MASARYK UNIVERSITY
FACULTY OF INFORMATICS

Modal Transition Systems: Extensions and Analysis

PH.D. THESIS

Jan Křetínský

Brno, 2014

Declaration

Hereby I declare, that this thesis is my original authorial work, which I have worked out by my own. All sources, references and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Advisor: prof. RNDr. Antonín Kučera, Ph.D.

Acknowledgement

I would like to thank my advisor Antonín Kučera and my consultant Tomáš Brázdil for introducing me to research, their guidance and care, and teaching me "more is always more", be it knowledge, the deadline thrill, or the number of PhD theses. I thank to all co-authors of papers that gave rise to this thesis, in order of appearance in my scientific life: Jiří Srba, who was the decisive factor for coming to Aalborg and taught me to do reasearch with joy, rigour and no stress; Kim Guldstrand Larsen, who introduced me to MTS and taught me that research enthusiasm can prevail even if all hope is repeatedly lost; Nikola Beneš, with whom I shamelessly practiced co-induction whenever we felt too lazy to do the base step, discussed all topics of life, universe and everything, and made fun of the same, and simply had a good time; Javier Esparza, who taught me more than I can list; Ivana Černá, who offered me the first paid work for the university; Mikael Harkjær Møller, who showed me you can get a good beer even in Denmark; Axel Legay, who never gives up; Uli Fahrenberg, who can appreciate the beuaty of math even in its technicalities; Benoît Delahaye, who fights for independence; Salomon Sickert, who always implements and analyzes everything already right before being asked; and all other colleagues all over Europe, particularly to Jan Krčál for feedback on the thesis, all the lot we experienced together, and keeping my clear conscience when it comes to travel allowance. Last, but not least, I want to thank to my parents and Zuzanka for their love, support, and not complaining how many theses I have to write.

Abstract

In this thesis, we deal with the specification formalism of modal transition systems (MTS). We introduce several extensions thereof and discuss their relationships and modelling capabilities. Apart from extending the modalities of MTS, we consider timed and priced extensions as well as systems with infinite state space. We also compare the behavioural framework of MTS and its extensions to logical frameworks.

The main focus of the thesis lies on algorithms for verification and analysis of the introduced MTS extensions. We cover refinement checking, model checking, standard logical and structural operations as used in specification theories as well as synthesis of implementations satisfying given logical formulae or optimality with respect to a given price scheme. Beside theoretical aspects such as establishing complexity of investigated problems, we also deal with practical issues such as heuristics and tool support.

Shrnutí

Tato dizertační práce studuje specifikační formalizmus zvaný modální přechodové systémy (MTS). Zavádíme několik rozšíření MTS a rozebíráme jejich vzájemné vztahy a modelovací sílu. Zaprvé rozšiřujeme modality MTS, zadruhé uvažujeme rozšíření s prvky času a ceny a zatřetí zavádíme různé třídy MTS s nekonečně mnoha stavy. Rovněž tato rozšíření MTS srovnáváme jako formalizmy automatové s formalizmy logickými.

Těžiště práce se nachází v oblasti algoritmické analýzy a verifikace uvedených systémů. Zajímá nás především kontrola zjemňování, ověřování modelu, obvyklé logické a strukturální operace používané v kontextu specifikačních teorií, automatická tvorba implementací splňujících danou logickou formuli či optimalitu vzhledem k danému cenovému schématu. Věnujeme se jak teroretickým otázkám, zejména složitosti zkoumaných problémů, tak i praktickým aspektům, zejména heuristikám a softwarovému nástroji pro podporu práce s MTS.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Year by year, software systems tend to be larger, more complex and reusing more code. Under these conditions, it is virtually impossible to create error-free systems despite extensive testing. In contrast, *formal methods* can be used not only for bug finding but also for verifying the absence of errors and even for development of correct-by-design software. The major advantage of the formal frameworks are guarantees given with mathematical certainty.

The key idea of formal methods and, in particular, formal verification is to first *specify* a property the system should satisfy and then to *verify* that this is indeed the case. The specification language should be equipped with mathematically precise meaning so that the verification can give reliable results and be done automatically. An alternative to verification is *refinement* of the original specification into an implementation, which is guaranteed to satisfy the specification, for the refinement is designed to preserve the properties of interest. The refinement can be done in one step, where the implementation is *synthesized* from the specification, or in more steps in a process of *stepwise refinement*. The latter is particularly useful when some details of the requirements are not known at the beginning of the design process, or synthesis of the whole system is unfeasible, or in the component-based design where other systems can be reused as parts of the new system.

The difference between verifying and refining systems is reflected in two fundamentally different approaches to specifying properties of systems. Firstly, the *logical* approach makes use of specifications given as formulae of temporal or modal logics. Secondly, the *behavioural* approach requires specifications to be given in the same formalism as implementations. The former relies on efficient model checking algorithms in order to verify systems. The latter exploits various equivalence and refinement checking methods. In this thesis, we combine the two approaches.

Example 1.1 *Consider the scenario of developing a piece of software illustrated in Fig. 1.1. We start with a viewpoint V_1 on the system, e.g. the client's view on the service functionality. This gets iteratively refined into a more concrete description V_m. Further, assume there is also another viewpoint W_1, e.g. a description of the service from the server point of view, which is refined in a similar fashion resulting in W_n. After these viewpoints are precise enough (although still very under-*

specified), we merge them into one, say S*, using an operation of* conjunction*. The complete description is now modelled by* S*, which is to be implemented. Suppose we have components* C *and* D *at our disposal, which perform subroutines needed in* S*. We put* C *and* D *together into a component* T *using an operation of* parallel composition*. What remains to be designed is a component* X *that we can compose with* T *in parallel so that the result conforms to the specification* S*. The most general such* X *is called the* quotient *of* S *by* T*. Once we have* X *we can further refine the underspecified behaviour in any desired way resulting in a specification* Y*. The final step is to automatically* synthesize *an implementation* Z *that, for instance, satisfies additional temporal* logic constraints φ *and/or is the* cheapest *implementation with respect to specified costs* $\mathfrak{C}$*.*

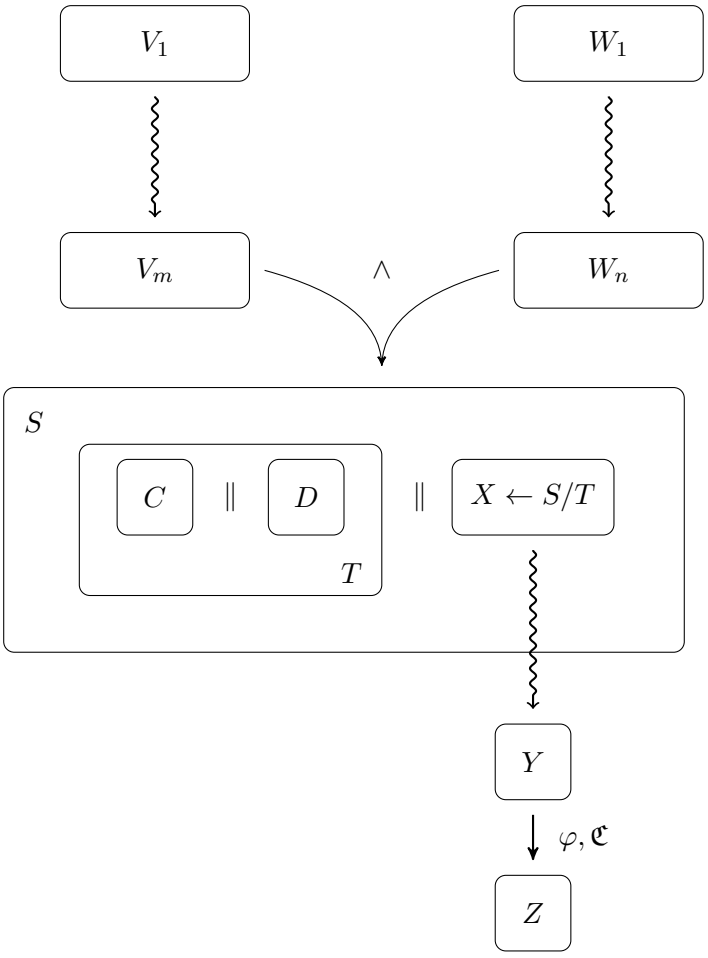


Figure 1.1: An example of a component-based step-wise design scheme

A good specification theory should allow for all the operations mentioned in the example and efficient algorithms to compute them. Moreover, it should be expressive enough to allow for convenient modelling. The behavioural formalism of *modal transition systems* (MTS) [LT88] provides a convenient basis for such a theory and has already attracted a lot of attention. Unfortunately, it does not satisfy either of the stipulations completely. In this thesis, we design *extensions* of MTS that meet all these demands and provide efficient algorithms for their *analysis* such as the mentioned operations, refinements, verification and synthesis. Moreover, we provide a link between the MTS extensions and logics, thus building a bridge between the behavioural and the logical world, allowing us to combine them, enjoying the best of both worlds.

1.1 History of modal transition systems

Modal transition systems (MTS) were introduced by Larsen and Thomsen [LT88] a quarter of a century ago. The goal was to obtain an operational, yet expressive specification formalism meeting the demands discussed above. Their main advantage is that they are a simple extension of labelled transition systems, which have proved appropriate for behavioural description of systems as well as their compositions.

MTS consist of a set of states and two transition relations. The *must* transitions prescribe what behaviour has to be present in every refinement of the system; the *may* transitions describe the behaviour that is allowed, but need not be realized in the refinements. This allows us to underspecify non-critical behaviour in the early stages of design, focusing on the main properties, verifying them and sorting out the details of the yet unimplemented non-critical behaviour later.

Example 1.2 *An MTS specification of a coffee machine is diplayed in Fig. 1.2 on the left. May transitions are depicted using dashed arrows, must transitions using solid arrows. In the left state, the machine can either start to* clean *or accept a* coin*. It may not always be possible to take the* coin *action, but if we do so the machine must offer* coffee *and possibly supplement the choice with* tea*. An implementation of this specification is displayed on the right. Here the* clean *is scheduled regularly after every two beverages. In addition,* tea *can always be chosen instead of* coffee*.*

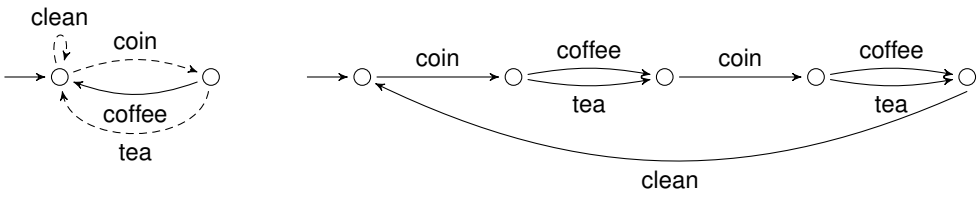


Figure 1.2: An MTS specification and its implementation

The formalism of MTS has many applications, most importantly in compositional reasoning and design as recognized in the ARTIST Network of Excellence [ART] and several other related European projects. It has proven to be useful in practice. Industrial applications are as old as [Bru97] where MTS have found use for an air-traffic system at Heathrow airport. Besides, MTS are advocated as an appropriate base for interface theories in [AHL+08a, RBB+09b, RBB+09a, RBB+11] and for product line theories in [LNW07a, Nym08]. Further, MTS-based software engineering methodology for design via merging partial descriptions of behaviour has been established in [UC04, BCU06, UBC07] and using residuation in [Rac07, Rac08, Ben08]. The MTS specification formalism is supported by several tools, e.g. [BLS95, DFFU07, BML11]. Furthermore, MTS are used for program analysis using abstraction [GHJ01, DGG97, Nam03, DN04, dAGJ04, NNN08, CGLT09, GNRT10].

Over the years, many extensions of MTS have been proposed. While MTS can only specify whether or not a particular transition is required, some extensions equip MTS with more general abilities to describe what *combinations* of transitions are possible. Disjunctive MTS (DMTS) [LX90] can specify that at least one of a given set of transitions is present. One-selecting MTS [FS08] specify that exactly one of them is present. Acceptance automata [Rac07] can even express any Boolean combination of transitions, but only for deterministic systems. In all the mentioned cases, every must transition is also automatically a may transition, modelling that whatever is required is also allowed. Sometimes this assumption is dropped and the two transition relations are given independently, giving rise to mixed transition systems (MixTS) [DGG97].

These formalisms have also been studied under other names in different contexts. To some extent equivalent variations of MTS have been adapted for model-checking: Kripke Modal Transition Systems (KMTS) [HJS01], Partial Kripke Structures [BG00], and 3-valued Kripke structures [CDEG03]. In the same manner MixTS correspond to Belnap transition systems [GWC06a]. Further, DMTS correspond to Generalized KMTS [SG04] or Abstract transition systems [dAGJ04]. While the variants of MTS and MixTS have been used in practical symbolic model-checkers (e.g. [CDEG03, GC06, GWC06b]), the "hypermust" transitions in DMTS are hard to encode efficiently into BDDs. A comparison of usability of these systems for symbolic model checking can be found in [WGC09]. Acceptance automata were also studied as acceptance trees [Hen85].

The relationship of MTS to logic was studied in [BL92, FP07]. It is established that MTS are equivalent to a fragment of Hennessy-Milner logic (HML) [HM80] where the formulae are "consistent and prime". This raises further questions on the relationship of the form "What corresponds to the notion X of one world in the other world?" For instance, we would like implementations of an MTS to be exactly models of the corresponding formula. Further, one would like to capture refinement by implication similarly to the refinement calculus for HML with re-

cursion of [Hol89]. Moreover, we would like both formalisms to be closed under the desired operations and describe the operations in one world using some notions of the other world, see Fig. 1.3. Finally, it would be elegant if both formalisms were defined naturally and not only as subclasses of other formalisms defined ad hoc in an artificial manner.

logic		MTS
model	$\sim$	implementation
entailment $\models$	$\sim$	refinement $\leq$
conjunction $\wedge$	$\sim$	?
disjunction $\vee$	$\sim$	?
?	$\sim$	parallel composition $\|$
?	$\sim$	quotient $/$

Figure 1.3: Correspondences between the logical and the behavioural world

1.2 Contribution of the thesis

Here we only give a very brief and high-level account on the contribution of the thesis. For a technical summary, we refer the reader to Chapter 6.

We introduce several extensions of MTS. Firstly, we extend the mechanism to specify which transitions are present and which not (Paper A and C) and examine their expressive power (Paper A and F). We identify a robust class of disjunctive MTS with more initial states. We provide a translation between this class (and several equally expressive classes) and the Hennessy-Milner logic with greatest fixed points (also denoted νHML or the ν-calculus). This enables us to use mixtures of behavioural and logical specifications (Paper G).

Secondly, we extend the underlying graph structures of MTS. We consider systems with time durations of actions and pricing of actions, where we combine one-shot investment price for the hardware and cost for running it per each time unit it is active. This is useful for modelling embedded systems, where safety comes along with economical requirements (Paper D). Further, we consider infinite state systems generated by finite rules to model infinite memory or communication between dynamically created threads (Paper E).

Furthermore, on the computational side, we provide algorithms for refinements (Paper B, C, E, and F), operations (Paper B and G), model checking (Paper B and I), and computing the cheapest implementation (Paper D). Finally, we also provide a tool support for most of the discussed functionality (Paper H).

1.3 Publication summary

In Appendix, we present the following papers:

A Nikola Beneš and Jan Křetínský. Process algebra for modal transition systemses. *MEMICS*, 2010. [BK10]

B Nikola Beneš, Ivana Černá, and Jan Křetínský. Modal transition systems: Composition and LTL model checking. *ATVA*, 2011. [BČK11]

C Nikola Beneš, Jan Křetínský, Kim G. Larsen, Mikael H. Moller, and Jiri Srba. Parametric modal transition systems. *ATVA*, 2011. [BKL+11]

D Nikola Beneš, Jan Křetínský, Kim Guldstrand Larsen, Mikael H. Moller, and Jiri Srba. Dual-priced modal transition systems with time durations. *LPAR*, 2012. [BKL+12]

E Nikola Beneš and Jan Křetínský. Modal process rewrite systems. *ICTAC*, 2012. [BK12]

F Jan Křetínský and Salomon Sickert. On refinements of Boolean and parametric modal transition systems. *ICTAC*, 2013. [KS13b]

G Nikola Beneš, Benoît Delahaye, Uli Fahrenberg, Jan Křetínský, and Axel Legay. Hennessy-Milner logic with greatest fixed points as a complete behavioural specification theory. *CONCUR*, 2013. [BDF+13]

H Jan Křetínský and Salomon Sickert. MoTraS: A tool for modal transition systems and their extensions. *ATVA*, 2013. [KS13a] (tool paper)

I Javier Esparza and Jan Křetínský. From LTL to Deterministic Automata: A Safraless Compositional Approach. *CAV*, 2014. [EK14]

1.3.1 Other co-authored papers

For the sake of completeness, apart from the presented papers, we also list other co-authored papers, which are, however, not a part of this thesis.

More papers on modal transition systems

- Nikola Beneš, Jan Křetínský, Kim G. Larsen, and Jiri Srba. Checking Thorough Refinement on Modal Transition Systems Is EXPTIME-Complete. *ICTAC*, 2009. [BKLS09a]
- Nikola Beneš, Jan Křetínský, Kim Guldstrand Larsen, and Jiri Srba. On determinism in modal transition systems. *Theoretical Computer Science*, 2009. [BKLS09b]
- Nikola Beneš, Jan Křetínský, Kim G. Larsen, and Jiri Srba. EXPTIME-completeness of thorough refinement on modal transition systems. *Information and Computation*, 2012. [BKLS12] Extended journal version of [BKLS09a]

- Holger Hermanns, Jan Krčál, and Jan Křetínský. Compositional verification and optimisation of interactive Markov chains. *CONCUR*, 2013. [HKK13]

Papers on temporal logics

- Tomáš Brázdil, Vojtěch Forejt, Jan Křetínský, and Antonín Kučera. The satisfiability problem for probabilistic CTL. *LICS*, 2008. [BFKK08]
- Jan Křetínský and Javier Esparza. Deterministic automata for the (F,G)-fragment of LTL. *CAV*, 2012. [KE12]
- Andreas Gaiser, Jan Křetínský, and Javier Esparza. Rabinizer: Small deterministic automata for LTL($\mathbf{F}$, $\mathbf{G}$). *ATVA*, 2012. [GKE12] (tool paper)
- Krishnendu Chatterjee, Andreas Gaiser, and Jan Křetínský. Automata with generalized Rabin pairs for probabilistic model checking and LTL synthesis. *CAV*, 2013. [CGK13]
- Jan Křetínský and Ruslán Ledesma Garza. Rabinizer 2: Small deterministic automata for LTL$_{\setminus \mathbf{GU}}$. *ATVA*, 2013. [KLG13] (tool paper)

Papers on stochastic continuous-time systems

- Tomáš Brázdil, Vojtěch Forejt, Jan Krčál, Jan Křetínský, and Antonín Kučera. Continuous-time stochastic games with time-bounded reachability. *FSTTCS*, 2009. [BFK+09].
- Tomáš Brázdil, Jan Krčál, Jan Křetínský, Antonín Kučera, and Vojtěch Řehák. Stochastic real-time games with qualitative timed automata objectives. *CONCUR*, 2010. [BKK+10]
- Tomáš Brázdil, Jan Krčál, Jan Křetínský, Antonín Kučera, and Vojtěch Řehák. Measuring performance of continuous-time stochastic processes using timed automata. *HSCC*, 2011. [BKK+11]
- Tomáš Brázdil, Jan Krčál, Jan Křetínský, and Vojtěch Řehák. Fixed-delay events in generalized semi-Markov processes revisited. *CONCUR*, 2011. [BKKŘ11]
- Tomáš Brázdil, Holger Hermanns, Jan Krčál, Jan Křetínský, and Vojtěch Řehák. Verification of open interactive Markov chains. *FSTTCS*, 2012. [BHK+12]
- Tomáš Brázdil, Luboš Korenčiak, Jan Krčál, Jan Křetínský, and Vojtěch Řehák. On time-average limits in deterministic and stochastic Petri nets. *ICPE*, 2013. [BKK+13] (poster paper)

- Tomáš Brázdil, Vojtěch Forejt, Jan Krčál, Jan Křetínský, and Antonín Kučera. Continuous-time stochastic games with time-bounded reachability. *Information and Computation*, 2013. [BFK+13] Extended journal version of [BFK+09]

1.3.2 Summary

The author has published 3 journal papers (2×Information and Computation, Theoretical Computer Science), 17 conference regular papers (2×ATVA, 3×CAV, 4×CONCUR, 2×FSTTCS, HSCC, 3×ICTAC, LICS, LPAR), 3 conference tool papers (3×ATVA), 1 workshop paper (MEMICS) and 1 conference poster paper (ICPE), altogether 25 papers. The current number of citations according to Google Scholar is 238 including self-citations with the h-index 9, and 140 without self-citations with the h-index 6.

1.4 Outline of the thesis

Chapter 2 introduces modal transition systems formally, recalls several logics used later in the thesis, and explains the stipulations on good specification formalisms. Chapter 3 discusses previous and new extensions of MTS with respect to specifying the combinations of present transitions. Chapter 4 discusses previous and new extensions of MTS with respect to the underlying graph structure of the MTS. In Chapter 5, results on refinements, operations, implementation synthesis, and the tool are presented and related to previously known results. Chapter 6 summarizes the contribution of the thesis and new results of the respective papers and gives an account of ongoing and possible future work. In Appendix, we present preprints of Papers A–I [BK10, BČK11, BKL+11, BKL+12, BK12, KS13b, BDF+13, KS13a, EK14] without appendices. We also include permissions to publish the preprints of the papers within this thesis as they were sent to the publisher.

The reader is assumed to have some familiarity with labelled transition systems, logic, and complexity to the extent of standard basic courses.

Chapter 2

Preliminaries

Properties of systems can be specified and verified using two fundamentally different approaches. Firstly, the *behavioural* approach exploits various equivalence or preorder relations and methods to check them, provided the specifications are given in the same formalism as implementations. Secondly, the *logical* approach uses formulae of temporal or modal logics as specifications and relies on model checking algorithms. In this chapter, we introduce a behavioural framework of modal transition systems as well as several logics. Later in the thesis, we relate and combine them.

2.1 Modal transition systems

The original modal transition systems were introduced in [LT88] as follows:

Definition 2.1 (Modal transition system) *A* modal transition system (MTS) *over an action alphabet Σ is a triple $(P, \dashrightarrow, \longrightarrow)$, where P is a set of* processes *and $\longrightarrow \subseteq \dashrightarrow \subseteq P \times \Sigma \times P$ are* must *and* may *transition relations, respectively.*

Observe that P is not required to be finite. We often use letters $s, t, \ldots$ to denote processes of MTS. Whenever it is clear from the context, we refer to processes without explicitly mentioning their underlying MTS.

The most fundamental notion of the theory of modal transition systems is the modal refinement. Intuitively, a process s refines a process t if the specification s concretizes the one of t (or in other words, t is more abstract than s). Technically, this is ensured by (1) only allowing in s what is already allowed in t and (2) requiring in s what is already required in t.

Definition 2.2 (Modal refinement) *Let $(P_1, \dashrightarrow_1, \longrightarrow_1)$, $(P_2, \dashrightarrow_2, \longrightarrow_2)$ be MTS over the same action alphabet and $s \in P_1$, $t \in P_2$ be processes. We say that s* modally refines *t, written $s \leq_m t$, if there is a* refinement relation *$R \subseteq P_1 \times P_2$ satisfying $(s,t) \in R$ and for every $(p,q) \in R$ and every $a \in \Sigma$:*

1. *if $p \overset{a}{\dashrightarrow}_1 p'$ then there is a transition $q \overset{a}{\dashrightarrow}_2 q'$ s.t. $(p', q') \in R$, and*
2. *if $q \overset{a}{\longrightarrow}_2 q'$ then there is a transition $p \overset{a}{\longrightarrow}_1 p'$ s.t. $(p', q') \in R$.*

Example 2.3 *In the course of the refinement process, must transitions are preserved, may transitions can turn into must transitions or disappear, and no new transitions are added. Note that refinement is a more complex notion than that of subgraph. Indeed, the same transition can be refined in different ways in different places as illustrated in Fig. 2.1. Note that whenever there is a must transition in an MTS, we do not depict its underlying may transitions. Here $i \leq_m s$ is witnessed by the relation $\{(i,s),(j_1,t),(j_2,t),(k_1,s),(k_2,s),(\ell,t)\}$.*

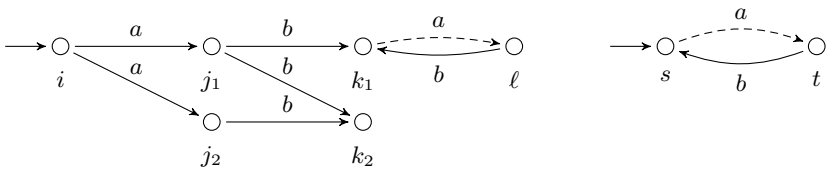


Figure 2.1: $i \leq_m s$

Whenever $s \leq_m t$, we call s a refinement of t and t an abstraction of s. We often consider MTS with a designed initial state; in such a case we say that an MTS refines another one if this is true of their initial states.

One may refine MTS in a stepwise manner until $\dashrightarrow = \longrightarrow$ is obtained and no further refinement is possible. MTS with $\dashrightarrow = \longrightarrow$ are called *implementations* and can be considered as the standard labelled transition systems (LTS). Given a process s we denote by $[\![s]\!] = \{i \mid i \text{ is an implementation and } i \leq_m s\}$ the set of all implementations of s.(*) In the previous example, j_1 is not an implementation, while j_2 is considered an implementation since all reachable transitions satisfy the requirement. Further notice that $k_2 \in [\![s]\!]$.

Note that on implementations the refinement coincides with the strong bisimilarity, and on modal transition systems without any must transitions it corresponds to the simulation preorder. Further, the refinement has a respective game characterization [BKLS09b] similar to (bi)simulation games, which is often useful in proofs.

2.2 Logics

In this thesis we use three logics defined below.

Propositional logic First, we recall the standard *propositional logic*. A Boolean formula over a set X of atomic propositions is given by the following syntax

$$\varphi ::= \mathbf{tt} \mid \mathbf{ff} \mid x \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \neg\varphi$$

where x ranges over X. The set of all Boolean formulae over X is denoted by

(∗). We introduced this notation in [BKLS09b], but since then it has become the standard.

$\mathcal{B}(X)$. For a truth assignment $\nu \subseteq X$, the satisfaction relation is given as follows:

$$\begin{array}{lcl}
\nu \models \mathbf{tt} & & \\
\nu \not\models \mathbf{ff} & & \\
\nu \models p & \iff & p \in \nu \\
\nu \models \varphi \wedge \psi & \iff & \nu \models \varphi \text{ and } \nu \models \psi \\
\nu \models \varphi \vee \psi & \iff & \nu \models \varphi \text{ or } \nu \models \psi \\
\nu \models \neg\varphi & \iff & \nu \not\models \varphi
\end{array}$$

We also use the standard derived operators like exclusive-or $\varphi \oplus \psi = (\varphi \wedge \neg\psi) \vee (\neg\varphi \wedge \psi)$, implication $\varphi \Rightarrow \psi = \neg\varphi \vee \psi$ and equivalence $\varphi \Leftrightarrow \psi = (\neg\varphi \vee \psi) \wedge (\varphi \vee \neg\psi)$.

A set of implementations can be specified not only by a behavioural specification such as an MTS, but also by a formula of a logic. In this thesis, we focus on two logics and their fragments: μ-calculus [Koz83] and LTL [Pnu77].

μ-calculus Let Ap be a set of atomic propositions, Var a set of variables, and Σ an action alphabet. The branching time logic *μ-calculus* is given by the following syntax:

$$\varphi ::= \mathbf{tt} \mid \mathbf{ff} \mid p \mid \neg p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid [a]\varphi \mid \langle a \rangle \varphi \mid \mu X.\varphi \mid \nu X.\varphi$$

where p ranges over Ap, X over Var, and a over Σ. We call μ the least fixpoint and ν the greatest fixpoint. Given an implementation $(P, \longrightarrow)$ and a valuation $\nu : P \to 2^{Ap}$ over its state space, the semantics is defined as follows:

$$\begin{array}{lcl}
[\![\mathbf{tt}]\!]_\nu & = & P \\
[\![\mathbf{ff}]\!]_\nu & = & \emptyset \\
[\![p]\!]_\nu & = & \{s \in P \mid p \in \nu(s)\} \\
[\![\neg p]\!]_\nu & = & \{s \in P \mid p \notin \nu(s)\} \\
[\![\varphi \wedge \psi]\!]_\nu & = & [\![\varphi]\!]_\nu \cap [\![\psi]\!]_\nu \\
[\![\varphi \vee \psi]\!]_\nu & = & [\![\varphi]\!]_\nu \cup [\![\psi]\!]_\nu \\
[\![[a]\varphi]\!]_\nu & = & \{s \in P \mid \forall s \xrightarrow{a} t : t \in [\![\varphi]\!]_\nu\} \\
[\![\langle a \rangle \varphi]\!]_\nu & = & \{s \in P \mid \exists s \xrightarrow{a} t : t \in [\![\varphi]\!]_\nu\} \\
[\![\mu X.\varphi]\!]_\nu & = & \bigcap\{Q \subseteq P \mid [\![\varphi]\!]_{\nu[X \mapsto Q]} \subseteq T\} \\
[\![\nu X.\varphi]\!]_\nu & = & \bigcup\{Q \subseteq P \mid [\![\varphi]\!]_{\nu[X \mapsto Q]} \supseteq T\}
\end{array}$$

Linear temporal logic Finally, the *linear temporal logic (LTL)* is given by the following syntax:

$$\varphi ::= \mathbf{tt} \mid \mathbf{ff} \mid p \mid \neg p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathbf{X}\varphi \mid \mathbf{X}_a\varphi \mid \mathbf{F}\varphi \mid \mathbf{G}\varphi \mid \varphi \mathbf{U} \varphi$$

where p ranges over Ap and a over Σ. For a word $w = \nu_0 a_0 \nu_1 a_1 \cdots \in (2^{Ap} \times \Sigma)^{\mathbb{N}}$, w_i denotes the suffix $\nu_i a_i \nu_{i+1} a_{i+1} \cdots$. The semantics of a formula on w is defined

inductively as follows:(†)

$$
\begin{array}{lcl}
w \models \mathbf{tt} & & \\
w \not\models \mathbf{ff} & & \\
w \models p & \iff & p \in \nu_0 \\
w \models \neg p & \iff & p \notin \nu_0 \\
w \models \varphi \wedge \psi & \iff & w \models \varphi \text{ and } w \models \psi \\
w \models \varphi \vee \psi & \iff & w \models \varphi \text{ or } w \models \psi \\
w \models \mathbf{X}\varphi & \iff & w_1 \models \varphi \\
w \models \mathbf{X}_a\varphi & \iff & a_0 = a \text{ and } w_1 \models \varphi \\
w \models \mathbf{F}\varphi & \iff & \exists\, k \in \mathbb{N} : w_k \models \varphi \\
w \models \mathbf{G}\varphi & \iff & \forall\, k \in \mathbb{N} : w_k \models \varphi \\
w \models \varphi \mathbf{U} \psi & \iff & \exists\, k \in \mathbb{N} : w_k \models \psi \text{ and} \\
 & & \forall\, 0 \leq j < k : w_j \models \varphi
\end{array}
$$

Notice that the sematics is a mixture of state-based and action-based properties. In the context of MTS, [Ben12] elaborates on the differences of the two.

Given an implementation $(P, \longrightarrow)$ and a valuation $\nu : P \to 2^{Ap}$ over its state space, any run (maximal path in the directed graph of the LTS) induces a word over 2^{Ap}. An LTS then satisfies a formula if all runs from its initial state satisfy the formula.

Example 2.4 *Consider the LTS and its state valuation depicted in Fig. 2.2. While it satisfies* $\mathbf{G}p$ *and* $\nu X.p \wedge [a]X$*, it does not satisfy* $\mathbf{F}q$ *and* $\mu X.q \vee [a]X$ *due to the run looping in* s*.*

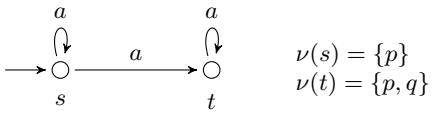


Figure 2.2: An LTS with a state valuation

2.3 Specification theories

In order to support component based development, many specification theories have been designed. One usually requires existence and effective computability of several operations subject to various axioms. In the following, let s and t be processes, arguments of the operations.

Some operations are structural stemming from the nature of behavioural descriptions, such as the operations of parallel composition and quotient. The *parallel composition* $\|$ should satisfy

(†). For a straightforward extension to possibly finite words, see Paper B.

(parallel) for any processes x and y, $x \parallel y \leq_m s \parallel t$ iff $x \leq_m s$ and $y \leq_m t$.

The *quotient* is an *adjoint* to parallel composition, hence the quotient s/t of s by t must satisfy

(quotient) for any process x, $x \leq_m s/t$ if and only if $t \parallel x \leq_m s$.

Given a specification s of the whole system and t of its component, the quotient s/t is thus a compact description of all systems that can be put in parallel with t to get a system complying with s.

Other operations are inherited from the logical view, such as Boolean operations. A *conjunction* of two systems is the most general refinement of the two systems. As the greatest lower bound with respect to $\leq_m$ it must satisfy

(conjunction) for any process x, $x \leq_m s \wedge t$ if and only if $x \leq_m s$ and $x \leq_m t$.

A bit weaker notion is that of *consistency* relation: systems are consistent if they have a common implementation, i.e. if the conjunction has a non-empty set of implementations. Dually, one can define *disjunction* by requiring

(disjunction) for any process x, $s \vee t \leq_m x$ if and only if $s \leq_m x$ and $t \leq_m x$.

The remaining Boolean operation is that of *complement*:

(complement) for any process x, $x \leq_m \bar{s}$ if and only if $x \not\leq_m s$.

For the related notion of difference, see e.g. [SCU11].

It is usually not possible to satisfy all axioms in this strong form. For instance, the "only-if" part of **(parallel)** is difficult to achieve in MTS [HL89, BKLS09b], see Fig. 2.3.

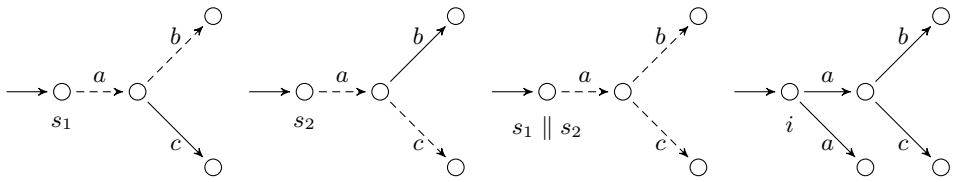


Figure 2.3: $i \leq_m s_1 \parallel s_2$, but i cannot be written as $i_1 \parallel i_2$ for any $i_1 \leq_m s_1, i_2 \leq_m s_2$

The "complete specification theories" of [BDH+12] only require the "if" form of **(parallel)**, called *independent implementability*. Further, existence of quotients and conjunctions is required if they have non-empty set of implementations. Here we presented a simpler version, which is equivalent for MTS enriched by a specification with no implementations.

Chapter 3

Extensions of modalities

Since the modelling capabilities of basic MTS are quite limited, many extensions have appeared in the literature. In this chapter, we focus on extensions of may and must transition relations.

3.1 State of the art

Standard MTS have two transition relations $\longrightarrow, \dashrightarrow \subseteq P \times \Sigma \times P$ satisfying, moreover, $\longrightarrow \subseteq \dashrightarrow$, which is called the syntactic consistency requirement. If this requirement is not imposed we obtain *mixed transition systems* as introduced in [DGG97].

Definition 3.1 (Mixed transition system) *A* mixed transition system (MixTS) *over an action alphabet Σ is a triple $(P, \dashrightarrow, \longrightarrow)$, where P is a set of* processes *and $\longrightarrow, \dashrightarrow \subseteq P \times \Sigma \times P$ are* must *and* may *transition relations, respectively.*

This extension allows us not only to have inconsistent specifications, but also an enforced non-deterministic choice as the following example shows.

Example 3.2 *Since must transitions are not necessarily also may transitions in MixTS, we depict may transitions explicitly for mixed systems even if there is a corresponding must transition. The specification of Fig. 3.1 requires an a transition followed by either only b's or only c's. Indeed, the must transition under a enforces a transition, but does not automatically allow it; only the two may transitions under a are allowed.*

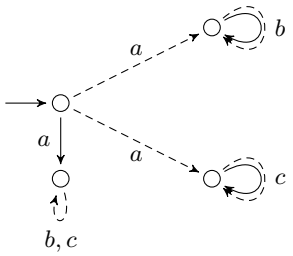


Figure 3.1: A mixed transition system

Nevertheless, even this feature is often insufficient to specify which *combinations* of transitions can be implemented.

