-32759-9. doi: 10.1007/978-3-642-32759-9_6. (Cited on pages 58, 95, 101, 139, and 145.)
- [24] Étienne André, Vincent Bloemen, Laure Petrucci, and Jaco van de Pol. Minimal-Time Synthesis for Parametric Timed Automata. In *25th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '19)*, volume 11428 of *LNCS*, pages 211–228. Springer International Publishing, 2019. ISBN 978-3-030-17465-1. doi: 10.1007/978-3-030-17465-1_12. (Cited on pages 100, 102, 143, 146, and 204.)
- [25] Roberto Bagnara, Patricia M. Hill, and Enea Zaffanella. The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *Science of Computer Programming (SciCo)*, 72(1):3–21, 2008. ISSN 0167-6423. doi: 10.1016/j.scico.2007.08.001. (Cited on page 95.)
- [26] Paolo Baldan, Andrea Corradini, and Ugo Montanari. History Preserving Bisimulation for Contextual Nets. In *14th International Workshop on Algebraic Development Techniques (WADT '99)*, volume 1827 of *LNCS*, pages 291–310. Springer Berlin Heidelberg, 2000. ISBN 978-3-540-44616-3. doi: 10.1007/978-3-540-44616-3_17. (Cited on page 155.)
- [27] Hauke Baller, Sascha Lity, Malte Lochau, and Ina Schaefer. Multi-objective Test Suite Optimization for Incremental Product Family Testing. In *IEEE 7th International Conference on Software Testing, Verification and Validation*, pages 303–312. IEEE, 2014. doi: 10.1109/ICST.2014.43. (Cited on page 101.)

- [28] Don S. Batory. Feature Models, Grammars, and Propositional Formulas. In *9th International Conference on Software Product Lines (SPLC '05)*, pages 7–20. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-32064-7. doi: 10.1007/11554844_3. (Cited on page 17.)
- [29] Gerd Behrmann, Kim G. Larsen, Justin Pearson, Carsten Weise, and Wang Yi. Efficient Timed Reachability Analysis Using Clock Difference Diagrams. In *11th International Conference on Computer Aided Verification (CAV '99)*, volume 1633 of *LNCS*, pages 341–353. Springer Berlin Heidelberg, 1999. ISBN 978-3-540-48683-1. doi: 10.1007/3-540-48683-6_30. (Cited on pages 99 and 143.)
- [30] Gerd Behrmann, Thomas Hune, and Frits Vaandrager. Distributing Timed Model Checking — How the Search Order Matters. In *12th International Conference on Computer Aided Verification (CAV '00)*, volume 1855 of *LNCS*, pages 216–231. Springer Berlin Heidelberg, 2000. ISBN 978-3-540-45047-4. doi: 10.1007/10722167_19. (Cited on pages 99 and 143.)
- [31] Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim G. Larsen, Paul Pettersson, Judi Romijn, and Frits Vaandrager. Minimum-Cost Reachability for Priced Time Automata. In *4th International Workshop on Hybrid Systems: Computation and Control (HSCC '01)*, volume 2034 of *LNCS*, pages 147–161. Springer Berlin Heidelberg, 2001. ISBN 978-3-540-45351-2. doi: 10.1007/3-540-45351-2_15. (Cited on pages 64 and 100.)
- [32] Gerd Behrmann, Patricia Bouyer, Emmanuel Fleury, and Kim G. Larsen. Static Guard Analysis in Timed Automata Verification. In *9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '03)*, volume 2619 of *LNCS*, pages 254–270. Springer Berlin Heidelberg, 2003. ISBN 978-3-540-36577-8. doi: 10.1007/3-540-36577-X_18. (Cited on pages 99 and 143.)
- [33] Gerd Behrmann, Kim G. Larsen, and Jacob I. Rasmussen. Optimal Scheduling Using Priced Timed Automata. *SIGMETRICS Performance Evaluation Review*, 32(4):34–40, 2005. ISSN 0163-5999. doi: 10.1145/1059816.1059823. (Cited on page 100.)
- [34] Gerd Behrmann, Alexandre David, Kim G. Larsen, John Håkansson, Paul Pettersson, Wang Yi, and Martijn Hendriks. UPPAAL 4.0. In *3rd International Conference on the Quantitative Evaluation of Systems (QEST '06)*, pages 125–126. IEEE, 2006. doi: 10.1109/QEST.2006.59. (Cited on pages 101, 110, 139, 144, and 145.)
- [35] Axel Belinfante. JTorX: A Tool for On-Line Model-Driven Test Derivation and Execution. In *16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '10)*, volume 6015 of *LNCS*, pages 266–270. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-12002-2. doi: 10.1007/978-3-642-12002-2_21. (Cited on page 145.)

- [36] Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957. (Cited on pages 34 and 192.)
- [37] Fabian Benduhn, Thomas Thüm, Malte Lochau, Thomas Leich, and Gunter Saake. A Survey on Modeling Techniques for Formal Behavioral Verification of Software Product Lines. In *9th International Workshop on Variability Modelling of Software-intensive Systems (VaMoS '15)*, pages 80–87. ACM, 2015. ISBN 978-1-4503-3273-6. doi: 10.1145/2701319.2701332. (Cited on page 16.)
- [38] Johan Bengtsson and Wang Yi. *Timed Automata: Semantics, Algorithms and Tools*, volume 3098 of *LNCS*, pages 87–124. Springer Berlin Heidelberg, 2004. ISBN 978-3-540-27755-2. doi: 10.1007/978-3-540-27755-2_3. (Cited on pages 12, 33, 34, 35, and 64.)
- [39] Johan Bengtsson, Kim G. Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. UPPAAL — a Tool Suite for Automatic Verification of Real-Time Systems. In *5th International Hybrid Systems Workshop (HS '96)*, volume 1066 of *LNCS*, pages 232–243. Springer Berlin Heidelberg, 1996. ISBN 978-3-540-68334-6. doi: 10.1007/BFb0020949. (Cited on pages 101, 110, 139, 144, 145, and 192.)
- [40] Harsh Beohar and Mohammad Reza Mousavi. Input-Output Conformance Testing Based on Featured Transition Systems. In *29th Annual Symposium on Applied Computing (SAC '14)*, pages 1272–1278. ACM, 2014. ISBN 978-1-4503-2469-4. doi: 10.1145/2554850.2554949. (Cited on page 144.)
- [41] Harsh Beohar and Mohammad Reza Mousavi. Spinal Test Suites for Software Product Lines. In *9th Workshop on Model-Based Testing (MBT '14)*, volume 141 of *EPTCS*, pages 44–55. arXiv, 2014. doi: 10.4204/EPTCS.141.4. (Cited on pages 146 and 205.)
- [42] Harsh Beohar and Mohammad Reza Mousavi. Input-output conformance testing for software product lines. *Journal of Logical and Algebraic Methods in Programming (JLAMP)*, 85(6):1131–1153, 2016. ISSN 2352-2208. doi: 10.1016/j.jlamp.2016.09.007. (Cited on page 144.)
- [43] Nathalie Bertrand, Sophie Pinchinat, and Jean-Baptiste Raclet. Refinement and Consistency of Timed Modal Specifications. In *3rd International Conference on Language and Automata Theory and Applications (LATA '09)*, volume 5457 of *LNCS*, pages 152–163. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-00982-2. doi: 10.1007/978-3-642-00982-2_13. (Cited on page 65.)
- [44] Dirk Beyer, Adam J. Chlipala, Thomas A. Henzinger, Ranjit Jhala, and Rupak Majumdar. Generating tests from counterexamples. In *26th International Conference on Software Engineering (ICSE '04)*, pages 326–335. IEEE, 2004. doi: 10.1109/ICSE.2004.1317455. (Cited on page 110.)
- [45] Behzad Bordbar and Kozo Okano. Verification of Timeliness QoS Properties in Multimedia Systems. In *5th International Conference on Formal Engineering Methods (ICFEM '03)*, volume 2885 of *LNCS*, pages 523–540. Springer Berlin Heidelberg, 2003. ISBN 978-3-540-39893-6. doi: 10.1007/978-3-540-39893-6_30. (Cited on pages 99 and 143.)

- [46] Patricia Bouyer, Catherine Dufourd, Emmanuel Fleury, and Antoine Petit. Are Timed Automata Updatable? In *12th International Conference on Computer Aided Verification (CAV '00)*, volume 1855 of *LNCS*, pages 464–479. Springer Berlin Heidelberg, 2000. ISBN 978-3-540-45047-4. doi: 10.1007/10722167_35. (Cited on page 64.)
- [47] Patricia Bouyer, Catherine Dufourd, Emmanuel Fleury, and Antoine Petit. Updatable timed automata. *Theoretical Computer Science (TCS)*, 321(2):291–345, 2004. ISSN 0304-3975. doi: 10.1016/j.tcs.2004.04.003. (Cited on page 64.)
- [48] Laura Brandán Briones and Mathias Röhl. *Test Derivation from Timed Automata*, volume 3472 of *LNCS*, pages 201–231. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-32037-1. doi: 10.1007/11498490_10. (Cited on page 144.)
- [49] Antonio Brogi, Andrea Canciani, and Jacopo Soldani. Fault-aware management protocols for multi-component applications. *Journal of Systems and Software (JSS)*, 139:189–210, 2018. ISSN 0164-1212. doi: 10.1016/j.jss.2018.02.005. (Cited on page 144.)
- [50] Ferenc Bujtor, Sascha Fendrich, Gerald Lüttgen, and Walter Vogler. Non-deterministic Modal Interfaces. In *41st International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM '15)*, volume 8939 of *LNCS*, pages 152–163. Springer Berlin Heidelberg, 2015. ISBN 978-3-662-46078-8. doi: 10.1007/978-3-662-46078-8_13. (Cited on page 144.)
- [51] Béatrice Bérard, Antoine Petit, Volker Diekert, and Paul Gastin. Characterization of the Expressive Power of Silent Transitions in Timed Automata. *Fundamenta Informaticae (FI)*, 36:145–182, 1998. doi: 10.3233/FI-1998-36233. (Cited on page 13.)
- [52] Johannes Bürdek. *Rekonfigurierbare Software-Systeme: Spezifikation und Testfall-generierung*. PhD thesis, TU Darmstadt, 2018. (Cited on page 17.)
- [53] Johannes Bürdek, Malte Lochau, Stefan Bauregger, Andreas Holzer, Alexander von Rhein, Sven Apel, and Dirk Beyer. Facilitating Reuse in Multi-goal Test-Suite Generation for Software Product Lines. In *18th International Conference on Fundamental Approaches to Software Engineering (FASE '15)*, volume 9033 of *LNCS*, pages 84–99. Springer Berlin Heidelberg, 2015. ISBN 978-3-662-46675-9. doi: 10.1007/978-3-662-46675-9_6. (Cited on pages 2, 3, 7, 26, 39, 80, 103, 113, 116, 118, 120, 122, 123, and 144.)
- [54] Rachel Cardell-Oliver and Tim Glover. A Practical and Complete Algorithm for Testing Real-Time Systems. In *5th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT '98)*, volume 1486 of *LNCS*, pages 251–261. Springer Berlin Heidelberg, 1998. ISBN 978-3-540-49792-9. doi: 10.1007/BFb0055352. (Cited on pages 39, 105, 109, 110, 111, 121, and 198.)
- [55] Franck Cassez and Kim G. Larsen. The Impressive Power of Stopwatches. In *11th International Conference on Concurrency Theory (CONCUR '00)*, vol-

- ume 1877 of *LNCS*, pages 138–152. Springer Berlin Heidelberg, 2000. ISBN 978-3-540-44618-7. doi: 10.1007/3-540-44618-4_12. (Cited on page 64.)
- [56] Kārlis Čerāns. Decidability of Bisimulation Equivalences for Parallel Timer Processes. In *4th International Conference on Computer Aided Verification (CAV '92)*, volume 663 of *LNCS*, pages 302–315. Springer Berlin Heidelberg, 1992. ISBN 978-3-540-47572-9. doi: 10.1007/3-540-56496-9_24. (Cited on pages 7, 33, 45, 151, 198, 199, and 203.)
- [57] Kārlis Čerāns, Jens C. Godskesen, and Kim G. Larsen. Timed Modal Specification — Theory and Tools. In *5th International Conference on Computer Aided Verification (CAV '93)*, volume 697 of *LNCS*, pages 253–267. Springer Berlin Heidelberg, 1993. ISBN 978-3-540-47787-7. doi: 10.1007/3-540-56922-7_21. (Cited on page 65.)
- [58] Harald Cichos, Sebastian Oster, Malte Lochau, and Andy Schürr. Model-Based Coverage-Driven Test Suite Generation for Software Product Lines. In *14th International Conference on Model Driven Engineering Languages and Systems (MODELS '11)*, volume 6981 of *LNCS*, pages 425–439. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-24485-8. doi: 10.1007/978-3-642-24485-8_31. (Cited on pages 7, 39, 114, and 116.)
- [59] Dave Clarke, Michiel Helvensteijn, and Ina Schaefer. Abstract Delta Modeling. *SIGPLAN Notices*, 46:13–22, 2010. ISSN 0362-1340. doi: 10.1145/1942788.1868298. (Cited on page 144.)
- [60] Andreas Classen, Patrick Heymans, Pierre-Yves Schobbens, Axel Legay, and Jean-François Raskin. Model Checking Lots of Systems: Efficient Verification of Temporal Properties in Software Product Lines. In *32nd International Conference on Software Engineering (ICSE '10)*, pages 335–344. ACM, 2010. ISBN 978-1-60558-719-6. doi: 10.1145/1806799.1806850. (Cited on pages 19, 21, 65, and 144.)
- [61] Andreas Classen, Patrick Heymans, Pierre-Yves Schobbens, and Axel Legay. Symbolic Model Checking of Software Product Lines. In *33rd International Conference on Software Engineering (ICSE '11)*, pages 321–330. ACM, 2011. ISBN 978-1-4503-0445-0. doi: 10.1145/1985793.1985838. (Cited on page 19.)
- [62] Guillermina Cledou, José Proença, and Luís S. Barbosa. Composing Families of Timed Automata. In *7th International Conference on Fundamentals of Software Engineering (FSEN '17)*, volume 10522 of *LNCS*, pages 51–66. Springer International Publishing, 2017. ISBN 978-3-319-68972-2. doi: 10.1007/978-3-319-68972-2_4. (Cited on page 65.)
- [63] Guillermina Cledou, José Proença, and Luís S. Barbosa. A Refinement Relation for Families of Timed Automata. In *20th Brazilian Symposium on Formal Methods: Foundations and Applications (SBMF '17)*, volume 10623 of *LNCS*, pages 161–178. Springer International Publishing, 2017. ISBN 978-3-319-70848-5. doi: 10.1007/978-3-319-70848-5_11. (Cited on page 65.)