Example 3.3 *Figure 3.2 depicts an MTS that specifies the following. A* request *from a client may arrive. Then we can* process *it directly on the server or make a* query *to a database where we are guaranteed an* answer*. In both cases we send a* response*.*

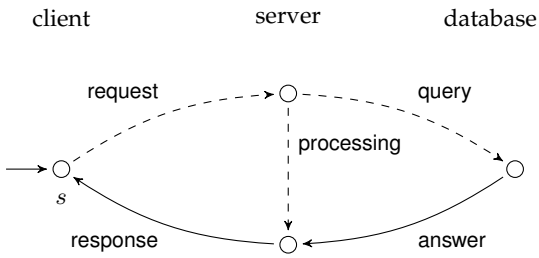


Figure 3.2: An example of a potentially deadlocking MTS

An MTS can be refined in two ways: a may transition is either implemented (and becomes a must transition) or omitted (and disappears as a transition). On the left of Fig. 3.3 there is an implementation of the system, where the processing *branch is implemented and the database* query *branch is omitted. Similarly, there is also an implementation omitting the* process *branch and implementing only the* query*. However, there is also an undesirable implementation that does not implement any option and deadlocks as seen on the right of Fig. 3.3.*

Figure 3.3: Two implementations i_1, i_2 of s of Fig. 3.2

To avoid deadlocking, we want to specify that *either* processing *or* query will be implemented. This is possible in *disjunctive modal transition systems* [LX90]. They were actually introduced as natural means for solutions to process equations since they can express both conjunctions and disjunctions of properties.

Definition 3.4 (Disjunctive modal transition system) *A* disjunctive modal transition system *(DMTS) over an action alphabet Σ is a triple $(P, \dashrightarrow, \longrightarrow)$, where P is a set of* processes *and $\dashrightarrow \subseteq P \times \Sigma \times P$ is the* may *and $\longrightarrow \subseteq P \times 2^{\Sigma \times P}$ the* must *(or* hyper-must*) transition relation.*

Example 3.5 *Now we can easily enforce a choice between arbitrary transitions, not just with the same action as in Example 3.2. Instead of forcing a particular transition, a must transition in DMTS specifies a whole set of transitions at least one of which must be present. In our example, it is the set consisting of* processing *and* query *transitions.*

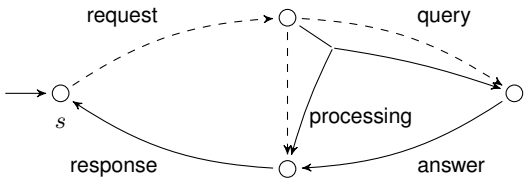


Figure 3.4: A disjunctive modal transition system

Note that DMTS are capable of forcing any positive Boolean combination of transitions, simply by turning it into the conjunctive normal form.

If the choice is supposed to be exclusive, we can use *one-selecting MTS (1MTS)* introduced in [FS08] with the property that *exactly* one transition from the set must be present. In 1MTS and also in *underspecified transition systems (UTS)* [FS05], both (hyper)must and (hyper)may transition relations are subsets of $P \times 2^{\Sigma \times P}$. For UTS, the syntactic consistency is required, i.e. the hyper-may is larger than the hyper-must.

Finally, explicit listing of all allowed combinations of outgoing transitions is used in *acceptance automata* [Rac08]. However, it is limited to deterministic systems.

Definition 3.6 (Acceptance automaton) *An* acceptance automaton *(AA) over an action alphabet Σ is a pair $(P, PossibleTranSets)$, where P is a prefix-closed language over Σ and $PossibleTranSets : P \to 2^{2^{\Sigma}} \setminus \emptyset$ satisfies that $w.a \in P$ if and only if $a \in TranSet \in PossibleTranSets(w)$ for some $TranSet$.*

Each of the formalisms presented so far in this section was an automata-based behavioural formalism. These are often preferred as they are easier to read than, for instance, formulae of modal logics. The alternative between logical and behavioural specifications is not only a question of preference. Automata-based specifications [Lar89, BG99] have a focus on compositional and incremental design in which logical specifications are somewhat lacking, with the trade-off of generally being less expressive than logics. Logical specification formalisms put a powerful logical language at the disposal of the user, and the logical approach to model checking [QS82, CE81] has seen a lot of success and tool implementations. As a result, one would like to establish connections between behavioural

and logical formalisms to exploit advantages of both at once. The relationship of MTS to logic was studied in [BL92, FP07]. It is established that MTS are equivalent to a fragment of μ-calculus where formulae are (1) consistent, (2) "prime", meaning the disjunction is allowed only in very special cases, and (3) do not contain the least fixpoint.

3.2 New results

We consider several extensions of MTS. First, we motivate and introduce the most general one and then discuss the relationships of its special cases. Their differences with respect to efficiency are discussed in Chapter 5. The motivation for DMTS was explained in Example 3.5. However, as the following example shows, convenient modelling requires more features such as conditional or persistent choices.

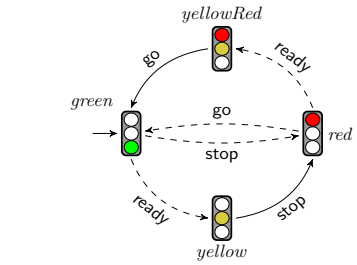
Example 3.7 *Consider a simple specification of a traffic light controller for several national variants for vehicles as well as for pedestrians in Fig. 3.5. At any moment it is in one of the four states* red, $green$, $yellow$ *or* $yellowRed$. *The requirements of the specification are: if* $green$ *is on then the traffic light may either change to* red *or* $yellow$, *and if it turned* $yellow$ *it must go to* red *afterward; if* red *is on then it may either turn to* $green$ *or* $yellowRed$, *and if it turns* $yellowRed$ *(as it is the case in some countries) it must go to* $green$ *afterwards.*

Figure 3.5a shows the respective MTS specification. In Figure 3.5c, Figure 3.5d and Figure 3.5e there are three different implementations of the MTS specification that are undesirable: the light is constantly $green$, *the lights switch nondeterministically or* $yellow$ *is only displayed sometimes. While the first problem can be avoided using DMTS (see Figure 3.5b), the latter two cannot. To eliminate the second implementation, we introduce* Boolean MTS, *which can model exclusive choice. For the third implementation to be removed, we need persistent choice, which can be modelled in* parametric MTS *where a parameter describes whether and when the yellow light is used making the choices permanent in the whole implementation.*

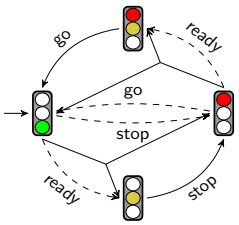
Now we define parametric MTS and instantiate other extended specification formalisms as its subclasses.

Definition 3.8 (Parametric modal transition system) *A* parametric MTS (PMTS) *over an action alphabet* Σ *is a tuple* (P, T, Par, Φ) *where*

- P *is a set of* processes,
- $T \subseteq P \times \Sigma \times P$ *is a* transition relation,
- Par *is a finite set of* parameters, *and*

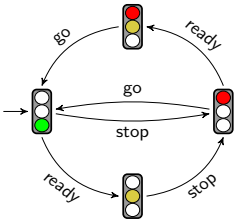


(a) MTS specification S_1

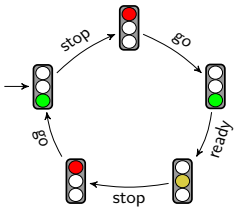


(b) DMTS specification S_2

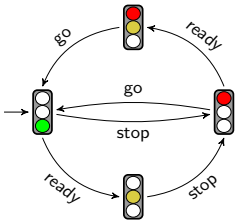
(c) Implementation I_1



(d) Implementation I_2

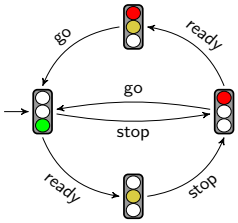


(e) Implementation I_3



Obligation function:
$\Phi(green) = (\mathsf{stop}, red) \oplus (\mathsf{ready}, yellow)$
$\Phi(red) = (\mathsf{go}, green) \oplus (\mathsf{ready}, yellowRed)$

(f) BMTS specification S_3



Parameters: $\{reqYfromR, reqYfromG\}$
Obligation function:
$\Phi(green) = ((\mathsf{stop}, red) \oplus (\mathsf{ready}, yellow))$
$\wedge (reqYfromG \Leftrightarrow (\mathsf{ready}, yellow))$
$\Phi(red) = ((\mathsf{go}, green) \oplus (\mathsf{ready}, yellowRed))$
$\wedge (reqYfromR \Leftrightarrow (\mathsf{ready}, yellowRed))$

(g) PMTS specification S_4

Figure 3.5: Specifications and implementations of a traffic light controller

- $\Phi : P \to \mathcal{B}((\Sigma \times P) \cup Par)$ *is an* obligation function *over outgoing transitions and parameters.*

A PMTS is positive *if for every* $s \in P$ *only parameters can be negated in* $\Phi(s)$*. If the following syntactic consistency*

$$\forall s \in P : \forall (a,t) \in \Phi(s) : (s,a,t) \in T \qquad \textbf{(SC)}$$

holds we call the PMTS pure*, otherwise* mixed*. A PMTS* (P, T, Par, Φ) *is called*

- Boolean MTS (BMTS) *if it is* parameter-free*, i.e. if* $Par = \emptyset$*,*
- transition system with obligation (OTS) *if it is parameter-free and positive.*

Intuitively, a set S of transitions from s is allowed if $\Phi(s)$ is true under the valuation induced by S and the fixed parameters.

Note that

- *DMTS* is an OTS with $\Phi(s)$ in the conjunctive normal form for all $s \in P$,
- *MixTS* is a DMTS with $\Phi(s)$ being a conjunction of positive literals (transitions) for all $s \in P$,
- *MTS* is a pure MixTS,
- *LTS* is an MTS with $\Phi(s) = \bigwedge T(s)$ for all $s \in P$, where $T(s) = \{(a,t) \mid (s,a,t) \in T\}$ is the set of all outgoing transitions of s.

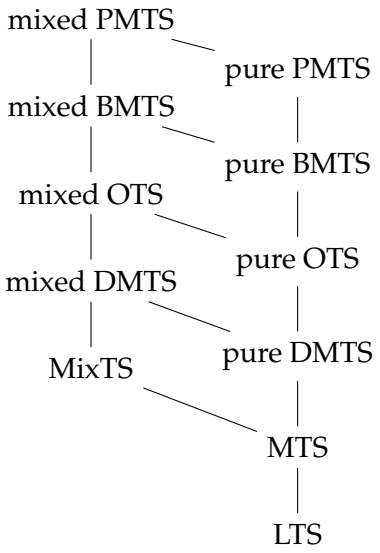


Figure 3.6: The syntactic hierarchy of MTS extensions

In this chapter we show that mixed variants usually do not have more expressive power than their pure counterparts. Apart from the already discussed extra power of MixTS over MTS, pure DMTS cannot express empty specification (with no implementations) while mixed DMTS can. Since this difference is not very important, we shall deal with pure systems only unless stated otherwise.

In order to define modal refinement, we first set the following notation. For a PMTS $M = (P, T, Par, \Phi)$, a valuation $\nu \subseteq Par$ of parameters induces a BMTS $M^\nu = (P, T, \emptyset, \Phi')$ where each occurrence of $p \in \nu$ in Φ is replaced by $\mathbf{tt}$ and of $p \notin \nu$ by $\mathbf{ff}$, i.e. for each $s \in P$, $\Phi'(s) = \Phi(s)[\mathbf{tt} \mapsto p \text{ for } p \in \nu, \mathbf{ff} \mapsto p \text{ for } p \notin \nu]$. Further, for $s \in P$, we denote by $\mathrm{Tran}_\nu(s) = \{E \subseteq T(s) \mid E \models \Phi'(s)\}$ the set

of all admissible sets of transitions from s under the fixed truth values of the parameters. Finally, we extend the notation to processes and let s^ν denote the process of M^ν corresponding to the process s of M.

We can now define the notion of modal refinement between PMTS as in Paper F.

Definition 3.9 (Modal refinement of PMTS) *Let $(P_1, T_1, Par_1, \Phi_1), (P_2, T_2, Par_2, \Phi_2)$ be PMTS over the same action alphabet and $s \in P_1$, $t \in P_2$ be processes. We say that s* modally refines *t, written $s \leq_m t$, if for every $\mu \subseteq Par_1$ there exists $\nu \subseteq Par_2$ such that (s^μ, t^ν) is contained in a* refinement relation *$R \subseteq P_1 \times P_2$ satisfying for every $(p, q) \in R$:*

$$\forall M \in \mathrm{Tran}_\mu(p) : \exists N \in \mathrm{Tran}_\nu(q) : \; \forall (a, p') \in M : \exists (a, q') \in N : (p', q') \in R \; \wedge \\ \forall (a, q') \in N : \exists (a, p') \in M : (p', q') \in R \, .$$

Intuitively, whatever parameters of the refining system we pick, the abstract system can emulate the same behaviour for some choice of its parameters.

Example 3.10 *Consider the rightmost PMTS in Fig. 3.7. It has two parameters $reqYfromG$ and $reqYfromR$ whose values can be set independently and it can be refined by the system in the middle of the figure having only one parameter $reqY$. This single parameter simply binds the two original parameters to the same value. The PMTS in the middle can be further refined into the implementations where either $yellow$ is always used in both cases, or never at all.*

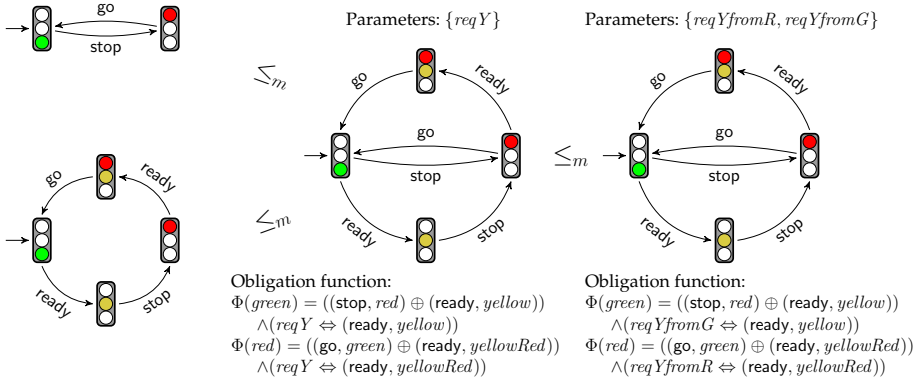


Figure 3.7: Example of modal refinement

The definition given above slightly differs from the one of Paper C. However, although this one is closer to the semantically defined notion of thorough refinement, it still keeps the same complexity as the modal refinement defined in Paper C. For the details, see Paper F.

3.2.1 Expressive power

Here we survey the results on the expressive power of the subclasses of PMTS achieved in Papers A, F and G. Firstly, in Paper G we define a *de-parametrization operator* $\mathcal{B}$ that blows up a PMTS into an equivalent (potentially exponentially larger) BMTS.

Proposition 3.11 *Let s be a process of a PMTS. Then $[\![s]\!] = [\![\mathcal{B}(s)]\!]$ and $s \leq_m \mathcal{B}(s)$.*

Secondly, we define an operator *de-negation* that transforms a BMTS with the set P of processes into a DMTS with the set $\{t \in \mathrm{Tran}(s) \mid s \in P\}$ of new processes. This way we can transform (again at an exponential cost) a BMTS into an equivalent DMTS with more, but still finitely many initial states. These are introduced formally in Paper G.

Proposition 3.12 *Let s be a process of a BMTS, $[\![s]\!] = \bigcup_{t \in \mathrm{Tran}_\emptyset(s)} [\![t]\!]$.*

Thirdly, in Paper A we show how to transform a mixed DMTS with non-empty set of implementations into a pure DMTS again at an exponential cost. The method used is a variant of a powerset construction and is not limited to disjunctive MTS. The only exceptions are the inconsistent specification, i.e. with no implementations. We disregard this minor technical difference here.

Finally, in Paper F we prove that DMTS with more initial states are equivalent to ν-calculus (or Hennessy-Milner logic with greatest fixpoints, abbreviated νHML), which is a fragment of μ-calculus without the least fixpoint. To this end, we use a non-deterministic extension of alternating automata (NAA) as an intermediate step.

Example 3.13 *Consider the following property: "at all time points after executing* request*, no* idle *nor further* request*s but only* work *is allowed until* grant *is executed". The property can be written in e.g. CTL [CE81] as*

$$AG(\mathsf{request} \Rightarrow AX(\mathsf{work}\ AW\ \mathsf{grant}))$$

Figure 3.8 shows an example of an equivalent νHML formula and a DMTS corresponding to this property.

$X = [\mathsf{grant}, \mathsf{idle}, \mathsf{work}]X \wedge [\mathsf{request}]Y$
$Y = (\langle \mathsf{work} \rangle Y \vee \langle \mathsf{grant} \rangle X) \wedge [\mathsf{idle}, \mathsf{request}]\mathbf{ff}$

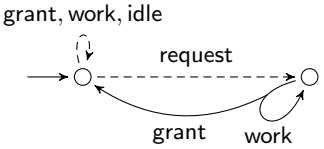


Figure 3.8: Example of a νHML formula and an equivalent DMTS

Summing up the results, we obtain the following:

Theorem 3.14 *For any set $\mathcal{S}$ of LTS, the following are equivalent:*

1. *There exists a finite PMTS process s with $[\![s]\!] = \mathcal{S}$.*
2. *There exists a finite BMTS process s with $[\![s]\!] = \mathcal{S}$.*
3. *There exists a finite set S of finite DMTS processes with $\bigcup_{s \in S} [\![s]\!] = \mathcal{S}$.*
4. *There exists a μ-calculus formula φ without μ with $[\![\varphi]\!] = \mathcal{S}$.*

Furthermore, the first three statements are equivalent even if we drop the finiteness constraints. In the fourth, we would have to allow infinite conjunctions and disjunctions.

Apart from the logical characterization, we can also describe processes using a process algebra [Fok00]. Here we do it only for (mixed) OTS and not consider parallel composition, for the latter see Paper A. Let $\mathcal{X}$ be a set of process names. A *term of process algebra for OTS* is given by the following syntax:

$$P ::= \text{nil} \mid \text{co-nil} \mid a.P \mid X \mid P \wedge P \mid P \vee P \mid \not\downarrow P$$

where X ranges over $\mathcal{X}$ and every $X \in \mathcal{X}$ is assigned a defining equality of the form $X := P$ where P is a term. The semantics is given by the following structural operational semantics rules:

$$\frac{}{a.P \xrightarrow{a} P} \qquad \frac{P \xrightarrow{a} P'}{X \xrightarrow{a} P'} X := P \qquad \frac{P \xrightarrow{a} P'}{P \wedge Q \xrightarrow{a} P'} \qquad \frac{P \xrightarrow{a} P'}{P \vee Q \xrightarrow{a} P'}$$

and, since the processes represent sets of implemented systems (i.e. sets of sets of behaviours), by an obligation function defined structurally as follows:

$$\begin{array}{lll} \boldsymbol{\Omega}(\text{nil}) & = \mathbf{tt} & \boldsymbol{\Omega}(P \wedge Q) = \boldsymbol{\Omega}(P) \wedge \boldsymbol{\Omega}(Q) \\ \boldsymbol{\Omega}(\text{co-nil}) & = \mathbf{ff} & \boldsymbol{\Omega}(P \vee Q) = \boldsymbol{\Omega}(P) \vee \boldsymbol{\Omega}(Q) \\ \boldsymbol{\Omega}(a.P) & = (a, P) & \boldsymbol{\Omega}(\not\downarrow P) = \boldsymbol{\Omega}(P) \\ \boldsymbol{\Omega}(X) & = \boldsymbol{\Omega}(P) \quad \text{for } X := P & \end{array}$$

We now obtain the discussed subclasses of OTS as syntactic subclasses generated by the following syntax equations (modulo transformation to conjunctive normal form):

mixed DMTS	$P ::= \text{nil} \mid a.P \mid X \mid P \wedge P \mid P \vee P \mid \not\downarrow P \mid \text{co-nil}$
pure DMTS	$P ::= \text{nil} \mid a.P \mid X \mid P \wedge P \mid P \vee P$
MixTS	$P ::= \text{nil} \mid a.P \mid X \mid P \wedge P \mid P \vee \text{nil} \mid \not\downarrow P \mid \text{co-nil}$
MTS	$P ::= \text{nil} \mid a.P \mid X \mid P \wedge P \mid P \vee \text{nil}$
LTS	$P ::= \text{nil} \mid a.P \mid X \mid P \wedge P$

The results on the relationships are summarized in Fig. 3.9, where Cm denotes a class C where systems are considered with more initial states.

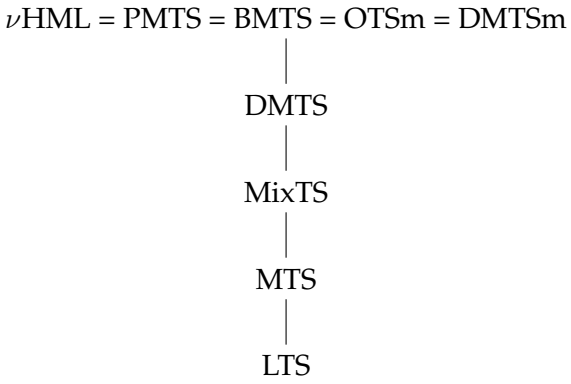


Figure 3.9: The semantic hierarchy of MTS extensions not considering empty specifications

Chapter 4

Extensions of transition systems

In the previous chapter, we extended the *modalities* of the systems. In this chapter, we focus on extensions of *systems* the modalities operate on. We focus on (1) systems that are executed in real time and (2) systems with infinite state space.

4.1 State of the art

Over the years, many extensions of MTS have been proposed. While many focus on what *combinations* of transitions are possible as discussed in Chapter 3, some lifted MTS to *quantitative settings* [LL12] with clear applications in the embedded systems design. This includes probabilistic specifications [JL91, CDL$^+$10, DKL$^+$11] and various weighted specifications, where weights stand for various quantitative aspects (e.g. time, power or memory), which are highly relevant in the area of embedded systems. As far as the particular case of timed systems is concerned, the quantity of time can be refined in various ways. In the early work [CGL93, LSW95], the precise quantities are almost disregarded. More recently [JLS12, BPR09, BLPR09, DLL$^+$10], the possible times are usually specified as time intervals, which can be narrowed down and thus made more specific. A more general option is to permit changes to anything smaller with respect to some abstract ordering; [BJL$^+$12a] provides the following conservative extension of MTS modal refinement along these lines:

Definition 4.1 (Modal refinement of MTS with structured labels) *Let Σ be an alphabet with an ordering $\sqsubseteq$. Let $(P_1, \dashrightarrow_1, \longrightarrow_1), (P_2, \dashrightarrow_2, \longrightarrow_2)$ be MTS over Σ and $s \in P_1$, $t \in P_2$ be processes. We say that s* modally refines *t, written $s \leq_m t$, if there is a relation $\mathcal{R} \subseteq P_1 \times P_2$ such that $(s,t) \in \mathcal{R}$ and for every $(p,q) \in R$ and every $a \in \Sigma$:*

1. *if $p \overset{a}{\dashrightarrow}_1 p'$ then there is a transition $q \overset{\bar{a}}{\dashrightarrow}_2 q'$ with $a \sqsubseteq \bar{a}$ and $(p',q') \in R$, and*
2. *if $q \overset{\bar{a}}{\longrightarrow}_2 q'$ then there is a transition $p \overset{a}{\longrightarrow}_1 p'$ with $a \sqsubseteq \bar{a}$ and $(p',q') \in R$.*

Example 4.2 *Consider $\Sigma = L \times \mathfrak{I}$ where L is a finite set ordered by identity and $\mathfrak{I}$ is the set of intervals ordered by inclusion and Σ is ordered point-wise. A transition labelled by $(\ell, [a, b])$ can thus be implemented by a transition (ℓ, c) for any $c \in [a, b]$.*

Moreover, one can also consider probabilistic and timed extension of MTS at once [HKKG13], more weights at once [BJL$^+$12b], or MTS with timed-automata clocks [BLPR12, FL12]. In all the quantitative settings, it is also natural to extend the qualitative notion of refinement into a quantitative notion of distance of systems [BFJ$^+$11, BFLT12].

Other extensions of MTS consider *infinite state* systems. Here only a few more or less ad hoc extensions have been proposed, such as systems with asynchronous communication based on FIFO [BHJ10] or Petri nets [EBHH10]. Other extensions consider also data to some extent [BHB10, BHW10, BLL$^+$14]. A systematic exploration of infinite state MTS was missing.

A convenient unifying framework for (non-modal) infinite-state systems is provided by process rewrite systems (PRS) [May00]. They encompass many standard models such as pushdown automata (PDA) or Petri nets (PN) as syntactic subclasses. A PRS consists of a finite set of rewriting rules, which model the computation. These rules may contain sequential and parallel composition. Formally, let $\mathcal{X}$ be a set of *process constants*. We define the set of process expressions $\mathcal{E}$ by the following abstract syntax:

$$E ::= \mathrm{nil} \mid X \mid E \parallel E \mid E; E$$

where X ranges over $\mathcal{X}$. The process expressions are considered modulo the usual structural congruence, i.e. the smallest congruence such that the operator ; is associative, $\parallel$ is associative and commutative and nil is a unit for both ; and $\parallel$.

Definition 4.3 (Process rewrite system) *A* process rewrite system (PRS) *is a finite relation $\Delta \subseteq (\mathcal{E} \setminus \{\mathrm{nil}\}) \times \Sigma \times \mathcal{E}$, elements of which are called* rewrite rules. *A PRS Δ induces a labelled transition system $LTS(\Delta) = (\mathcal{E}, \longrightarrow)$ where $\longrightarrow$ is the least relation satisfying the following rules:*

$$\frac{(E, a, E') \in \Delta}{E \xrightarrow{a} E'} \qquad \frac{E \xrightarrow{a} E'}{E; F \xrightarrow{a} E'; F} \qquad \frac{E \xrightarrow{a} E'}{E \parallel F \xrightarrow{a} E' \parallel F}$$

We consider four distinguished classes of process expressions. Class **S** stands for expressions with no $\parallel$ (purely sequential expressions) and class **P** stands for expressions with no ; (purely parallel expressions). Further, we use **G** for the whole $\mathcal{E}$ (general expressions) and **1** for $\mathcal{X}$ (one process constant and no operators). Now restricting the left and right sides of rules of PRS to these classes yields

subclasses of PRS as depicted in Figure 4.1; for the standard shortcuts used, see Paper E. Note that the hierarchy is strict with respect to the bisimulation equivalence.

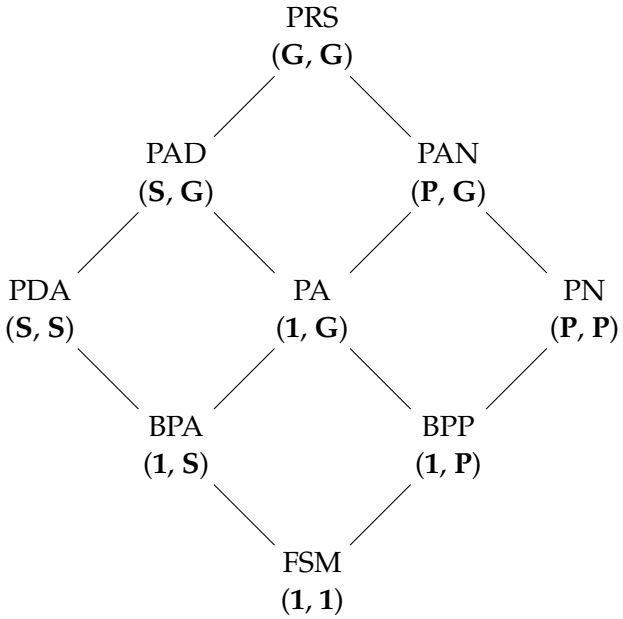


Figure 4.1: PRS hierarchy

Example 4.4 *A transition t of a Petri net with input places p, q and output places r, s can be described by the rule $p \parallel q \xrightarrow{t} r \parallel s$. A transition of a pushdown automaton in a state s with a top stack symbol X reading a letter a resulting in changing the state to q and pushing Y onto the stack can be written as $sX \xrightarrow{a} qY$.*

4.2 New results

4.2.1 Modal transition systems with durations

We present a timed extension of BMTS (the corresponding extension of PMTS would be straightforward). The time durations of transitions are modelled as controllable or uncontrollable intervals. Controllable intervals can be further refined into narrower intervals, whereas uncontrollable are considered under the control of an unpredictable environment and cannot be further narrowed down.

Definition 4.5 (MTSD) *A* modal transition system with durations (MTSD) *is a tuple $\mathcal{S} = (P, T, \Phi, D)$ where $(P, T, \emptyset, \Phi)$ is a BMTS and $D : T \to \mathfrak{I}$ is a* duration interval function*, where $\mathfrak{I} = \{\langle m, n\rangle, [m, n] \mid m, n \in \mathbb{N}_0, m \leq n\}$.*

Moreover, we require that there are no Zeno cycles, i.e. there is no sequence $s_1 a_1 s_2 a_2 \cdots s_n$ *where* $(s_i, a_i, s_{i+1}) \in T$ *and* $s_n = s_1$ *such that for all* i*, the interval* $D((s_i, a_i, s_{i+1}))$ *is of the form either* $\langle 0, m\rangle$ *or* $[0, m]$ *for some* m*.*

Example 4.6 *Consider the specification* $\mathcal{S}$ *of Figure 4.2 describing the work of a shuttle bus driver. He drives a bus between a hotel and the airport. First, the driver has to* wait *for the passengers at the hotel. This can take one to five minutes. Then the driver has to* drive *the bus to the airport (this takes six to ten minutes) where he has to do a* small_cleanup*, then* wait *for passengers before he can* drive *the bus back to the hotel. When he returns he can do either a* small_cleanup*,* big_cleanup *or* skip_cleanup *of the bus before he starts over again. Note that next time the choice in* t *may differ.*

There are two types of durations on the transitions. First, there are controllable *intervals, written in angle brackets. The meaning of e.g.* $\langle 1, 5\rangle$ *is that in the implementation we can instruct the driver to wait for a fixed number of minutes in the range. Second, there are* uncontrollable *intervals, written in square brackets. The interval* $[6, 10]$ *on the* drive *transition means that in the implementation we cannot fix any particular time and the time can vary, say, depending on the traffic and it is chosen non-deterministically by the environment.*

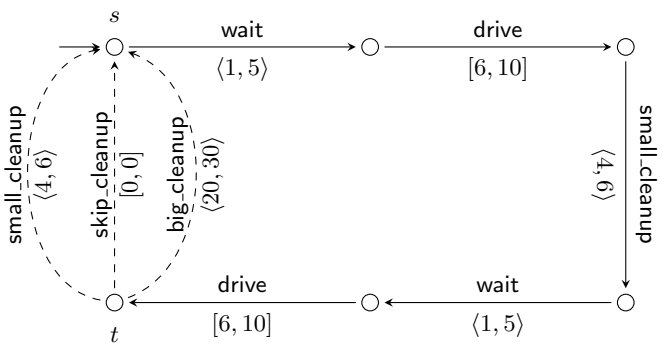


Figure 4.2: Example of a modal transition system with time durations

We now define a notion of modal refinement. In order to do that, we first need to define refinement of intervals as a binary relation $\leq \subseteq \mathfrak{I} \times \mathfrak{I}$ such that

- $\langle m', n'\rangle \sqsubseteq \langle m, n\rangle$ whenever $m' \geq m$ and $n' \leq n$, and
- $[m', n'] \sqsubseteq \langle m, n\rangle$ whenever $m' \geq m$ and $n' \leq n$.

Thus controllable intervals can be refined by narrowing them, at most until they become singleton intervals, or until they are changed to uncontrollable intervals. The modal refinement is now a combination of BMTS refinement with refinement on structured labels over $\Sigma \times \mathfrak{I}$ ordered by $= \times \sqsubseteq$. The latter compound is thus a special case of [BJL$^+$12a] (published after Paper D).

Definition 4.7 (Modal refinement of MTSD) *Let (P_1, T_1, Φ_1, D_1), (P_2, T_2, Φ_2, D_2) be MTSD and $s_1 \in S_1, s_2 \in S_2$ be processes. We say that s_1 modally refines s_2, written $s_1 \leq_m s_2$, if there is a relation $R \subseteq P_1 \times P_2$ containing (s_1, s_2) such that for every $(s,t) \in R$ the following holds:*

$$\begin{aligned}
&\forall M \in \mathrm{Tran}(s) : \exists N \in \mathrm{Tran}(t) : \\
&\qquad \forall (a, s') \in M : \exists (a, t') \in N : D_1(s,a,s') \sqsubseteq D_2(t,a,t') \ \wedge \ (s', t') \in R, \text{ and} \\
&\qquad \forall (a, t') \in N : \exists (a, s') \in M : D_1(s,a,s') \sqsubseteq D_2(t,a,t') \ \wedge \ (s', t') \in R \,.
\end{aligned}$$

In Chapter 5, we further equip the model with a *dual price scheme*, i.e. two kinds of quantitative costs: each action has its own running cost per time unit, and actions may require several hardware components of different costs. We then discuss the problem of finding an implementation with the cheapest long-run average running costs per time unit (so that we pay the driver the least possible amount of money) given a fixed budget for the investment into the hardware components.

4.2.2 Modal process rewrite systems

In this section, we introduce modalities into a general framework for infinite-state systems, namely process rewrite systems. In Chapter 5, we study modal extensions of well-established classes of infinite-state systems and the complexity of their analysis.

One can naturally lift PRS to the modal world by having two sets of rules: may and must rules. The finite set of rules then generates a generally infinite MTS.

Definition 4.8 (Modal process rewrite system) *A* modal process rewrite system (mPRS) *is a tuple $\Delta = (\Delta_{\mathrm{may}}, \Delta_{\mathrm{must}})$ where $\Delta_{\mathrm{must}} \subseteq \Delta_{\mathrm{may}}$ are two PRS.*

The mPRS Δ induces an MTS

$$MTS(\Delta) = (\mathcal{E}, \dashrightarrow, \longrightarrow)$$

defined by $LTS(\Delta_{\mathrm{may}}) = (\mathcal{E}, \dashrightarrow)$ and $LTS(\Delta_{\mathrm{may}}) = (\mathcal{E}, \longrightarrow)$.

In other words the MTS is induced by the mPRS structurally as follows:

$$\frac{(E, a, E') \in \Delta_{\mathrm{may}}}{E \overset{a}{\dashrightarrow} E'} \qquad \frac{E \overset{a}{\dashrightarrow} E'}{E; F \overset{a}{\dashrightarrow} E'; F} \qquad \frac{E \overset{a}{\dashrightarrow} E'}{E \parallel F \overset{a}{\dashrightarrow} E' \parallel F}$$

$$\frac{(E, a, E') \in \Delta_{\text{must}}}{E \xrightarrow{a} E'} \qquad \frac{E \xrightarrow{a} E'}{E; F \xrightarrow{a} E'; F} \qquad \frac{E \xrightarrow{a} E'}{E \parallel F \xrightarrow{a} E' \parallel F}$$

Each subclass $\mathcal{C}$ of PRS has a corresponding modal extension $\mathrm{m}\mathcal{C}$ containing all mPRS $(\Delta_{\text{may}}, \Delta_{\text{must}})$ with both Δ_{may} and Δ_{must} in $\mathcal{C}$. For instance, mFSM correspond to the standard finite MTS and mPN are modal Petri nets as introduced in [EBHH10].

Definition 4.9 (Modal refinement) *Given mPRS $\Delta_1 \in m\mathcal{C}_1, \Delta_2 \in m\mathcal{C}_2$ and process terms δ_1, δ_2, we say δ_1 refines δ_2, written $\delta_1 \leq_m \delta_2$, if $\delta_1 \leq_m \delta_2$ as processes of $MTS(\Delta_1)$ and $MTS(\Delta_2)$, respectively.*

What is the use of mPRS? Firstly, potentially infinite-state systems such as Petri nets are very popular for modelling whenever communication and/or synchronization between processes occurs. This is true even in cases where they are actually bounded and thus with a finite state space.

Example 4.10 *Consider the following may rule (we use dashed arrows to denote may rules) generating a small Petri net.*

$$\mathsf{resource} \overset{\mathsf{produce}}{\dashrightarrow} \mathsf{money} \parallel \mathsf{trash}$$

If this is the only rule with trash *on the right side a safety property is guaranteed for all implementations of this system, namely that* trash *can only arise if there is at least one* resource*. On the other hand, it is not guaranteed that* money *can indeed be produced in such a situation. This is very useful as during the design process new requirements can arise, such as necessity of adding more participants to perform this transition. For instance,*

$$\mathsf{resource} \parallel \mathsf{permit} \overset{\mathsf{produce}}{\dashrightarrow} \mathsf{money} \parallel \mathsf{trash}$$

expresses an auxiliary condition required to produce trash*, namely that* permit *is available. Replacing the old rule with the new one is equivalent to adding an input place* permit *to the modal Petri net, see Figure 4.3 in red. In the modal transition system view, the new system* refines *the old one. Indeed, the new system is only more specific about the allowed behaviour than the old one and does not permit any previously forbidden behaviour.*

One can further refine the system into the one given by

$$\mathsf{resource} \parallel \mathsf{permit} \parallel \mathsf{bribe} \overset{\mathsf{produce}}{\longrightarrow} \mathsf{money} \parallel \mathsf{trash}$$

where additional condition is imposed and now the money*-producing transition has to be available (denoted by an unbroken arrow) whenever the left hand side condition is satisfied.*

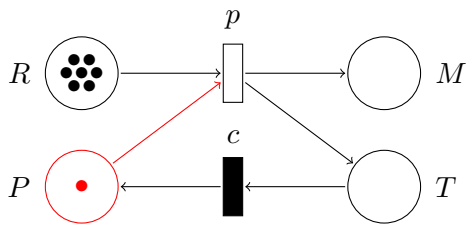


Figure 4.3: A modal Petri net given by rules Resource|| Permit $\overset{\text{produce}}{\dashrightarrow}$ Money || Trash and Trash $\overset{\text{clean}}{\longrightarrow}$ Permit with may transitions drawn as empty boxes and must transitions as full boxes

Further, infinitely many states are useful to capture unbounded memory. For instance, consider a specification where the total amount of permits is not explicitly limited. In an implementation, the number of permits might need to be remembered in the state of the system.