- [64] Paul Clements and Linda M. Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley Reading, 2001. (Cited on page 16.)
- [65] Anastasia Cmyrev and Ralf Reissing. Efficient and Effective Testing of Automotive Software Product Lines. *Applied Science and Engineering Progress*, 7(2):53–57, 2014. doi: 10.14416/j.ijast.2014.05.001. (Cited on page 100.)
- [66] David M. Cohen, Siddhartha R. Dalal, Michael L. Fredman, and Gardner C. Patton. The AETG System: An Approach to Testing Based on Combinatorial Design. *IEEE Transactions on Software Engineering (TSE)*, 23(7):437–444, 1997. ISSN 0098-5589. doi: 10.1109/32.605761. (Cited on pages 26 and 100.)
- [67] Aurore Collomb-Annichini and Mihaela Sighireanu. Parameterized Reachability Analysis of the IEEE 1394 Root Contention Protocol using TReX. Technical report, 2001. URL <https://hal.archives-ouvertes.fr/hal-00110454/>. (Cited on page 193.)
- [68] Keith D. Cooper and Linda Torczon. *Engineering a Compiler*. Morgan Kaufmann, 2011. (Cited on page 96.)
- [69] Maxime Cordy and Axel Legay. Verification and abstraction of real-time variability-intensive systems. *International Journal on Software Tools for Technology Transfer (STTT)*, 21(6):635–649, 2019. ISSN 1433-2787. doi: 10.1007/s10009-019-00537-z. (Cited on page 145.)
- [70] Maxime Cordy, Pierre-Yves Schobbens, Patrick Heymans, and Axel Legay. Behavioural Modelling and Verification of Real-time Software Product Lines. In *16th International Software Product Line Conference (SPLC '12)*, pages 66–75. ACM, 2012. ISBN 978-1-4503-1094-9. doi: 10.1145/2362536.2362549. (Cited on pages 3, 16, 19, 21, 38, 65, 101, and 201.)
- [71] Maxime Cordy, Andreas Classen, Patrick Heymans, Pierre-Yves Schobbens, and Axel Legay. ProVeLines: A Product Line of Verifiers for Software Product Lines. In *17th International Software Product Line Conference Co-located Workshops (SPLC '13 Workshops)*, pages 141–146. ACM, 2013. ISBN 978-1-4503-2325-3. doi: 10.1145/2499777.2499781. (Cited on pages 101 and 145.)
- [72] Maxime Cordy, Axel Legay, Pierre-Yves Schobbens, and Louis-Marie Traonouez. A Framework for the Rigorous Design of Highly Adaptive Timed Systems. In *1st FME Workshop on Formal Methods in Software Engineering (FormaliSE '13)*, pages 64–70. IEEE, 2013. doi: 10.1109/FormaliSE.2013.6612279. (Cited on pages 101 and 145.)
- [73] Maxime Cordy, Marco Willemart, Bruno Dawagne, Patrick Heymans, and Pierre-Yves Schobbens. An Extensible Platform for Product-line Behavioural Analysis. In *18th International Software Product Line Conference Co-located Workshops (SPLC '14 Workshops)*, pages 102–109. ACM, 2014. ISBN 978-1-4503-2739-8. doi: 10.1145/2647908.2655973. (Cited on pages 101 and 145.)

- [74] Costas Courcoubetis and Mihalis Yannakakis. Minimum and Maximum Delay Problems in Real-Time Systems. *Formal Methods in System Design*, 1(4): 385–415, 1992. ISSN 1572-8102. doi: 10.1007/BF00709157. (Cited on pages 7, 40, and 100.)
- [75] Krzysztof Czarnecki and Ulrich W. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison Wesley, 2000. (Cited on page 1.)
- [76] Paulo Anselmo da Mota Silveira Neto, Ivan do Carmo Machado, John D. McGregor, Eduardo Santana de Almeida, and Silvio Romero de Lemos Meira. A systematic mapping study of software product lines testing. *Information and Software Technology (IST)*, 53(5):407–423, 2011. ISSN 0950-5849. doi: 10.1016/j.infsof.2010.12.003. (Cited on page 144.)
- [77] Carlos Diego N. Damasceno, Mohammad Reza Mousavi, and Adenilso Simão. Learning from Difference: An Automated Approach for Learning Family Models from Software Product Lines. In *23rd International Software Product Line Conference (SPLC '19)*, pages 52—63. ACM, 2019. ISBN 9781450371384. doi: 10.1145/3336294.3336307. (Cited on page 74.)
- [78] Carlos Diego N. Damasceno, Mohammad Reza Mousavi, and Adenilso Simão. Learning by Sampling: Learning Behavioral Family Models from Software Product Lines. Accepted for publication in *Empirical Software Engineering*, 2020. (Cited on page 74.)
- [79] Alexandre David, Kim G. Larsen, Axel Legay, Ulrik Nyman, and Andrzej Wasowski. Timed I/O Automata: A Complete Specification Theory for Real-time Systems. In *13th ACM International Conference on Hybrid Systems: Computation and Control (HSCC '10)*, pages 91–100. ACM, 2010. ISBN 978-1-60558-955-8. doi: 10.1145/1755952.1755967. (Cited on page 32.)
- [80] Leonardo de Moura and Nikolaj Bjørner. Z3: An Efficient SMT Solver. In *14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '08)*, volume 4963 of LNCS, pages 337–340. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-78800-3. doi: 10.1007/978-3-540-78800-3_24. (Cited on page 95.)
- [81] Jeremy Dick and Alain Faivre. Automating the Generation and Sequencing of Test Cases from Model-Based Specifications. In *1st International Symposium of Formal Methods Europe (FME '93)*, volume 670 of LNCS, pages 268–284. Springer Berlin Heidelberg, 1993. ISBN 978-3-540-47623-8. doi: 10.1007/BFb0024651. (Cited on pages 6 and 30.)
- [82] David L. Dill. Timing Assumptions and Verification of Finite-State Concurrent Systems. In *1st International Workshop on Automatic Verification Methods for Finite State Systems (CAV '89)*, volume 407 of LNCS, pages 197–212. Springer Berlin Heidelberg, 1989. ISBN 978-3-540-46905-6. doi: 10.1007/3-540-52148-8_17. (Cited on pages 33, 34, and 192.)

- [83] Uli Fahrenberg and Axel Legay. Quantitative properties of featured automata. *International Journal on Software Tools for Technology Transfer (STTT)*, 21(6): 667–677, 2019. ISSN 1433-2787. doi: 10.1007/s10009-019-00538-y. (Cited on page 100.)
- [84] Stefan Feldmann, Julia Fuchs, and Birgit Vogel-Heuser. Modularity, variant and version management in plant automation – future challenges and state of the art. In *12th International Design Conference (DESIGN '12)*, pages 1689–1698. The Design Society, 2012. (Cited on page 1.)
- [85] Elena Fersman, Pavel Krčál, Paul Pettersson, and Wang Yi. Task automata: Schedulability, decidability and undecidability. *Information and Computation (InCo)*, 205(8):1149–1172, 2007. ISSN 0890-5401. doi: 10.1016/j.ic.2007.01.009. (Cited on page 64.)
- [86] Gerard Florin, Céline Fraize, and Stéphane Natkin. Stochastic Petri nets: Properties, applications and tools. *Microelectronics Reliability*, 31(4):669–697, 1991. ISSN 0026-2714. doi: 10.1016/0026-2714(91)90009-V. (Cited on page 64.)
- [87] Laurent Fribourg and Ulrich Kühne. Parametric Verification and Test Coverage for Hybrid Automata Using the Inverse Method. In *5th International Workshop on Reachability Problems (RP '11)*, volume 6945 of LNCS, pages 191–204. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-24288-5. doi: 10.1007/978-3-642-24288-5_17. (Cited on page 65.)
- [88] Roman Froschauer, Deepak Dhungana, and Paul Grünbacher. Managing the Life-cycle of Industrial Automation Systems with Product Line Variability Models. In *34th Euromicro Conference on Software Engineering and Advanced Applications (SEAA '08)*, pages 35–42. IEEE, 2008. doi: 10.1109/SEAA.2008.21. (Cited on page 1.)
- [89] Bharat Garhewal, Frits Vaandrager, Falk Howar, Timo Schrijvers, Toon Lenaerts, and Rob Smits. Grey-Box Learning of Register Automata. In *18th International Conference on Integrated Formal Methods (IFM '20)*, volume 12546 of LNCS, pages 22–40. Springer International Publishing, 2020. ISBN 978-3-030-63461-2. doi: 10.1007/978-3-030-63461-2_2. (Cited on page 74.)
- [90] Shibashis Guha, Chinmay Narayan, and S. Arun-Kumar. Deciding Timed Bisimulation for Timed Automata Using Zone Valuation Graph, 2012. (Cited on page 154.)
- [91] Shibashis Guha, Chinmay Narayan, and S. Arun-Kumar. On Decidability of Prebisimulation for Timed Automata. In *24th International Conference on Computer Aided Verification (CAV '12)*, volume 7358 of LNCS, pages 444–461. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-31424-7. doi: 10.1007/978-3-642-31424-7_33. (Cited on pages 7, 151, 153, 198, 199, and 203.)
- [92] Shibashis Guha, Shankara Narayanan Krishna, Chinmay Narayan, and S. Arun-Kumar. A Unifying Approach to Decide Relations for Timed Automata and their Game Characterization. In *Combined 20th International*

- Workshop on Expressiveness in Concurrency and 10th Workshop on Structural Operational Semantics (EXPRESS/SOS '13)*, volume 120 of *EPTCS*, pages 47–62. arXiv, 2013. doi: 10.4204/EPTCS.120.5. (Cited on pages 7, 151, 198, 199, and 203.)
- [93] Hendrik Göttmann, Lars Luthmann, Malte Lochau, and Andy Schürr. Real-Time-Aware Reconfiguration Decisions for Dynamic Software Product Lines. In *24th International Systems and Software Product Line Conference (SPLC '20)*, pages 1–11. ACM, 2020. ISBN 978-1-4503-7569-6. doi: 10.1145/3382025.3414945. (Cited on page 144.)
- [94] Vanderson Hafemann Fragal, Adenilso Simão, and Mohammad Reza Mousavi. Validated Test Models for Software Product Lines: Featured Finite State Machines. In *13th International Conference on Formal Aspects of Component Software (FACS '16)*, volume 10231 of *LNCS*, pages 210–227. Springer International Publishing, 2016. ISBN 978-3-319-57665-7. doi: 10.1007/978-3-319-57666-4_13. (Cited on page 145.)
- [95] Vanderson Hafemann Fragal, Adenilso Simão, Mohammad Reza Mousavi, and Uraz Cengiz Türker. Extending HSI Test Generation Method for Software Product Lines. *The Computer Journal*, 62(1):109–129, 2018. ISSN 0010-4620. doi: 10.1093/comjnl/bxy046. (Cited on page 145.)
- [96] Evelyn N. Haslinger, Roberto E. Lopez-Herrejon, and Alexander Egyed. Using Feature Model Knowledge to Speed Up the Generation of Covering Arrays. In *7th International Workshop on Variability Modelling of Software-intensive Systems (VaMoS '13)*, pages 16:1–16:6. ACM, 2013. ISBN 978-1-4503-1541-8. doi: 10.1145/2430502.2430524. (Cited on pages 26 and 100.)
- [97] Klaus Havelund, Arne Skou, Kim G. Larsen, and Kristian Lund. Formal modeling and analysis of an audio/video protocol: An industrial case study using UPPAAL. In *Proceedings Real-Time Systems Symposium*, pages 2–13. IEEE, 1997. doi: 10.1109/REAL.1997.641264. (Cited on page 193.)
- [98] Martijn Hendriks, Gerd Behrmann, Kim Larsen, Peter Niebert, and Frits Vaandrager. Adding Symmetry Reduction to UPPAAL. In *1st International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS '03)*, volume 2791 of *LNCS*, pages 46–59. Springer Berlin Heidelberg, 2003. ISBN 978-3-540-40903-8. doi: 10.1007/978-3-540-40903-8_5. (Cited on pages 99 and 143.)
- [99] Thomas A. Henzinger, Zohar Manna, and Amir Pnueli. Temporal Proof Methodologies for Real-time Systems. In *18th Symposium on Principles of Programming Languages, (POPL '91)*, pages 353–366. ACM, 1991. ISBN 0-89791-419-8. doi: 10.1145/99583.99629. (Cited on pages 7 and 31.)
- [100] Thomas A. Henzinger, Zohar Manna, and Amir Pnueli. Timed Transition Systems. In *Workshop on Real-Time: Theory in Practice (REX '91)*, volume 600 of *LNCS*, pages 226–251. Springer Berlin Heidelberg, 1991. ISBN 978-3-540-47218-6. doi: 10.1007/BFb0031995. (Cited on pages 7 and 31.)

- [101] Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic Model Checking for Real-Time Systems. *Information and Computation (InCo)*, 111(2):193–244, 1994. ISSN 0890-5401. doi: 10.1006/inco.1994.1045. (Cited on page 14.)
- [102] Anders Hessel, Kim G. Larsen, Marius Mikucionis, Brian Nielsen, Paul Pettersson, and Arne Skou. *Testing Real-Time Systems Using UPPAAL*, volume 4949 of *LNCS*, pages 77–117. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-78917-8. doi: 10.1007/978-3-540-78917-8_3. (Cited on page 143.)
- [103] Thomas Hune, Judi Romijn, Mariëlle Stoelinga, and Frits Vaandrager. Linear parametric model checking of timed automata. *The Journal of Logic and Algebraic Programming (JLAP)*, 52–53:183–220, 2002. ISSN 1567-8326. doi: 10.1016/S1567-8326(02)00037-1. (Cited on pages 38, 186, 187, and 188.)
- [104] *IBM ILOG CPLEX Optimization Studio CPLEX User’s Manual*. IBM Corp., 2017. https://www.ibm.com/support/knowledgecenter/SSSA5P_12.8.0/ilog.odms.studio.help/pdf/usrcplex.pdf. (Cited on page 192.)
- [105] Henrik E. Jensen, Kim G. Larsen, and Arne Skou. Modelling and Analysis of a Collision Avoidance Protocol using SPIN and UPPAAL. Technical report, Aalborg University, 1996. (Cited on page 193.)
- [106] Martin F. Johansen, Øystein Haugen, and Franck Fleurey. An Algorithm for Generating t-wise Covering Arrays from Large Feature Models. In *16th International Software Product Line Conference (SPLC ’12)*, pages 46–55. ACM, 2012. ISBN 978-1-4503-1094-9. doi: 10.1145/2362536.2362547. (Cited on pages 26, 67, and 100.)
- [107] Martin F. Johansen, Øystein Haugen, Franck Fleurey, Anne G. Eldegard, and Torbjørn Syversen. Generating Better Partial Covering Arrays by Modeling Weights on Sub-product Lines. In *15th International Conference on Model Driven Engineering Languages and Systems (MODELS ’12)*, volume 7590 of *LNCS*, pages 269–284. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-33666-9. doi: 10.1007/978-3-642-33666-9_18. (Cited on pages 26 and 100.)
- [108] Kyo C. Kang, Sholom G. Cohen, James A. Hess, William E. Novak, and A. Spencer Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical report, Software Engineering Institute, Carnegie Mellon University, 1990. (Cited on pages 1, 16, and 18.)
- [109] Ahmet S. Karataş, Halit Oğuztüzün, and Ali Doğru. Mapping Extended Feature Models to Constraint Logic Programming over Finite Domains. In *14th International Conference on Software Product Lines (SPLC ’10)*, volume 6287 of *LNCS*, pages 286–299. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-15579-6. doi: 10.1007/978-3-642-15579-6_20. (Cited on pages 48 and 50.)
- [110] Joost-Pieter Katoen, David N. Jansen, and Jasper Berendsen. Probably on Time and within Budget: On Reachability in Priced Probabilistic Timed