Example 4.11 *Consider a basic process algebra (BPA) given by rules* $X \overset{(}{\longrightarrow} XX$ *and* $X \overset{)}{\longrightarrow} \varepsilon$ *for correctly parenthesized expressions with* $X \overset{a}{\dashrightarrow} X$ *for all other symbols* a*, i.e. with no restriction on the syntax of expressions. One can easily refine this system into a PDA that accepts correct arithmetic expressions by remembering in a control state whether the last symbol read was an operand or an operator.*

Chapter 5

Analysis

In this chapter, we give algorithms for and establish complexities of the most important problems on MTS and their extensions.

5.1 State of the art

5.1.1 Refinements

Modal refinement is a syntactically defined notion extending both bisimulation and simulation. Similarly to bisimulation having a semantic counterpart in trace equivalence, here the semantic counterpart of modal refinement is the *thorough refinement*. As opposed to the syntactic definition using local notions, the semantic definition relates (by inclusion) the sets of implementations of the specifications. The definition is universal for all extensions of MTS as it only depends on the notion of implementation and not on syntax of the particular extension.

Definition 5.1 (Thorough refinement) *Given processes s and t, we say that s* thoroughly refines *t, written $s \leq_t t$, if $[\![s]\!] \subseteq [\![t]\!]$.*

Note that the two refinements are in general different as we illustrate in the following example:

Example 5.2 *Consider processes s and t of Figure 5.1. On the one hand, the sets of implementations of s and t are the same, namely those that can perform either no action or one a or two a's or combine the latter two options. On the other hand, s does not modally refine t. Indeed, whenever $s \leq_m t$ then either $s' \leq_m t_1$ or $s' \leq_m t_2$. However, neither is true, as s' allows a transition while t_1 does not, and s' does not require any transition while t_2 does.*

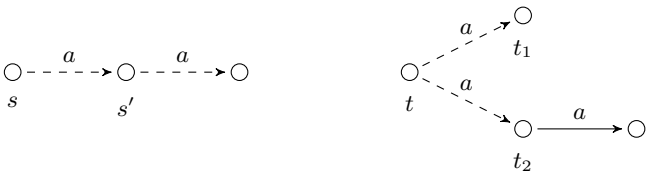


Figure 5.1: $s \leq_t t$, but $s \not\leq_m t$

Although the two refinements differ, modal refinement is a sound under-approximation of the thorough refinement. Indeed, whenever we have $i \leq_m s$ and $s \leq_m t$, by transitivity of the modal refinement we obtain $i \leq_m t$.

Proposition 5.3 *Let s, t be processes. If $s \leq_m t$ then $s \leq_t t$.*

Moreover, in [BKLS09b] we show the other direction holds whenever the refined system is deterministic. A process is *deterministic* if, for each process s of its underlying MTS and for each $a \in \Sigma$, there is at most one s' such that $s \xrightarrow{a} s'$.

Proposition 5.4 *Let s, t be processes and t deterministic. If $s \leq_t t$ then $s \leq_m t$.*

In Table 5.1 we give an overview of the results related to deciding modal and thorough refinements for different combinations of processes on the left- and right-hand side (here D stands for deterministic processes and N for non-deterministic processes). Note that the co-inductive refinement relations are easy to compute using a fixed-point computation, although other methods are also possible, e.g. logical programming [AKRU11] or QBF solving described later.

Table 5.1: Refinement complexities for various cases of (non)determinism

	modal refinement	thorough refinement
D,D	NL-complete [BKLS09b]	NL-complete [BKLS09b]
N,D	NL-complete [BKLS09b]	NL-complete [BKLS09b]
D,N	$\in$ P [KS90, PT87] P-hard [BKLS09b]	$\in$ EXP [AHL$^+$08b] EXP-hard [BKLS12]
N,N	$\in$ P [KS90, PT87] P-hard [BGS92]	$\in$ EXP [AHL$^+$08b, BKLS09a] EXP-hard [BKLS09a]

Since the thorough refinement is EXP-hard, it is much harder than the modal refinement. Therefore, we also investigate how the thorough refinement can be approximated by the modal refinement. While under-approximation is easy, as modal refinement implies thorough refinement, over-approximation is more difficult. Here one can use our method of the deterministic hull for MTS [BKLS09b]. The *deterministic hull* operator $\mathcal{D}$ is a generalization of the powerset construction on finite automata and it is the *smallest* (w.r.t. modal refinement) deterministic system refined by the original system.

Proposition 5.5 *Let s be an arbitrary MTS process. Then $\mathcal{D}(s)$ is a deterministic MTS process such that $s \leq_m \mathcal{D}(s)$ and, for every deterministic MTS process t, if $s \leq_t t$ then $\mathcal{D}(s) \leq_m t$.*

Corollary 5.6 *For any processes s, t, if $s \not\leq_m \mathcal{D}(t)$ then $s \not\leq_t t$.*

There are also other notions of refinements of systems close to MTS, such as alternating refinements [AHKV98, AFdFE+11], branching refinement [FBU09] or refinement preserving the termination possibility [CR12].

5.1.2 Operations

We are interested in classes of modal systems being closed under certain operations introduced in Chapter 2. Whenever they are closed we want to know how to compute the result and how hard it is.

Firstly, we address the closure properties of previously investigated classes of systems, see Table 5.2. While the automata based formalisms automatically allow to compose systems structurally, logical operations are either difficult to compute or cannot be expressed in the formalism at all. Therefore, most of the focus has been directed to the very simple deterministic case, where some operations can be defined using local syntactic rules.

Table 5.2: Closure properties of previously known classes of modal systems

	$\wedge$	$\vee$	$\neg$	$\parallel$	$/$
deterministic MTS	✓	×	×	✓	✓
MTS	?	×	×	✓	×
MixTS	✓	×	×	✓	?
DMTS	?	×	×	✓	?

The parallel composition can often be lifted to the modal setting simply by applying the same rules for both may and must transition functions. This holds for a wide class of operators as described in our [BKLS09b]. Here we consider a simple case of synchronous message passing with the synchronization alphabet $\Sigma' \subseteq \Sigma$, i.e. full synchronization on Σ' and interleaving on $\Sigma \setminus \Sigma'$:

$$\frac{s \overset{a}{\dashrightarrow} s' \quad t \overset{a}{\dashrightarrow} t'}{s \parallel t \overset{a}{\dashrightarrow} s' \parallel t'} \forall a \in \Sigma' \qquad \frac{s \overset{a}{\dashrightarrow} s'}{s \parallel t \overset{a}{\dashrightarrow} s' \parallel t} \forall a \in \Sigma \setminus \Sigma' \qquad \frac{t \overset{a}{\dashrightarrow} t'}{s \parallel t \overset{a}{\dashrightarrow} s \parallel t'} \forall a \in \Sigma \setminus \Sigma'$$

$$\frac{s \overset{a}{\longrightarrow} s' \quad t \overset{a}{\longrightarrow} t'}{s \parallel t \overset{a}{\longrightarrow} s' \parallel t'} \forall a \in \Sigma' \qquad \frac{s \overset{a}{\longrightarrow} s'}{s \parallel t \overset{a}{\longrightarrow} s' \parallel t} \forall a \in \Sigma \setminus \Sigma' \qquad \frac{t \overset{a}{\longrightarrow} t'}{s \parallel t \overset{a}{\longrightarrow} s \parallel t'} \forall a \in \Sigma \setminus \Sigma'$$

The quotient is more complex. For deterministic MTS, we can define it syntac-

tically as in [Rac07, Rac08]:

$$\frac{s \overset{a}{\dashrightarrow} s' \quad t \overset{a}{\dashrightarrow} t'}{s \wedge t \overset{a}{\dashrightarrow} s' \wedge t'} \qquad \frac{t \overset{a}{\not\dashrightarrow} t'}{s \wedge t \overset{a}{\dashrightarrow} univ}$$

$$\frac{s \overset{a}{\longrightarrow} s' \quad t \overset{a}{\longrightarrow} t'}{s \wedge t \overset{a}{\longrightarrow} s' \wedge t'} \qquad \frac{s \overset{a}{\longrightarrow} s' \quad t \overset{a}{\not\longrightarrow} t'}{s \wedge t \in \lightning} \qquad \frac{}{univ \overset{a}{\dashrightarrow} univ}$$

where states in $\lightning$ are *inconsistent* and must be pruned. Pruning $t \in \lightning$ means t must be removed and whenever there is $s \overset{a}{\longrightarrow} t$ we also recursively prune s. For non-deterministic MTS, the problem was open. Further related questions such as decomposition of a system into several components put in parallel have also been investigated [SUBK12], but again only for deterministic systems.

The situation is similar with conjunction. For deterministic MTS, we can again define it syntactically:

$$\frac{s \overset{a}{\dashrightarrow} s' \quad t \overset{a}{\dashrightarrow} t'}{s \wedge t \overset{a}{\dashrightarrow} s' \wedge t'} \qquad \frac{s \overset{a}{\longrightarrow} s' \quad t \overset{a}{\dashrightarrow} t'}{s \wedge t \overset{a}{\longrightarrow} s' \wedge t'} \qquad \frac{s \overset{a}{\dashrightarrow} s' \quad t \overset{a}{\longrightarrow} t'}{s \wedge t \overset{a}{\longrightarrow} s' \wedge t'}$$

$$\frac{s \overset{a}{\longrightarrow} s' \quad t \overset{a}{\not\dashrightarrow} t'}{s \wedge t \in \lightning} \qquad \frac{s \overset{a}{\not\dashrightarrow} s' \quad t \overset{a}{\longrightarrow} t'}{s \wedge t \in \lightning}$$

using the same pruning procedure. For non-deterministic systems, there were several attempts. Unfortunately, the resulting MTS is not minimal (with respect to modal refinement) [UC04], or not finite even when claimed to be finite [FU08]: their "clone" operation may not terminate even in cases when it is supposed to, for example, for processes s_1, s_2 of Fig. 5.4 where the self-loops are redirected back to the initial states.

We are also interested in questions closely related to the discussed conjunction. The *common implementation* decision problem (CI) contains tuples of systems, such that there is an implementation refining each system of the tuple. For tuples of size two this is equivalent to non-emptiness of the conjunction, for one system (for instance a MixTS) this is equivalent to semantic consistency (or non-emptiness) [LNW07b], i.e. existence of implementation. Note that despite the lack of results on conjunction of non-deterministic systems the complexity is known here. The complexity improves when the input processes are deterministic ($\mathrm{CI_D}$ problem). Finally, rather surprisingly, the problem whether there is a deterministic common implementation (dCI) is hard. We display the known results in Table 5.3 for several cases depending on whether the number of input processes is fixed or a part of the input. The results again indicate that several problems become more tractable if the given specifications are deterministic .

While MTS are not closed under complement (not even deterministic ones), there have been attempts at characterizing symmetric difference [SCU11].

Table 5.3: Complexity of the common implementation problems

	single MTS	single MixTS	fixed # of systems	arbitrary # of systems
CI	trivial	EXP-c. [AHL$^+$09]	P-c. [BGS92, HH08]	EXP-c. [AHL$^+$09]
$\mathrm{CI_D}$	trivial	trivial	NL-c.[BKLS09b]	PSPACE-c.[BKLS09b]
dCI	EXP-c.[BKLS09b]	EXP-c.[BKLS09b]	EXP-c.[BKLS09b]	EXP-c.[BKLS09b]

5.1.3 Model checking

Given a valuation $\nu : P \to 2^{Ap}$ assigning to each process a set of atomic propositions valid in the process, one can check whether an MTS satisfies an LTL formula φ over Ap. Since an MTS stands for a class of implementations, the question of satisfaction can be posed in two flavours:

($\models_\forall$-problem) Do all implementations satisfy φ?

($\models_\exists$-problem) Is there an implementation satisfying φ?

In [GP09] the generalized model checking of LTL over partial Kripke structures (PKS) is shown to be 2-EXP-hard. Further, [GJ03] describes a reduction from generalized model checking of μ-calculus over PKS to μ-calculus over MTS [Hut02, Hut99, GHJ01]. However, the hardness for LTL does not follow since the encoding of an LTL formula into μ-calculus includes an exponential blow-up. There is thus no straightforward way to use the result of [GJ03] to provide a polynomial reduction.

On the one hand, answering the $\models_\forall$-problem is easy. Indeed, it is sufficient to perform standard model checking on the "greatest" implementation, i.e. such where all mays are turned into musts and thus all possible runs are present. On the other hand, the $\models_\exists$-problem is much harder and trickier. Since LTL is usually interpreted over infinite words, all formulae are satisfied whenever there is an implementation without infinite runs, i.e. without a lasso of must transitions. There are several ways to avoid this vacuous satisfaction. Firstly, we can define LTL also on finite words, which we consider later in this chapter. Secondly, we can consider only implementations without deadlocks, which we also discuss. The deadlock-free approach has been studied in [UBC09] and the proposed solution was implemented in the tool MTSA [DFCU08]. Their approach attempts to find a *deadlock-free* implementation of a given MTS that satisfies a given formula. However, the solution given in [UBC09] is incorrect. In particular, existence of a deadlock-free implementation satisfying a given formula is claimed even in some cases where no such implementation exists.

Example 5.7 *The flaw can be seen on a simple counterexample given in Fig. 5.2. Clearly, S has no deadlock-free implementation with action a only, i.e. satisfying*

$\mathbf{GX}_a\mathbf{tt}$. *Yet the method of [UBC09] as well as the tool [DFCU08] claim that such an implementation exists.*

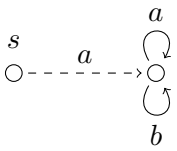


Figure 5.2: No deadlock-free implementation of s satisfies $\mathbf{GX}_a\mathbf{tt}$

While the solution attempt of [UBC09] yields a PSPACE algorithm, the problem is actually 2-EXP-complete.

Generalized model checking of MTS with respect to computation tree logic (CTL) is also investigated [AHL+08a, GAW13] as well as a variant of safety [DDM10].

5.1.4 Tools

Although the tool support is quite extensive, e.g. [BLS95, DFFU07, BML11, BČK11], it is so far limited to basic MTS and, moreover, partially limited to deterministic systems. The currently available tools are `MTSA` (Modal transition system analyzer) [DFFU07] and `MIO` (MIO Workbench) [BML11]. While `MTSA` is a tool for MTS, `MIO` is a tool for modal I/O automata (MIOA) [LNW07a, RBB+11], which combine MTS and interface automata based on I/O automata. Although MIOA have three types of may and must transitions (input, output, and internal), if we restrict to say only input transitions, the refinement works the same as for MTS, and some other operations, too. Further, there are also tools for loosely related formalisms of I/O automata (with no modalities) such as `ECDAR` (Environment for Compositional Design and Analysis of Real Time Systems) [DLL+10], which supports their timed extension. The comparison of the functionality of the tools is depicted in Table 5.12.

5.2 New results

5.2.1 Refinements

In Chapter 3, we show several new extensions of MTS, see Fig. 3.6. We also show that most of them have the same expressive power. However, the transformations are exponential and thus the extensions differ in succinctness. Therefore, the respective refinement problems are expected to be harder for the more succinct extensions, which is indeed the case. We focus both on theoretical and practical complexity of the refinement problems.

Modal refinement

We first show the complexities of modal refinement for systems without parameters depending on the form of the obligation function. We consider formulae of the general form (BMTS), positive formulae (OTS), conjunctive (DMTS) and disjunctive normal form, conjunctions of literals (MTS) and implementations (LTS), see Table 5.4. Observe that in most cases the refinement can be decided in polynomial time or using a SAT solver. The notable exception is the case when a general BMTS is refined.

Table 5.4: Complexity of modal refinement checking of parameter-free systems. The refining system is displayed in the first column, the refined system in the first row.

	Boolean	Positive	pCNF	pDNF	MTS
Boolean	Π_2^p–c.	coNP–c.	$\in$ coNP P-hard	coNP–c.	$\in$ coNP P-hard
Positive	Π_2^p–c.	coNP–c.	P–c.	coNP–c.	P–c.
pCNF	Π_2^p–c.	coNP–c.	P–c.	coNP–c.	P–c.
pDNF	Π_2^p–c.	P–c.	P–c.	P–c.	P–c.
MTS	Π_2^p–c.	P–c.	P–c.	P–c.	P–c.
LTS	NP–c.	P–c.	P–c.	P–c.	P–c.

For systems with parameters, the complexity is significantly higher, see Table 5.5.

Table 5.5: Complexity of modal refinement checking with parameters

	Boolean	positive	pCNF	pDNF
Boolean	Π_4^p–c.	Π_3^p–c.	$\in \Pi_3^p$ Π_2^p-hard	Π_3^p–c.
positive	Π_4^p–c.	Π_3^p–c.	Π_2^p–c.	Π_3^p–c.
pCNF	Π_4^p–c.	Π_3^p–c.	Π_2^p–c.	Π_3^p–c.
pDNF	Π_4^p–c.	Π_2^p–c.	Π_2^p–c.	Π_2^p–c.
MTS	Σ_3^p–c.	NP–c.	NP–c.	NP–c.
LTS	NP–c.	NP–c.	NP–c.	NP–c.

Since all the complexities are included in PSPACE, the huge success of SAT (Boolean satisfiability) solvers and also QBF (true quantified Boolean formulae) solvers inspired us to reduce these refinement problems to problems solvable by a QBF solver. By the complexity results it is possible to reduce the modal refinement over PMTS to a Π_4^p query and over BMTS to a Π_2^p query. However, for practical purposes it is better to use Σ_3^p query for the latter case.

Proposition 5.8 *Given BMTS processes s, t, we can construct in polynomial time*

a formula $\Psi_{s,t}$ with no quantifiers such that

$$s \leq_m t \quad \textit{if and only if} \quad \exists R\, \forall T_1\, \exists T_2\, \Psi_{s,t}$$

Proposition 5.9 *Given PMTS processes s, t, we can construct in polynomial time a formula $\Psi_{s,t}$ with no quantifiers such that*

$$s \leq_m t \quad \textit{if and only if} \quad \forall Par_1\, \exists Par_2\, \exists R\, \forall T_1\, \exists T_2\, \Psi_{s,t}$$

We have also performed experiments showing that this solution scales well in the size of the system as well as in the number of parameters, while a direct naive solution is infeasible, given the exponential complexity. We have implemented the reduction and linked it to the QBF solver Quantor. In order to evaluate whether our solution scales, we generate random systems of various sizes (as displayed in tables below in columns). We consider MTS, DMTS, BMTS and PMTS with different numbers of parameters (displayed in parenthesis). The entries in the tables are average running times in seconds. For the details, see Paper F and [KS13c]. In Table 5.6 we work with random systems, the refining system is identical to the abstract system except for a stronger obligation. The interesting part of the results is also depicted in Fig. 5.3.

Table 5.6: Experimental results: systems over alphabet of size 2 with branching degree 2 in the upper part, and systems over alphabet of size 10 with branching degree 10 in the lower part

	25	50	75	100	125	150	175	200
MTS	0.03	0.15	0.29	0.86	0.87	0.96	1.88	2.48
DMTS	0.04	0.22	0.39	0.91	1.13	1.34	2.61	3.19
BMTS	0.03	0.15	0.30	0.62	0.83	0.87	1.61	2.17
PMTS(1)	0.03	0.20	0.37	0.84	0.97	1.23	2.44	3.15
PMTS(5)	0.04	0.22	0.42	0.91	1.26	1.59	2.83	3.66
MTS	0.18	0.84	2.12	3.88	5.63	7.64	10.30	14.18
DMTS	0.44	2.23	5.31	8.59	10.13	14.14	13.96	66.92
BMTS	0.21	1.08	2.65	4.58	6.70	9.63	12.44	17.06
PMTS(1)	0.26	1.12	2.74	4.57	7.58	10.31	11.26	16.41
PMTS(5)	0.25	1.17	2.94	6.36	7.80	10.01	11.90	36.51

In Table 5.7, we first consider the systems as above, i.e. with edges generated randomly so that they create a tree and with some additional "noise" edges thus making the branching degree constant. Second, we consider systems where we have different "clusters", each of which is interconnected with many edges. Each

of these clusters has a couple of "interface" states, which are used to connect to other clusters. We use this class of systems to model system descriptions with more "organic" structure.

Table 5.7: Experimental results: systems over alphabet of size 2 with branching degree 5; systems with random structure in the upper part, and systems with "organic" structure in the lower part

	25	50	75	100	125	150	175	200
BMTS	0.32	1.57	3.46	7.18	10.24	15.18	20.6	27.05
PMTS(1)	0.34	1.57	3.21	8.25	12.46	19.88	24.53	31.01
PMTS(5)	0.33	1.65	4.48	8.21	13.14	21.5	20.55	25.82
BMTS	0.01	0.03	0.18	0.22	0.3	0.48	0.73	1.02
PMTS(1)	0.01	0.07	0.14	0.22	0.43	0.43	0.72	0.83
PMTS(5)	0.01	0.05	0.1	0.17	0.31	0.43	0.88	1.39

On the one hand, observe that the number of parameters does not play any major role in the running time, see the graph in Fig. 5.3. The running times on PMTS with 5 parameters are very close to BMTS, i.e. PMTS with zero parameters, as can be seen in the graph. Therefore, the greatest theoretical complexity threat—the number of parameters allowing in general only for searching all exponentially many combinations—is in practice eliminated by the use of QBF solvers. On the other hand, observe that the running time is more affected by the level of non-determinism. However, the level of non-determinism is often quite low [BKLS09b], hence this dependency does not pose a serious problem in practice. Further, even this most difficult setting with a high level of non-determinism allows for fast analysis if systems with a natural organic structure are considered, cf. upper and lower part of Table 5.7.

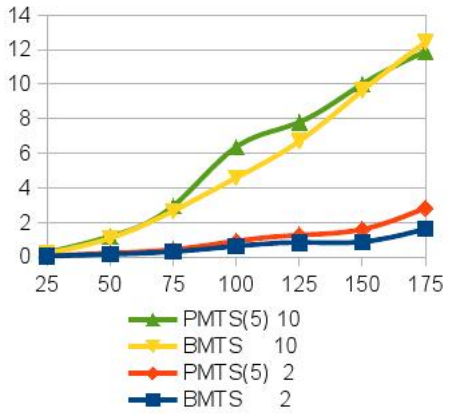


Figure 5.3: Graph of experimental results of Table 5.6 for systems with alphabets of size 10 and 2

Thorough refinement

Furthermore, we extend the decision algorithm for thorough refinement checking over MTS [BKLS12, BKLS12] to the setting of DMTS (Paper B) and of BMTS and PMTS (Paper F). We show how PMTS can be translated to BMTS and BMTS can then be transformed to DMTS. As we also show we can decide the problem on DMTS in EXP, this shows decidability for BMTS and PMTS, but each of the translations is inevitably exponential. However, we show better upper bounds than doubly and triply exponential. To this end, we give also a direct algorithm for showing the problem is in NEXP for BMTS and 2-EXP for PMTS. The results are summarized in Table 5.8.

Since the thorough refinement is very hard, we also extend the method to approximate the thorough refinement by the modal refinement. Firstly, under-approximation is easy, as modal refinement implies thorough refinement.

Proposition 5.10 *Let s and t be PMTS processes. If $s \leq_m t$ then $s \leq_t t$.*

For over-approximation, modal refinement can be used on deterministic BMTS, as both refinements coincide there, but not on PMTS.

Proposition 5.11 *Let s be a PMTS process and t a deterministic BMTS process. If $s \leq_t t$ then $s \leq_m t$.*

Proposition 5.12 *There is a PMTS process s and a deterministic PMTS process t such that $s \leq_t t$ but $s \not\leq_m t$.*

For the general case, we extend our method of the deterministic hull for MTS [BKLS09b] to both BMTS and PMTS. Since for BMTS modal and thorough refinements coincide if the refined system is deterministic, we obtain a method to over-approximate using the deterministic hull. Finally, in the case with PMTS, we need to over-approximate the behaviour dependent on the parameters using *parameter-free hull* $\mathcal{P}$, because the coincidence of the refinements on deterministic systems fails for PMTS.

Proposition 5.13 *Let s be a PMTS process. Then $\mathcal{D}(s)$ is a deterministic PMTS process such that $s \leq_m \mathcal{D}(s)$.*

Proposition 5.14 *Let s be a PMTS state. Then*

- *for every deterministic PMTS state t, if $s \leq_m t$ then $\mathcal{D}(s) \leq_m t$;*
- *for every deterministic BMTS state t, if $s \leq_t t$ then $\mathcal{D}(s) \leq_m t$.*

Proposition 5.15 *Let s and t be PMTS states. If $s \leq_t t$ then $s \leq_m \mathcal{P}(\mathcal{D}(t))$.*

Table 5.8: Complexity of the thorough refinement and the relationship to the modal refinement

	MTS	DMTS	BMTS	PMTS
$\leq_t \in$	EXP	EXP	NEXP	2-EXP
for t deterministic	$\leq_m = \leq_t$	$\leq_m = \leq_t$	$\leq_m = \leq_t$	$\leq_m \neq \leq_t$

Modal refinement of infinite state systems

We now turn our attention to the problems whether modal refinement holds between systems of given classes $m\mathcal{C}_1$ and $m\mathcal{C}_2$. We denote each such problem by $m\mathcal{C}_1 \leq_m m\mathcal{C}_2$ and consider classes of mMPRS as introduced in Chapter 4.

Unfortunately, simulation—and thus also refinement—is undecidable already on BPP [Hüt94] and BPA [GH94]. When considering the case where one of the two classes is FSM, the undecidability holds for PA [KM99]. Thus we are left with the problems mFSM$\leq_m$mPDA, mPDA$\leq_m$mFSM and mFSM$\leq_m$mPN, mPN$\leq_m$mFSM.

On the one hand, we show the former two problems to be decidable using non-modal methods for simulation of [KM02b]. On the other hand, the non-modal methods for simulation of [JM95] cannot be extended to the latter two problems. We show that (surprisingly) they are both undecidable and, moreover, even for mBPP. The results are summarized in Table 5.9.

Moreover, for the decidable case, we obtain the precise complexity using [KM02a].

Proposition 5.16 *The problem mPDA$\leq_m$mFSM is EXP-complete in both ways, even if the mFSM is of a fixed size.*

The problem mBPA$\leq_m$mFSM is EXP-complete in both ways, but if the mFSM is of a fixed size, it is P-complete.

Furthermore, the problem is decidable even for infinite MTS on both sides if we restrict to *visibly* PDA (vPDA) [AM04] and its subclass of visibly BPA (vBPA). The complexity can then be proved using complexity bounds for μ-calculus model checking, as in [Srb06].

Proposition 5.17 *The problem mvPDA$\leq_m$mvPDA is EXP-complete.*

The problem mvBPA$\leq_m$mvBPA is P-complete.

Finally, in the spirit of [AHKV98], we also consider a symmetric version of refinement resulting into a bisimulation notion over MTS.

Definition 5.18 *A* birefinement relation *is a symmetric refinement relation. Given processes s, t, we say that s birefines t, written $s \sim_{\mathrm{m}} t$, if there exists a birefinement relation containing (s, t).*

Table 5.9: Decidability of modal refinement on mPRS

decidable	mFSM $\leq_{\geq}^{\mathrm{m}}$ mPDA, mvPDA $\leq_{\geq}^{\mathrm{m}}$ mvPDA, mFSM $\sim_{\mathrm{m}}$ mPRS
undecidable	mFSM $\leq_{\geq}^{\mathrm{m}}$ mBPP, mBPA $\leq_{\geq}^{\mathrm{m}}$ mBPA

Using [KŘS05] we obtain decidability of birefinement between a finite MTS and (surprisingly) arbitrary mPRS.

5.2.2 Operations

We first show that MTS are not closed under conjunction.

Example 5.19 *In Fig. 5.4 one can see two MTS with two incomparable maximal MTS solutions for conjunction. However, there is a unique greatest DMTS solution. This gives another justification for using DMTS instead of MTS.*

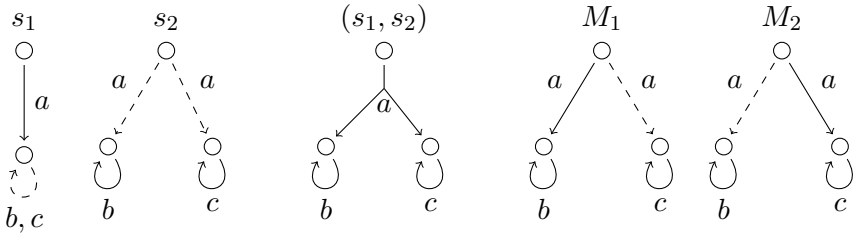


Figure 5.4: MTS processes s_1, s_2, their greatest lower bound (s_1, s_2), and their two maximal MTS lower bounds M_1, M_2

Nevertheless, we show that DMTS with one or more initial states, and thus also BMTS and PMTS are closed under conjunction. The result of the construction is based on the synchronous product. Thus it is a system over tuples of processes where the length of the tuple is the number of input systems. This means that the conjunction (and thus also a common implementation) can be constructed in polynomial time, if n is fixed; and in exponential, if n is a part of the input. This also yields the respective complexity bounds for the problem of deciding whether the input systems have a common implementation in these two settings. Further, if the MTS processes are deterministic, then the greatest lower bound is—as our algorithm computes it—also a deterministic MTS. Moreover, the conjunction is also the greatest lower bound with respect to the thorough refinement.

Theorem 5.20 *Let s_1, s_2, s_3 be BMTS or DMTSm processes. Then $[\![s_1 \wedge s_2]\!] = [\![s_1]\!] \cap [\![s_2]\!]$. Further, $s_1 \leq_m s_2 \wedge s_3$ if and only if $s_1 \leq_m s_2$ and $s_1 \leq_m s_3$.*

The conjunction construction was later extended to systems with different alphabets by [BDCU13].

For disjunction, it is easy to obtain similar results for DMTS with more initial states or BMTS or PMTS.

Theorem 5.21 *Let s_1, s_2, s_3 be BMTS or DMTSm processes. Then $[\![s_1 \vee s_2]\!] = [\![s_1]\!] \cup [\![s_2]\!]$. Further, $s_1 \vee s_2 \leq_m s_3$ iff $s_1 \leq_m s_3$ and $s_2 \leq_m s_3$.*

However, for MTS (deterministic or not) and DMTS with a single initial state this is not possible.

Example 5.22 *Consider the MTS specifications in Fig. 5.5. While the disjunction can be described simply as a BMTS with obligation $\Omega(s_2) = ((a, \bullet) \wedge (b, \bullet)) \vee (\neg(a, \bullet) \wedge \neg(b, \bullet))$, no DMTS can express this.*

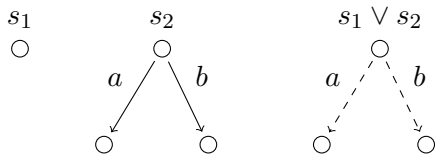


Figure 5.5: MTS processes s_1 and s_2, and their MTS and BMTS least upper bounds (s_1, s_2)

Proposition 5.23 *With operations $\wedge$ and $\vee$, the sets of BMTS (or DMTSm) processes form bounded distributive lattices up to $(\leq_m \cap \geq_m)$-equivalence.*

We also define parallel composition for DMTS and other classes, see Paper G and [BČK10]. Unfortunately, they inherit the incompleteness with respect to modal refinement from MTS, see [HL89, BKLS09b]. Therefore, we can only satisfy one direction in the axiom (parallel), the so-called independent implementability:

Theorem 5.24 *For all BMTS processes s_1, s_2, s_3, s_4, if $s_1 \leq_m s_3$ and $s_2 \leq_m s_4$ then $s_1 \parallel s_2 \leq_m s_3 \parallel s_4$.*

The quotient (of non-deterministic systems) is considerably more complex and the question was open for a long time. We give a construction for BMTS and an exponentially smaller one for MTS.

Theorem 5.25 *For all BMTS processes s, t and x, $x \parallel t \leq_m s$ iff $x \leq_m s/t$.*

We briefly sketch the construction for MTS. Let $(S, \dashrightarrow_S, \longrightarrow_S), (T, \dashrightarrow_T, \longrightarrow_T)$ be non-deterministic MTS and s, t their processes, respectively. We define the quotient $S/T = (Q, \dashrightarrow_Q, \longrightarrow_Q)$ using $Q = \mathcal{P}(S \times T)$ with the initial state $q = \{(s, t)\}$.

For $q = \{s_1/t_1, \ldots, s_n/t_n\} \in Q$ and $a \in \Sigma$, we define the may transitions. We first compute which actions may be present: $\gamma(q) = \bigcap_i \big(\alpha(s_i) \cup (\Sigma \setminus \alpha(t_i))\big)$ where $\alpha(x) = \{a \in \Sigma \mid \exists y, M : (a, y) \in M \in \mathrm{Tran}(x)\}$ is the set of all available actions in x. Now, for every $a \in \gamma(q)$ and for each $i \in \{1, \ldots, n\}$, denote by $May_a(t_i) = \{t_{i,1}, \ldots, t_{i,m_i}\}$ all the may a-successors of t_i, and define all possible transitions from q, where each successor in T is matched with a successor in S:

$$May_a(q) = \big\{\{s_{i,j}/t_{i,j} \mid i \in \{1, \ldots, n\}, j \in \{1, \ldots, m_i\}\} \mid \\ \forall i \in \{1, \ldots, n\} : \forall j \in \{1, \ldots, m_i\} : s_{i,j} \in May_a(s_i)\big\}.$$

For the (disjunctive) must-transitions, we let, for every $s_i \xrightarrow{a} s'$,

$$q \longrightarrow \{(a, M) \in \{a\} \times May_a(q) \mid \exists t' : s'/t' \in M,\ t_i \xrightarrow{a} t'\}.$$

be the matching preserving $s_i \xrightarrow{a} s'$ in the composition of T and the quotient.

Theorem 5.26 *For all MTS processes s, t and x, $x \parallel t \leq_m s$ iff $x \leq_m s/t$.*

Example 5.27 *We illustrate the construction on an example in Fig. 5.6. For details, see Paper G.*

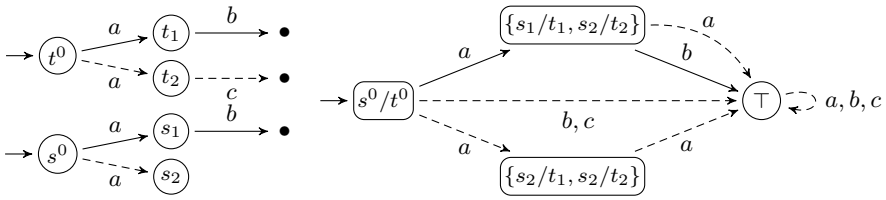


Figure 5.6: Two non-deterministic MTS and their quotient

Proposition 5.28 *With operations $\wedge$, $\vee$, $\parallel$ and $/$, the set of BMTS forms a commutative residuated lattice up to $(\leq_m \cap \geq_m)$-equivalence.*

We sum up the results in the following theorem and Table 5.10.

Table 5.10: Newly established closure properties (marked in red)

	$\wedge$	$\vee$	$\parallel$	$/$
deterministic MTS	✓	×	✓	✓
MTS	×	×	✓	×
DMTS	✓	×	✓	✓
DMTSm/BMTS/PMTS	✓	✓	✓	✓

5.2.3 Model checking

LTL model checking and realizability

Recall that the $\models_\forall$-problem for DMTS can be solved by standard model checking where we consider the LTS that is the "greatest" implementation of the DMTS, i.e. the one with all mays replaced by musts. Thus all possible runs are present in the implementation. The solution works for infinite runs, as well as deadlock-free systems and general systems. Similarly for $\models_\exists$-problem, we can take the minimal implementation of the MTS if deadlocks are allowed and finite runs ignored. However, if this is not the case this solution does not work for the $\models_\exists$-problem. Indeed, there are no minimal implementations; non-trivial decisions have to be made which transitions to implement.

Example 5.29 *An MTS with only one may a-successor and one may b-successor cannot avoid deadlock in a unique way. Moreover, even if deadlocks are allowed, not implementing any choice may result in not satisfying* $\mathbf{Xtt}$.

The solution is to assign the task of decision making to a "player" and another player then chooses which of the implemented transitions is taken. Decisions of the players determine a run. The objective of the game is to satisfy the formula on the run. The first player can always succeed irrespective of what the other player does if and only if there is an implementation satisfying the formula. These LTL games are in general 2-EXP-complete[PR89]. The consequences are summarized in Table 5.11.