- Automata. In *3rd International Conference on the Quantitative Evaluation of Systems (QEST '06)*, pages 311–322. IEEE, 2006. doi: 10.1109/QEST.2006.43. (Cited on page 64.)
- [111] Robert M. Keller. Formal verification of parallel programs. *Communications of the ACM*, 19(7):371–384, 1976. ISSN 0001-0782. doi: 10.1145/360248.360251. (Cited on page 41.)
- [112] Jin Hyun Kim, Axel Legay, Louis-Marie Traonouez, Mathieu Acher, and Sungwon Kang. A Formal Modeling and Analysis Framework for Software Product Line of Preemptive Real-time Systems. In *31st Annual ACM Symposium on Applied Computing (SAC '16)*, pages 1562–1565. ACM, 2016. ISBN 978-1-4503-3739-7. doi: 10.1145/2851613.2851977. (Cited on page 65.)
- [113] Andrew King, Oleg Sokolsky, and Insup Lee. A Modal Specification Theory for Timing Variability. Technical report, University of Pennsylvania, 2013. URL https://repository.upenn.edu/cis_reports/987/. (Cited on page 65.)
- [114] Sergiy S. Kolesnikov, Sven Apel, Norbert Siegmund, Stefan Sobernig, Christian Kästner, and Semah Senkaya. Predicting Quality Attributes of Software Product Lines Using Software and Network Measures and Sampling. In *7th International Workshop on Variability Modelling of Software-intensive Systems (VaMoS '13)*, pages 6:1–6:5. ACM, 2013. ISBN 978-1-4503-1541-8. doi: 10.1145/2430502.2430511. (Cited on page 101.)
- [115] Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer, 2012. (Cited on page 29.)
- [116] Moez Krichen and Stavros Tripakis. Real-Time Testing with Timed Automata Testers and Coverage Criteria. In *2nd International Conference on Formal Modeling and Analysis of Timed and Systems (FORMATS '04)*, volume 3253 of LNCS, pages 134–151. Springer Berlin Heidelberg, 2004. ISBN 978-3-540-30206-3. doi: 10.1007/978-3-540-30206-3_11. (Cited on pages 108, 120, and 143.)
- [117] Marta Kwiatkowska, Gethin Norman, Roberto Segala, and Jeremy Sproston. Verifying Quantitative Properties of Continuous Probabilistic Timed Automata. In *11th International Conference on Concurrency Theory (CONCUR '00)*, volume 1877 of LNCS, pages 123–137. Springer Berlin Heidelberg, 2000. ISBN 978-3-540-44618-7. doi: 10.1007/3-540-44618-4_11. (Cited on page 64.)
- [118] Kim G. Larsen and Bent Thomsen. A Modal Process Logic. In *3rd Annual Symposium on Logic in Computer Science (LICS '88)*, pages 203–210. IEEE, 1988. doi: 10.1109/LICS.1988.5119. (Cited on page 65.)
- [119] Kim G. Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. Efficient Verification of Real-Time Systems: Compact Data Structure and State-Space Reduction. In *18th IEEE Real-Time Systems Symposium (RTSS '97)*, pages 14–24. IEEE, 1997. doi: 10.1109/REAL.1997.641265. (Cited on pages 99 and 143.)

- [120] Kim G. Larsen, Marius Mikucionis, and Brian Nielsen. Online Testing of Real-time Systems Using UPPAAL. In *4th International Workshop on Formal Approaches to Software Testing (FATES '04)*, volume 3395 of LNCS, pages 79–94. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-31848-4. doi: 10.1007/978-3-540-31848-4_6. (Cited on page 145.)
- [121] Jihyun Lee, Sungwon Kang, and Danhyung Lee. A Survey on Software Product Line Testing. In *16th International Software Product Line Conference (SPLC '12)*, pages 31–40. ACM, 2012. ISBN 9781450310949. doi: 10.1145/2362536.2362545. (Cited on page 144.)
- [122] Christoph Legat, Jens Folmer, and Birgit Vogel-Heuser. Evolution in Industrial Plant Automation: A Case Study. In *39th Annual Conference of the IEEE Industrial Electronics Society (IECON '13)*, pages 4386–4391. IEEE, 2013. doi: 10.1109/IECON.2013.6699841. (Cited on pages 1, 9, and 96.)
- [123] Magnus Lindahl, Paul Pettersson, and Wang Yi. Formal Design and Analysis of a Gear Controller. In *4th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '98)*, volume 1384 of LNCS, pages 281–297. Springer Berlin Heidelberg, 1998. ISBN 978-3-540-69753-4. doi: 10.1007/BFb0054178. (Cited on page 193.)
- [124] Sascha Lity, Thomas Morbach, Thomas Thüm, and Ina Schaefer. Applying Incremental Model Slicing to Product-Line Regression Testing. In *15th International Conference on Software Reuse (ICSR '16)*, volume 9679 of LNCS, pages 3–19. Springer International Publishing, 2016. ISBN 978-3-319-35122-3. doi: 10.1007/978-3-319-35122-3_1. (Cited on page 145.)
- [125] Giovanni Liva, Muhammad T. Khan, and Martin Pinzger. Extracting Timed Automata from Java Methods. In *17th International Working Conference on Source Code Analysis and Manipulation (SCAM '17)*, pages 91–100. IEEE, 2017. doi: 10.1109/SCAM.2017.9. (Cited on pages 66 and 204.)
- [126] Malte Lochau, Ina Schaefer, Jochen Kamischke, and Sascha Lity. Incremental Model-Based Testing of Delta-Oriented Software Product Lines. In *6th International Conference on Tests and Proofs (TAP '12)*, volume 7305 of LNCS, pages 67–82. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-30473-6. doi: 10.1007/978-3-642-30473-6_7. (Cited on page 144.)
- [127] Malte Lochau, Johannes Bürdek, Sascha Lity, Matthias Hagner, Christoph Legat, Ursula Goltz, and Andy Schürr. Applying Model-based Software Product Line Testing Approaches to the Automation Engineering Domain. *Automatisierungstechnik*, 62(11):771–780, 2014. doi: 10.1515/auto-2014-1099. (Cited on page 1.)
- [128] Malte Lochau, Lars Luthmann, Hendrik Göttmann, and Isabelle Bacher. Parametric Timed Bisimulation. In *9th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA '20)*, volume 12477 of LNCS, pages 55–71. Springer International Publishing, 2020. ISBN

- 978-3-030-61470-6. doi: 10.1007/978-3-030-61470-6_5. (Cited on pages 152, 178, and 179.)
- [129] Roberto E. Lopez-Herrejon, Stefan Fischer, Rudolf Ramler, and Alexander Egyed. A First Systematic Mapping Study on Combinatorial Interaction Testing for Software Product Lines. In *8th International Conference on Software Testing, Verification and Validation Workshops (ICSTW '15)*, pages 1–10. IEEE, 2015. doi: 10.1109/ICSTW.2015.7107435. (Cited on pages 26 and 100.)
- [130] Lars Luthmann, Stephan Mennicke, and Malte Lochau. Towards an I/O Conformance Testing Theory for Software Product Lines based on Modal Interface Automata. In *6th International Workshop on Formal Methods and Analysis in SPL Engineering (FMSPLE '15)*, volume 182 of *EPTCS*, pages 1–13. arXiv, 2015. doi: 10.4204/EPTCS.182.1. (Cited on page 144.)
- [131] Lars Luthmann, Stephan Mennicke, and Malte Lochau. Compositionality, Decompositionality and Refinement in Input/Output Conformance Testing. In *13th International Conference on Formal Aspects of Component Software (FACS '16)*, volume 10231 of *LNCS*, pages 54–72. Springer International Publishing, 2016. ISBN 978-3-319-57666-4. doi: 10.1007/978-3-319-57666-4_5. (Cited on page 144.)
- [132] Lars Luthmann, Stephan Mennicke, and Malte Lochau. Compositionality, Decompositionality and Refinement in Input/Output Conformance Testing – Technical Report. arXiv, 2016. URL <https://arxiv.org/abs/1606.09035>. (Cited on page 144.)
- [133] Lars Luthmann, Andreas Stephan, Johannes Bürdek, and Malte Lochau. Modeling and Testing Product Lines with Unbounded Parametric Real-Time Constraints. In *21st International Systems and Software Product Line Conference (SPLC '17)*, pages 104–113. ACM, 2017. ISBN 978-1-4503-5221-5. doi: 10.1145/3106195.3106204. (Cited on pages 50, 69, and 104.)
- [134] Lars Luthmann, Timo Gerecht, and Malte Lochau. Sampling Strategies for Product Lines with Unbounded Parametric Real-Time Constraints. *International Journal on Software Tools for Technology Transfer (STTT)*, 21:613–633, 2019. ISSN 1433-2787. doi: 10.1007/s10009-019-00532-4. (Cited on pages 69 and 100.)
- [135] Lars Luthmann, Timo Gerecht, Andreas Stephan, Johannes Bürdek, and Malte Lochau. Minimum/maximum delay testing of product lines with unbounded parametric real-time constraints. *Journal of Systems and Software (JSS)*, 149:535–553, 2019. ISSN 0164-1212. doi: 10.1016/j.jss.2018.12.028. (Cited on pages 105 and 143.)
- [136] Lars Luthmann, Hendrik Göttmann, Isabelle Bacher, and Malte Lochau. Checking Timed Bisimulation with Bounded Zone-History Graphs – Technical Report. arXiv, 2019. URL <https://arxiv.org/abs/1910.08992>. (Cited on page 191.)

- [137] Lars Luthmann, Hendrik Göttmann, and Malte Lochau. Compositional Liveness-Preserving Conformance Testing of Timed I/O Automata. In *16th International Conference on Formal Aspects of Component Software (FACS '19)*, volume 12018 of *LNCS*, pages 147–169. Springer International Publishing, 2019. ISBN 978-3-030-40914-2. doi: 10.1007/978-3-030-40914-2_8. (Cited on page 144.)
- [138] Lars Luthmann, Hendrik Göttmann, and Malte Lochau. Compositional Liveness-Preserving Conformance Testing of Timed I/O Automata – Technical Report. arXiv, 2019. URL <https://arxiv.org/abs/1909.03703>. (Cited on page 144.)
- [139] Lars Luthmann, Stephan Mennicke, and Malte Lochau. Unifying modal interface theories and compositional input/output conformance testing. *Science of Computer Programming (SciCo)*, 172:27–47, 2019. ISSN 0167-6423. doi: 10.1016/j.scico.2018.09.008. (Cited on page 144.)
- [140] Lars Luthmann, Hendrik Göttmann, Isabelle Bacher, and Malte Lochau. Checking Timed Bisimulation with Bounded Zone-History Graphs. Submitted to *Acta Informatica*, 2020. (Cited on pages 152, 191, and 196.)
- [141] Gerald Lüttgen and Walter Vogler. Modal Interface Automata. In *7th International Conference on Theoretical Computer Science (TCS '12)*, volume 7604 of *LNCS*, pages 265–279. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-33475-7. doi: 10.1007/978-3-642-33475-7_19. (Cited on page 144.)
- [142] C. R. Maga and N. Jazdi. An Approach for Modeling Variants of Industrial Automation Systems. In *17th IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR '10)*, pages 1–6. IEEE, 2010. doi: 10.1109/AQTR.2010.5520918. (Cited on page 1.)
- [143] Oded Maler, Amir Pnueli, and Joseph Sifakis. On the synthesis of discrete controllers for timed systems. In *12th Annual Symposium on Theoretical Aspects of Computer Science (STACS '95)*, volume 900 of *LNCS*, pages 229–242. Springer Berlin Heidelberg, 1995. ISBN 978-3-540-49175-0. doi: 10.1007/3-540-59042-0_76. (Cited on page 64.)
- [144] Philip M. Merlin and David J. Farber. Recoverability of Communication Protocols – Implications of a Theoretical Study. *IEEE Transactions on Communications*, 24(9):1036–1043, 1976. ISSN 0090-6778. doi: 10.1109/TCOM.1976.1093424. (Cited on page 64.)
- [145] Joseph S. Miller. Decidability and Complexity Results for Timed Automata and Semi-linear Hybrid Automata. In *3rd International Workshop on Hybrid Systems: Computation and Control (HSCC '00)*, volume 1790 of *LNCS*, pages 296–310. Springer Berlin Heidelberg, 2000. ISBN 978-3-540-46430-3. doi: 10.1007/3-540-46430-1_26. (Cited on page 64.)
- [146] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *LNCS*. Springer, 1980. ISBN 3-540-10235-3. doi: 10.1007/3-540-10235-3. (Cited on page 198.)

- [147] Ralf Mitsching, Carsten Weise, and Stefan Kowalewski. Design Patterns for Integrating Variability in Timed Automata. In *4th International Conference on Secure Software Integration and Reliability Improvement (SSIRI '10)*, pages 38–45. IEEE, 2010. doi: 10.1109/SSIRI-C.2010.21. (Cited on page 65.)
- [148] Faron Moller and Chris Tofts. A Temporal Calculus of Communicating Systems. In *1st International Conference on Concurrency Theory (CONCUR '90)*, volume 458 of *LNCS*, pages 401–415. Springer Berlin Heidelberg, 1990. ISBN 978-3-540-46395-5. doi: 10.1007/BFb0039073. (Cited on pages 43 and 198.)
- [149] Xavier Nicollin and Joseph Sifakis. The Algebra of Timed Processes, ATP: Theory and Application. *Information and Computation (InCo)*, 114(1):131–178, 1994. ISSN 0890-5401. doi: 10.1006/inco.1994.1083. (Cited on page 198.)
- [150] Changhai Nie and Hareton Leung. A Survey of Combinatorial Testing. *ACM Computing Surveys (CSUR)*, 43(2):11:1–11:29, 2011. ISSN 0360-0300. doi: 10.1145/1883612.1883618. (Cited on pages 2, 26, 27, 29, and 67.)
- [151] Peter Niebert, Stavros Tripakis, and Sergio Yovine. Minimum-Time Reachability for Timed Automata. In *Mediterranean Control Conference*. IEEE, 2000. (Cited on page 100.)
- [152] Harald Niederreiter. *Random Number Generation and Quasi-Monte Carlo Methods*. SIAM, 1992. (Cited on pages 74 and 101.)
- [153] Jeho Oh, Don Batory, Margaret Myers, and Norbert Siegmund. Finding Near-Optimal Configurations in Product Lines by Random Sampling. In *11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE '17)*, pages 61–71. ACM, 2017. ISBN 978-1-4503-5105-8. doi: 10.1145/3106237.3106273. (Cited on page 101.)
- [154] Sebastian Oster, Florian Markert, and Philipp Ritter. Automated Incremental Pairwise Testing of Software Product Lines. In *14th International Conference on Software Product Lines (SPLC '10)*, volume 6287 of *LNCS*, pages 196–210. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-15579-6. doi: 10.1007/978-3-642-15579-6_14. (Cited on pages 26, 28, and 101.)
- [155] Sebastian Oster, Andreas Wübbecke, Gregor Engels, and Andy Schürr. A Survey of Model-Based Software Product Lines Testing. *Model-Based Testing for Embedded Systems*, pages 339–381, 2012. (Cited on page 144.)
- [156] Sebastian Panek, Olaf Stursberg, and Sebastian Engell. Optimization of Timed Automata Models Using Mixed-Integer Programming. In *2nd International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS '04)*, volume 2791 of *LNCS*, pages 73–87. Springer Berlin Heidelberg, 2004. ISBN 978-3-540-40903-8. doi: 10.1007/978-3-540-40903-8_7. (Cited on page 100.)
- [157] David M. R. Park. Concurrency and automata on infinite sequences. In *Theoretical Computer Science (TCS)*, volume 104 of *LNCS*, pages 167–183.