Table 5.11: Complexities of generalized LTL model checking (ω denoting finite runs are ignored, df deadlock-free implementations are ignored, ∞ no restriction)

	$\models_\forall$	$\models_\exists$
MTS $\models^\omega$	PSPACE-complete	PSPACE-complete
MTS $\models^{\mathrm{df}}$	PSPACE-complete	2-EXP-complete
MTS $\models^\infty$	PSPACE-complete	2-EXP-complete
DMTS	PSPACE-complete	2-EXP-complete

The best known time complexity bounds with respect to the size of system $|S|$ and the size of LTL formula $|\varphi|$ are the following. In all PSPACE-complete cases the time complexity is $\mathcal{O}(|S|\cdot 2^{|\varphi|})$; in all 2-EXP-complete cases the time complexity is $|S|^{2^{\mathcal{O}(|\varphi|)}} \cdot 2^{2^{\mathcal{O}(|\varphi|\log|\varphi|)}}$. The latter upper bound is achieved by translating the LTL formula into a deterministic Rabin automaton of size $2^{2^{\mathcal{O}(|\varphi|\log|\varphi|)}}$ with $2^{\mathcal{O}(|\varphi|)}$ accepting pairs, thus changing the LTL game into a Rabin game. State of the art algorithm for solving Rabin games can be found e.g. in [PP06].

The high complexity poses a serious problem also in practice. Whenever the Rabin automaton is too large, the game cannot be solved in reasonable time.

The automata are usually large because there is a two-stage translation. First, we turn an LTL formula into a non-deterministic Büchi automaton; secondly, we determinize it using Safra's construction [Saf88] or its modifications [MS95, Pit06, Sch09]. Since Safra's determinization works not only for automata obtained from LTL but for all automata, it is too general and the blow-up is usually large. Therefore, we propose a direct translation from LTL to deterministic Rabin automata, which works much better in practice [KE12]. Moreover, we can use *generalized Rabin automata* instead, which can be used as well and are even smaller for complex formulae [CGK13]. As opposed to our previous work [KE12, GKE12, KLG13, Kře13], in Paper I we provide a translation for the whole LTL and actually even a more expressive logic of μ-calculus (interpreted over linear structures) with "well-nested" fix-points.

LTL model checking can also help us with the problem of incompleteness of the parallel composition. We have seen there are processes s_1, s_2 such that their composition $s_1 \parallel s_2$ has an implementation i that does *not* arise as a composition $i_1 \parallel i_2$ of any two implementations $i_1 \leq_m s_1, i_2 \leq_m s_2$. Completeness can be achieved only under some restrictive conditions [BKLS09b]. Here we show that composition is sound and complete with respect to every logic of linear time, i.e. it preserves and reflects all linear time properties. The completeness of composition with respect to linear time logics holds for all discussed cases: both for MTS and DMTS, both for infinite and all runs, and both universally and existentially. We do not define linear properties formally here, see e.g. [BK08]. As a special case, one may consider LTL formulae.

Proposition 5.30 *Let s_1 and s_2 be DMTS processes, φ a linear time property, and $\star \in \{\omega, \infty\}$. Then $s_1 \parallel s_2 \models^\star_\forall \varphi$ if and only if $i_1 \parallel i_2 \models^\star \varphi$ for all implementations $i_1 \leq_m s_1$ and $i_2 \leq_m s_2$.*

Proposition 5.31 *Let s_1 and s_2 be DMTS processes, φ a linear time property, and $\star \in \{\omega, \infty\}$. Then $s_1 \parallel s_2 \models^\star_\exists \varphi$ if and only if $i_1 \parallel i_2 \models^\star \varphi$ for some implementations $i_1 \leq_m s_1$ and $i_2 \leq_m s_2$.*

Thus $\parallel$ is "LTL complete", i.e. preserves and reflects all LTL properties. Therefore, the only spurious implementations are sums of legal implementations.

The same approach of reduction to an LTL game has later been used [DBPU12] to solve a very similar problem of deciding whether all/some implementation can be pruned to satisfy a given LTL formula.

Cheapest implementation

Now let us consider the problem of finding an implementation of MTSD, so that we spend the least possible amount of money (e.g. the pay to the driver) per time unit while conforming to the specification and the hardware budget. We first formally define the price scheme.

Definition 5.32 (Dual-Price Scheme) *A* dual-price scheme *over an alphabet* Σ *is a tuple* $\mathcal{P} = (r, H, \Psi, h)$ *where*

- $r : \Sigma \to \mathbb{Z}$ *is a* running cost *function of actions per time unit,*
- H *is a finite set of available* hardware*,*
- $\Psi : \Sigma \to \mathcal{B}(H)$ *is a* hardware requirement *function, and*
- $h : H \to \mathbb{N}_0$ *is a* hardware investment cost *function.*

Example 5.33 *Consider Example 4.6 of the bus driver. Each action is assigned a* running cost *per time unit, e.g.* drive *costs 10 each time unit it is being performed. In addition, in order to perform an action, some* hardware *may be needed, e.g.* vacuum_cleaner *for the* big_cleanup *costing 100. This* investment cost *is paid only once. The price scheme is illustrated in Fig. 5.7. Notice that hardware requirements are specified as a Boolean combination of hardware components. This allows for much more variability than a possible alternative of a simple investment cost* $\Sigma \to \mathbb{N}_0$.

$a \in \Sigma$	$r(a)$
wait	8
drive	10
small_cleanup	6
big_cleanup	7
skip_cleanup	0

$$H = \{\mathsf{vacuum\_cleaner}, \mathsf{sponge}\}$$

$$\Psi(a) = \begin{cases} \mathsf{vacuum\_cleaner} & \text{if } a = \mathsf{big\_cleanup} \\ \mathsf{sponge} \vee \mathsf{vacuum\_cleaner} & \text{if } a = \mathsf{small\_cleanup} \\ \mathbf{tt} & \text{otherwise} \end{cases}$$

$\eta \in H$	$h(\eta)$
vacuum_cleaner	100
sponge	5

Figure 5.7: Example of a dual price scheme

A set $G \subseteq H$ of hardware is *sufficient* for an MTSD implementation i_0, written $G \models i_0$, if $G \models a$ for every action a reachable from i_0. The *investment cost* of i_0 is then defined as

$$\mathsf{ic}(i_0) = \min_{G \models i_0} \sum_{g \in G} h(g) \ .$$

Further, a *run* from i_0 is an infinite sequence $i_0 a_0 t_0 i_1 a_1 t_1 \cdots$ with $(i_k, a_k, i_{k+1}) \in T$ and $t_k \in D(i_k, a_k, i_{k+1})$. Hence, in such a run, a concrete time duration in each uncontrollable interval is selected. We denote the set of all runs from i_0 by $\mathcal{R}(i_0)$. The *running cost* of an implementation i_0 is the worst-case long-run average

$$\mathsf{rc}(i_0) = \sup_{i_0 a_0 t_0 i_1 a_1 t_1 \cdots \in \mathcal{R}(s_0)} \limsup_{n \to \infty} \frac{\sum_{k=0}^{n} r(a_k) \cdot t_k}{\sum_{k=0}^{n} t_k} .$$

Our *cheapest-implementation problem* is now defined as follows: given an MTSD specification process s together with a dual-price scheme over the same alphabet, and given an upper-bound max_{ic} for the investment cost, find an implementation i of s such that $\mathsf{ic}(i) \leq max_{\mathsf{ic}}$ and for every implementation i' of s with $\mathsf{ic}(i') \leq max_{\mathsf{ic}}$, we have $\mathsf{rc}(i) \leq \mathsf{rc}(i')$.

Further, we introduce the respective decision problem, the *implementation problem*, as follows: given an MTSD specification process s together with a dual-price scheme over the same alphabet, and given an upper-bound max_{ic} for the investment cost and an upper bound max_{rc} on the running cost, decide whether there is an implementation i of s such that both $\mathsf{ic}(i) \leq max_{\mathsf{ic}}$ and $\mathsf{rc}(i) \leq max_{\mathsf{rc}}$.

Example 5.34 *Consider the dual-price scheme from the previous example and a specification process s of Example 4.6. On the one hand, for maximum investment cost at least 100, the optimal running cost of the implementation of s is $(1 \cdot 8 + 10 \cdot 10 + 6 \cdot 6 + 1 \cdot 8 + 10 \cdot 10 + 30 \cdot 7)/(1 + 10 + 6 + 1 + 10 + 30) \approx 7.97$ where we instruct the driver to do the* big_cleanup *and to* wait *as short as possible. On the other hand, if the maximum investment cost is 99 or less the optimal implementation has running cost $(5 \cdot 8 + 10 \cdot 10 + 6 \cdot 5)/(5 + 10 + 6) \approx 8.10$ where we instruct the driver to do the* small_cleanup *and to* wait *as long as possible. For details, see Paper D.*

In general, the problem is quite hard.

Theorem 5.35 *The implementation problem is NP-complete.*

We can obtain a more efficient algorithm for the implementation problem if we restrict the hardware requirement and obligation functions.

Theorem 5.36 *The implementation problem with positive obligation function and a constant number of hardware components is polynomially equivalent to mean-payoff games and thus it is in NP $\cap$ coNP and solvable in pseudo-polynomial time.*

5.2.4 Tool $\overrightarrow{\text{MoTraS}}$

In Paper H, we provide a tool $\overrightarrow{\text{MoTraS}}$ for design and analysis of MTS and its extensions. It comes not only with a graphical user interface, but as opposed to other mentioned tools also with a command line interface, which allows for batch processing. The Netbeans-based GUI offers all the standard components such as a canvas for drawing systems, windows for editing their properties, algorithms menu, possibility to view more systems at once etc. Both the GUI and the independent algorithms package, which contains all data-structures, algorithms and the CLI, are written in Java. In order to make the tool easily extensible, we introduced a file format *xmts*, which facilitates textual representation of different extensions of modal transition systems. The description of the format can be found on the web page of the tool [MTS].

As to the available algorithms, MoTraS supports all operations required for complete specification theories [BDH+12] and more. This includes modal refinement checking, parallel composition (for quotient see below), conjunction (or merge) and the related consistency checking and maximal implementation generation, deterministic hull and generalized LTL model checking. This functionality comes for MTS as well as more general DMTS and in all cases also *non-deterministic* systems are supported; in particular, the algorithm for conjunction is now considerably more complex.

In contrast, `MTSA` supports only modal refinement, parallel composition and consistency using the cloning operation, which may not terminate. It also offers a model checking procedure, which is, unfortunately, fundamentally flawed, see Section 5.1.3. This was shown in Paper B from where we adopt the corrected implementation. `MIO` offers modal refinement, the MIOA parallel composition, and conjunction for deterministic systems. On the top, it also offers quotient for deterministic systems. Note that both `MTSA` and `MIO` can only handle modal systems, not their disjunctive extension. MoTraS supports DMTS, which have more expressive power. In contrast to (non-deterministic) MTS, DMTS are rich enough to express solutions to process equations [LX90] (hence a specification of a missing component in a system can be computed) and are closed under all operations, particularly conjunction (which is necessary for merging viewpoints on a system).

Further, on the top of this functionality for MTS and DMTS, we also provide an implementation of our new method for modal refinement checking of BMTS and PMTS. While modal refinement on MTS and DMTS can be decided in polynomial time, on BMTS and PMTS it is higher in the polynomial hierarchy, namely Π_2 and Π_4, respectively. The new method, however, reduces the refinement problem to a problem directly and efficiently solvable by a QBF solver. The experimental results of Section 5.2.1 show that this solution scales well in the size of the system as well as in the number of parameters, while a direct naive solution is infeasible.

Moreover, we also implement the deterministic hull and the parameter-free

hull for BMTS and PMTS, which enables us to both over- and under-approximate the very hard thorough refinement using the fast modal refinement, now even for the most general class of PMTS.

Table 5.12 summarizes the functionality: ✓indicates a MoTraS implementation; for the other tools, the name indicates an implementation; "det." denotes a functionality limited to deterministic systems.

Table 5.12: Functionality of the available tools

Operation	MTS			DMTS	BMTS	PMTS
Parallel composition	`MTSA`	`MIO`(MIAO)	✓	✓		
Consistency	`MTSA`(of 2 systems)	`MIO`(det.)	✓	✓		
Conjunction		`MIO`(det.)	✓	✓		
Quotient (det.)		`MIO`	✓	×		
Generalized LTL	`MTSA`(incorrect)		✓	✓		
Det./Par. hull			✓	✓	✓	✓
Refinement	`MTSA`	`MIO`	✓	✓	✓	✓

Furthermore, as a result of a Bachelor's thesis [Man13], MoTraS has been extended with MTSD and solving the cheapest implementation problem recently.

$\longrightarrow\!\!\!\!\Rightarrow\!\!\!\!\dashrightarrow$MoTraS is open source and the sources as well as examples, user's guide etc. are accessible at `http://www.model.in.tum.de/~kretinsk/motras.html`

Chapter 6

Summary of the results and future work

The thesis contributes to the area of specification and verification of component-based software using modal transition systems. Firstly, in order to extend modelling abilities of the MTS formalism we have extended MTS with the following features:

- the obligation function to express practically useful more involved modalities (combined, exclusive, persistent or conditional choices) obtaining OTS, BMTS and PMTS;
- time durations and a price scheme to model running cost and hardware investment cost of an implementation;
- an infinite state space generated by finitely many rules for sequential and parallel compositions to model systems with unbounded memory, synchronization or dynamic threads/process creation.

We have compared the resulting formalisms and established relationships among them. This has led to identifying a robust class of DMTS with more initial states. Moreover, we have shown its equivalence to the modal ν-calculus. This unifies the behavioural and logical approach to specification and verification and enables us to mix the two.

Secondly, in order to make the extensions applicable, we have developed algorithms and provided complexities for the following problems:

- operations required for complete specification theories, most interestingly lowering the complexity of conjunction form exponential to polynomial and the first solution to the quotient of non-deterministic MTS;
- modal and thorough refinements over the extensions necessary for stepwise design process;
- the cheapest implementation using mean-payoff games, useful in the embedded design;
- LTL generalized model checking using LTL games and deterministic ω-automata.

Thirdly, apart from theoretical contributions, we have also focused on practical efficiency of the proposed solutions:

- computing modal refinements with higher complexity using QBF solvers;
- approximating thorough refinement with yet higher complexity using the modal refinement and hulls;
- decreasing the size of the LTL games used for model checking MTS using a new translation of LTL to deterministic ω-automata.

Each of the proposed methods leads to speed-ups in orders of magnitude compared to standard methods.

Finally, we have provided a tool called $\overrightarrow{\text{MoTraS}}$ supporting most of the discussed functionality also for non-deterministic systems.

As for future work, one may consider the following directions. Firstly, there are several purely *theoretical questions* left open, such as logical characterization of refinements or a complete syntactic criterion for consistent mixed transition systems. Further, although the complexity of many problems has been established, there are still several complexity gaps left open, for instance, some cases of the refinements, the quotient construction (we conjecture the exponential blow-up is in general unavoidable), or conditions on decidability of refinement over infinite systems with parallelism, e.g. determinism as in [BKLS09b, EBHH10].

Secondly, one may *further extend* the MTS with additional features, e.g. input, output and internal actions as it is usual in interface theories [dAH01, CdAHS03] similarly to [RBB+09a, BMSH10, BHW10, BHB10, LV12], or include even more time features, such as clocks in priced timed automata [BLR04, BBL08] similarly to [BLPR12, FL12]. One may also extend the model checking algorithm to more complex settings such as the cheapest implementation with an additional requirement that the partial sums stay within given bounds as done in [BFL+08], or cheapest implementation satisfying a temporal property as suggested in [CdAHS03, CD10], model checking metric temporal logic (LTL with time durations) [Koy90], model checking infinite-state MTS similarly to PDA in [Wal96], or cheapest implementation of mPDA using methods like [CV12].

Thirdly, on the *practical* side, $\overrightarrow{\text{MoTraS}}$ only offers a limited support for BMTS and PMTS and the non-deterministic quotient is not implemented at all. Refinement algorithms are implemented using fixed-point iteration and waiting-queue skeleton classes, which allows for an easy introduction of multi-threading to all algorithms with conjectured speed up factor close to the number of cores used. Moreover, one could use a combined modal refinement checker, which uses the standard modal refinement checker to prune the initial relation before the QBF-based checker is called. Further, model checking could be speeded up by inte-

grating Rabinizer 3, which we currently develop. Finally, the cheapest implementation problem has been implemented in a Bachelor's thesis [Man13] under the author's supervision and is readily incorporated in an unreleased version of the tool.

6.1 Summary of the papers

In Appendix, we present the following papers:

A *Process algebra for modal transition systemses.* (MEMICS 2010)

We introduce the obligation function, OTS and a process algebra and examine the expressivity of the previously studied subclasses of OTS.

B *Modal transition systems: Composition and LTL model checking.* (ATVA 2011)

We provide a polynomial algorithm for conjunction and solve LTL generalized model checking over (D)MTS using LTL games.

C *Parametric modal transition systems.* (ATVA 2011)

We introduce BMTS and PMTS and study the complexity of the modal refinement problems.

D *Dual-priced modal transition systems with time durations.* (LPAR 2012)

We introduce MTSD and price schemes and solve the cheapest implementation problem using mean-payoff games.

E *Modal process rewrite systems.* (ICTAC 2012)

We introduce mPRS and study decidability and complexity of the induced modal refinement problems.

F *On refinements of Boolean and parametric modal transition systems.* (ICTAC 2013)

We study the complexity of thorough refinement on BMTS and PMTS, provide its approximation using modal refinement, which we further reduce to QBF solving. We also show BMTS to be as expressive as PMTS.

G *Hennessy-Milner logic with greatest fixed points as a complete behavioural specification theory.* (CONCUR 2013)

We show DMTS with more initial states, BMTS, and ν-calculus to be equivalent, establish them as complete specification theories and provide the first non-deterministic quotient construction.

H *MoTraS: A tool for modal transition systems and their extensions.* (ATVA 2013)

This tool paper presents $\overrightarrow{\text{MoTraS}}$ and compares it to the existing tools for MTS.

I *From LTL to deterministic automata: A Safraless compositional approach.* (CAV 2014)

We provide an efficient translation of LTL into generalized Rabin automata yielding a basis for faster model checking MTS.

In Appendix, each paper is summarized and the author's contribution is listed. The percentage indicating the author's contribution has been approved by the respective co-authors.

Bibliography

[AFdFE+11] Luca Aceto, Ignacio Fábregas, David de Frutos-Escrig, Anna Ingólfsdóttir, and Miguel Palomino. Relating modal refinements, covariant-contravariant simulations and partial bisimulations. In Farhad Arbab and Marjan Sirjani, editors, *FSEN*, volume 7141 of *Lecture Notes in Computer Science*, pages 268–283. Springer, 2011.

[AHKV98] Rajeev Alur, Thomas A. Henzinger, Orna Kupferman, and Moshe Y. Vardi. Alternating refinement relations. In Davide Sangiorgi and Robert de Simone, editors, *CONCUR*, volume 1466 of *Lecture Notes in Computer Science*, pages 163–178. Springer, 1998.

[AHL+08a] Adam Antonik, Michael Huth, Kim G. Larsen, Ulrik Nyman, and Andrzej Wasowski. 20 years of modal and mixed specifications. *Bulletin of the EATCS*, 95:94–129, 2008.

[AHL+08b] Adam Antonik, Michael Huth, Kim Guldstrand Larsen, Ulrik Nyman, and Andrzej Wasowski. Complexity of decision problems for mixed and modal specifications. In Roberto M. Amadio, editor, *FoSSaCS*, volume 4962 of *Lecture Notes in Computer Science*, pages 112–126. Springer, 2008.

[AHL+09] Adam Antonik, Michael Huth, Kim Guldstrand Larsen, Ulrik Nyman, and Andrzej Wasowski. Exptime-complete decision problems for modal and mixed specifications. *Electr. Notes Theor. Comput. Sci.*, 242(1):19–33, 2009.

[AKRU11] Dalal Alrajeh, Jeff Kramer, Alessandra Russo, and Sebastián Uchitel. An inductive approach for modal transition system refinement. In John P. Gallagher and Michael Gelfond, editors, *ICLP (Technical Communications)*, volume 11 of *LIPIcs*, pages 106–116. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.

[AM04] Rajeev Alur and P. Madhusudan. Visibly pushdown languages. In László Babai, editor, *STOC*, pages 202–211. ACM, 2004.

[ART] ARTIST European Network of Excellence on Embedded Systems Design. `http://www.artist-embedded.org/`.

[BBL08] Patricia Bouyer, Ed Brinksma, and Kim Guldstrand Larsen. Optimal infinite scheduling for multi-priced timed automata. *Formal Methods in System Design*, 32(1):3–23, 2008.

[BČK10] Nikola Beneš, Ivana Černá, and Jan Křetínský. Disjunctive modal transition systems and generalized LTL model checking. Technical report FIMU-RS-2010-12, Faculty of Informatics, Masaryk University, Brno, 2010.

[BČK11] Nikola Beneš, Ivana Černá, and Jan Křetínský. Modal transition systems: Composition and LTL model checking. In Bultan and Hsiung [BH11], pages 228–242.

[BCU06] Greg Brunet, Marsha Chechik, and Sebastián Uchitel. Properties of behavioural model merging. In Misra et al. [MNS06], pages 98–114.

[BDCU13] Shoham Ben-David, Marsha Chechik, and Sebastián Uchitel. Merging partial behaviour models with different vocabularies. In D'Argenio and Melgratti [DM13], pages 91–105.

[BDF+13] Nikola Beneš, Benoît Delahaye, Uli Fahrenberg, Jan Křetínský, and Axel Legay. Hennessy-Milner logic with greatest fixed points as a complete behavioural specification theory. In D'Argenio and Melgratti [DM13], pages 76–90.

[BDH+12] Sebastian S. Bauer, Alexandre David, Rolf Hennicker, Kim Guldstrand Larsen, Axel Legay, Ulrik Nyman, and Andrzej Wasowski. Moving from specifications to contracts in component-based design. In Juan de Lara and Andrea Zisman, editors, *FASE*, volume 7212 of *Lecture Notes in Computer Science*, pages 43–58. Springer, 2012.

[Ben08] Albert Benveniste. Multiple viewpoint contracts and residuation. In *2nd International Workshop on Foundations of Interface Technologies (FIT)*, 2008.

[Ben12] Nikola Beneš. *Disjunctive Modal Transition Systems*. PhD thesis, Masaryk University, 2012.

[BFJ+11] Sebastian S. Bauer, Uli Fahrenberg, Line Juhl, Kim G. Larsen, Axel Legay, and Claus R. Thrane. Quantitative refinement for weighted modal transition systems. In Filip Murlak and Piotr Sankowski, editors, *MFCS*, volume 6907 of *Lecture Notes in Computer Science*, pages 60–71. Springer, 2011.

[BFK+09] Tomáš Brázdil, Vojtěch Forejt, Jan Krčál, Jan Křetínský, and Antonín Kučera. Continuous-time stochastic games with time-bounded

reachability. In Ravi Kannan and K. Narayan Kumar, editors, *FSTTCS*, volume 4 of *LIPIcs*, pages 61–72. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2009.

[BFK+13] Tomáš Brázdil, Vojtech Forejt, Jan Krčál, Jan Křetínský, and Antonín Kucera. Continuous-time stochastic games with time-bounded reachability. *Inf. Comput.*, 224:46–70, 2013.

[BFKK08] Tomáš Brázdil, Vojtěch Forejt, Jan Křetínský, and Antonín Kučera. The satisfiability problem for probabilistic CTL. In *LICS*, pages 391–402. IEEE Computer Society, 2008.

[BFL+08] Patricia Bouyer, Ulrich Fahrenberg, Kim Guldstrand Larsen, Nicolas Markey, and Jirí Srba. Infinite runs in weighted timed automata with energy constraints. In Franck Cassez and Claude Jard, editors, *FORMATS*, volume 5215 of *Lecture Notes in Computer Science*, pages 33–47. Springer, 2008.

[BFLT12] Sebastian S. Bauer, Uli Fahrenberg, Axel Legay, and Claus R. Thrane. General quantitative specification theories with modalities. In Edward A. Hirsch, Juhani Karhumäki, Arto Lepistö, and Michail Prilutskii, editors, *CSR*, volume 7353 of *Lecture Notes in Computer Science*, pages 18–30. Springer, 2012.

[BG99] Glenn Bruns and Patrice Godefroid. Model checking partial state spaces with 3-valued temporal logics. In Nicolas Halbwachs and Doron Peled, editors, *CAV*, volume 1633 of *Lecture Notes in Computer Science*, pages 274–287. Springer, 1999.

[BG00] Glenn Bruns and Patrice Godefroid. Generalized model checking: Reasoning about partial state spaces. In Catuscia Palamidessi, editor, *CONCUR*, volume 1877 of *Lecture Notes in Computer Science*, pages 168–182. Springer, 2000.

[BGS92] José L. Balcázar, Joaquim Gabarró, and Miklos Santha. Deciding bisimilarity is p-complete. *Formal Asp. Comput.*, 4(6A):638–648, 1992.

[BH11] Tevfik Bultan and Pao-Ann Hsiung, editors. *Automated Technology for Verification and Analysis, 9th International Symposium, ATVA 2011, Taipei, Taiwan, October 11-14, 2011. Proceedings*, volume 6996 of *Lecture Notes in Computer Science*. Springer, 2011.

[BHB10] Sebastian S. Bauer, Rolf Hennicker, and Michel Bidoit. A modal interface theory with data constraints. In Jim Davies, Leila Silva, and Adenilso da Silva Simão, editors, *SBMF*, volume 6527 of *Lecture Notes in Computer Science*, pages 80–95. Springer, 2010.

[BHJ10] Sebastian S. Bauer, Rolf Hennicker, and Stephan Janisch. Interface theories for (a)synchronously communicating modal I/O-transition systems. In Axel Legay and Benoît Caillaud, editors, *FIT*, volume 46 of *EPTCS*, pages 1–8, 2010.

[BHK+12] Tomáš Brázdil, Holger Hermanns, Jan Krčál, Jan Křetínský, and Vojtech Řehák. Verification of open interactive Markov chains. In Deepak D'Souza, Telikepalli Kavitha, and Jaikumar Radhakrishnan, editors, *FSTTCS*, volume 18 of *LIPIcs*, pages 474–485. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.

[BHW10] Sebastian S. Bauer, Rolf Hennicker, and Martin Wirsing. Building a modal interface theory for concurrency and data. In Till Mossakowski and Hans-Jörg Kreowski, editors, *WADT*, volume 7137 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2010.

[BJL+12a] Sebastian S. Bauer, Line Juhl, Kim G. Larsen, Axel Legay, and Jirí Srba. Extending modal transition systems with structured labels. *Mathematical Structures in Computer Science*, 22(4):581–617, 2012.

[BJL+12b] Sebastian S. Bauer, Line Juhl, Kim G. Larsen, Jirí Srba, and Axel Legay. A logic for accumulated-weight reasoning on multiweighted modal automata. In Tiziana Margaria, Zongyan Qiu, and Hongli Yang, editors, *TASE*, pages 77–84. IEEE, 2012.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008.

[BK10] Nikola Beneš and Jan Křetínský. Process algebra for modal transition systemses. In Ludek Matyska, Michal Kozubek, Tomas Vojnar, Pavel Zemcik, and David Antos, editors, *MEMICS*, volume 16 of *OASICS*, pages 9–18. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2010.

[BK12] Nikola Beneš and Jan Křetínský. Modal process rewrite systems. In Roychoudhury and D'Souza [RD12], pages 120–135.

[BKK+10] Tomáš Brázdil, Jan Krčál, Jan Křetínský, Antonín Kučera, and Vojtěch řehák. Stochastic real-time games with qualitative timed automata objectives. In Paul Gastin and François Laroussinie, editors, *CONCUR*, volume 6269 of *Lecture Notes in Computer Science*, pages 207–221. Springer, 2010.

[BKK+11] Tomáš Brázdil, Jan Krčál, Jan Křetínský, Antonín Kučera, and Vojtěch řehák. Measuring performance of continuous-time stochas-

tic processes using timed automata. In Marco Caccamo, Emilio Frazzoli, and Radu Grosu, editors, *HSCC*, pages 33–42. ACM, 2011.

[BKK+13] Tomáš Brázdil, Lubos Korenciak, Jan Krčál, Jan Křetínský, and Vojtech Řehák. On time-average limits in deterministic and stochastic Petri nets. In Seetharami Seelam, Petr Tuma, Giuliano Casale, Tony Field, and José Nelson Amaral, editors, *ICPE*, pages 421–422. ACM, 2013.

[BKKŘ11] Tomáš Brázdil, Jan Krčál, Jan Křetínský, and Vojtech Řehák. Fixed-delay events in generalized semi-Markov processes revisited. In Joost-Pieter Katoen and Barbara König, editors, *CONCUR*, volume 6901 of *Lecture Notes in Computer Science*, pages 140–155. Springer, 2011.

[BKL+11] Nikola Beneš, Jan Křetínský, Kim G. Larsen, Mikael H. Moller, and Jiri Srba. Parametric modal transition systems. In Bultan and Hsiung [BH11], pages 275–289.

[BKL+12] Nikola Beneš, Jan Křetínský, Kim Guldstrand Larsen, Mikael H. Moller, and Jiri Srba. Dual-priced modal transition systems with time durations. In Nikolaj Bjørner and Andrei Voronkov, editors, *LPAR*, volume 7180 of *Lecture Notes in Computer Science*, pages 122–137. Springer, 2012.

[BKLS09a] Nikola Beneš, Jan Křetínský, Kim Guldstrand Larsen, and Jiri Srba. Checking thorough refinement on modal transition systems is EXPTIME-complete. In Leucker and Morgan [LM09], pages 112–126.

[BKLS09b] Nikola Beneš, Jan Křetínský, Kim Guldstrand Larsen, and Jiri Srba. On determinism in modal transition systems. *Theor. Comput. Sci.*, 410(41):4026–4043, 2009.

[BKLS12] Nikola Beneš, Jan Křetínský, Kim G. Larsen, and Jiri Srba. Exptime-completeness of thorough refinement on modal transition systems. *Inf. Comput.*, 218:54–68, 2012.

[BL92] Gérard Boudol and Kim Guldstrand Larsen. Graphical versus logical specifications. *Theor. Comput. Sci.*, 106(1):3–20, 1992.

[BLL+14] Sebastian S. Bauer, Kim G. Larsen, Axel Legay, Ulrik Nyman, and Andrzej Wasowski. A modal specification theory for components with data. *Sci. Comput. Program.*, 83:106–128, 2014.

[BLPR09] Nathalie Bertrand, Axel Legay, Sophie Pinchinat, and Jean-Baptiste Raclet. A compositional approach on modal specifications for timed

systems. In Karin Breitman and Ana Cavalcanti, editors, *ICFEM*, volume 5885 of *Lecture Notes in Computer Science*, pages 679–697. Springer, 2009.

[BLPR12] Nathalie Bertrand, Axel Legay, Sophie Pinchinat, and Jean-Baptiste Raclet. Modal event-clock specifications for timed component-based design. *Sci. Comput. Program.*, 77(12):1212–1234, 2012.

[BLR04] Gerd Behrmann, Kim Guldstrand Larsen, and Jacob Illum Rasmussen. Priced timed automata: Algorithms and applications. In Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf, and Willem P. de Roever, editors, *FMCO*, volume 3657 of *Lecture Notes in Computer Science*, pages 162–182. Springer, 2004.

[BLS95] Anders Børjesson, Kim Guldstrand Larsen, and Arne Skou. Generality in design and compositional verification using tav. *Formal Methods in System Design*, 6(3):239–258, 1995.

[BML11] Sebastian S. Bauer, Philip Mayer, and Axel Legay. MIO workbench: A tool for compositional design with modal input/output interfaces. In Bultan and Hsiung [BH11], pages 418–421.

[BMSH10] Sebastian S. Bauer, Philip Mayer, Andreas Schroeder, and Rolf Hennicker. On weak modal compatibility, refinement, and the MIO workbench. In Javier Esparza and Rupak Majumdar, editors, *TACAS*, volume 6015 of *Lecture Notes in Computer Science*, pages 175–189. Springer, 2010.

[BPR09] Nathalie Bertrand, Sophie Pinchinat, and Jean-Baptiste Raclet. Refinement and consistency of timed modal specifications. In Adrian Horia Dediu, Armand-Mihai Ionescu, and Carlos Martín-Vide, editors, *LATA*, volume 5457 of *Lecture Notes in Computer Science*, pages 152–163. Springer, 2009.

[Bru97] Glenn Bruns. An industrial application of modal process logic. *Sci. Comput. Program.*, 29(1-2):3–22, 1997.

[CD10] Krishnendu Chatterjee and Laurent Doyen. Energy parity games. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *ICALP (2)*, volume 6199 of *Lecture Notes in Computer Science*, pages 599–610. Springer, 2010.

[CdAHS03] Arindam Chakrabarti, Luca de Alfaro, Thomas A. Henzinger, and Mariëlle Stoelinga. Resource interfaces. In Rajeev Alur and Insup Lee, editors, *EMSOFT*, volume 2855 of *Lecture Notes in Computer Science*, pages 117–133. Springer, 2003.

[CDEG03] Marsha Chechik, Benet Devereux, Steve M. Easterbrook, and Arie Gurfinkel. Multi-valued symbolic model-checking. *ACM Trans. Softw. Eng. Methodol.*, 12(4):371–408, 2003.

[CDL+10] Benoît Caillaud, Benoît Delahaye, Kim G. Larsen, Axel Legay, Mikkel L. Pedersen, and Andrzej Wasowski. Compositional design methodology with constraint Markov chains. In *QEST*, pages 123–132. IEEE Computer Society, 2010.

[CE81] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In Dexter Kozen, editor, *Logic of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer, 1981.

[CGK13] Krishnendu Chatterjee, Andreas Gaiser, and Jan Křetínský. Automata with generalized Rabin pairs for probabilistic model checking and LTL synthesis. In Natasha Sharygina and Helmut Veith, editors, *CAV*, volume 8044 of *Lecture Notes in Computer Science*, pages 559–575. Springer, 2013.

[CGL93] Karlis Cerans, Jens Chr. Godskesen, and Kim Guldstrand Larsen. Timed modal specification - theory and tools. In Costas Courcoubetis, editor, *CAV*, volume 697 of *Lecture Notes in Computer Science*, pages 253–267. Springer, 1993.

[CGLT09] Alarico Campetelli, Alexander Gruler, Martin Leucker, and Daniel Thoma. *Don't Know* for multi-valued systems. In Zhiming Liu and Anders P. Ravn, editors, *ATVA*, volume 5799 of *Lecture Notes in Computer Science*, pages 289–305. Springer, 2009.

[CR12] Benoît Caillaud and Jean-Baptiste Raclet. Ensuring reachability by design. In Roychoudhury and D'Souza [RD12], pages 213–227.

[CV12] Krishnendu Chatterjee and Yaron Velner. Mean-payoff pushdown games. In *LICS*, pages 195–204. IEEE, 2012.

[dAGJ04] Luca de Alfaro, Patrice Godefroid, and Radha Jagadeesan. Three-valued abstractions of games: Uncertainty, but with precision. In LICS04 [LIC04], pages 170–179.

[dAH01] Luca de Alfaro and Thomas A. Henzinger. Interface automata. In *ESEC / SIGSOFT FSE*, pages 109–120. ACM, 2001.

[DBPU12] Nicolás D'Ippolito, Víctor A. Braberman, Nir Piterman, and Sebastián Uchitel. The modal transition system control problem. In Giannakopoulou and Méry [GM12], pages 155–170.

[DDM10] Philippe Darondeau, Jérémy Dubreil, and Hervé Marchand. Supervisory control for modal specifications of services. In *WODES*, pages 428–435, 2010.

[DFCU08] Nicolás D'Ippolito, Dario Fischbein, Marsha Chechik, and Sebastián Uchitel. MTSA: The modal transition system analyser. In *ASE*, pages 475–476. IEEE, 2008.

[DFFU07] Nicolás D'Ippolito, Dario Fischbein, Howard Foster, and Sebastián Uchitel. MTSA: Eclipse support for modal transition systems construction, analysis and elaboration. In Li-Te Cheng, Alessandro Orso, and Martin P. Robillard, editors, *ETX*, pages 6–10. ACM, 2007.

[DGG97] Dennis Dams, Rob Gerth, and Orna Grumberg. Abstract interpretation of reactive systems. *ACM Trans. Program. Lang. Syst.*, 19(2):253–291, 1997.

[DKL+11] Benoît Delahaye, Joost-Pieter Katoen, Kim G. Larsen, Axel Legay, Mikkel L. Pedersen, Falak Sher, and Andrzej Wasowski. Abstract probabilistic automata. In Ranjit Jhala and David A. Schmidt, editors, *VMCAI*, volume 6538 of *Lecture Notes in Computer Science*, pages 324–339. Springer, 2011.

[DLL+10] Alexandre David, Kim Guldstrand Larsen, Axel Legay, Ulrik Nyman, and Andrzej Wasowski. Ecdar: An environment for compositional design and analysis of real time systems. In Ahmed Bouajjani and Wei-Ngan Chin, editors, *ATVA*, volume 6252 of *Lecture Notes in Computer Science*, pages 365–370. Springer, 2010.

[DM13] Pedro R. D'Argenio and Hernán C. Melgratti, editors. *CONCUR 2013 - Concurrency Theory - 24th International Conference, CONCUR 2013, Buenos Aires, Argentina, August 27-30, 2013. Proceedings*, volume 8052 of *Lecture Notes in Computer Science*. Springer, 2013.