- Springer Berlin Heidelberg, 1981. ISBN 978-3-540-38561-5. doi: 10.1007/BFb0017309. (Cited on page 42.)
- [158] Gilles Perrouin, Sagar Sen, Jacques Klein, Benoit Baudry, and Yves le Traon. Automated and Scalable T-wise Test Case Generation Strategies for Software Product Lines. In *3rd International Conference on Software Testing, Verification and Validation (ICST '10)*, pages 459–468. IEE, 2010. doi: 10.1109/ICST.2010.43. (Cited on pages 26, 67, and 100.)
- [159] Justyna Petke, Shin Yoo, Myra B. Cohen, and Mark Harman. Efficiency and Early Fault Detection with Lower and Higher Strength Combinatorial Interaction Testing. In *9th Joint Meeting of the European Software Engineering Conference and the Symposium on the Foundations of Software Engineering (ESEC/FSE '13)*, pages 26—36. ACM, 2013. doi: 10.1145/2491411.2491436. (Cited on page 97.)
- [160] Justyna Petke, Myra B. Cohen, Mark Harman, and Shin Yoo. Practical Combinatorial Interaction Testing: Empirical Findings on Efficiency and Early Fault Detection. *Transactions on Software Engineering (TSE)*, 41(9):901–924, 2015. (Cited on page 97.)
- [161] Paul Pettersson. *Modelling and Verification of Real-Time Systems Using Timed Automata: Theory and Practice*. PhD thesis, Uppsala University, 1999. (Cited on pages 37 and 164.)
- [162] David A. Plaisted and Steven Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, 2(3):293–304, 1986. ISSN 0747-7171. doi: 10.1016/S0747-7171(86)80028-1. (Cited on page 18.)
- [163] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer Science+Business Media, 2005. (Cited on pages 1 and 16.)
- [164] C. V. Ramamoorthy and Gary S. Ho. Performance Evaluation of Asynchronous Concurrent Systems Using Petri Nets. *IEEE Transactions on Software Engineering (TSE)*, SE-6(5):440–449, 1980. ISSN 0098-5589. doi: 10.1109/TSE.1980.230492. (Cited on page 64.)
- [165] Stuart C. Reid. An Empirical Analysis of Equivalence Partitioning, Boundary Value Analysis and Random Testing. In *4th International Software Metrics Symposium (METRICS '97)*, pages 64–73. IEEE, 1997. doi: 10.1109/METRIC.1997.637166. (Cited on pages 6 and 30.)
- [166] Dennis Reuling, Johannes Bürdek, Serge Rotärmel, Malte Lochau, and Udo Kelter. Fault-based Product-line Testing: Effective Sample Generation Based on Feature-diagram Mutation. In *19th International Software Product Line Conference (SPLC '15)*, pages 131–140. ACM, 2015. ISBN 978-1-4503-3613-0. doi: 10.1145/2791060.2791074. (Cited on page 101.)

- [167] Tomas G. Rokicki. *Modelling and Verification of Real-Time Systems Using Timed Automata: Theory and Practice*. PhD thesis, Stanford University, 1994. (Cited on pages 37 and 164.)
- [168] Sebastian Ruland, Lars Luthmann, Johannes Bürdek, Sascha Lity, Thomas Thüm, Malte Lochau, and Márcio Ribeiro. Measuring Effectiveness of Sample-Based Product-line Testing. In *17th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences (GPCE '18)*, pages 119–133. ACM, 2018. ISBN 978-1-4503-6045-6. doi: 10.1145/3278122.3278130. (Cited on pages 96, 97, and 101.)
- [169] Hamideh Sabouri, Mohammad M. Jaghoori, Frank de Boer, and Ramtin Khosravi. Scheduling and Analysis of Real-Time Software Families. In *36th Annual Computer Software and Applications Conference (COMPSAC '12)*, pages 680–689. IEEE, 2012. doi: 10.1109/COMPSAC.2012.95. (Cited on page 65.)
- [170] Atri Sarkar, Jianmei Guo, Norbert Siegmund, Sven Apel, and Krzysztof Czarnecki. Cost-Efficient Sampling for Performance Prediction of Configurable Systems. In *30th International Conference on Automated Software Engineering (ASE '15)*, pages 342–352. IEEE, 2015. doi: 10.1109/ASE.2015.45. (Cited on page 101.)
- [171] Ina Schaefer and Rainer Hähnle. Formal Methods in Software Product Line Engineering. *Computer*, 44(2):82–85, 2011. ISSN 0018-9162. doi: 10.1109/MC.2011.47. (Cited on page 25.)
- [172] Julien Schmaltz and Jan Tretmans. On Conformance Testing for Timed Systems. In *6th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS '08)*, volume 5215 of LNCS, pages 250–264. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-85778-5. doi: 10.1007/978-3-540-85778-5_18. (Cited on page 144.)
- [173] Pierre-Yves Schobbens, Patrick Heymans, Jean-Christophe Trigaux, and Yves Bontemps. Feature Diagrams: A Survey and a Formal Semantics. In *14th IEEE International Requirements Engineering Conference (RE '06)*, pages 139–148. IEEE, 2006. doi: 10.1109/RE.2006.23. (Cited on page 18.)
- [174] Alexander Schrijver. *Theory of Linear and Integer Programming*. Wiley, 1998. (Cited on pages 90 and 124.)
- [175] Bran Selic. Using UML for Modeling Complex Real-Time Systems. In *Workshop on Languages, Compilers, and Tools for Embedded Systems (LCTES '98)*, volume 1474 of LNCS, pages 250–260. Springer Berlin Heidelberg, 1998. ISBN 978-3-540-49673-1. doi: 10.1007/BFb0057795. (Cited on page 64.)
- [176] Liang Shi, Changhai Nie, and Baowen Xu. A Software Debugging Method Based on Pairwise Testing. In *5th International Conference on Computational Science (ICCS '05)*, volume 3516 of LNCS, pages 1088–1091. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-32118-7. doi: 10.1007/11428862_179. (Cited on pages 27 and 28.)