[DN04] Dennis Dams and Kedar S. Namjoshi. The existence of finite abstractions for branching time model checking. In LICS04 [LIC04], pages 335–344.

[EBHH10] Dorsaf Elhog-Benzina, Serge Haddad, and Rolf Hennicker. Process refinement and asynchronous composition with modalities. In Susanna Donatelli, Jetty Kleijn, Ricardo Jorge Machado, and João M. Fernandes, editors, *ACSD/Petri Nets Workshops*, volume 827 of *CEUR Workshop Proceedings*, pages 385–401. CEUR-WS.org, 2010.

[EK14] Javier Esparza and Jan Křetínský. From LTL to deterministic automata: A Safraless compositional approach. In *CAV*, Lecture Notes in Computer Science. Springer, 2014. To appear. Technical report accessible at `http://arxiv.org/abs/1402.3388`.

[FBU09] Dario Fischbein, Víctor A. Braberman, and Sebastián Uchitel. A sound observational semantics for modal transition systems. In Leucker and Morgan [LM09], pages 215–230.

[FL12] Uli Fahrenberg and Axel Legay. A robust specification theory for modal event-clock automata. In Sebastian S. Bauer and Jean-Baptiste Raclet, editors, *FIT*, volume 87 of *EPTCS*, pages 5–16, 2012.

[Fok00] Wan Fokkink. *Introduction to process algebra*. Texts in theoretical computer science. Springer, 2000.

[FP07] Guillaume Feuillade and Sophie Pinchinat. Modal specifications for the control theory of discrete event systems. *Discrete Event Dynamic Systems*, 17(2):211–232, 2007.

[FS05] Harald Fecher and Martin Steffen. Characteristic mu-calculus formulas for underspecified transition systems. *Electr. Notes Theor. Comput. Sci.*, 128(2):103–116, 2005.

[FS08] Harald Fecher and Heiko Schmidt. Comparing disjunctive modal transition systems with an one-selecting variant. *J. Log. Algebr. Program.*, 77(1-2):20–39, 2008.

[FU08] Dario Fischbein and Sebastián Uchitel. On correct and complete strong merging of partial behaviour models. In Mary Jean Harrold and Gail C. Murphy, editors, *SIGSOFT FSE*, pages 297–307. ACM, 2008.

[GAW13] Paulo T. Guerra, Aline Andrade, and Renata Wassermann. Toward the revision of CTL models through Kripke modal transition systems. In Juliano Iyoda and Leonardo Mendonça de Moura, editors, *SBMF*, volume 8195 of *Lecture Notes in Computer Science*, pages 115–130. Springer, 2013.

[GC06] Arie Gurfinkel and Marsha Chechik. Why waste a perfectly good abstraction? In Holger Hermanns and Jens Palsberg, editors, *TACAS*, volume 3920 of *Lecture Notes in Computer Science*, pages 212–226. Springer, 2006.

[GH94] Jan Friso Groote and Hans Hüttel. Undecidable equivalences for basic process algebra. *Inf. Comput.*, 115(2):354–371, 1994.

[GHJ01] Patrice Godefroid, Michael Huth, and Radha Jagadeesan. Abstraction-based model checking using modal transition systems. In Kim Guldstrand Larsen and Mogens Nielsen, editors, *CONCUR*, volume 2154 of *Lecture Notes in Computer Science*, pages 426–440. Springer, 2001.

[GJ03] Patrice Godefroid and Radha Jagadeesan. On the expressiveness of 3-valued models. In Lenore D. Zuck, Paul C. Attie, Agostino Cortesi, and Supratik Mukhopadhyay, editors, *VMCAI*, volume 2575 of *Lecture Notes in Computer Science*, pages 206–222. Springer, 2003.

[GKE12] Andreas Gaiser, Jan Křetínský, and Javier Esparza. Rabinizer: Small deterministic automata for LTL(F, G). In Supratik Chakraborty and Madhavan Mukund, editors, *ATVA*, volume 7561 of *Lecture Notes in Computer Science*, pages 72–76. Springer, 2012.

[GM12] Dimitra Giannakopoulou and Dominique Méry, editors. *FM 2012: Formal Methods - 18th International Symposium, Paris, France, August 27-31, 2012. Proceedings*, volume 7436 of *Lecture Notes in Computer Science*. Springer, 2012.

[GNRT10] Patrice Godefroid, Aditya V. Nori, Sriram K. Rajamani, and SaiDeep Tetali. Compositional may-must program analysis: unleashing the power of alternation. In Manuel V. Hermenegildo and Jens Palsberg, editors, *POPL*, pages 43–56. ACM, 2010.

[GP09] Patrice Godefroid and Nir Piterman. LTL generalized model checking revisited. In Jones and Müller-Olm [JMO09], pages 89–104.

[GWC06a] Arie Gurfinkel, Ou Wei, and Marsha Chechik. Systematic construction of abstractions for model-checking. In E. Allen Emerson and Kedar S. Namjoshi, editors, *VMCAI*, volume 3855 of *Lecture Notes in Computer Science*, pages 381–397. Springer, 2006.

[GWC06b] Arie Gurfinkel, Ou Wei, and Marsha Chechik. Yasm: A software model-checker for verification and refutation. In Thomas Ball and Robert B. Jones, editors, *CAV*, volume 4144 of *Lecture Notes in Computer Science*, pages 170–174. Springer, 2006.

[Hen85] Matthew Hennessy. Acceptance trees. *J. ACM*, 32(4):896–928, 1985.

[HH08] Altaf Hussain and Michael Huth. On model checking multiple hybrid views. *Theor. Comput. Sci.*, 404(3):186–201, 2008.

[HJS01] Michael Huth, Radha Jagadeesan, and David A. Schmidt. Modal transition systems: A foundation for three-valued program analysis. In David Sands, editor, *ESOP*, volume 2028 of *Lecture Notes in Computer Science*, pages 155–169. Springer, 2001.

[HKK13] Holger Hermanns, Jan Krčál, and Jan Křetínský. Compositional verification and optimization of interactive Markov chains. In D'Argenio and Melgratti [DM13], pages 364–379.

[HKKG13] Tingting Han, Christian Krause, Marta Z. Kwiatkowska, and Holger Giese. Modal specifications for probabilistic timed systems. In Luca Bortolussi and Herbert Wiklicky, editors, *QAPL*, volume 117 of *EPTCS*, pages 66–80, 2013.

[HL89] Hans Hüttel and Kim Guldstrand Larsen. The use of static constructs in a modal process logic. In Albert R. Meyer and Michael A. Taitslin, editors, *Logic at Botik*, volume 363 of *Lecture Notes in Computer Science*, pages 163–180. Springer, 1989.

[HM80] Matthew Hennessy and Robin Milner. On observing nondeterminism and concurrency. In J. W. de Bakker and Jan van Leeuwen, editors, *ICALP*, volume 85 of *Lecture Notes in Computer Science*, pages 299–309. Springer, 1980.

[HO13] Dang Van Hung and Mizuhito Ogawa, editors. *Automated Technology for Verification and Analysis - 11th International Symposium, ATVA 2013, Hanoi, Vietnam, October 15-18, 2013. Proceedings*, volume 8172 of *Lecture Notes in Computer Science*. Springer, 2013.

[Hol89] Sören Holmström. A refinement calculus for specifications in Hennessy-Milner logic with recursion. *Formal Asp. Comput.*, 1(3):242–272, 1989.

[Hüt94] Hans Hüttel. Undecidable equivalences for basic parallel processes. In Masami Hagiya and John C. Mitchell, editors, *TACS*, volume 789 of *Lecture Notes in Computer Science*, pages 454–464. Springer, 1994.

[Hut99] Michael Huth. A unifying framework for model checking labeled Kripke structures, modal transition systems and interval transition systems. In C. Pandu Rangan, Venkatesh Raman, and Ramaswamy Ramanujam, editors, *FSTTCS*, volume 1738 of *Lecture Notes in Computer Science*, pages 369–380. Springer, 1999.

[Hut02] Michael Huth. Model checking modal transition systems using Kripke structures. In Agostino Cortesi, editor, *VMCAI*, volume 2294

of *Lecture Notes in Computer Science*, pages 302–316. Springer, 2002.

[JL91] Bengt Jonsson and Kim Guldstrand Larsen. Specification and refinement of probabilistic processes. In *LICS*, pages 266–277. IEEE Computer Society, 1991.

[JLS12] Line Juhl, Kim G. Larsen, and Jirí Srba. Modal transition systems with weight intervals. *J. Log. Algebr. Program.*, 81(4):408–421, 2012.

[JM95] Petr Jancar and Faron Moller. Checking regular properties of Petri nets. In Insup Lee and Scott A. Smolka, editors, *CONCUR*, volume 962 of *Lecture Notes in Computer Science*, pages 348–362. Springer, 1995.

[JMO09] Neil D. Jones and Markus Müller-Olm, editors. *Verification, Model Checking, and Abstract Interpretation, 10th International Conference, VMCAI 2009, Savannah, GA, USA, January 18-20, 2009. Proceedings*, volume 5403 of *Lecture Notes in Computer Science*. Springer, 2009.

[KE12] Jan Křetínský and Javier Esparza. Deterministic automata for the (F,G)-fragment of LTL. In P. Madhusudan and Sanjit A. Seshia, editors, *CAV*, volume 7358 of *Lecture Notes in Computer Science*, pages 7–22. Springer, 2012.

[KLG13] Jan Křetínský and Ruslán Ledesma-Garza. Rabinizer 2: Small deterministic automata for LTL\GU. In Hung and Ogawa [HO13], pages 446–450.

[KM99] Antonín Kučera and Richard Mayr. Simulation preorder on simple process algebras. In Jirí Wiedermann, Peter van Emde Boas, and Mogens Nielsen, editors, *ICALP*, volume 1644 of *Lecture Notes in Computer Science*, pages 503–512. Springer, 1999.

[KM02a] Antonín Kučera and Richard Mayr. On the complexity of semantic equivalences for pushdown automata and BPA. In Krzysztof Diks and Wojciech Rytter, editors, *MFCS*, volume 2420 of *Lecture Notes in Computer Science*, pages 433–445. Springer, 2002.

[KM02b] Antonín Kučera and Richard Mayr. Why is simulation harder than bisimulation? In Lubos Brim, Petr Jancar, Mojmír Křetínský, and Antonín Kučera, editors, *CONCUR*, volume 2421 of *Lecture Notes in Computer Science*, pages 594–610. Springer, 2002.

[Koy90] Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.

[Koz83] Dexter Kozen. Results on the propositional mu-calculus. *Theor. Comput. Sci.*, 27:333–354, 1983.

[Kře13] Jan Křetínský. *Verification of Discrete- and Continuous-Time Non-Deterministic Markovian Systems*. PhD thesis, Technische Universität München, 2013.

[KŘS05] Mojmír Křetínský, Vojtech Řehák, and Jan Strejček. Reachability of Hennessy-Milner properties for weakly extended PRS. In Ramaswamy Ramanujam and Sandeep Sen, editors, *FSTTCS*, volume 3821 of *Lecture Notes in Computer Science*, pages 213–224. Springer, 2005.

[KS90] Paris C. Kanellakis and Scott A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Inf. Comput.*, 86(1):43–68, 1990.

[KS13a] Jan Křetínský and Salomon Sickert. MoTraS: A tool for modal transition systems and their extensions. In Hung and Ogawa [HO13], pages 487–491.

[KS13b] Jan Křetínský and Salomon Sickert. On refinements of Boolean and parametric modal transition systems. In Zhiming Liu, Jim Woodcock, and Huibiao Zhu, editors, *ICTAC*, volume 8049 of *Lecture Notes in Computer Science*, pages 213–230. Springer, 2013.

[KS13c] Jan Křetínský and Salomon Sickert. On refinements of Boolean and parametric modal transition systems. Technical Report abs/1304.5278, arXiv.org, 2013.

[Lar89] Kim Guldstrand Larsen. Modal specifications. In Joseph Sifakis, editor, *Automatic Verification Methods for Finite State Systems*, volume 407 of *Lecture Notes in Computer Science*, pages 232–246. Springer, 1989.

[LIC04] *19th IEEE Symposium on Logic in Computer Science (LICS 2004), 14-17 July 2004, Turku, Finland, Proceedings*. IEEE Computer Society, 2004.

[LIC06] *21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12-15 August 2006, Seattle, WA, USA, Proceedings*. IEEE Computer Society, 2006.

[LL12] Kim G. Larsen and Axel Legay. Quantitative modal transition systems. In Narciso Martí-Oliet and Miguel Palomino, editors, *WADT*, volume 7841 of *Lecture Notes in Computer Science*, pages 50–58. Springer, 2012.

[LM09] Martin Leucker and Carroll Morgan, editors. *Theoretical Aspects of Computing - ICTAC 2009, 6th International Colloquium, Kuala Lumpur, Malaysia, August 16-20, 2009. Proceedings*, volume 5684 of *Lecture Notes in Computer Science*. Springer, 2009.

[LNW07a] Kim Guldstrand Larsen, Ulrik Nyman, and Andrzej Wasowski. Modal I/O automata for interface and product line theories. In Rocco De Nicola, editor, *ESOP*, volume 4421 of *Lecture Notes in Computer Science*, pages 64–79. Springer, 2007.

[LNW07b] Kim Guldstrand Larsen, Ulrik Nyman, and Andrzej Wasowski. On modal refinement and consistency. In Luís Caires and Vasco Thudichum Vasconcelos, editors, *CONCUR*, volume 4703 of *Lecture Notes in Computer Science*, pages 105–119. Springer, 2007.

[LSW95] Kim Guldstrand Larsen, Bernhard Steffen, and Carsten Weise. Fischer's protocol revisited: A simple proof using modal constraints. In Rajeev Alur, Thomas A. Henzinger, and Eduardo D. Sontag, editors, *Hybrid Systems*, volume 1066 of *Lecture Notes in Computer Science*, pages 604–615. Springer, 1995.

[LT88] Kim Guldstrand Larsen and Bent Thomsen. A modal process logic. In *LICS*, pages 203–210. IEEE Computer Society, 1988.

[LV12] Gerald Lüttgen and Walter Vogler. Modal interface automata. In Jos C. M. Baeten, Thomas Ball, and Frank S. de Boer, editors, *IFIP TCS*, volume 7604 of *Lecture Notes in Computer Science*, pages 265–279. Springer, 2012.

[LX90] Kim Guldstrand Larsen and Liu Xinxin. Equation solving using modal transition systems. In *LICS*, pages 108–117. IEEE Computer Society, 1990.

[Man13] Alexander Manta. *Implementation of algorithms for modal transition systems with durations*. Bachelor's thesis, Technische Universität München, 2013.

[May00] Richard Mayr. Process rewrite systems. *Inf. Comput.*, 156(1-2):264–286, 2000.

[MNS06] Jayadev Misra, Tobias Nipkow, and Emil Sekerinski, editors. *FM 2006: Formal Methods, 14th International Symposium on Formal Methods, Hamilton, Canada, August 21-27, 2006, Proceedings*, volume 4085 of *Lecture Notes in Computer Science*. Springer, 2006.

[MS95] David E. Muller and Paul E. Schupp. Simulating alternating tree automata by nondeterministic automata: New results and new proofs

of the theorems of rabin, mcnaughton and safra. *Theor. Comput. Sci.*, 141(1&2):69–107, 1995.

[MTS] Motras. Available at `http://www7.in.tum.de/~kretinsk/motras.html`.

[Nam03] Kedar S. Namjoshi. Abstraction for branching time properties. In Warren A. Hunt Jr. and Fabio Somenzi, editors, *CAV*, volume 2725 of *Lecture Notes in Computer Science*, pages 288–300. Springer, 2003.

[NNN08] Sebastian Nanz, Flemming Nielson, and Hanne Riis Nielson. Modal abstractions of concurrent behaviour. In María Alpuente and Germán Vidal, editors, *SAS*, volume 5079 of *Lecture Notes in Computer Science*, pages 159–173. Springer, 2008.

[Nym08] Ulrik Nyman. *Modal Transition Systems as the Basis for Interface Theories and Product Lines*. PhD thesis, Aalborg Universitet, 2008.

[Pit06] Nir Piterman. From nondeterministic Buchi and Streett automata to deterministic parity automata. In LICS06 [LIC06], pages 255–264.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57. IEEE Computer Society, 1977.

[PP06] Nir Piterman and Amir Pnueli. Faster solutions of rabin and streett games. In LICS06 [LIC06], pages 275–284.

[PR89] Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *POPL*, pages 179–190. ACM Press, 1989.

[PT87] Robert Paige and Robert Endre Tarjan. Three partition refinement algorithms. *SIAM J. Comput.*, 16(6):973–989, 1987.

[QS82] Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in cesar. In Mariangiola Dezani-Ciancaglini and Ugo Montanari, editors, *Symposium on Programming*, volume 137 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 1982.

[Rac07] Jean-Baptiste Raclet. *Quotient de spécifications pour la réutilisation de composants*. PhD thesis, Université de Rennes I, 2007. (In French).

[Rac08] Jean-Baptiste Raclet. Residual for component specifications. *Electr. Notes Theor. Comput. Sci.*, 215:93–110, 2008.

[RBB+09a] Jean-Baptiste Raclet, Eric Badouel, Albert Benveniste, Benoît Caillaud, Axel Legay, and Roberto Passerone. Modal interfaces: unifying interface automata and modal specifications. In Samarjit Chakraborty and Nicolas Halbwachs, editors, *EMSOFT*, pages 87–96. ACM, 2009.

[RBB+09b] Jean-Baptiste Raclet, Eric Badouel, Albert Benveniste, Benoît Caillaud, and Roberto Passerone. Why are modalities good for interface theories? In *ACSD*, pages 119–127. IEEE Computer Society, 2009.

[RBB+11] Jean-Baptiste Raclet, Eric Badouel, Albert Benveniste, Benoît Caillaud, Axel Legay, and Roberto Passerone. A modal interface theory for component-based design. *Fundam. Inform.*, 108(1-2):119–149, 2011.

[RD12] Abhik Roychoudhury and Meenakshi D'Souza, editors. *Theoretical Aspects of Computing - ICTAC 2012 - 9th International Colloquium, Bangalore, India, September 24-27, 2012. Proceedings*, volume 7521 of *Lecture Notes in Computer Science*. Springer, 2012.

[Saf88] Shmuel Safra. On the complexity of omega-automata. In *FOCS*, pages 319–327. IEEE Computer Society, 1988.

[Sch09] Sven Schewe. Tighter bounds for the determinisation of Büchi automata. In Luca de Alfaro, editor, *FOSSACS*, volume 5504 of *Lecture Notes in Computer Science*, pages 167–181. Springer, 2009.

[SCU11] Mathieu Sassolas, Marsha Chechik, and Sebastián Uchitel. Exploring inconsistencies between modal transition systems. *Software and System Modeling*, 10(1):117–142, 2011.

[SG04] Sharon Shoham and Orna Grumberg. Monotonic abstraction-refinement for CTL. In Kurt Jensen and Andreas Podelski, editors, *TACAS*, volume 2988 of *Lecture Notes in Computer Science*, pages 546–560. Springer, 2004.

[Sic12] Salomon Sickert. *Refinement algorithms for parametric modal transition systems*. Bachelor's thesis, Technische Universität München, 2012.

[Srb06] Jirí Srba. Visibly pushdown automata: From language equivalence to simulation and bisimulation. In Zoltán Ésik, editor, *CSL*, volume 4207 of *Lecture Notes in Computer Science*, pages 89–103. Springer, 2006.

[SUBK12] German E. Sibay, Sebastián Uchitel, Víctor A. Braberman, and Jeff Kramer. Distribution of modal transition systems. In Giannakopoulou and Méry [GM12], pages 403–417.

[UBC07] Sebastián Uchitel, Greg Brunet, and Marsha Chechik. Behaviour model synthesis from properties and scenarios. In *ICSE*, pages 34–43. IEEE Computer Society, 2007.

[UBC09] Sebastián Uchitel, Greg Brunet, and Marsha Chechik. Synthesis of partial behavior models from properties and scenarios. *IEEE Trans. Software Eng.*, 35(3):384–406, 2009.

[UC04] Sebastián Uchitel and Marsha Chechik. Merging partial behavioural models. In Richard N. Taylor and Matthew B. Dwyer, editors, *SIGSOFT FSE*, pages 43–52. ACM, 2004.

[Wal96] Igor Walukiewicz. Pushdown processes: Games and model checking. In Rajeev Alur and Thomas A. Henzinger, editors, *CAV*, volume 1102 of *Lecture Notes in Computer Science*, pages 62–74. Springer, 1996.

[WGC09] Ou Wei, Arie Gurfinkel, and Marsha Chechik. Mixed transition systems revisited. In Jones and Müller-Olm [JMO09], pages 349–365.

Appendix

Paper A:

Process algebra for modal transition systemses

Nikola Beneš and Jan Křetínský

This paper has been published in Luděk Matyska, Michal Kozubek, Tomáš Vojnar, Pavel Zemčík, and David Antoš, editors, MEMICS, volume 16 of OASICS, pages 9–18. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2010.

Summary

There have been several extensions of MTS, which offer various possibilities to specify modalities in specifications. We extend the framework of MTS and introduce the obligation function as a more general and more succinct way to express modality. Furthermore, it allows for more efficient composition. We then compare the expressiveness of subclasses of this new framework (OTS) that have been studied before (DMTS, MixTS, MTS). We establish several equivalences and several strict inequalities. Namely, we show that the syntactic consistency requirement for DMTS does not decrease the expressiveness except for the empty specifications. In contrast, it matters for MixTS. Further, we show DMTS to be more expressive than MixTS and not less expressive than general OTS. In order to establish the relationships, we introduce a process algebra to describe the systems and, moreover, characterize the examined classes as syntactic subclasses of this algebra.

Author's contribution: 50 %

- participating in the discussions,
- contributing, in particular, to the design of OTS and the process algebra,
- writing Introduction and parts of the paper.

Process Algebra for Modal Transition Systemses*

Nikola Beneš[†1] and Jan Křetínský[‡2]

1 **Faculty of Informatics, Masaryk University**
Botanická 68a, 602 00 Brno, Czech Republic
xbenes3@fi.muni.cz

2 **Institut für Informatik, TU München**
Boltzmannstr. 3, D-85748, Garching, Germany
Faculty of Informatics, Masaryk University
Botanická 68a, 602 00 Brno, Czech Republic
jan.kretinsky@fi.muni.cz

Abstract

The formalism of modal transition systems (MTS) is a well established framework for systems specification as well as abstract interpretation. Nevertheless, due to incapability to capture some useful features, various extensions have been studied, such as e.g. mixed transition systems or disjunctive MTS. Thus a need to compare them has emerged. Therefore, we introduce transition system with obligations as a general model encompassing all the aforementioned models, and equip it with a process algebra description. Using these instruments, we then compare the previously studied subclasses and characterize their relationships.

Keywords and phrases modal transition systems, process algebra, specification

Digital Object Identifier 10.4230/OASIcs.xxx.yyy.p

1 Introduction

Design and verification of parallel systems is a difficult task for several reasons. Firstly, a system usually consists of a number of components working in parallel. Component based design thus receives much attention and *composition* is a crucial element to be supported in every reasonable specification framework for parallel systems. Secondly, the behaviour of the components themselves is not trivial. One thus begins the design process with an underspecified system where some behaviour is already prescribed and some may or may not be present. The specification is then successively refined until a real implementation is obtained, where all details of the behaviour are settled. Therefore, a need for support of *stepwise refinement* design arises. This is indispensable, either due to incapability of capturing all the required behaviour in the early design phase, or due to leaving a bunch of possibilities for the implementations, such as in e.g. product lines [6]. Modal transition systems is a framework supporting both these fundamental features.

Modal transition systems (MTS) is a specification formalism introduced by Larsen and Thomsen [7, 1] allowing both for stepwise refinement design of systems and their composition. A considerable attention has been recently paid to MTS due to many applications, e.g. component-based software development [9], interface theories [10], or modal abstractions and program analysis [5], to name just a few.

* The word "Systemses" in the title is deliberate. Modal transition systems is a formalism. We consider several formalisms based on modal transition systems here.

† The author has been supported by Czech Grant Agency grant no. GD102/09/H042.
‡ The author is a holder of Brno PhD Talent Financial Aid.

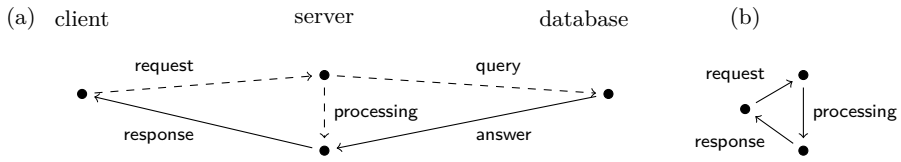


Figure 1 An example of (a) a modal transition system (b) its implementation

The MTS formalism is based on transparent and simple to understand model of *labelled transition systems* (LTS). While LTS has only one labelled transition relation between the states determining the behaviour of the system, MTS as a specification formalism is equipped with two types of transitions: the *must* transitions capture the required behaviour, which is present in all its implementations; the *may* transitions capture the allowed behaviour, which need not be present in all implementations. Such a system can be refined in two ways: a may transition is either implemented (and becomes a must transition) or omitted (and disappears as a transition). Figure 1 depicts an MTS that has arisen as a composition of three systems and specifies the following. A request from a client may arrive. Then we can process it directly or make a query to a database where we are guaranteed an answer. In both cases we send a response. On the right there is an implementation of the system where the processing branch is implemented and the database query branch is omitted. Note that in this formalism we can easily compose implementations as well as specifications.

While specifying may transitions brings guarantees on safety, liveness can be guaranteed to some extent using must transitions. Nevertheless, at an early stage of design we may not know which of several possible different ways to implement a particular functionality will later be chosen, although we know at least one of them has to be present. We want to specify e.g. that either processing or query will be implemented, otherwise we have no guarantee on receiving response eventually. Therefore, several formalisms extending MTS have been introduced. *Disjunctive modal transition systems* (DMTS) do not enforce a particular transition, but specify a whole set of transitions at least one of which must be present. (In our example, it would be the set consisting of processing and query transitions.) DMTS have been introduced in several flavours [8, 4, 2]. Another extension guaranteeing more structured requirements on the behaviour are *mixed transition systems* (MixTS) [3]. Here the required behaviour is not automatically allowed (not all must transitions are necessarily also may transitions) and it must be realized using other allowed behaviour. This corresponds to the situation where a new requirement can be implemented using some reused components. Moreover, it allows for some liveness properties as well. All in all, a need for more structured requirements has emerged. Therefore, we want to compare these formalisms and their expressive power.

We introduce *transition system with obligations* (OTS), a framework that encompasses all the aforementioned systems. Further, we introduce a new process algebra, since there was none for any of the discussed classes of systems. The algebra comes with the respective structural operational semantics, and thus enriches the ways to reason about all these systems. More importantly it allows us to obtain their alternative characterization and provide a more compact description language for them. Altogether, these two new tools allow us to compare all the variants of MTS and we indeed show interesting relationships among the discussed systems. We characterize the process algebra fragments corresponding to the various subclasses of OTS, such as MTS, MixTS or variants of DMTS. Since bisimulation is a congruence w.r.t. all operators of the algebra, this allows for modular analysis of the systems and also for practical optimizations based on minimization by bisimulation

quotienting. Finally, since OTS allow to specify requirements in quite a general form, we can perform some important optimizations in the composition of systems. E.g., when composing DMTS we can avoid an additional exponential blowup that was unavoidable so far.

2 Preliminaries

In order to define the framework we will work in, we need a tool to handle complex requirements imposed on the systems. For this we use positive boolean formulae.

▶ **Definition 1.** A *positive boolean formula* over set X of atomic propositions is given by the following syntax:

$$\varphi ::= x \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathbf{tt} \mid \mathbf{ff}$$

where x ranges over X. The set of all positive boolean formulae over X is denoted as $\mathcal{B}^+(X)$. The semantics $[\![\varphi]\!]$ of a positive boolean formula φ is a set of subsets of X satisfying φ. It is inductively defined as follows:

$$[\![x]\!] = \{Y \subseteq X \mid x \in Y\} \quad [\![\varphi \wedge \psi]\!] = [\![\varphi]\!] \cap [\![\psi]\!] \quad [\![\mathbf{tt}]\!] = 2^X \quad [\![\mathbf{ff}]\!] = \emptyset \quad [\![\varphi \vee \psi]\!] = [\![\varphi]\!] \cup [\![\psi]\!]$$

Every positive boolean formula can be uniquely represented in conjunctive normal form (CNF). It can also be uniquely represented in disjunctive normal form (DNF). In the disjunctive normal form of φ, the disjuncts are precisely the minimal elements of $[\![\varphi]\!]$ (with set inclusion). The formulae $\mathbf{tt}$ and $\mathbf{ff}$ are never needed as proper subformulae of any other formula.

We now proceed with the definition of the systems that are general enough to capture features of all the systems that we discuss in the paper.

▶ **Definition 2.** A *transition system with obligations* (OTS) over an action alphabet Σ is a triple $(\mathcal{P}, \dashrightarrow, \boldsymbol{\Omega})$, where P is a set of *processes*, $\dashrightarrow \subseteq P \times \Sigma \times P$ is the *may* transition relation and $\boldsymbol{\Omega} \colon P \to \mathcal{B}^+(\Sigma \times P)$ is the set of *obligations*.

For simplicity we also require the systems to be finitely branching, i.e. for every $P \in \mathcal{P}$ there are only finitely many $P' \in \mathcal{P}$ with $(P, a, P') \in \dashrightarrow$ for some a. Nevertheless, we could easily drop this assumption if we allowed conjunctions and disjunctions of infinite arities.

Various subclasses of OTS have been studied. We list the most important ones and depict their syntactic relationships in Fig. 2.

- A *disjunctive modal transition system* (DMTS) [8] is an OTS where the must obligations are in CNF. An arbitrary OTS can thus be expressed as a DMTS. Indeed, as noted above, any formula can be translated into CNF. However, this can cost an exponential blowup.
- A *mixed transition system* (MixTS) [3] is an OTS where the must obligations are just conjunctions of atomic predicates.

Moreover, we can impose the following *consistency requirement*

$$\boldsymbol{\Omega}(S) \neq \mathbf{ff} \text{ and if } \boldsymbol{\Omega}(S) \text{ contains } (a, T) \text{ then } S \overset{a}{\dashrightarrow} T,$$

which guarantees that all required behaviour is also allowed. This gives rise to the following systems:

- A *consistent DMTS* (cDMTS) [2] is a DMTS satisfying the consistency requirement.
- A *modal transition system* (MTS) [7] is a MixTS satisfying the consistency requirement.

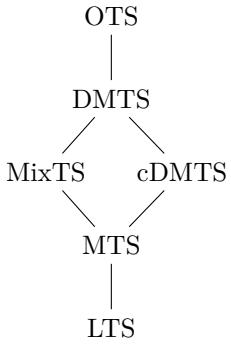


Figure 2 The syntactic hierarchy of MTS extensions

- A *labelled transition system* (LTS) is an MTS such that whenever $S \xrightarrow{a} T$ then $\mathbf{\Omega}(S) = (a, T) \wedge \varphi$ for some φ. Since all behaviour of an LTS is both allowed and required at the same time, we also call LTS an *implementation*.

In order to define the refinement relation on the systems, we need the following auxiliary notion of refinement on formulae motivated by the following example.

▶ **Example 3.** Let us assume formulae $\varphi = (a \wedge b) \vee c$ and $\psi = A \vee C \vee D$. The renaming $R: \ a = A, c = C$ then guarantees that $\varphi \Rightarrow \psi$. This logical refinement (entailment) up to renaming is formalized in the following definition.

▶ **Definition 4.** Let $R \subseteq X \times X$, let $\varphi, \psi \in \mathcal{B}^+(X)$. We write $\varphi \sqsubseteq_R \psi$ to denote

$$\forall M \in [\![\varphi]\!] \ \exists N \in [\![\psi]\!] \ \forall n \in N \ \exists m \in M : (m, n) \in R$$

Note that if we take $R = id$, $\varphi \sqsubseteq_{id} \psi$ if and only if $\varphi \Rightarrow \psi$ (i.e. $[\![\varphi]\!] \subseteq [\![\psi]\!]$). Before proceeding to the fundamental definition of OTS, we prove the following lemmata that will be useful in later proofs. The first lemma is straightforward.

▶ **Lemma 5.** *Let $\varphi \in \mathcal{B}^+(X)$. Then $[\![\varphi]\!]$ is an upwards closed set in $(2^X, \subseteq)$.*

For the two following lemmata, assume this situation: Let X be an arbitrary set and let Y_x be an arbitrary finite set for all $x \in X$. Let $\varphi \in \mathcal{B}^+(X)$ and $\widehat{\varphi}$ be the formula that is created from φ by replacing all occurrences of x by $\bigvee Y_x$ (where $\bigvee \emptyset = \mathbf{ff}$).

▶ **Lemma 6.** *Let $Z \subseteq X$ and let $Z' \subseteq \bigcup_{z \in Z} Y_z$ such that for all $z \in Z$, there is some $y \in Y_z \cap Z'$. Then $Z \in [\![\varphi]\!]$ implies $Z' \in [\![\widehat{\varphi}]\!]$.*

Proof. The proof is done by induction on φ.

- The cases of $\varphi = \mathbf{tt}$ and $\varphi = \mathbf{ff}$ are trivial.
- $\varphi = x$. Then $\widehat{\varphi} = \bigvee Y_x$. $Z \in [\![\varphi]\!]$ implies $x \in Z$ and thus there is some $y \in Y_x \cap Z'$. Therefore $Z' \in [\![\widehat{\varphi}]\!]$ as $[\![\widehat{\varphi}]\!]$ contains $\{y\}$ and it is an upwards closed set.
- $\varphi = \psi \wedge \xi$, then $\widehat{\varphi} = \widehat{\psi} \wedge \widehat{\xi}$. Let $Z \in [\![\varphi]\!] = [\![\psi]\!] \cap [\![\xi]\!]$. Then $Z \in [\![\psi]\!]$ and $Z \in [\![\xi]\!]$. Due to the induction hypothesis, $Z' \in [\![\widehat{\psi}]\!]$ and $Z' \in [\![\widehat{\xi}]\!]$, thus also $Z' \in [\![\widehat{\psi} \wedge \widehat{\xi}]\!] = [\![\widehat{\varphi}]\!]$.
- The case of $\vee$ is similar to the previous case. ◀

▶ **Lemma 7.** *Let $Z' \subseteq \bigcup_x Y_x$ and let $Z = \{x \mid \exists y \in Y_x \cap Z'\}$. Then $Z' \in [\![\widehat{\varphi}]\!]$ implies $Z \in [\![\varphi]\!]$.*

Proof. The proof is done by induction on φ.

- The cases of $\varphi = \mathbf{tt}$ and $\varphi = \mathbf{ff}$ are trivial.
- $\varphi = x$. Then $\widehat{\varphi} = \bigvee Y_x$. As $Z' \in [\![\widehat{\varphi}]\!]$, there has to be some $y \in Y_x \cap Z'$. Thus $x \in Z$, which means that $Z \in [\![\varphi]\!]$.
- The cases of $\wedge$ and $\vee$ are similar to the proof of the previous lemma. ◀

We can now proceed to the fundamental definition of refinement of OTS.

▶ **Definition 8.** Let $(\mathcal{P}_1, \dashrightarrow_1, \mathbf{\Omega}_1)$, $(\mathcal{P}_2, \dashrightarrow_2, \mathbf{\Omega}_2)$ be two OTS and $R \subseteq \mathcal{P}_1 \times \mathcal{P}_2$. We say that R is a *refinement relation*, if $(S,T) \in R$ implies that:

- Whenever $S \overset{a}{\dashrightarrow} S'$ there is $T \overset{a}{\dashrightarrow} T'$ such that $(S',T') \in R$.
- $\mathbf{\Omega}_1(S) \sqsubseteq_{\Sigma R} \mathbf{\Omega}_2(T)$ where $\Sigma R = \{((a,S),(a,T)) \mid a \in \Sigma, (S,T) \in R\}$.

We say that S refines T (denoted as $S \leq T$) if there is a refinement relation R such that $(S,T) \in R$. Further, we say that a process I is an *implementation of a process* S if I is an implementation and $I \leq S$. We denote the set of all implementations of S by $[\![S]\!] = \{I \mid I \leq S, I \text{ is an implementation}\}$.

▶ Remark. Clearly, our definition of refinement coincides with modal refinements on all discussed subclasses of OTS.

One can easily see that every system satisfying the consistency requirement has an implementation, whereas DMTS and MixTS do not necessarily have one. We can compare various flavours of modal transition systems according to expressivity. Due to previous observation, we only consider nonempty sets of implementations.

▶ **Definition 9.** Let $\mathcal{C}, \mathcal{D}$ be subclasses of OTS. We say that $\mathcal{D}$ is at least as expressive as $\mathcal{C}$, written $\mathcal{C} \preceq \mathcal{D}$, if for every $C \in \mathcal{C}$ with $[\![C]\!] \neq \emptyset$ there is $D \in \mathcal{D}$ such that $[\![D]\!] = [\![C]\!]$.We write $\mathcal{C} \equiv \mathcal{D}$ to indicate $\mathcal{C} \preceq \mathcal{D}$ and $\mathcal{C} \succeq \mathcal{D}$, and $\mathcal{C} \prec \mathcal{D}$ to indicate $\mathcal{C} \preceq \mathcal{D}$ and not $\mathcal{C} \equiv \mathcal{D}$.

3 Process Algebra for DMTS

In this section we define a process algebra for OTS. However, since the processes represent sets of implemented systems (i.e. sets of sets of behaviours), we still need the obligation function to fully capture them. For the sake of simplicity, we introduce the parallel composition operator only in the following subsection.

▶ **Definition 10.** Let $\mathcal{X}$ be a set of process names. A *term of process algebra for OTS* is given by the following syntax:

$$P ::= \text{nil} \mid \text{co-nil} \mid a.P \mid X \mid P \wedge P \mid P \vee P \mid \not\downarrow P$$

where X ranges over $\mathcal{X}$ and every $X \in \mathcal{X}$ is assigned a defining equality of the form $X := P$ where P is a term. The semantics is given by the following structural operational semantics rules:

$$\frac{}{a.P \overset{a}{\dashrightarrow} P} \qquad \frac{P \overset{a}{\dashrightarrow} P'}{X \overset{a}{\dashrightarrow} P'} X := P \qquad \frac{P \overset{a}{\dashrightarrow} P'}{P \wedge Q \overset{a}{\dashrightarrow} P'} \qquad \frac{P \overset{a}{\dashrightarrow} P'}{P \vee Q \overset{a}{\dashrightarrow} P'}$$

The obligation function on terms is defined structurally as follows:

$$\begin{array}{lll} \mathbf{\Omega}(\text{nil}) & = \mathbf{tt} & \mathbf{\Omega}(P \wedge Q) = \mathbf{\Omega}(P) \wedge \mathbf{\Omega}(Q) \\ \mathbf{\Omega}(\text{co-nil}) & = \mathbf{ff} & \mathbf{\Omega}(P \vee Q) = \mathbf{\Omega}(P) \vee \mathbf{\Omega}(Q) \\ \mathbf{\Omega}(a.P) & = (a, P) & \mathbf{\Omega}(\not\downarrow P) = \mathbf{\Omega}(P) \\ \mathbf{\Omega}(X) & = \mathbf{\Omega}(P) \quad \text{for } X := P & \end{array}$$

As a convenient shortcut we introduce $?P \equiv (P \vee \text{nil})$ to capture the may transitions, i.e. an allowed behaviour that is not necessarily forced. Hence we easily obtain the following using the rules above:

$$\frac{P \overset{a}{\dashrightarrow} P'}{?P \overset{a}{\dashrightarrow} P'} \qquad \mathbf{\Omega}(?P) = \mathbf{tt}$$

We now obtain the discussed subclasses of OTS as syntactic subclasses generated by the following syntax equations (modulo transformation to CNF):

DMTS	$P ::= \text{nil} \mid a.P \mid X \mid P \wedge P \mid P \vee P \mid \lightning P \mid \text{co-nil}$
cDMTS	$P ::= \text{nil} \mid a.P \mid X \mid P \wedge P \mid P \vee P$
MixTS	$P ::= \text{nil} \mid a.P \mid X \mid P \wedge P \mid P \vee \text{nil} \mid \lightning P \mid \text{co-nil}$
MTS	$P ::= \text{nil} \mid a.P \mid X \mid P \wedge P \mid P \vee \text{nil}$
LTS	$P ::= \text{nil} \mid a.P \mid X \mid P \wedge P$

3.1 Composition

We define the composition operator based on synchronous message passing, as it encompasses the synchronous product as well as interleaving.

▶ **Definition 11.** Let $\Gamma \subseteq \Sigma$ be a *synchronizing alphabet*. For processes S_1 and S_2 we define the process $S_1 \parallel S_2$ as follows.

$$\frac{S_1 \overset{a}{\dashrightarrow} S_1' \quad S_2 \overset{a}{\dashrightarrow} S_2'}{S_1 \parallel S_2 \overset{a}{\dashrightarrow} S_1' \parallel S_2'} \; a \in \Gamma$$

$$\frac{S_1 \overset{a}{\dashrightarrow} S_1'}{S_1 \parallel S_2 \overset{a}{\dashrightarrow} S_1' \parallel S_2} \; a \in \Sigma \setminus \Gamma \qquad \frac{S_2 \overset{a}{\dashrightarrow} S_2'}{S_1 \parallel S_2 \overset{a}{\dashrightarrow} S_1 \parallel S_2'} \; a \in \Sigma \setminus \Gamma$$

As we may assume obligations to be in disjunctive normal form, let us denote $\mathbf{\Omega}(S_1) = \bigvee_i \bigwedge_j (a_{ij}, P_{ij})$ and $\mathbf{\Omega}(S_2) = \bigvee_k \bigwedge_\ell (b_{k\ell}, Q_{k\ell})$. We define $\mathbf{\Omega}(S_1 \parallel S_2)$ by

$$\bigvee_{i,k} \Big(\bigwedge_{j,\ell : a_{ij} = b_{k\ell} \in \Gamma} (a_{ij}, P_{ij} \parallel Q_{kl}) \wedge \bigwedge_{j : a_{ij} \notin \Gamma} (a_{ij}, P_{ij} \parallel S_2) \wedge \bigwedge_{\ell : b_{k\ell} \notin \Gamma} (b_{k\ell}, S_1 \parallel Q_{kl}) \Big)$$

Intuitively, for a process S, the set $[\![\mathbf{\Omega}(S)]\!] \subseteq 2^{\Sigma \times P}$ consists of all possible choices of successors of S that realize all obligations. Composing $[\![\mathbf{\Omega}(S_1)]\!]$ and $[\![\mathbf{\Omega}(S_2)]\!]$ in the same manner as may transitions above generates $[\![\mathbf{\Omega}(S_1 \parallel S_2)]\!]$.

Note that $[\![\mathbf{\Omega}(S)]\!]$ corresponds to DNF of obligations. Nevertheless, they can also be written equivalently in the form of a set of must transitions of DMTS, which corresponds to CNF. During the design process CNF is more convenient to use, whereas the composition has to be done in DNF even for DMTS and then translated back, thus causing an exponential blowup. However, using OTS allows for only one transformation and then the compositions are done using DNF, as the result is again in DNF. As our definition extends the previous definitions on all the discussed models, this shows another use of OTS.

▶ Remark. Refinement is a precongruence with respect to all operators of the process algebra (including the composition operator). Hence, refinemental equivalence, i.e. $\leq \cap \leq^{-1}$, is a congruence.

4 Hierarchy Results

In this section, we study the relationship between the OTS subclasses and establish the following complete result:

$$\text{LTS (implementations)} \prec \text{MTS} \prec \text{MixTS} \prec \text{cDMTS} \equiv \text{DMTS (OTS)}$$

We first show that cDMTS $\equiv$ DMTS. We do that by showing that every OTS process that has an implementation can be substituted by an OTS process that satisfies the consistency requirement and has the same set of implementations. To that end, we use an auxiliary definition of a consistency relation. This definition is a slight modification of the consistency relation defined in [8]. In the definition, the notation $2^{\mathcal{P}}_{\text{Fin}}$ stands for the set of all finite subsets of $\mathcal{P}$.

▶ **Definition 12** (consistency)**.** Let $(\mathcal{P}, \dashrightarrow, \mathbf{\Omega})$ be a OTS. A subset C of $2^{\mathcal{P}}_{\text{Fin}}$ is called a *consistency relation* if for all $\{S_1, \ldots, S_n\} \in C$ and $i \in \{1, \ldots, n\}$ there is $X \in [\![\mathbf{\Omega}(S_i)]\!]$ such that for all $(a, U) \in X$ there are $S_j \overset{a}{\dashrightarrow} T_j$ (for all j) such that $\{U, T_1, \ldots, T_n\} \in C$.

It may be easily seen that an arbitrary union of consistency relations (for given OTS) is also a consistency relation. Therefore, we may talk about the greatest consistency relation. The following lemma explains the motivation behind the consistency relation, i.e. that a set of processes is consistent if it has a common implementation.

▶ **Lemma 13.** *Let $S_1, \ldots, S_n$ be processes. There exists a consistency relation C containing $\{S_1, \ldots, S_n\}$ if and only if $\bigcap_{1 \le i \le n} [\![S_i]\!] \neq \emptyset$.*

Proof. Recall that $\varphi \sqsubseteq_{\Sigma\le} \psi$ if and only if for all $M \in [\![\varphi]\!]$ there is some $N \in [\![\psi]\!]$ such that for all $(a, T) \in N$ there is some $(a, S) \in M$ such that $S \le T$.

We show that $C = \{\{S_1, \ldots, S_k\} \mid k \in \mathbb{N}, \bigcap_i [\![S_i]\!] \neq \emptyset\}$ is a consistency relation. Let $\{S_1, \ldots, S_n\} \in C$, let $I \in \bigcap_i [\![S_i]\!]$ and let $i \in \{1, \ldots, n\}$ be arbitrary. Take $M = \{(a, J) \mid I \overset{a}{\dashrightarrow} J\}$. Clearly, $M \in [\![\mathbf{\Omega}(I)]\!]$ as I is an implementation. Due to $\mathbf{\Omega}(I) \sqsubseteq_{\Sigma\le} \mathbf{\Omega}(S_i)$ there has to be some $N \in [\![\mathbf{\Omega}(S_i)]\!]$ such that for each $(a, U) \in N$ there is $(a, J) \in M$ such that $J \le U$.

Let now $X = N$ and let $(a, U) \in X$. Then $I \overset{a}{\dashrightarrow} J$ with $J \le U$. Therefore, as $I \le S_j$, $S_j \overset{a}{\dashrightarrow} T_j$ and $J \le T_j$ for all j. Thus $J \in [\![U]\!] \cap \bigcap_i [\![T_i]\!]$ and $\{U, T_1, \ldots, T_n\} \in C$.

To show the converse, assume that there is a consistency relation C containing $\{S_1, \ldots, S_n\}$. We know that for all i there is some $X \in \mathbf{\Omega}(S_i)$ such that for all $(a, U) \in X$ there are $S_j \overset{a}{\dashrightarrow} T_j$ (for all j) such that $\{U, T_1, \ldots, T_n\} \in C$. For fixed i, we denote the chosen X as X_i. We construct I coinductively as follows:

$$\mathbf{\Omega}(I) = \bigwedge_i \bigwedge_{(a,U) \in X_i} (a, J_i^U)$$

with $I \dashrightarrow$ transitions to all (a, J_i^U), where J_i^U is a common implementation of U, T_1, …, T_n with T_i given above. Clearly, I is an implementation of all S_i. ◀

We now proceed with the construction of a new consistent OTS that is equivalent to the original OTS.

▶ **Definition 14.** Let $(\mathcal{P}, \dashrightarrow, \mathbf{\Omega})$ be a OTS, Con its greatest consistency relation. We create a new OTS as $(\mathsf{Con}, \dashrightarrow, \mathbf{\Omega})$ where

- $\mathcal{S} \overset{a}{\dashrightarrow} \mathcal{T}$ whenever for all $S \in \mathcal{S}$, $S \overset{a}{\dashrightarrow} T$ with $T \in \mathcal{T}$.

- $\mathbf{\Omega}(\mathcal{S}) = \bigwedge_{S\in\mathcal{S}} \widehat{\mathbf{\Omega}(S)}$ where $\widehat{\mathbf{\Omega}(S)}$ is the formula that is created from $\mathbf{\Omega}(S)$ by replacing all occurrences of (a, U) by $\bigvee\{(a, \{U, T_1, \ldots, T_n\}) \mid \forall i : S_i \xrightarrow{a} T_i, \{U, T_1, \ldots, T_n\} \in \mathsf{Con}\}$ (where $\bigvee \emptyset = \mathbf{ff}$).

Note that due to the properties of Con, $\mathbf{\Omega}(\mathcal{S})$ is never $\mathbf{ff}$. We prove that the construction is correct, i.e. for every consistent process of the original OTS, we have indeed a process of the new OTS with the same set of implementations.

▶ **Theorem 15.** *Let S be a process. Then $[\![S]\!] \neq \emptyset$ if and only if $\{S\} \in \mathsf{Con}$. Moreover, if $\{S\} \in \mathsf{Con}$ then $[\![S]\!] = [\![\{S\}]\!]$.*

Proof. The first part of the theorem is already included in Lemma 13. We thus prove the second part. We first show that $I \in [\![S]\!]$ implies $I \in [\![\{S\}]\!]$. We define R as:

$$R = \{(I, \{S_1, \ldots, S_n\}) \mid n \in \mathbb{N}, \forall i : I \in [\![S_i]\!], \{S_1, \ldots, S_n\} \in \mathsf{Con}\}$$

and prove that R is a refinement relation. Let $(I, \{S_1, \ldots, S_n\}) \in R$.

- Let $I \xrightarrow{a} J$. Then, as $I \leq S_i$, $S_i \xrightarrow{a} T_i$ with $J \leq T_i$ for all i. Thus also $\{S_1, \ldots, S_n\} \xrightarrow{a} \{T_1, \ldots, T_n\}$ and $(J, \{T_1, \ldots, T_n\}) \in R$.
- Let $\mathbf{\Omega}(I) = \varphi$, $\mathbf{\Omega}(\{S_1, \ldots, S_n\}) = \psi$. We need to show that $\varphi \sqsubseteq_{\Sigma R} \psi$.
  Let $M \in [\![\varphi]\!]$. Then, as $I \leq S_i$ for all i, there exist $N_i \in [\![\mathbf{\Omega}(S_i)]\!]$ such that for all $(a, U) \in N_i$ exists $(a, J) \in M$ with $J \leq U$ (due to $\varphi \sqsubseteq_{\Sigma\leq} \mathbf{\Omega}(S_I)$). We use the notation $J_{(a,U)}$ to denote such J.
  Let now $N = \{(a, \{U, T_1, \ldots, T_n\}) \mid \exists i : (a, U) \in N_i, \forall j : S_j \xrightarrow{a} T_j, J_{(a,U)} \leq T_j, \{U, T_1, \ldots, T_n\} \in \mathsf{Con}\}$. Clearly, for all $(a, \{U, T_1, \ldots, T_n\}) \in N$ there is some $(a, J) \in M$ such that $(J, \{U, T_1, \ldots, T_n\}) \in R$ (we take $J = J_{(a,U)}$).
  We need to prove that $N \in [\![\psi]\!]$. In other words, we need to prove that for all i, $N \in [\![\widehat{\mathbf{\Omega}(S_i)}]\!]$. That is, however, a straightforward corollary of Lemma 6 (take $Z = N_i$, $Z' = N$).

We now show that $I \in [\![\{S\}]\!]$ implies $I \in [\![S]\!]$. We define R as:

$$R = \{(I, S) \mid I \leq \mathcal{S} \text{ with } S \in \mathcal{S} \in \mathsf{Con}\}$$

and prove that R is again a refinement relation. Let $(I, S) \in R$ and let $\mathcal{S}$ be such that $I \leq \mathcal{S}$ and $S \in \mathcal{S}$.

- Let $I \xrightarrow{a} J$. Then $\mathcal{S} \xrightarrow{a} \mathcal{T}$ with $J \leq \mathcal{T}$ and thus $S \xrightarrow{a} T$ with $T \in \mathcal{T}$. Thus also $(J, T) \in R$.
- Let $\mathbf{\Omega}(I) = \varphi$, $\mathbf{\Omega}(S) = \psi$. We need to show that $\varphi \sqsubseteq_{\Sigma R} \psi$. Let $M \in [\![\varphi]\!]$. Due to the fact that $\varphi \sqsubseteq_{\Sigma\leq} \mathbf{\Omega}\mathcal{S}$, we know that there exists $N' \in [\![\mathbf{\Omega}(\mathcal{S})]\!]$ such that for all $(a, \{U, T_1, \ldots, T_k\}) \in N'$ there exists $(a, J) \in M$ with $J \leq \{U, T_1, \ldots, T_k\}$. Take $N = \{(a, U) \mid (a, \mathcal{T}) \in N' \text{ with } U \in \mathcal{T}\}$. Clearly, as $N' \in [\![\mathbf{\Omega}(\mathcal{S})]\!]$ also $N' \in [\![\widehat{\mathbf{\Omega}(S)}]\!]$. Using Lemma 7, we get that $N \in [\![\mathbf{\Omega}(S)]\!]$ (take $Z' = N', Z = N$). ◀

The following lemma shows that MixTS $\prec$ cDMTS.

▶ **Lemma 16.** *There is no MixTS M such that $[\![M]\!] = [\![a.\mathrm{nil} \vee b.\mathrm{nil}]\!]$.*

Proof. We first note that any equation defining a MixTS may be written in the following normal form:

$$X := \bigwedge_i ?a_i.S_i \wedge \bigwedge_j \not\downarrow a_j.T_j$$

Clearly, there are three implementations of $a.\text{nil} \vee b.\text{nil}$, namely $a.\text{nil}$, $b.\text{nil}$ and $a.\text{nil} \wedge b.\text{nil}$. Let thus M have these three implementations. Clearly, the $\not\downarrow$ part of M has to be empty (i.e. $\mathbf{\Omega}(M) = \mathbf{tt}$) as M can force neither a transition nor b transition. But then $\text{nil} \in [\![M]\!]$. ◀

Due to Theorem 15, we have a syntactic characterization of consistent OTS. Since we now know MixTS $\prec$ DMTS, a question arises whether such a characterization can be obtained also for consistent MixTS. Observe that the previous construction transforms every MixTS into a consistent OTS with formulae in CNF where all literals in one clause have the same action. One might be tempted to consider the following syntactic characterization of consistent MixTS:

$$P ::= \text{nil} \mid X \mid a.P \mid P \wedge P \mid \bigvee_i a.P_i$$

However, that is not the case, as shown by the following lemma. Hence, this question remains open.

▶ **Lemma 17.** *There is no MixTS M such that $[\![M]\!] = [\![(a.(a.\text{nil} \wedge b.\text{nil}) \vee a.nil) \wedge ?a.a.\text{nil}]\!]$.*

Proof. Let $M = \bigwedge_i ?a.N_i \wedge \bigwedge_j \not\downarrow a.O_j$. (All outgoing transitions from M have to be a-transitions.) We make the following observations:

- For all j, $\mathbf{\Omega}(O_j) = \mathbf{tt}$. Otherwise, $a.\text{nil}$ could not be an implementation of M.
- Also, for all j, $O_j \overset{a}{\dashrightarrow}$. Otherwise, $a.(a.\text{nil} \wedge b.\text{nil})$ could not be an implementation of M.
- There has to be some k such that $a.\text{nil} \in [\![N_k]\!]$, as $a.\text{nil} \wedge a.a.\text{nil}$ also has to be an implementation of M.

We now show that $a.a.\text{nil}$ is an implementation of M. Let R' be an arbitrary refinement relation such that $(a.\text{nil}, N_k) \in R'$ (we know that such R' exists as $a.\text{nil} \leq N_k$). Take R as

$$R = id \cup R' \cup \{(a.a.\text{nil}, M)\} \cup \{(a.\text{nil}, O_j) \mid \forall j\}$$

We now show that R is a refinement.

- $a.a.\text{nil} \overset{a}{\dashrightarrow} a.\text{nil}$ is matched by $M \overset{a}{\dashrightarrow} N_k$.
- $\mathbf{\Omega}(a.a.\text{nil}) = (a, a.\text{nil})$, $\mathbf{\Omega}(M) = \bigwedge_j (a, O_j)$, thus $\mathbf{\Omega}(a.a.\text{nil}) \sqsubseteq_{\Sigma R} \mathbf{\Omega}(M)$.
- $((a.\text{nil}), N_k) \in R'$, therefore the conditions of refinement are satisfied.
- $a.\text{nil} \overset{a}{\dashrightarrow} \text{nil}$ is matched by $O_j \overset{a}{\dashrightarrow} \text{nil}$
- As $\mathbf{\Omega}(O_j) = \mathbf{tt}$, clearly $\mathbf{\Omega}(a.\text{nil}) \sqsubseteq_{\Sigma R} \mathbf{\Omega}(O_j)$.

However, $a.a.\text{nil}$ is not an implementation of $(a.(a.\text{nil} \wedge b.\text{nil}) \vee a.nil) \wedge ?a.a.\text{nil}$. ◀

Finally, we show that MTS $\prec$ MixTS.

▶ **Lemma 18.** *There is no MTS S such that $[\![S]\!] = [\![?a.b.\text{nil} \wedge ?a.c.\text{nil} \wedge \not\downarrow a.(?b.\text{nil} \wedge ?c.\text{nil})]\!]$.*

Proof. Similarly to MixTS, any equation defining a MTS can be written in the following normal form:

$$X = \bigwedge_i ?a_i.S_i \wedge \bigwedge_j a_j.T_j$$

There are three implementations which S has to possess and those are $a.b.\text{nil}$, $a.c.\text{nil}$, and $a.b.\text{nil} \wedge a.c.\text{nil}$ and S cannot possess any other implementation. Clearly, S cannot be of the form $a.T \wedge P$, as then T would have to satisfy $b.\text{nil} \leq T$ (as $a.b.\text{nil} \leq S$), also $c.\text{nil} \leq T$ (as $a.c.\text{nil} \leq S$), yet it cannot satisfy $(b.\text{nil} \wedge c.\text{nil}) \leq T$ (as $a.(b.\text{nil} \wedge c.\text{nil}) \not\leq S$). This is not possible as it can be proven that if $P \leq T$ and $Q \leq T$ then also $P \wedge Q \leq T$ for all OTS. Therefore $S = \bigvee_i ?a.S_i$ and thus $\mathbf{\Omega}(S) = \mathbf{tt}$. But then $\text{nil} \leq S$ and S has more implementations than $?a.b.\text{nil} \wedge ?a.c.\text{nil} \wedge \not\downarrow a.(?b.\text{nil} \wedge ?c.\text{nil})$. ◀

For the sake of completeness, we also state that LTS $\prec$ MTS. This trivially follows, as every LTS only has one implementation, whereas e.g. $?a.\text{nil}$ has two implementations $a.\text{nil}$ and nil.

5 Conclusion and Future Work

We have introduced a new formalism of transition system with obligations together with its process algebra. We have used it to compare various previously studied systems. The main result shows that general DMTS are not more powerful than consistent DMTS, whereas mixed transition systems are strictly less expressive. Furthermore, we have given an alternative syntactic characterizations of the studied systems, although a complete syntactic criterion for consistent mixed transition systems remains as a future work. Surprisingly, using more general OTS leads to some optimizations in computation of the composition that were not possible in the previously used frameworks (as discussed in [2]).

References

1 A. Antonik, M. Huth, K. G. Larsen, U. Nyman, and A. Wasowski. 20 years of modal and mixed specifications. *Bulletin of the EATCS no. 95*, pages 94–129, 2008.

2 N. Beneš, I. Černá, and J. Křetínský. Disjunctive modal transition systems and generalized LTL model checking. Submitted.

3 Dennis Dams, Rob Gerth, and Orna Grumberg. Abstract interpretation of reactive systems. *ACM Trans. Program. Lang. Syst.*, 19(2):253–291, 1997.

4 H. Fecher and M. Steffen. Characteristic mu-calculus formulas for underspecified transition systems. *ENTCS*, 128(2):103–116, 2005.

5 M. Huth, R. Jagadeesan, and D. A. Schmidt. Modal transition systems: A foundation for three-valued program analysis. In *Proc. of ESOP'01*, volume 2028 of *LNCS*, pages 155–169. Springer, 2001.

6 K. G. Larsen, U. Nyman, and A. Wasowski. Modeling software product lines using color-blind transition systems. *STTT*, 9(5-6):471–487, 2007.

7 K. G. Larsen and B. Thomsen. A modal process logic. In *LICS*, pages 203–210. IEEE Computer Society, 1988.

8 K. G. Larsen and L. Xinxin. Equation solving using modal transition systems. In *LICS*, pages 108–117. IEEE Computer Society, 1990.

9 J.-B. Raclet. Residual for component specifications. In *Proc. of the 4th International Workshop on Formal Aspects of Component Software*, 2007.

10 J.-B. Raclet, E. Badouel, A. Benveniste, B. Caillaud, and R. Passerone. Why are modalities good for interface theories? In *ACSD*, pages 119–127. IEEE, 2009.

Paper B:

Modal transition systems: Composition and LTL model checking

Nikola Beneš, Ivana Černá, and Jan Křetínský

This paper has been published in Tevfik Bultan and Pao-Ann Hsiung (eds.): Automated Technology for Verification and Analysis, 9th International Symposium, ATVA 2011, Taipei, Taiwan, October 11-14, 2011. Proceedings, volume 6996 of Lecture Notes in Computer Science, pages 228–242. Springer, 2011. [BČK11]

Summary

We study syntactically consistent DMTS. We first establish the complexities of modal refinement (P-complete), thorough refinement (EXP-complete) and refinements of deterministic systems (NL-complete). Second, we provide a unified algorithm for common implementation and specification problems and establish their complexities for different cases. As opposed to the general case with exponential complexity, we give a new polynomial solution when the number of systems on the input is fixed. Third, we show a fundamental flaw in previous attempts at (generalized) LTL model checking MTS, provide a new solution based on games extending it to DMTS, establish the complexities for several cases (the general case is 2-EXP-complete) and show how the model checking approach can help us to cope with the incompleteness of parallel composition of (D)MTS.

Author's contribution: 45 %

- participating in the discussions,
- contributing, in particular, to solutions to the thorough refinement, parallel composition, and model checking using games,
- writing Introduction and parts of the paper.

Modal Transition Systems: Composition and LTL Model Checking

Nikola Beneš[1⋆], Ivana Černá[1⋆⋆], and Jan Křetínský[1,2⋆⋆⋆]

[1] Faculty of Informatics, Masaryk University, Brno, Czech Republic
[2] Institut für Informatik, Technische Universität München, Germany
{xbenes3, cerna, jan.kretinsky}@fi.muni.cz

Abstract. Modal transition systems (MTS) is a well established formalism used for specification and for abstract interpretation. We consider its disjunctive extension (DMTS) and we provide algorithms showing that refinement problems for DMTS are not harder than in the case of MTS. There are two main results in the paper. Firstly, we identify an error in a previous attempt at LTL model checking of MTS and provide algorithms for LTL model checking of MTS and DMTS. Moreover, we show how to apply this result to compositional verification and circumvent the general incompleteness of the MTS composition. Secondly, we give a solution to the common implementation and conjunctive composition problems lowering the complexity from EXPTIME to PTIME.

1 Introduction

Specification and verification of programs is a fundamental part of theoretical computer science and is nowadays regarded indispensable when designing and implementing safety critical systems. Therefore, many specification formalisms and verification methods have been introduced. There are two main approaches to this issue. The *behavioural* approach exploits various equivalence or refinement checking methods, provided the specifications are given in the same formalism as implementations. The *logical* approach makes use of specifications given as formulae of temporal or modal logics and relies on efficient model checking algorithms. In this paper, we combine these two methods.

The specifications are rarely complete, either due to incapability of capturing all the required behaviour in the early design phase, or due to leaving a bunch of possibilities for the implementations, such as in e.g. product lines [1]. One thus begins the design process with an underspecified system where some behaviour is already prescribed and some may or may not be present. The specification is then successively refined until a real implementation is obtained, where all the

⋆ The author has been supported by Czech Grant Agency, grant no. GD102/09/H042.
⋆⋆ The author has been supported by Czech Grant Agency, grant no. GAP202/11/0312.
⋆⋆⋆ The author is a holder of Brno PhD Talent Financial Aid and is supported by the Czech Science Foundation, grant No. P202/10/1469.

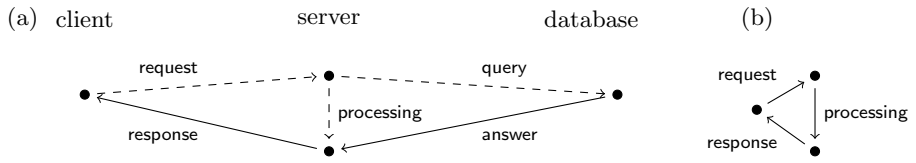


Fig. 1. An example of (a) a modal transition system (b) its implementation

behaviour is completely determined. Of course, we require that our formalism allow for this *stepwise refinement*.

Furthermore, since supporting the *component based design* is becoming crucial, we need to allow also for the compositional verification. To illustrate this, let us consider a partial specification of a component that we design, and a third party component that comes with some guarantees, such as a formula of a temporal logic describing the most important behaviour. Based on these underspecified models of the systems we would like to prove that their interaction is correct, no matter what the hidden details of the particular third party component are. Also, we want to know if there is a way to implement our component specification so that the composition fulfills the requirements. Moreover, we would like to synthesize the respective implementation. We address all these problems.

Modal transition systems (MTS) is a specification formalism introduced by Larsen and Thomsen [2, 3] allowing for stepwise refinement design of systems and their composition. A considerable attention has been recently paid to MTS due to many applications, e.g. component-based software development [4, 5], interface theories [6, 7], or modal abstractions and program analysis [8–10], to name just a few.

The MTS formalism is based on transparent and simple to understand model of *labelled transition systems* (LTS). While LTS has only one labelled transition relation between the states determining the behaviour of the system, MTS as a specification formalism is equipped with two types of transitions: the *must* transitions capture the required behaviour, which is present in all its implementations; the *may* transitions capture the allowed behaviour, which need not be present in all implementations. Figure 1 depicts an MTS that has arisen as a composition of three systems and specifies the following. A request from a client may arrive. Then we can process it directly or make a query to a database where we are guaranteed an answer. In both cases we send a response.

Such a system can be refined in two ways: a may transition is either implemented (and becomes a must transition) or omitted (and disappears as a transition). On the right there is an implementation of the system where the processing branch is implemented and the database query branch is omitted. Note that an implementation with both branches realized is also possible. This may model e.g. behaviour dependent on user input. Moreover, implementations may even be non-deterministic, thus allowing for modelling e.g. unspecified environment.

On the one hand, specifying may transitions brings guarantees on safety. On the other hand, liveness can be guaranteed to some extent using must transi-

tions. Nevertheless, at an early stage of design we may not know which of several possible different ways to implement a particular functionality will later be chosen, although we know at least one of them has to be present. We want to specify e.g. that either processing or query will be implemented, otherwise we have no guarantee on receiving response eventually. However, MTS has no way to specify liveness in this setting. Therefore, *disjunctive modal transition systems* (DMTS) (introduced in [11] as solutions to process equations) are the desirable extension appropriate for specifying liveness. This has been advocated also in [12] where a slight modification of DMTS is investigated under the name *underspecified transition systems*. Instead of forcing a particular transition, the must transitions in DMTS specify a whole set of transitions at least one of which must be present. In our example, it would be the set consisting of processing and query transitions. DMTS turn out to be capable of forcing any positive Boolean combination of transitions, simply by turning it into the conjunctive normal form. Another possible solution to this issue is offered in [13] where one-selecting MTS are introduced with the property that *exactly* one transition from the set must be present.

As DMTS is a strict extension of MTS a question arises whether all fundamental problems decidable in the context of MTS remain decidable for DMTS, and if so, whether their complexities remain unchanged. We show that this is indeed the case. Therefore, using the more powerful DMTS is not more costly than using MTS.

There is also another good reason to employ the greater power of DMTS instead of using MTS. Often a set of requirements need to be satisfied at once. Therefore, we are interested in the *common implementation* (CI) problem, where one asks whether there is an implementation that refines all specifications in a given set, i.e. whether the specifications are consistent. (In accordance with the traditional usage, the states of (D)MTS specifications shall be called processes.) Moreover, we also want to construct the most general process refining all processes, i.e. the greatest lower bound with respect to the refinement. We call this process a *conjunction* as this composition is the analog of logical and. We show there may not be any process that is a conjunction of a given set of processes, when only considering MTS processes. However, we also show that there is always a DMTS process that is a conjunction of a given set of (D)MTS processes. This again shows that DMTS is a more appropriate framework than MTS.

As the first main result, we show a new perspective on these problems, namely we give a simple co-inductive characterization yielding a straightforward fixpoint algorithm. This characterization unifies the view not only (i) in the MTS vs. DMTS aspect, but also (ii) in the cases of number of specifications being fixed or a part of the input, and most importantly (iii) establishes connection between CI and the conjunction. Our new view provides a solution for DMTS and yields algorithms for the aforementioned cases with the respective complexities being the same as for CI over MTS as determined in [14, 15]. So far, conjunction has been solved for MTS enriched with weights on transitions in [16], however, only for the deterministic case. Previous results on conjunction over DMTS [11] yield

an algorithm that requires exponential time (even for only two processes on input). Our algorithm runs in polynomial time both for conjunction and CI for any fixed number of processes on input.

As the second main result, as already mentioned we would like to supplement the refinement based framework of (D)MTS with model checking methods. Since a specification induces a set of implementations, we apply the thorough approach of generalized model checking of Kripke structures with partial valuations [17, 18] in our setting. Thus a specification either satisfies a formula φ if all its implementations satisfy φ; or refutes it if all implementations refute it; or neither of the previous holds, i.e. some of the implementations satisfy and some refute φ. This classification has also been adopted in [3] for CTL model checking MTS. Similarly, [19] provides a solution to LTL model checking over deadlock-free MTS, which was implemented in the tool support for MTS [20]. However, we identify an error in this LTL solution and provide correct model checking algorithms. The erroneous algorithm for the deadlock-free MTS was running in PSPACE, nevertheless, we show that this problem is 2-EXPTIME-complete by reduction to and from LTL games. The generalized model checking problem is equivalent to solving the problems (i) whether all implementations satisfy the given formula and if they do not then (ii) whether there exists an implementation satisfying the formula. We provide algorithms for both the universal and the existential case, and moreover, for the cases of MTS, deadlock-free MTS and DMTS, providing different complexities. Due to our reduction, the resulting algorithm can be also used for synthesis, i.e. if there is a satisfying implementation, we automatically receive it. Not only is the application in the specification area clear, but there is also an important application to abstract interpretation. End-users are usually more comfortable with linear time logic and the analysis of path properties requires to work with abstractions capturing over- and under-approximation of a system simultaneously. MTS are a perfect framework for this task, as may and must transitions can capture over- and under-approximations, respectively [8]. Our results thus allow for LTL model-checking of system abstractions, including counterexample generation.

Finally, we show how the model checking approach can help us getting around the fundamental problem with the parallel composition. There are MTS processes S and T, where the composed process $S \parallel T$ contains more implementations than what can be obtained by composing implementations of S and T. Hence the composition is not complete with respect to the semantic view. Some conditions to overcome this difficulty were identified in [15]. Here we show the general completeness of the composition with respect to the LTL formulae satisfaction, and generally to all linear time properties.

The rest of the paper is organized as follows. We provide basic definitions and results on refinements in Section 2. The results on LTL model checking and its relation to the parallel composition can be found in Section 3. The "logical and" composition is investigated in Section 4. Section 5 concludes and discusses future work. Due to space limitations the proofs are omitted and can be found in [21].

2 Preliminaries

In this section we define the specification formalism of disjunctive modal transition systems (DMTS). A DMTS can be gradually refined until we get a labelled transition system (LTS) where all the behaviour is fully determined. The semantics of a DMTS will thus be the set of its refining LTSs. The following definition is a slight modification of the original definition in [11].

Definition 2.1. *A* disjunctive modal transition system (DMTS) *over an action alphabet Σ and a set of propositions Ap is a tuple $(\mathcal{P}, \dashrightarrow, \longrightarrow, \nu)$ where $\mathcal{P}$ is a set of* processes*, $\dashrightarrow \subseteq \mathcal{P} \times \Sigma \times \mathcal{P}$ and $\longrightarrow \subseteq \mathcal{P} \times 2^{\Sigma \times \mathcal{P}}$ are* may *and* must transition relations*, respectively, and $\nu : \mathcal{P} \to 2^{Ap}$ is a* valuation*. We write $S \overset{a}{\dashrightarrow} T$ meaning $(S, a, T) \in \dashrightarrow$, and $S \longrightarrow \mathcal{T}$ meaning $(S, \mathcal{T}) \in \longrightarrow$. We require that whenever $S \longrightarrow \mathcal{T}$ then (i) $\mathcal{T} \neq \emptyset$ and (ii) for all $(a, T) \in \mathcal{T}$ we also have $S \overset{a}{\dashrightarrow} T$.*

The original definition of DMTS does not include the two requirements, thus allowing for *inconsistent* DMTS, which have no implementations. Due to the requirements, our DMTS guarantee that all must obligations can be fulfilled. Hence, we do not have to expensively check for consistency* when working with our DMTS. And there is yet another difference to the original definition. Since one of our aims is model checking state *and* action based LTL, we not only have labelled transitions, but we also equip DMTS with a valuation over states.

Clearly, the must transitions of DMTS can be seen as a positive boolean formula in conjunctive normal form. Arbitrary requirements expressible as positive boolean formulae can be thus represented by DMTS, albeit at the cost of possible exponential blowup, as commented on in [22].

Example 2.2. Figure 2 depicts three DMTSs. The may transitions are drawn as dashed arrows, while each must transition of the form $(S, \mathcal{T})$ is drawn as a solid arrow from S branching to all elements in $\mathcal{T}$. Due to requirement (ii) it is redundant to draw the dashed arrow when there is a solid arrow and we never depict it explicitly.

While in DMTS we can specify that at least one of the selected transitions has to be present, in modal transition systems (MTS) we can only specify that a particular transition has to be present, i.e. we need to know from the beginning which one. Thus MTS is a special case of DMTS. Further, when the may and must transition relations coincide, we get labelled transition systems (with valuation).