- [177] Norbert Siegmund, Marko Rosenmüller, Martin Kuhlemann, Christian Kästner, Sven Apel, and Gunter Saake. SPL Conqueror: Toward optimization of non-functional properties in software product lines. *Software Quality Journal*, 20(3–4):487–517, 2012. ISSN 1573-1367. doi: 10.1007/s11219-011-9152-9. (Cited on page 101.)
- [178] Jacopo Soldani, Lars Luthmann, Malte Lochau, and Antonio Brogi. Compositionally Testing Management Conformance in Multi-Component Applications. Submitted to *Acta Informatica*, 2020. (Cited on page 144.)
- [179] Jacopo Soldani, Lars Luthmann, Malte Lochau, and Antonio Brogi. Testing conformance in multi-component enterprise application management. In *8th European Conference on Service-Oriented and Cloud Computing (ESOCC '20)*, volume 12054 of LNCS, pages 3–18. Springer International Publishing, 2020. ISBN 978-3-030-44769-4. doi: 10.1007/978-3-030-44769-4_1. (Cited on page 144.)
- [180] Andreas Spillner and Tilo Linz. *Basiswissen Softwaretest*. dpunkt.verlag, 2012. (Cited on pages 108 and 109.)
- [181] Jan Springintveld, Frits Vaandrager, and Pedro R. D'Argenio. Testing timed automata. *Theoretical Computer Science*, 254(1):225–257, 2001. ISSN 0304-3975. doi: 10.1016/S0304-3975(99)00134-6. (Cited on page 143.)
- [182] Jun Sun, Yang Liu, Jin Song Dong, and Xian Zhang. Verifying Stateful Timed CSP Using Implicit Clocks and Zone Abstraction. In *11th International Conference on Formal Engineering Methods (ICFEM '09)*, volume 5885 of LNCS, pages 581–600. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-10373-5. doi: 10.1007/978-3-642-10373-5_30. (Cited on pages 99 and 143.)
- [183] Tadaaki Tanimoto, Suguru Sasaki, Akio Nakata, and Teruo Higashino. A Global Timed Bisimulation Preserving Abstraction for Parametric Time-Interval Automata. In *2nd International Conference on Automated Technology for Verification and Analysis (ATVA '04)*, volume 3299 of LNCS, pages 179–195. Springer Berlin Heidelberg, 2004. ISBN 978-3-540-30476-0. doi: 10.1007/978-3-540-30476-0_18. (Cited on page 198.)
- [184] Paul Temple, Mathieu Acher, Gilles Perrouin, Battista Biggio, Jean-Marc Jézéquel, and Fabio Roli. Towards Quality Assurance of Software Product Lines with Adversarial Configurations. In *23rd International Software Product Line Conference (SPLC '19)*, pages 277–288. ACM, 2019. ISBN 9781450371384. doi: 10.1145/3336294.3336309. (Cited on pages 74 and 101.)
- [185] Thomas Thüm, Sven Apel, Christian Kästner, Ina Schaefer, and Gunter Saake. A Classification and Survey of Analysis Strategies for Software Product Lines. *ACM Computing Survey (CSUR)*, 47(1):6:1–6:45, 2014. ISSN 0360-0300. doi: 10.1145/2580950. (Cited on pages 25, 112, and 113.)
- [186] Jan Tretmans. Test Generation with Inputs, Outputs and Repetitive Quiescence. Technical report, University of Twente, 1996. URL <http://doc.utwente.nl/65463>. (Cited on page 144.)

- [187] Jan Tretmans and Hendrik Brinksma. TorX: Automated model-based testing. In *First European Conference on Model-Driven Software Engineering*, pages 31–43, 2003. (Cited on page 145.)
- [188] Jan Tretmans and Pi erre van de Laar. Model-Based Testing with TorXakis. In *30th Central European Conference on Information and Intelligent Systems (CECIIS '19)*, pages 247–258. Faculty of Organization and Informatics, 2019. (Cited on page 145.)
- [189] Grigori S. Tseitin. *On the Complexity of Derivation in Propositional Calculus*, pages 466–483. Springer Berlin Heidelberg, 1983. ISBN 978-3-642-81955-1. doi: 10.1007/978-3-642-81955-1_28. (Cited on page 18.)
- [190] Mark Utting and Bruno Legeard. *Practical Model-Based Testing: A Tools Approach*. Morgan Kaufmann, 2006. (Cited on pages 2, 39, 67, 85, 101, 106, 108, and 120.)
- [191] Frits Vaandrager and Abhisek Midya. A Myhill-Nerode Theorem for Register Automata and Symbolic Trace Languages. In *17th International Colloquium on Theoretical Aspects of Computing (ICTAC '20)*, volume 12545 of LNCS, pages 43–63. Springer International Publishing, 2020. ISBN 978-3-030-64276-1. doi: 10.1007/978-3-030-64276-1_3. (Cited on page 74.)
- [192] Mahsa Varshosaz, Harsh Beohar, and Mohammad Reza Mousavi. Delta-Oriented FSM-Based Testing. In *17th International Conference on Formal Engineering Methods (ICFEM '15)*, volume 9407 of LNCS, pages 366–381. Springer International Publishing, 2015. ISBN 978-3-319-25423-4. doi: 10.1007/978-3-319-25423-4_24. (Cited on page 145.)
- [193] Mahsa Varshosaz, Mohammad Reza Mousavi, Lars Luthmann, and Malte Lochau. Expressive Power and Encoding of Transition System Models for Software Product Lines. In *29th Nordic Workshop on Programming Theory (NWPT '17)*, TUCS Lecture Notes, pages 57–59. Turku Center for Computer Science, 2017. URL <https://research.it.abo.fi/nwpt17/proceedings/NWPT2017proceedings.pdf>. (Cited on page 65.)
- [194] Mahsa Varshosaz, Mustafa Al-Hajjaji, Thomas Th um, Tobias Runge, Mohammad Reza Mousavi, and Ina Schaefer. A Classification of Product Sampling for Software Product Lines. In *Proceedings of the 22nd International Systems and Software Product Line Conference - Volume 1*, pages 1–13. ACM, 2018. ISBN 9781450364645. doi: 10.1145/3233027.3233035. (Cited on pages 26 and 100.)
- [195] Mahsa Varshosaz, Lars Luthmann, Paul Mohr, Malte Lochau, and Mohammad Reza Mousavi. Modal transition system encoding of featured transition systems. *Journal of Logical and Algebraic Methods in Programming (JLAMP)*, 106:1–28, 2019. ISSN 2352-2208. doi: 10.1016/j.jlamp.2019.03.003. (Cited on page 65.)
- [196] Birgit Vogel-Heuser, Christoph Legat, Jens Folmer, and Stefan Feldmann. Researching Evolution in Industrial Plant Automation: Scenarios and Docu-

- mentation of the Pick and Place Unit. Technical report, Institute of Automation and Information Systems, Technische Universität München, 2014. URL <https://mediatum.ub.tum.de/node?id=1208973>. (Cited on pages 9 and 96.)
- [197] Birgit Vogel-Heuser, Safa Bougouffa, and Michael Sollfrank. Researching Evolution in Industrial Plant Automation: Scenarios and Documentation of the extended Pick and Place Unit. Technical report, Institute of Automation and Information Systems, Technische Universität München, 2018. (Cited on pages 1 and 10.)
- [198] Md Tawhid Bin Waez, Juergen Dingel, and Karen Rudie. A survey of timed automata for the development of real-time systems. *Computer Science Review (CSR)*, 9:1–26, 2013. ISSN 1574-0137. doi: 10.1016/j.cosrev.2013.05.001. (Cited on pages 11 and 64.)
- [199] Jiacun Wang. *Timed Petri Nets*. Springer Science+Business Media New York, 2012. (Cited on page 64.)
- [200] Yingxu Wang. The Real-Time Process Algebra (RTPA). *Annals of Software Engineering*, 14(1):235–274, 2002. ISSN 1573-7489. doi: 10.1023/A:1020561826073. (Cited on page 64.)
- [201] Carsten Weise and Dirk Lenzkes. Efficient scaling-invariant checking of timed bisimulation. In *14th Annual Symposium on Theoretical Aspects of Computer Science (STACS '97)*, volume 1200 of *LNCS*, pages 177–188. Springer Berlin Heidelberg, 1997. ISBN 978-3-540-68342-1. doi: 10.1007/BFb0023458. (Cited on pages 7, 151, 152, 154, 198, 199, and 203.)
- [202] David M. Weiss. The Product Line Hall of Fame. In *12th International Software Product Line Conference (SPLC '08)*, page 395. IEEE, 2008. doi: 10.1109/SPLC.2008.56. (Cited on pages 1 and 16.)
- [203] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. The Worst-case Execution-time Problem—Overview of Methods and Survey of Tools. *ACM Transactions on Embedded Computing Systems (TECS)*, 7(3):36:1–36:53, 2008. ISSN 1539-9087. doi: 10.1145/1347375.1347389. (Cited on pages 6, 30, 40, 68, 83, 104, and 128.)
- [204] Wang Yi. Real-Time Behaviour of Asynchronous Agents. In *1st International Conference on Concurrency Theory (CONCUR '90)*, volume 458 of *LNCS*, pages 502–520. Springer Berlin Heidelberg, 1990. ISBN 978-3-540-46395-5. doi: 10.1007/BFb0039080. (Cited on pages 43 and 198.)