Definition 2.3. *A DMTS $\mathfrak{S} = (\mathcal{P}, \dashrightarrow, \longrightarrow, \nu)$ is an* MTS (with valuation) *if $S \longrightarrow \mathcal{T}$ implies that $\mathcal{T}$ is a singleton. We then write $S \xrightarrow{a} T$ for $\mathcal{T} = \{(a, T)\}$.*

* Checking consistency is an EXPTIME-complete problem. It is polynomial [11] only under an assumption that all "conjunctions" of processes are also present in the given DMTS which is very artificial in our setting. For more details, see [21].

If moreover $S \overset{a}{\dashrightarrow} T$ *implies* $S \overset{a}{\longrightarrow} T$*, then* $\mathfrak{S}$ *is an* LTS*. Processes of an LTS are called* implementations*.*

A DMTS $\mathfrak{S} = (\mathcal{P}, \dashrightarrow, \longrightarrow, \nu)$ *is* deterministic *if for every process* S *and action* a *there is at most one process* T *with* $S \overset{a}{\dashrightarrow} T$*.*

For the sake of readable notation, when speaking of a process, we often omit the underlying DMTS if it is clear from the context. Moreover, we say that S is deterministic (an MTS etc.) meaning that the DMTS on processes reachable from S is deterministic (MTS etc.). Further, when analyzing the complexity we assume we are given finite DMTSs.

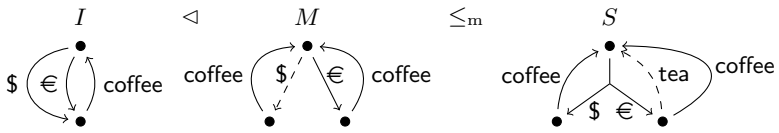


Fig. 2. An implementation I, a process M of an MTS, and a process S of a DMTS such that $I \lhd M \leq_{\mathrm{m}} S$

When refining a process, we need to satisfy two conditions: (1) the respective refining process cannot allow any new behaviour not allowed earlier; and (2) if there is a requirement to implement an action by choosing among several options, the refining process can only have more restrictive set of these options.

Definition 2.4 (Modal refinement). *Let* $(\mathcal{P}, \dashrightarrow, \longrightarrow, \nu)$ *be a DMTS. Then* $R \subseteq \mathcal{P} \times \mathcal{P}$ *is called a* modal refinement relation *if for all* $(A, B) \in R$

- $\nu(A) = \nu(B)$*, and*
- *whenever* $A \overset{a}{\dashrightarrow} A'$ *then* $B \overset{a}{\dashrightarrow} B'$ *for some* B' *with* $(A', B') \in R$*, and*
- *whenever* $B \longrightarrow \mathcal{B}'$ *then* $A \longrightarrow \mathcal{A}'$ *for some* $\mathcal{A}'$ *such that for all* $(a, A') \in \mathcal{A}'$ *there is* $(a, B') \in \mathcal{B}'$ *with* $(A', B') \in R$*.*

We say that S modally refines T*, denoted by* $S \leq_{\mathrm{m}} T$*, if there exists a modal refinement relation* R *with* $(S, T) \in R$*.*

Note that since a union of modal refinement relations is a modal refinement relation, the relation $\leq_{\mathrm{m}}$ is the greatest modal refinement relation. Also note that on implementations the modal refinement coincides with bisimulation.

We now define the semantics of a process as a set of implementations that are refining it. The defined notion of thorough refinement is a semantic counterpart to the syntactic notion of modal refinement.

Definition 2.5 (Thorough refinement). *Let* I, S, T *be processes. We say that* I *is an* implementation of S*, denoted by* $I \lhd S$*, if* I *is an implementation and* $I \leq_{\mathrm{m}} S$*. We say that* S thoroughly refines T*, denoted by* $S \leq_{\mathrm{t}} T$*, if* $J \lhd S$ *implies* $J \lhd T$ *for every implementation* J*.*

While the syntactic characterization is sound, it is not complete since it is incomplete already for MTS. However, completeness can be achieved on a reasonable subclass.

Proposition 2.6. *Let S and T be processes. Then $S \leq_{\mathrm{m}} T$ implies $S \leq_{\mathrm{t}} T$. If T is deterministic then $S \leq_{\mathrm{t}} T$ implies $S \leq_{\mathrm{m}} T$.*

Next we show that both refinement problems are not harder for DMTS than for MTS. This allows for using more powerful DMTS instead of MTS. The following is proven similarly as in [15]. In order to prove the last claim significantly involved modifications of the approach of [23] are needed.

Theorem 2.7. *Deciding $\leq_{\mathrm{m}}$ is PTIME-complete. Deciding $\leq_{\mathrm{m}}$ when restricted to the refined (i.e. right-hand-side) process being deterministic is NLOGSPACE-complete. Deciding $\leq_{\mathrm{t}}$ is EXPTIME-complete.*

3 LTL Model Checking

This section discusses the model checking problem for linear temporal logic (LTL) [24] and its application on compositional verification. The following definition of state and action based LTL is equivalent to that of [25], with a slight difference in syntax.

Definition 3.1 (LTL syntax). *The formulae of state and action based LTL (LTL in the following) are defined as follows.*

$$\varphi ::= \mathbf{tt} \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \,\mathbf{U}\, \varphi \mid \mathbf{X}\,\varphi \mid \mathbf{X}_a\,\varphi$$

where p ranges over Ap and a ranges over Σ.

We use the standard derived operators, such as $\mathbf{F}\,\varphi = \mathbf{tt}\,\mathbf{U}\,\varphi$ and $\mathbf{G}\,\varphi = \neg\,\mathbf{F}\,\neg\varphi$.

Definition 3.2 (LTL semantics). *Let I be an implementation. A run of I is a maximal (finite or infinite) alternating sequence of state valuations and actions $\pi = \nu(I_0), a_0, \nu(I_1), a_1, \ldots$ such that $I_0 = I$ and $I_{i-1} \xrightarrow{a_{i-1}} I_i$ for all $i > 0$. If a run π is finite, we denote by $|\pi|$ the number of state valuations in π, we set $|\pi| = \infty$ if π is infinite. We also define the ith subrun of π as $\pi^i = \nu(I_i), a_i, \nu(I_{i+1}), \ldots$ Note that this definition only makes sense when $i < |\pi|$. The set of all runs of I is denoted by $\mathcal{R}^\infty(I)$, the set of all infinite runs is denoted by $\mathcal{R}^\omega(I)$.*

The semantics of LTL on $\pi = \nu_0, a_0, \nu_1, a_1, \ldots$ is then defined as follows:

$$\begin{aligned}
&\pi \models \mathbf{tt} && \textit{always}\\
&\pi \models p && \iff p \in \nu_0\\
&\pi \models \neg\varphi && \iff \pi \not\models \varphi\\
&\pi \models \varphi \wedge \psi && \iff \pi \models \varphi \textit{ and } \pi \models \psi\\
&\pi \models \varphi\,\mathbf{U}\,\psi && \iff \exists\, 0 \leq k < |\pi| : \pi^k \models \psi \textit{ and } \forall\, 0 \leq j < k : \pi^j \models \varphi\\
&\pi \models \mathbf{X}\,\varphi && \iff |\pi| > 1 \textit{ and } \pi^1 \models \varphi\\
&\pi \models \mathbf{X}_a\,\varphi && \iff |\pi| > 1,\ a_0 = a \textit{ and } \pi^1 \models \varphi
\end{aligned}$$

We say that an implementation I satisfies φ on infinite runs, denoted as $I \models^\omega \varphi$, if for all $\pi \in \mathcal{R}^\omega(I)$, $\pi \models \varphi$. We say that an implementation I satisfies φ on all runs, denoted as $I \models^\infty \varphi$, if for all $\pi \in \mathcal{R}^\infty(I)$, $\pi \models \varphi$.

The use of symbols ω and ∞ to distinguish between using only infinite runs or all runs is in accordance with standard usage in the field of infinite words.

It is common to define LTL over infinite runs only. In that respect, our definition of $\models^\omega$ matches the standard definition. In the following, we shall first talk about this satisfaction relation only, and comment on $\models^\infty$ afterwards.

The generalized LTL model checking problem for DMTS can be split into two subproblems – deciding whether all implementations satisfy a given formula, and deciding whether at least one implementation does. We therefore introduce the following notation: we write $S \models^\omega_\forall \varphi$ to mean $\forall I \lhd S : I \models^\omega \varphi$ and $S \models^\omega_\exists \varphi$ to mean $\exists I \lhd S : I \models^\omega \varphi$; similarly for $\models^\infty$.

Note that $\models^\omega_\exists$ contains a hidden alternation [26] of quantifiers, as it actually means $\exists I \lhd S : \forall \pi \in \mathcal{R}^\omega(I) : I \models^\omega \varphi$. No alternation is present in $\models^\omega_\forall$. This observation hints that the problem of deciding $\models^\omega_\forall$ is easier than deciding $\models^\omega_\exists$. Our first two results show that indeed, deciding $\models^\omega_\forall$ is not harder than the standard LTL model checking whereas deciding $\models^\omega_\exists$ is 2-EXPTIME-complete.

The only known correct result on LTL model checking of MTS is that deciding $MTS \models^\omega_\forall$ over MTS is PSPACE-complete [19]. This holds also for DMTS.

Theorem 3.3. *The problem of deciding $\models^\omega_\forall$ over DMTS is PSPACE-complete.*

Proof (Sketch). All implementations of S satisfy φ if and only if the may structure of S satisfies φ. □

In [18] the generalized model checking of LTL over partial Kripke structures (PKS) is shown to be 2-EXPTIME-hard. Further, [27] describes a reduction from generalized model checking of μ-calculus over PKS to μ-calculus over MTS. However, the hardness for *LTL* over MTS does not follow since the encoding of an LTL formula into μ-calculus includes an exponential blowup. There is thus no straightforward way to use the result of [27] to provide a polynomial reduction. Therefore, we prove the following theorem directly.

Theorem 3.4. *The problem of deciding $\models^\omega_\exists$ over DMTS is 2-EXPTIME-complete.*

Proof (Sketch). We show the reduction to and from the 2-EXPTIME-complete problem of deciding existence of a winning strategy in an LTL game [28]. An LTL game is a two player positional game over a finite Kripke structure. The winning condition is the set of all infinite plays (sequences of states) satisfying a given LTL formula.

Thus, an LTL game may be seen as a special kind of DMTS over unary action alphabet. Here the processes are the states of the Kripke structure, the may structure is the transition relation of the Kripke structure, and the must structure is built as follows. Every process corresponding to a state of Player I has one must transition spanning all may-successors; every process corresponding to a state of Player II has several must transitions, one to each may-successor. The implementations of such DMTS now correspond to strategies of Player I in the original LTL game. Thus follows the hardness part of the theorem.

For the containment part, we provide an algorithm that transforms the given DMTS into a Kripke structure with states assigned to the two players. This

construction bears some similarities to the construction transforming Kripke MTS into alternating tree automata in [29].

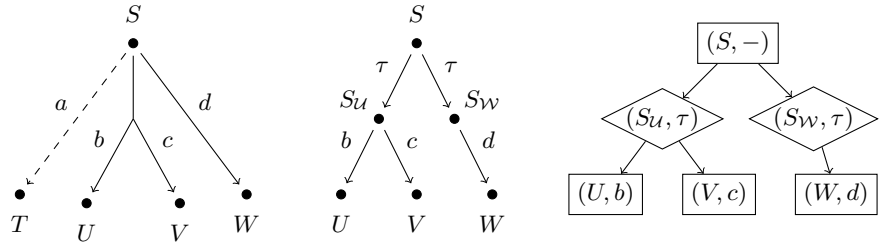


Fig. 3. Transformation from DMTS into a two player game

The transformation from a DMTS into a two player game proceeds as follows. We first eliminate all may transitions that are not covered by any must transitions. We then modify the must transitions. For each $S \longrightarrow \mathcal{U}$ we create a unique new process $S_{\mathcal{U}}$ and set $S \xrightarrow{\tau} S_{\mathcal{U}}$ and $S_{\mathcal{U}} \xrightarrow{a} T$ for all $(a, T) \in \mathcal{U}$. We thus now have a labelled transition system, possibly with valuation. We then eliminate actions by encoding them into their target state, thus obtaining a Kripke structure. States that were created from processes of the original DMTS belong to Player II, states created from must transitions belong to Player I. The construction is illustrated in Fig. 3. We then modify the LTL formula in two steps. First, we add the possibility of a τ action in every odd step. Second, we transform the state-and-action LTL formula into a purely state-based one. The resulting game over the Kripke structure together with the modified LTL formula form the desired LTL game. □

There are constructive algorithms for solving LTL games, i.e. not only do they decide whether a winning strategy exists, but they can also synthesize such a strategy. Furthermore, our reduction effectively transforms a winning strategy into an implementation satisfying the given formula. We can thus synthesize an implementation of a given DMTS satisfying a given formula in 2-EXPTIME.

Although the general complexity of the problem is very high, various subclasses of LTL have been identified in [30] for which the problem is computationally easier. These complexity results can be easily carried over to generalized model checking of DMTS.

Interestingly enough, deciding $\models^{\omega}_{\exists}$ is much easier over MTS.

Theorem 3.5. *The problem of deciding $\models^{\omega}_{\exists}$ over MTS is PSPACE-complete.*

Proof (Sketch). The proof is similar to the proof of Theorem 3.3, only instead of checking the may structure of S, we check the must structure of S. □

However, despite its lower complexity, $\models^{\omega}_{\exists}$ over MTS is not a very useful satisfaction relation. As we only considered infinite runs, an MTS may (and

frequently will) possess *trivial* implementations without infinite runs. The statement $S \models^{\omega}_{\exists} \varphi$ then holds vacuously for all φ. Two natural ways to cope with this issue are (a) using $\models^{\infty}_{\exists}$ (see below) and (b) considering only deadlock-free implementations, i.e. with infinite runs only.

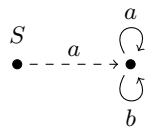


Fig. 4. No deadlock-free implementation of S satisfies $\mathbf{G}\,\mathbf{X}_a\,\mathbf{tt}$

The deadlock-free approach has been studied in [19] and the proposed solution was implemented in the tool MTSA [20]. However, the solution given in [19] is incorrect. In particular, existence of a deadlock-free implementation satisfying a given formula is claimed even in some cases where no such implementation exists. A simple counterexample is given in Fig. 4. Clearly, S has no deadlock-free implementation with action a only, i.e. satisfying $\mathbf{G}\,\mathbf{X}_a\,\mathbf{tt}$. Yet the method of [19] as well as the tool [20] claim that such an implementation exists.

Furthermore, there is no chance that the approach of [19] could be easily fixed to provide correct results. The reason is that this approach leads to a PSPACE algorithm, whereas we prove again by reduction from LTL games that finding a deadlock-free implementation of a given MTS is 2-EXPTIME-hard. For more details see [21]. The containment in 2-EXPTIME is then proved by reduction to the problem of deciding $\models^{\omega}_{\exists}$ for DMTS. The basic idea is to modify all processes without must transitions, enhancing them with one must transition spanning all may-successors.

Proposition 3.6. *The problem of deciding the existence of a deadlock-free implementation of a given MTS satisfying a given LTL formula, is 2-EXPTIME-complete.*

We now turn our attention to the (a) option, i.e. all (possibly finite) runs, and investigate the $\models^{\infty}$ satisfaction. Checking properties even on finite runs is indeed desirable when considering (D)MTS used for modelling non-reactive systems. We show that deciding $\models^{\infty}_{\exists}$ and $\models^{\infty}_{\forall}$ over DMTS has the same complexity as deciding $\models^{\omega}_{\exists}$ and $\models^{\omega}_{\forall}$ over DMTS, respectively. We also show that contrary to the case of infinite runs, the problem of deciding $\models^{\infty}_{\exists}$ remains 2-EXPTIME-hard even for standard MTS.

Theorem 3.7. *The problem of deciding $\models^{\infty}_{\exists}$ over (D)MTS is 2-EXPTIME-complete, the problem of deciding $\models^{\infty}_{\forall}$ over (D)MTS is PSPACE-complete.*

Although we have so far considered the more general state and action based LTL, this costs no extra overhead when compared to state-based or action-based LTL.

Table 1. Complexities of generalized LTL model checking

	$\models_\forall$	$\models_\exists$
MTS $\models^\omega$	PSPACE-complete	PSPACE-complete
MTS $\models^{\mathrm{df}}$	PSPACE-complete	2-EXPTIME-complete
MTS $\models^\infty$	PSPACE-complete	2-EXPTIME-complete
DMTS	PSPACE-complete	2-EXPTIME-complete

Proposition 3.8. *The complexity of deciding $\models^\star_\exists$ and $\models^\star_\forall$ for $\star \in \{\omega, \infty\}$ remains the same if the formula φ is a purely state-based or a purely action-based formula.*

The results of this section are summed up in Table 1. We use $\models^{\mathrm{df}}$ to denote that only deadlock-free implementations are considered. Recall that the surprising result for $\models^\omega_\exists$ over MTS is due to the fact that the formula may hold vacuously.

The best known time complexity bounds with respect to the size of system $|S|$ and the size of LTL formula $|\varphi|$ are the following. In all PSPACE-complete cases the time complexity is $\mathcal{O}(|S| \cdot 2^{|\varphi|})$; in all 2-EXPTIME-complete cases the time complexity is $|S|^{2^{\mathcal{O}(|\varphi|)}} \cdot 2^{2^{\mathcal{O}(|\varphi| \log |\varphi|)}}$. The latter upper bound is achieved by translating the LTL formula into a deterministic Rabin automaton of size $2^{2^{\mathcal{O}(|\varphi| \log |\varphi|)}}$ with $2^{\mathcal{O}(|\varphi|)}$ accepting pairs, thus changing the LTL game into a Rabin game. State of the art algorithm for solving Rabin games can be found e.g. in [31].

3.1 Parallel Composition

We conclude this section with an application to compositional verification. In [3] the composition of MTS is shown to be incomplete, i.e. there are processes S_1, S_2 such that their composition $S_1 \parallel S_2$ has an implementation I that does *not* arise as a composition $I_1 \parallel I_2$ of any two implementations $I_1 \lhd S_1, I_2 \lhd S_2$. Completeness can be achieved only under some restrictive conditions [15]. Here we show that composition is sound and complete with respect to every logic of linear time, i.e. it preserves and reflects all linear time properties.

For the sake of readability, we present the results on MTS only. Nevertheless, the same holds for the straightforward extension of $\parallel$ to DMTS, see [21].

The composition operator used is based on synchronous message passing, since it is the most general one. Indeed, it encompasses the synchronous product as well as interleaving. It is defined as follows. Let $\Gamma \subseteq \Sigma$ be a *synchronizing alphabet*. Then

- for $a \in \Gamma$, we set $S_1 \parallel S_2 \xrightarrow{a} S_1' \parallel S_2'$ whenever $S_1 \xrightarrow{a} S_1'$ and $S_2 \xrightarrow{a} S_2'$;
- for $a \in \Sigma \setminus \Gamma$, we set $S_1 \parallel S_2 \xrightarrow{a} S_1' \parallel S_2$ whenever $S_1 \xrightarrow{a} S_1'$, and similarly $S_1 \parallel S_2 \xrightarrow{a} S_1 \parallel S_2'$ whenever $S_2 \xrightarrow{a} S_2'$;

and analogously for the must transition relation. As for valuations, we can consider any function $f : 2^{Ap} \times 2^{Ap} \to 2^{Ap}$ to define $\nu(S_1 \parallel S_2) = f(\nu(S_1), \nu(S_2))$, such as e.g. union.

The completeness of composition with respect to linear time logics holds for all discussed cases: both for MTS and DMTS, both for infinite and all runs, and both universally and existentially. We do not define linear properties formally here, see e.g. [32]. As a special case, one may consider LTL formulae.

Theorem 3.9. *Let S_1, S_2 be processes, φ a linear time property, and $\star \in \{\omega, \infty\}$. Then $S_1 \parallel S_2 \models^\star_\forall \varphi$ if and only if $I_1 \parallel I_2 \models^\star \varphi$ for all $I_1 \lhd S_1$ and $I_2 \lhd S_2$.*

Theorem 3.10. *Let S_1, S_2 be processes, φ a linear time property, and $\star \in \{\omega, \infty\}$. Then $S_1 \parallel S_2 \models^\star_\exists \varphi$ if and only if there exist $I_1 \lhd S_1$ and $I_2 \lhd S_2$ such that $I_1 \parallel I_2 \models^\star \varphi$.*

The idea of the proof is that the minimal (w.r.t. the set of runs) implementations of $S_1 \parallel S_2$ are decomposable, i.e. they can be written as $I_1 \parallel I_2$ where $I_1 \lhd S_1$ and $I_2 \lhd S_2$. The same holds for the maximal implementations of $S_1 \parallel S_2$. The results imply that although the composition is incomplete with respect to thorough refinement no new behaviour arises in the composition.

4 Common Implementation Problem and Conjunction

In the following, we study composing (D)MTS in the sense of logical conjunction. The *common implementation problem* (CI) is to decide whether there is an implementation refining all processes from a given set. Furthermore, we also want to construct the *conjunction*, i.e. the process that is the greatest lower bound for a given set of processes w.r.t. the modal refinement, if it exists. We show that although MTSs may not have an MTS conjunction, there is always a conjunction expressible as a DMTS. The complexity depends on the number of the input processes. We examine the complexity both for the case when it is fixed and when it is a part of the input.

Theorem 4.1. *For the number of input processes being a part of the input, the CI problem is EXPTIME-complete and conjunction can be computed in exponential time. For any fixed number of input processes, CI is PTIME-complete and conjunction can be computed in polynomial time.*

We first give a coinductive syntactic characterization of the problem and proceed by constructing the greatest lower bound.

Definition 4.2 (Consistency relation). *Let $(\mathcal{P}, \dashrightarrow, \longrightarrow, \nu)$ be a DMTS and $n \geq 2$. Then $C \subseteq \mathcal{P}^n$ is called a* consistency relation *if for all $(A_1, \ldots, A_n) \in C$*

- *$\nu(A_1) = \nu(A_2) = \ldots = \nu(A_n)$, and*
- *whenever there exists i such that $A_i \longrightarrow \mathcal{B}_i$, then there is some $(a, B_i) \in \mathcal{B}_i$ such that there exist B_j for all $j \neq i$ with $A_j \overset{a}{\dashrightarrow} B_j$ and $(B_1, \ldots, B_n) \in C$.*

In the following, we will assume an arbitrary, but fixed n. Clearly, arbitrary union of consistency relations is also a consistency relation, we may thus assume the existence of the greatest consistency relation for a given DMTS. We now show how to use this relation to construct a DMTS that is the greatest lower bound with regard to modal refinement (taken as a preorder).

Definition 4.3. *Let $\mathfrak{S} = (\mathcal{P}, \dashrightarrow, \longrightarrow, \nu)$ be a DMTS and* Con *its greatest consistency relation. We define a new DMTS $\mathfrak{S}_{\mathsf{Con}} = (\mathsf{Con}, \dashrightarrow_{\mathsf{Con}}, \longrightarrow_{\mathsf{Con}}, \nu_{\mathsf{Con}})$, where*

- $\nu_{\mathsf{Con}}((A_1, \ldots, A_n)) = \nu(A_1)$,
- $(A_1, \ldots, A_n) \overset{a}{\dashrightarrow}_{\mathsf{Con}} (B_1, \ldots, B_n)$ *whenever* $\forall i : A_i \overset{a}{\dashrightarrow} B_i$, *and*
- *whenever* $\exists j : A_j \longrightarrow \mathcal{B}_j$, *then* $(A_1, \ldots, A_n) \longrightarrow_{\mathsf{Con}} \mathcal{B}$ *where* $\mathcal{B} = \{(a, (B_1, \ldots, B_n)) \mid (a, B_j) \in \mathcal{B}_j \text{ and } (A_1, \ldots, A_n) \overset{a}{\dashrightarrow}_{\mathsf{Con}} (B_1, \ldots, B_n)\}$.

Clearly, the definition gives a correct DMTS due to the properties of Con, notably, $\mathcal{B}$ is never empty. The following two theorems state the results about the CI problem and conjunction construction, respectively. The second theorem also states that the actual result is stronger than originally intended.

Theorem 4.4. *Let $S_1, \ldots, S_n$ be processes. Then $S_1, \ldots, S_n$ have a common implementation if and only if $(S_1, \ldots, S_n) \in \mathsf{Con}$.*

Theorem 4.5. *Let $(S_1, \ldots, S_n) \in \mathsf{Con}$. Then the set of all implementations of $(S_1, \ldots, S_n)$ is exactly the intersection of the sets of all implementations of all S_i. In other words, $I \lhd (S_1, \ldots, S_n)$ if and only if $I \lhd S_i$ for all i. Therefore, $(S_1, \ldots, S_n)$ as a process of $\mathfrak{S}_{\mathsf{Con}}$ is the greatest lower bound of S_1, ..., S_n with regard to the* modal *as well as the* thorough *refinement.*

The greatest consistency relation can be computed using standard greatest fixed point computation, i.e. we start with all $n$tuples of processes and eliminate those that violate the conditions. One elimination step can clearly be done in polynomial time. As the number of all $n$tuples is at most $|\mathcal{P}|^n$, this means that the common implementation problem may be solved in PTIME, if n is fixed; and in EXPTIME, if n is a part of the input. The problem is also PTIME/EXPTIME-hard, which follows from (a) PTIME-hardness of bisimulation of two LTSs and (b) EXPTIME-hardness of the common implementation problem for ordinary MTS [14]. The statement of Theorem 4.1 thus follows.

Note that even if $S_1, \ldots, S_n$ are MTSs, $(S_1, \ldots, S_n)$ may not be an MTS. Indeed, there exist MTSs without a greatest lower bound that is also an MTS; there may only be several maximal lower bounds, see Fig. 5. This gives another justification for using DMTS instead of MTS. However, if the MTSs are moreover *deterministic*, then the greatest lower bound is—as our algorithm computes it—also a deterministic MTS [16].

5 Conclusion and Future Work

Our generalization of the known algorithms has shown that refinement problems on DMTS are not harder than for MTS. As the first main result, we have solved the LTL model checking and synthesis problems and shown how the model checking approach helps overcoming difficulties with the parallel composition. We have implemented the algorithm in $\overset{\longrightarrow\Longrightarrow\dashrightarrow}{\text{MoTraS}}$, our prototype tool available

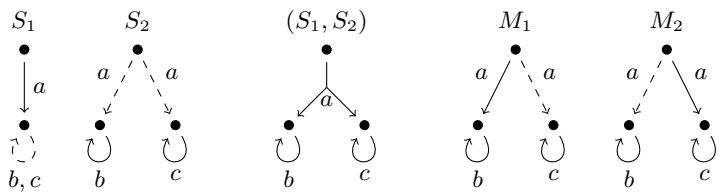


Fig. 5. MTSs S_1, S_2, their greatest lower bound (S_1, S_2), and their two maximal MTS lower bounds M_1, M_2

at `http://anna.fi.muni.cz/~xbenes3/MoTraS/` (the site includes further details about the tool and its functionality). As the second main result, we have given a general solution to the common implementation problem and conjunctive composition.

There are several possible extensions of DMTS such as the mixed variant (where must transition need not be syntactically under the may transitions) or systems with partial valuation on states [3]. Yet another modification adds weights on transitions [16]. It is not clear whether all results of this paper can be extended to these systems and whether the respective complexities remain the same.

References

1. Larsen, K.G., Nyman, U., Wasowski, A.: Modeling software product lines using color-blind transition systems. STTT **9**(5-6) (2007) 471–487
2. Larsen, K.G., Thomsen, B.: A modal process logic. In: LICS, IEEE Computer Society (1988) 203–210
3. Antonik, A., Huth, M., Larsen, K.G., Nyman, U., Wasowski, A.: 20 years of modal and mixed specifications. Bulletin of the EATCS no. 95 (2008) 94–129
4. Raclet, J.B.: Residual for component specifications. In: Proc. of the 4th International Workshop on Formal Aspects of Component Software. (2007)
5. Bertrand, N., Pinchinat, S., Raclet, J.B.: Refinement and consistency of timed modal specifications. In: Proc. of LATA'09. Volume 5457 of LNCS., Springer (2009) 152–163
6. Raclet, J.B., Badouel, E., Benveniste, A., Caillaud, B., Passerone, R.: Why are modalities good for interface theories? In: ACSD, IEEE (2009) 119–127
7. Uchitel, S., Chechik, M.: Merging partial behavioural models. In: Proc. of FSE'04, ACM (2004) 43–52
8. Huth, M., Jagadeesan, R., Schmidt, D.A.: Modal transition systems: A foundation for three-valued program analysis. In: Proc. of ESOP'01. Volume 2028 of LNCS., Springer (2001) 155–169
9. Godefroid, P., Huth, M., Jagadeesan, R.: Abstraction-based model checking using modal transition systems. In: Proc. CONCUR'01. Volume 2154 of LNCS., Springer (2001) 426–440
10. Nanz, S., Nielson, F., Nielson, H.R.: Modal abstractions of concurrent behaviour. In: Proc. of SAS'08. Volume 5079 of LNCS., Springer (2008) 159–173

11. Larsen, K.G., Xinxin, L.: Equation solving using modal transition systems. In: LICS, IEEE Computer Society (1990) 108–117
12. Fecher, H., Steffen, M.: Characteristic mu-calculus formulas for underspecified transition systems. ENTCS **128**(2) (2005) 103–116
13. Fecher, H., Schmidt, H.: Comparing disjunctive modal transition systems with an one-selecting variant. J. of Logic and Alg. Program. **77**(1-2) (2008) 20–39
14. Antonik, A., Huth, M., Larsen, K.G., Nyman, U., Wasowski, A.: EXPTIME-complete decision problems for mixed and modal specifications. In: 15th International Workshop on Expressiveness in Concurrency. (2008)
15. Beneš, N., Křetínský, J., Larsen, K., Srba, J.: On determinism in modal transition systems. Theoretical Computer Science **410**(41) (2009) 4026–4043
16. Juhl, L., Larsen, K.G., Srba, J.: Introducing modal transition systems with weight intervals. (Submitted.)
17. Bruns, G., Godefroid, P.: Generalized model checking: Reasoning about partial state spaces. In: CONCUR 2000. Volume 1877 of LNCS., Springer (2000) 168–182
18. Godefroid, P., Piterman, N.: LTL generalized model checking revisited. In: VMCAI. Volume 5403 of LNCS., Springer (2009) 89–104
19. Uchitel, S., Brunet, G., Chechik, M.: Synthesis of partial behavior models from properties and scenarios. IEEE Trans. Software Eng. **35**(3) (2009) 384–406
20. D'Ippolito, N., Fischbein, D., Chechik, M., Uchitel, S.: MTSA: The modal transition system analyser. In: Proc. of ASE'08, IEEE (2008) 475–476
21. Beneš, N., Černá, I., Křetínský, J.: Disjunctive modal transition systems and generalized LTL model checking. Technical report FIMU-RS-2010-12, Faculty of Informatics, Masaryk University, Brno (2010)
22. Beneš, N., Křetínský, J.: Process algebra for modal transition systemses. In: MEMICS. Volume 16 of OASICS., Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany (2010) 9–18
23. Beneš, N., Křetínský, J., Larsen, K.G., Srba, J.: Checking thorough refinement on modal transition systems is EXPTIME-complete. In: ICTAC 2009. Volume 5684 of LNCS., Springer (2009)
24. Pnueli, A.: The temporal logic of programs. In: FOCS, IEEE (1977) 46–57
25. Chaki, S., Clarke, E.M., Ouaknine, J., Sharygina, N., Sinha, N.: State/event-based software model checking. In: Proceedings of IFM'04. Volume 2999 of LNCS., Springer-Verlag (2004) 128–147
26. Godefroid, P., Jagadeesan, R.: Automatic abstraction using generalized model checking. In: CAV. Volume 2404 of LNCS. Springer (2002) 137–151
27. Godefroid, P., Jagadeesan, R.: On the expressiveness of 3-valued models. In: VMCAI. Volume 2575 of LNCS., Springer (2003) 206–222
28. Pnueli, A., Rosner, R.: On the synthesis of an asynchronous reactive module. In: ICALP. Volume 372 of LNCS., Springer (1989) 652–671
29. Dams, D., Namjoshi, K.S.: Automata as abstractions. In: VMCAI. Volume 3385 of LNCS. (2005) 216–232
30. Alur, R., Torre, S.L.: Deterministic generators and games for LTL fragments. ACM Trans. Comput. Log. **5**(1) (2004) 1–25
31. Piterman, N., Pnueli, A.: Faster solution of rabin and streett games. In: Proceedings of LICS'06, IEEE press (2006) 275–284
32. Baier, C., Katoen, J.P.: Principles of model checking. MIT Press (2008)

Paper C:

Parametric modal transition systems

Nikola Beneš, Jan Křetínský, Kim G. Larsen, Mikael H. Moller, and Jiri Srba

This paper has been published in Tevfik Bultan and Pao-Ann Hsiung (eds.): Automated Technology for Verification and Analysis, 9th International Symposium, ATVA 2011, Taipei, Taiwan, October 11-14, 2011. Proceedings, volume 6996 of Lecture Notes in Computer Science, pages 275–289. Springer, 2011. [BKL+11]

Summary

MTS and their mixed and disjunctive extensions lack the ability to express practically useful modalities such as exclusive choices (exactly one of the possible options is always available in the implementation), persistent choices (an implementation always offers the same choices in states implementing the same specification) or conditional choices (the choices are dependent on each other across the whole system). Therefore, we extend MTS with parameters (PMTS) and express modalities with an obligation function introduced in Paper A extended to general Boolean expressions over transitions as well as parameters. Further, we also provide the respective notion of modal refinement and provide a detailed account on its complexity for the most relevant subclasses of PMTS.

Author's contribution: 30 %

- participating in the discussions,
- contributing, in particular, to defining the framework and finding solutions to several complexity problems,
- writing parts of the paper.

Parametric Modal Transition Systems

Nikola Beneš[2⋆] Jan Křetínský[2,3⋆⋆] Kim G. Larsen[1]
Mikael H. Møller[1] Jiří Srba[1⋆⋆⋆]

[1] Aalborg University, Denmark
[2] Masaryk University, Czech Republic
[3] Technische Universität München, Germany

Abstract. Modal transition systems (MTS) is a well-studied specification formalism of reactive systems supporting a step-wise refinement methodology. Despite its many advantages, the formalism as well as its currently known extensions are incapable of expressing some practically needed aspects in the refinement process like exclusive, conditional and persistent choices. We introduce a new model called parametric modal transition systems (PMTS) together with a general modal refinement notion that overcome many of the limitations and we investigate the computational complexity of modal refinement checking.

1 Introduction

The specification formalisms of Modal Transition Systems (MTS) [11, 1] grew out of a series of attempts to achieve a flexible and easy-to-use compositional development methodology for reactive systems. In fact the formalism of MTS may be seen as a fragment of a temporal logic [5], while having a behavioural semantics allowing for an easy composition with respect to process constructs.

In short, MTS are labelled transition systems equipped with two types of transitions: *must* transitions which are mandatory for any implementation, and *may* transitions which are optional for an implementation. Refinement of an MTS now essentially consists of iteratively resolving the unsettled status of may transitions: either by removing them or by turning them into must transitions.

It is well admitted (see e.g. [15]) that MTS and their extensions like disjunctive MTS (DMTS) [12], 1-selecting MTS (1MTS) [6] and transition systems with obligations (OTS) [4] provide strong support for a specification formalism allowing for step-wise refinement process. Moreover, the MTS formalisms have applications in other contexts, which include verification of product lines [8, 10], interface theories [17, 15] and modal abstractions in program analysis [7, 9, 13].

Unfortunately, all of these formalisms lack the capability to express some intuitive specification requirements like exclusive, conditional and persistent

⋆ The author is supported by Czech Grant Agency, grant no. GAP202/11/0312.
⋆⋆ The author is a holder of Brno PhD Talent Financial Aid and is supported by the Czech Science Foundation, grant No. P202/10/1469.
⋆⋆⋆ The author is partially supported by Ministry of Education of The Czech Republic, grant no. MSM 0021622419.

choices. In this paper we extend considerably the expressiveness of MTS and its variants so that it can model arbitrary Boolean conditions on transitions and also allows to instantiate persistent transitions. Our model, called *parametric modal transition systems* (PMTS), is equipped with a finite set of parameters that are fixed prior to the instantiation of the transitions in the specification. The generalized notion of modal refinement is designed to handle the parametric extension and it specializes to the well-studied modal refinements on all the subclasses of our model like MTS, disjunctive MTS and MTS with obligations.

To the best of our knowledge, this is the first sound attempt to introduce persistence into a specification formalism based on modal transition systems. The most related work is by Fecher and Schmidt on 1-selecting MTS [6] where the authors allow to model exclusive-or and briefly mention the desire to extend the formalism with persistence. However, as in detail explained in [3], their definition does not capture the notion of persistence. Our formalism is in several aspects semantically more general and handles persistence in a complete and uniform manner.

The main technical contribution, apart from the formalism itself, is a comprehensive complexity characterization of modal refinement checking on all of the practically relevant subclasses of PMTS. We show that the complexity ranges from P-completeness to Π_4^{p}-completeness, depending on the requested generality of the PMTS specifications on the left-hand and right-hand sides.

2 Parametric Modal Transition Systems

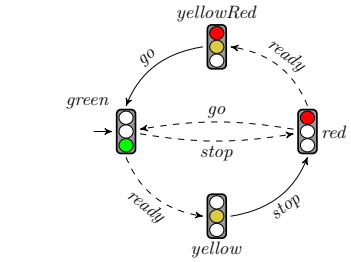
In this section we present the formalism of parametric modal transition systems (PMTS), starting with a motivating example and continuing with the formal definitions, followed by the general notion of modal refinement.

2.1 Motivation

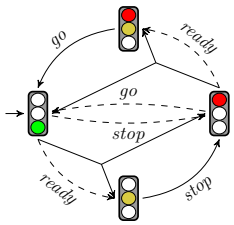
Modal transition systems and their extensions described in the literature are lacking the capability to express several specification requirements like exclusive, conditional and persistent choices. We shall now discuss these limitations on an example as a motivation for the introduction of parametric MTS formalism with general Boolean conditions in specification requirements.

Consider a simple specification of a traffic light controller that can be at any moment in one of the four predefined states: *red*, *green*, *yellow* or *yellowRed*. The requirements of the specification are: when *green* is on the traffic light may either change to *red* or *yellow* and if it turned *yellow* it must go to *red* afterward; when *red* is on it may either turn to *green* or *yellowRed*, and if it turns *yellowRed* (as it is the case in some countries) it must go to *green* afterwords.

Figure 1a shows an obvious MTS specification (defined formally later on) of the proposed specification. The transitions in the standard MTS formalism are either of type may (optional transitions depicted as dashed lines) or must (required transitions depicted as solid lines). In Figure 1c, Figure 1d and Figure 1e

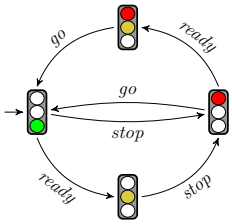


(a) MTS specification S_1

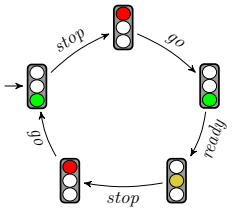


(b) DMTS specification S_2

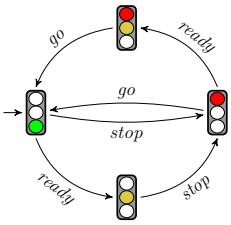
(c) Implementation I_1



(d) Implementation I_2

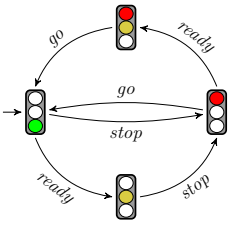


(e) Implementation I_3



Obligation function:
$\Phi(green) = (stop, red) \oplus (ready, yellow)$
$\Phi(red) = (go, green) \oplus (ready, yellowRed)$

(f) Specification S_3



Parameters: $\{reqYfromR, reqYfromG\}$
Obligation function:
$\Phi(green) = ((stop, red) \oplus (ready, yellow))$
$\wedge(reqYfromG \Leftrightarrow (ready, yellow))$
$\Phi(red) = ((go, green) \oplus (ready, yellowRed))$
$\wedge(reqYfromR \Leftrightarrow (ready, yellowRed))$

(g) PMTS specification S_4

Fig. 1: Specifications and implementations of a traffic light controller

we present three different implementations of the MTS specification where there are no more optional transitions. The implementation I_1 does not implement any may transition as it is a valid possibility to satisfy the specification S_1. Of course, in our concrete example, this means that the light is constantly *green* and it is clearly an undesirable behaviour that cannot be, however, easily avoided. The second implementation I_2 on the other hand implements all may transitions, again a legal implementation in the MTS methodology but not a desirable implementation of a traffic light as the next action is not always deterministically given. Finally, the implementation I_3 of S_1 illustrates the third problem with the MTS specifications, namely that the choices made in each turn are not persistent and the implementation alternates between entering *yellow* or not. None of these problems can be avoided when using the MTS formalism.

A more expressive formalism of disjunctive modal transition systems (DMTS) can overcome some of the above mentioned problems. A possible DMTS specification S_2 is depicted in Figure 1b. Here the *ready* and *stop* transitions, as well as *ready* and *go* ones, are disjunctive, meaning that it is still optional which one is implemented but at least one of them must be present. Now the system I_1 in Figure 1c is not a valid implementation of S_2 any more. Nevertheless, the undesirable implementations I_2 and I_3 are still possible and the modelling power of DMTS is insufficient to eliminate them.

Inspired by the recent notion of transition systems with obligations [4], we can model the traffic light using specification as a transition system with arbitrary[4] obligation formulae. These formulae are Boolean propositions over the outgoing transitions from each state, whose satisfying assignments yield the allowed combinations of outgoing transitions. A possible specification called S_3 is given in Figure 1f and it uses the operation of exclusive-or. We will follow an agreement that whenever the obligation function for some node is not listed in the system description then it is implicitly understood as requiring all the available outgoing transitions to be present. Due to the use of exclusive-or in the obligation function, the transition systems I_1 and I_2 are not valid implementation any more. Nevertheless, the implementation I_3 in Figure 1e cannot be avoided in this formalism either.

Finally, the problem with the alternating implementation I_3 is that we cannot enforce in any of the above mentioned formalisms a uniform (persistent) implementation of the same transitions in all its states. In order to overcome this problem, we propose the so-called parametric MTS where we can, moreover, choose persistently whether the transition to *yellow* is present or not via the use of parameters. The PMTS specification with two parameters *reqYfromR* and *reqYfromG* is shown in Figure 1g. Fixing a priori the (Boolean) values of the parameters makes the choices permanent in the whole implementation, hence we eliminate also the last problematic implementation I_3.

2.2 Definition of Parametric Modal Transition System

We shall now formally capture the intuition behind parametric MTS introduced above. First, we recall the standard propositional logic.

A Boolean formula over a set X of atomic propositions is given by the following abstract syntax

$$\varphi ::= \mathbf{tt} \mid x \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi$$

where x ranges over X. The set of all Boolean formulae over the set X is denoted by $\mathcal{B}(X)$. Let $\nu \subseteq X$ be a truth assignment, i.e. a set of variables with value true, then the satisfaction relation $\nu \models \varphi$ is given by $\nu \models \mathbf{tt}$, $\nu \models x$ iff $x \in \nu$, and the satisfaction of the remaining Boolean connectives is defined in the standard way. We also use the standard derived operators like exclusive-or $\varphi \oplus \psi = (\varphi \wedge$

[4] In the transition systems with obligations only positive Boolean formulae are allowed.

$\neg\psi) \vee (\neg\varphi \wedge \psi)$, implication $\varphi \Rightarrow \psi = \neg\varphi \vee \psi$ and equivalence $\varphi \Leftrightarrow \psi = (\neg\varphi \vee \psi) \wedge (\varphi \vee \neg\psi)$.

We can now proceed with the definition of parametric MTS.

Definition 1. *A* parametric MTS *(PMTS) over an action alphabet Σ is a tuple (S, T, P, Φ) where S is a set of* states*, $T \subseteq S \times \Sigma \times S$ is a* transition relation*, P is a finite set of* parameters*, and $\Phi : S \to \mathcal{B}((\Sigma \times S) \cup P)$ is an* obligation function *over the atomic propositions containing outgoing transitions and parameters. We implicitly assume that whenever $(a, t) \in \Phi(s)$ then $(s, a, t) \in T$. By $T(s) = \{(a, t) \mid (s, a, t) \in T\}$ we denote the set of all outgoing transitions of s.*

We recall the agreement that whenever the obligation function for some node is not listed in the system description then it is implicitly understood as $\Phi(s) = \bigwedge T(s)$, with the empty conjunction being $\mathbf{tt}$.

We call a PMTS *positive* if, for all $s \in S$, any negation occurring in $\Phi(s)$ is applied only to a parameter. A PMTS is called *parameter-free* if $P = \emptyset$. We can now instantiate the previously studied specification formalisms as subclasses of PMTS.

Definition 2. *A PMTS is called*

- transition system with obligation *(OTS) if it is parameter-free and positive,*
- disjunctive modal transition system *(DMTS) if it is an OTS and $\Phi(s)$ is in the conjunctive normal form for all $s \in S$,*
- modal transition system *(MTS) if it is a DMTS and $\Phi(s)$ is a conjunction of positive literals (transitions) for all $s \in S$, and*
- implementation *(or simply a labelled transition system) if it is an MTS and $\Phi(s) = \bigwedge T(s)$ for all $s \in S$.*

Note that positive PMTS, despite the absence of a general negation and the impossibility to define for example exclusive-or, can still express useful requirements like $\Phi(s) = p \Rightarrow (a, t) \wedge \neg p \Rightarrow (b, u)$ requiring in a state s a conditional presence of certain transitions. Even more interestingly, we can enforce binding of actions in different states, thus ensuring certain functionality. Take a simple two state-example: $\Phi(s) = p \Rightarrow (\mathit{request}, t)$ and $\Phi(t) = p \Rightarrow (\mathit{response}, s)$. We shall further study OTS with formulae in the disjunctive normal form that are dual to DMTS and whose complexity of parallel composition is lower [4] while still being as expressive as DMTS.

2.3 Modal Refinement

A fundamental advantage of MTS-based formalisms is the presence of *modal refinement* that allows for a step-wise system design (see e.g. [1]). We shall now provide such a refinement notion for our general PMTS model so that it will specialize to the well-studied refinement notions on its subclasses. In the definition, the parameters are fixed first (persistence) followed by all valid choices modulo the fixed parameters that now behave as constants.

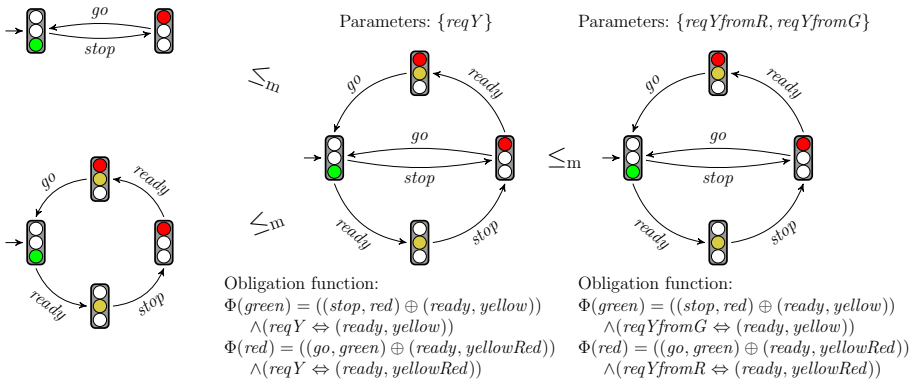


Fig. 2: Example of modal refinement

First we set the following notation. Let (S, T, P, Φ) be a PMTS and $\nu \subseteq P$ be a truth assignment. For $s \in S$, we denote by $\mathrm{Tran}_\nu(s) = \{E \subseteq T(s) \mid E \cup \nu \models \Phi(s)\}$ the set of all admissible sets of transitions from s under the fixed truth values of the parameters.

We can now define the notion of modal refinement between PMTS.

Definition 3 (Modal Refinement). *Let (S_1, T_1, P_1, Φ_1) and (S_2, T_2, P_2, Φ_2) be two PMTSs. A binary relation $R \subseteq S_1 \times S_2$ is a* modal refinement *if for each $\mu \subseteq P_1$ there exists $\nu \subseteq P_2$ such that for every $(s, t) \in R$ holds*

$$\forall M \in \mathrm{Tran}_\mu(s) : \exists N \in \mathrm{Tran}_\nu(t) : \begin{array}{l} \forall (a, s') \in M : \exists (a, t') \in N : (s', t') \in R \ \wedge \\ \forall (a, t') \in N : \exists (a, s') \in M : (s', t') \in R \ . \end{array}$$

We say that s modally refines t, denoted by $s \leq_\mathrm{m} t$, if there exists a modal refinement R such that $(s, t) \in R$.

Example 4. Consider the rightmost PMTS in Figure 2. It has two parameters *reqYfromG* and *reqYfromR* whose values can be set independently and it can be refined by the system in the middle of the figure having only one parameter *reqY*. This single parameter simply binds the two original parameters to the same value. The PMTS in the middle can be further refined into the implementations where either *yellow* is always used in both cases, or never at all. Notice that there are in principle infinitely many implementations of the system in the middle, however, they are all bisimilar to either of the two implementations depicted in the left of Figure 2.

In the next section, we shall investigate the complexity of positive subclasses of PMTS. For this reason we prove the following lemma showing how the definition of modal refinement can be simplified in this particular case.

We shall first realize that in positive PMTS and for any truth assignment ν, $\mathrm{Tran}_\nu(s)$ is *upward closed*, meaning that if $M \in \mathrm{Tran}_\nu(s)$ and $M \subseteq M' \subseteq T(s)$ then $M' \in \mathrm{Tran}_\nu(s)$.

Lemma 5. *Consider Definition 3 where the right-hand side PMTS is positive. Now the condition in Definition 3 can be equivalently rewritten as a conjunction of conditions* (1) *and* (2)

$$\forall M \in \mathrm{Tran}_\mu(s) : \forall (a, s') \in M : \exists (a, t') \in T(t) : (s', t') \in R \tag{1}$$

$$\forall M \in \mathrm{Tran}_\mu(s) : \mathrm{match}_t(M) \in \mathrm{Tran}_\nu(t) \tag{2}$$

where $\mathrm{match}_t(M)$ *denotes the set* $\{(a, t') \in T(t) \mid \exists (a, s') \in M : (s', t') \in R\}$*. If the left-hand side PMTS is moreover positive too, Condition* (1) *is equivalent to*

$$\forall (a, s') \in T(s) : \exists (a, t') \in T(t) : (s', t') \in R \ . \tag{3}$$

Proof. We shall first argue that the condition of modal refinement is equivalent to the conjunction of Conditions (4) and (5).

$$\forall M \in \mathrm{Tran}_\mu(s) : \exists N \in \mathrm{Tran}_\nu(t) : \forall (a, s') \in M : \exists (a, t') \in N : (s', t') \in R \tag{4}$$

$$\forall M \in \mathrm{Tran}_\mu(s) : \exists N \in \mathrm{Tran}_\nu(t) : \forall (a, t') \in N : \exists (a, s') \in M : (s', t') \in R \tag{5}$$

Let μ, ν, R, s and t be fixed. Definition 3 trivially implies both Conditions (4) and (5). We now prove that (4) and (5) imply the condition in Definition 3.

Let $M \in \mathrm{Tran}_\mu(s)$ be arbitrary. There is some $N_1 \in \mathrm{Tran}_\nu(t)$ satisfying (4) and some $N_2 \in \mathrm{Tran}_\nu(t)$ satisfying (5). Let now $N_1' = \{(a, t') \in N_1 \mid \exists (a, s') \in M : (s', t') \in R\}$. Consider $N = N_1' \cup N_2$. Clearly, as $\mathrm{Tran}_\nu(t)$ is upward closed, $N \in \mathrm{Tran}_\nu(t)$. Moreover, due to Condition (4) we have some $(a, t') \in N_1$ such that $(s', t') \in R$. Clearly, $(a, t') \in N_1'$ and thus also in N.

Now let $(a, t') \in N$ be arbitrary. If $(a, t') \in N_2$, due to Condition (5) we have some $(a, s') \in M$ such that $(s', t') \in R$. If $(a, t') \notin N_2$ then $(a, t') \in N_1'$. The existence of $(a, s') \in M$ such that $(s', t') \in R$ is then guaranteed by the definition of N_1'.

Let us now proceed with proving the claims of the lemma. Condition (4) is trivially equivalent to (1) since $\mathrm{Tran}_\nu(t)$ is upward closed. Condition (5) is equivalent to (2). Indeed, (2) clearly implies (5) and we show that also (5) implies (2). Let M be arbitrary. We then have some N satisfying (5). Clearly, $N \subseteq \mathrm{match}_t(M)$. Since $\mathrm{Tran}_\nu(t)$ is upward closed, $N \in \mathrm{Tran}_\nu(t)$ implies $\mathrm{match}_t(M) \in \mathrm{Tran}_\nu(t)$. Due to the upward closeness of both $\mathrm{Tran}_\mu(s)$ and $\mathrm{Tran}_\nu(t)$ in the case of a positive left-hand side, the equivalence of (1) and (3) follows. □

Theorem 6. *Modal refinement as defined on PMTS coincides with the standard modal refinement notions on MTS, DMTS and OTS. On implementations it coincides with bisimulation.*

Proof. The fact that Definition 3 coincides with modal refinement on OTS as defined in [4] is a straightforward corollary of Lemma 5 and its proof. Indeed, the two conditions given in [4] are exactly conditions (3) and (5). As the definition of modal refinement on OTS coincides with modal refinement on DMTS (as shown in [4]) and thus also on MTS, the proof is done.

However, for the reader's convenience, we present a direct proof that Definition 3 coincides with modal refinement on MTS. Assume a parameter-free PMTS

Table 1: Complexity of modal refinement checking of parameter-free systems

	Boolean	Positive	pCNF	pDNF	MTS
Boolean	Π_2^{p}-complete	coNP-complete	$\in$ coNP P-hard	coNP-complete	$\in$ coNP P-hard
Positive	Π_2^{p}-complete	coNP-complete	P-complete	coNP-complete	P-complete
pCNF	Π_2^{p}-complete	coNP-complete	P-complete	coNP-complete	P-complete
pDNF	Π_2^{p}-complete	P-complete	P-complete	P-complete	P-complete
MTS	Π_2^{p}-complete	P-complete	P-complete	P-complete	P-complete
Impl	NP-complete	P-complete	P-complete	P-complete	P-complete

(S, T, P, Φ) where $\Phi(s)$ is a conjunction of transitions for all $s \in S$, in other words it is a standard MTS where the must transitions are listed in the conjunction and the may transitions are simply present in the underlying transition system but not a part of the conjunction. Observe that every transition $(s, a, t) \in T$ is contained in some $M \in \mathrm{Tran}_{\emptyset}(s)$. Further, each must transition $(s, a, t) \in T$ is contained in all $M \in \mathrm{Tran}_{\emptyset}(s)$. Therefore, the first conjunct in Definition 3 requires that for all may transition from s there be a corresponding one from t with the successors in the refinement relation. Similarly, the second conjunct now requires that for all must transitions from t there be a corresponding must transition from s. This is exactly the standard notion of modal refinement as introduced in [11]. □

3 Complexity of Modal Refinement Checking

We shall now investigate the complexity of refinement checking on PMTS and its relevant subclasses. Without explicitly mentioning it, we assume that all considered PMTS are now finite and the decision problems are hence well defined. The complexity bounds include classes from the polynomial hierarchy (see e.g. [14]) where for example $\Sigma_0^{\mathrm{p}} = \Pi_0^{\mathrm{p}} = \mathrm{P}$, $\Pi_1^{\mathrm{p}} = \mathrm{coNP}$ and $\Sigma_1^{\mathrm{p}} = \mathrm{NP}$.

3.1 Parameter-Free Systems

Since even the parameter-free systems have interesting expressive power and the complexity of refinement on OTS has not been studied before, we first focus on parameter-free systems. Moreover, the results of this subsection are then applied to parametric systems in the next subsection. The results are summarized in Table 1. The rows in the table correspond to the restrictions on the left-hand side PMTS while the columns correspond to the restrictions on the right-hand side PMTS. Boolean denotes the general system with arbitrary negation. Positive denotes the positive systems, in this case exactly OTS. We use pCNF and pDNF to denote positive systems with formulae in conjunctive and disjunctive normal forms, respectively. In this case, pCNF coincides with DMTS. The special case of satisfaction relation, where the refining system is an implementation is denoted by Impl. We do not include Impl to the columns as it makes sense that an implementation is refined only to an implementation and here modal refinement

corresponds to bisimilarity that is P-complete [2] (see also [16]). The P-hardness is hence the obvious lower bound for all the problems mentioned in the table.

We start with the simplest NP-completeness result.

Proposition 7. *Modal refinement between an implementation and a parameter-free PMTS is NP-complete.*

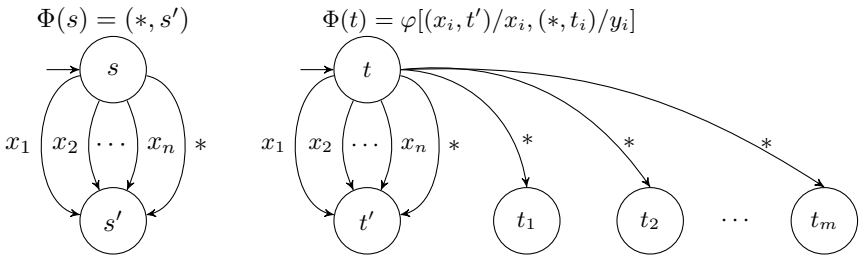
Proof. The containment part is straightforward. First we guess the relation R. As s is an implementation then the set $\mathrm{Tran}_\emptyset(s)$ is a singleton. We thus only need to further guess $N \in \mathrm{Tran}_\nu(t)$ and then in polynomial time verify the two conjuncts in Definition 3.

The hardness part is by a simple reduction from SAT. Let $\varphi(x_1, \ldots, x_n)$ be an given Boolean formula (instance of SAT). We construct two PMTSs (S, T, P, Φ) and (S', T', P', Φ') such that (i) $S = \{s, s'\}, T = (s, a, s'), P = \emptyset$, $\Phi(s) = (a, s')$ and $\Phi(s') = \mathbf{tt}$ and (ii) $S' = \{t, t_1, \ldots, t_n\}$, $T = \{(t, a, t_i) \mid 1 \leq i \leq n.\}$, $P' = \emptyset$, $\Phi(t) = \varphi[(a, t_i)/x_i]$ and $\Phi(t_i) = \mathbf{tt}$ for all i, $1 \leq i \leq n$. Clearly, φ is satisfiable if and only if $s \leq_\mathrm{m} t$. □

Next we show that modal refinement is Π_2^p-complete. The following lemma introduces a gadget used also later on in other hardness results. We will refer to it as the $*$-*construction*.

Proposition 8. *Modal refinement between two parameter-free PMTS is* Π_2^p*-hard even if the left-hand side is an MTS.*

Proof. The proof is by polynomial time reduction from the validity of the quantified Boolean formula $\psi \equiv \forall x_1 \ldots \forall x_n \exists y_1 \ldots \exists y_m : \varphi(x_1, \ldots, x_n, y_1, \ldots, y_m)$ to the refinement checking problem $s \leq_\mathrm{m} t$ where s and t are given as follows.



Assume that ψ is true. Let $M \in \mathrm{Tran}_\emptyset(s)$ (clearly $(*, s') \in M$) and we want to argue that there is $N \in \mathrm{Tran}_\emptyset(t)$ with $(*, t') \in N$ such that for all $(x_i, s') \in M$ there is $(x_i, t') \in N$ (clearly the states s', t' and t_i are in modal refinement) and for all $(x_i, t') \in N$ there is $(x_i, s') \in M$. Such an N can be found by simply including (x_i, t') whenever $(x_i, s') \in M$ and by adding also $(*, t')$ into N. As ψ is true, we include into N also all $(*, t_i)$ whenever y_i is set to true in ψ. Hence we get $s \leq_\mathrm{m} t$.

On the other hand if ψ is false then we pick $M \in \mathrm{Tran}_\emptyset(s)$ such that M corresponds to the values of x_i's such that there are no values of $y_1, \ldots, y_m$ that make ψ true. This means that from t there will be no transitions as $\mathrm{Tran}_\emptyset(t) = \emptyset$

assuming that (x_i, t') have to be set to true whenever $(x_i, s') \in M$, otherwise the refinement between s and t will fail. However, now $(*, s') \in M$ cannot be matched from t and hence $s \not\leq_{\mathrm{m}} t$. □

Proposition 9. *Modal refinement between two parameter-free PMTS is in* Π_2^{p}.

Proof. The containment follows directly from Definition 3 (note that the parameters are empty) and the fact that the last conjunction in Definition 3 is polynomially verifiable once the sets M and N were fixed. The relation R could be in principle guessed before it is verified, however, this would increase the complexity bound to Σ_3^{p}. Instead, we will initially include all pairs (polynomially many) into R and for each pair ask whether for every M there is N such that the two conjuncts are satisfied. If it fails, we remove the pair and continue until we reach (after polynomially many steps) the greatest fixed point. The complexity in this way remains in Π_2^{p}. We shall use this standard method also in further proofs and refer to it as a co-inductive computation of R. □

Positive Right-Hand Side. We have now solved all the cases where the right-hand side is arbitrary. We now look at the cases where the right-hand side is positive. In the proofs that follow we shall use the alternative characterization of refinement from Lemma 5. The following proposition determines the subclasses on which modal refinement can be decided in polynomial time.

Proposition 10. *Modal refinement on parameter-free PMTS is in P, provided that both sides are positive and either the left-hand side is in pDNF or the right-hand side is in pCNF.*

Proof. Due to Lemma 5, the refinement is equivalent to the conjunction of (3) and (2). Clearly, (3) can be checked in P. We show that Condition (2) can be verified in P too. Recall that (2) says that

$$\forall M \in \mathrm{Tran}_\mu(s) : \mathrm{match}_t(M) \in \mathrm{Tran}_\nu(t)$$

where $\mathrm{match}_t(M) = \{(a, t') \in T(t) \mid \exists (a, s') \in M : (s', t') \in R\}$.

First assume that the left-hand side is in pDNF. If for some M the Condition (2) is satisfied then it is also satisfied for all $M' \supseteq M$, as $\mathrm{Tran}_\mu(s)$ is upwards closed. It it thus sufficient to verify the condition for all minimal elements (wrt. inclusion) of $\mathrm{Tran}_\mu(s)$. In this case it correspond to the clauses of $\Phi(s)$. Thus we get a polynomial time algorithm as shown in Algorithm 1.

Second, assume that the right-hand side is in pCNF. Note that Condition (2) can be equivalently stated as

$$\forall M : \mathrm{match}_t(M) \notin \mathrm{Tran}_\nu(t) \Rightarrow M \notin \mathrm{Tran}_\mu(s) \tag{6}$$

As $\Phi(t)$ is in conjunctive normal form then $N \in \mathrm{Tran}_\nu(t)$ is equivalent to saying that N has nonempty intersection with each clause of $\Phi(t)$. We may thus enumerate all maximal $N \notin \mathrm{Tran}_\nu(t)$. Having a maximal $N \notin \mathrm{Tran}_\nu(t)$, we can easily construct M such that $N = \mathrm{match}_t(M)$. This leads to the polynomial time Algorithm 2.

The statement of the proposition thus follows. □

Algorithm 1: Test for Condition (2) of modal refinement (pDNF)

Input : states s and t such that $\Phi(s)$ is in positive DNF and $\Phi(t)$ is positive, relation R
Output: *true* if s, t satisfy the refinement condition, *false* otherwise
foreach *clause* $(a_1, s_1) \wedge \cdots \wedge (a_k, s_k)$ *in* $\Phi(s)$ **do**
  $N \leftarrow \{(a, t') \in T(t) \mid \exists i : a_i = a \wedge (s_i, t') \in R\}$;
  if $N \notin \mathrm{Tran}_\nu(t)$ **then return** false;
return true;

Algorithm 2: Test for Condition (2) of modal refinement (pCNF)

Input : states s and t such that $\Phi(s)$ is positive and $\Phi(t)$ is in positive CNF, relation R
Output: *true* if s, t satisfy the refinement condition, *false* otherwise
foreach *clause* $(a_1, t_1) \vee \cdots \vee (a_k, t_k)$ *in* $\Phi(t)$ **do**
  $M \leftarrow T(s) \setminus \{(a, s') \in T(s) \mid \exists i : a_i = a \wedge (s', t_i) \in R\}$;
  if $M \in \mathrm{Tran}_\mu(s)$ **then return** false;
return true;

Proposition 11. *Modal refinement on parameter-free PMTS is in coNP, if the right-hand side is positive.*

Proof. Due to Lemma 5 we can solve the two refinement conditions separately. Furthermore, both Condition (1) an (2) of Lemma 5 can be checked in coNP. The guessing of R is done co-inductively as described in the proof of Proposition 9. □

Proposition 12. *Modal refinement on parameter-free systems is coNP-hard, even if the left-hand side is in positive CNF and the right-hand side is in positive DNF.*

Proof. We reduce SAT into non-refinement. Let $\varphi(x_1, \ldots, x_n)$ be a formula in CNF. We modify φ into an equivalent formula φ' as follows: add new variables $\tilde{x}_1, \ldots, \tilde{x}_n$ and for all i change all occurrences of $\neg x_i$ into $\tilde{x}_i$ and add new clauses $(x_i \vee \tilde{x}_i)$ and $(\neg x_i \vee \neg \tilde{x}_i)$.

Observe now that all clauses contain either all positive literals or all negative literals. Let ψ^+ denote a CNF formula that contains all positive clauses of φ' and ψ^- denote a CNF formula that contains all negative clauses of φ'. As $\varphi' = \psi^+ \wedge \psi^-$ it is clear that φ' is satisfiable if and only if $(\psi^+ \Rightarrow \neg\psi^-)$ is not valid.

Now we construct two PMTSs (S, T, P, Φ) and (S', T', P', Φ') over $\Sigma = \{x_1, \ldots, x_n, \tilde{x}_1, \ldots, \tilde{x}_n\}$ as follows: (i) $S = \{s, s'\}$, $T = \{(s, x_i, s'), (s, \tilde{x}_i, s') \mid 1 \leq i \leq n\}$, $P = \emptyset$, $\Phi(s) = \psi^+[(x_i, s')/x_i, (\tilde{x}_i, s')/\tilde{x}_i]$ and $\Phi(s') = \mathbf{tt}$, and (ii) $S' = \{t, t'\}$, $T' = \{(t, x_i, t'), (t, \tilde{x}_i, t) \mid 1 \leq i \leq n\}$, $P' = \emptyset$, $\Phi(t) = \neg\psi^-[(x_i, t')/x_i, (\tilde{x}_i, t')/\tilde{x}_i]$ and $\Phi(t') = \mathbf{tt}$. Note that by pushing the negation of ψ^- inside, this formula can be written as pDNF. It is easy to see that now $s \leq_\mathrm{m} t$ if and only if $(\psi^+ \Rightarrow \neg\psi^-)$ is valid. Therefore, $s \not\leq_\mathrm{m} t$ if and only if φ is satisfiable. □

Table 2: Complexity of modal refinement checking with parameters

	Boolean	positive	pCNF	pDNF
Boolean	Π_4^{p}-complete	Π_3^{p}-complete	$\in \Pi_3^{\mathrm{p}}$ Π_2^{p}-hard	Π_3^{p}-complete
positive	Π_4^{p}-complete	Π_3^{p}-complete	Π_2^{p}-complete	Π_3^{p}-complete
pCNF	Π_4^{p}-complete	Π_3^{p}-complete	Π_2^{p}-complete	Π_3^{p}-complete
pDNF	Π_4^{p}-complete	Π_2^{p}-complete	Π_2^{p}-complete	Π_2^{p}-complete
MTS	Σ_3^{p}-complete	NP-complete	NP-complete	NP-complete
Impl	NP-complete	NP-complete	NP-complete	NP-complete

Note that the exact complexity of modal refinement with the right-hand side being in positive CNF or MTS and the left-hand side Boolean remains open.

3.2 Systems with Parameters

In the sequel we investigate the complexity of refinement checking in the general case of PMTS with parameters. The complexities are summarized in Table 2. We start with an observation of how the results on parameter-free systems can be applied to the parametric case.

Proposition 13. *The complexity upper bounds from Table 1 carry over to Table 2, as follows. If the modal refinement in the parameter-free case is in* NP, coNP *or* Π_2^{p}*, then the modal refinement with parameters is in* Π_2^{p}, Π_3^{p} *and* Π_4^{p}*, respectively. Moreover, if the left-hand side is an MTS, the complexity upper bounds shift from* NP *and* Π_2^{p} *to* NP *and* Σ_3^{p}*, respectively.*

Proof. In the first case, we first universally choose μ, we then existentially choose ν and modify the formulae $\Phi(s)$ and $\Phi(t)$ by evaluating the parameters. This does not change the normal form/positiveness of the formulae. We then perform the algorithm for the parameter-free refinement. For the second case note that implementations and MTS have no parameters and we may simply choose (existentially) ν and run the algorithm for the parameter-free refinement. □

We now focus on the respective lower bounds (proof of Proposition 15 can be found in [3]).

Proposition 14. *Modal refinement between an implementation and a right-hand side in positive CNF or in DNF is NP-hard.*

Proof. The proof is by reduction from SAT. Let $\varphi(x_1, \ldots, x_n)$ be a formula in CNF and let φ_1, φ_2, …, φ_k be the clauses of φ. We construct two PMTSs (S, T, P, Φ) and (S', T', P', Φ') over the action alphabet $\Sigma = \{a_1, \ldots, a_k\}$ as follows: (i) $S = \{s, s'\}$, $T = \{(s, a_i, s') \mid 1 \leq i \leq k\}$, $P = \emptyset$, $\Phi(s) = \bigwedge_{1 \leq i \leq k}(a_i, s')$ and $\Phi(s') = \mathbf{tt}$ and (ii) $S' = \{t\} \cup \{t_i \mid 1 \leq i \leq k\}$, $T' = \{(t, a_i, t_i) \mid 1 \leq i \leq k\}$, $P' = \{x_1, \ldots, x_n\}$, $\Phi'(t) = \bigwedge_{1 \leq i \leq k}(a_i, t_i)$ and $\Phi'(t_i) = \varphi_i$ for all $1 \leq i \leq k$. Notice that each φ_i in $\Phi'(t_i)$ is in positive form as we negate only the parameters x_i and every clause φ_i is trivially in DNF. Now we easily get that $s \leq_{\mathrm{m}} t$ if and only if φ is satisfiable. □

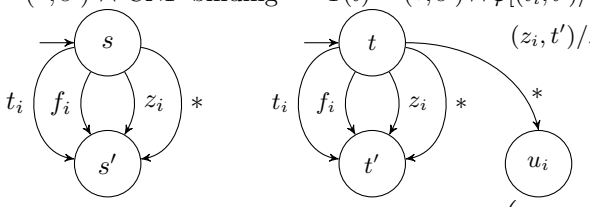
Proposition 15. *Modal refinement is Σ_3^{p}-hard even if the left-hand side is MTS.*

The following proof introduces a gadget used also later on in other hardness results. We refer to it as *CNF-binding*. Further, we use the $*$-construction here.

Proposition 16. *Modal refinement is Π_4^{p}-hard even if the left-hand side is in positive CNF.*

Proof (Sketch). Consider a Π_4^{p}-hard QSAT instance, a formula $\psi = \forall x \exists y \forall z \exists w : \varphi(x, y, z, w)$ with φ is in CNF and x, y, z, w vectors of length n. We construct two system s and t and use the variables $\{x_1, \ldots, x_n\}$ as parameters for the left-hand side system s, and $\{y_1, \ldots, y_n\}$ as parameters for the right-hand side system t.

$\Phi(s) = (*, s') \wedge$ CNF-binding $\quad \Phi(t) = (*, s') \wedge \varphi[(t_i, t')/x_i, (f_i, t')/\neg x_i,$ $(z_i, t')/z_i, (*, u_i)/w_i]$ for all $1 \leq i \leq n$



On the left we require $\Phi(s) = (*, s') \wedge \bigwedge_{1 \leq i \leq n} \Big((x_i \Rightarrow (t_i, s')) \wedge (\neg x_i \Rightarrow (f_i, s') \Big)$ and call the latter conjunct *CNF-binding*. Thus the value of each parameter x_i is "saved" into transitions of the system. Note that although both t_i and f_i may be present, a "minimal" implementation contains exactly one of them. On the right-hand side the transitions look similar but we require $\Phi(t) = (*, t) \wedge \varphi'$ where φ' is created from φ by changing every positive literal x_i into (t_i, t'), every negative literal $\neg x_i$ into (f_i, t'), every z_i into (z_i, t'), and every w_i into $(*, u_i)$.

We show that ψ is true iff $s \leq_{\mathrm{m}} t$. Assume first that ψ is true. Therefore, for every choice of parameters x_i there is a choice of parameters y_i so that $\forall z \exists w : \varphi(x, y, z, w)$ is true and, moreover, t_i or f_i is present on the left whenever x_i or $\neg x_i$ is true, respectively (and possibly even if it is false). We set exactly all these transitions t_i and f_i on the right, too. Further, for every choice of transitions z_i on the left there are w_i's so that $\varphi(x, y, z, w)$ holds. On the right, we implement a transition (z_i, t') for each z_i set to true and $(*, u_i)$ for each w_i set to true. Now φ' is satisfied as it has only positive occurrences of (t_i, t') and (f_i, t') and hence the extra t_i's and f_i's do not matter. Now for every implementation of s we obtained an implementation of t. Moreover, their transitions match. Indeed, t_i's and f_i's were set the same as on the left, similarly for z_i's. As for the $*$-transition, we use the same argumentation as in the original $*$-construction. On the left, there is always one. On the right, there can be more of them due to w_i's but at least one is also guaranteed by $\Phi(t)$.

Let now $s \leq_{\mathrm{m}} t$. Then for every choice of x_i's—and thus also for every choice of *exactly* one transition of t_i, f_i for each i—there are y_i's so that every choice of transitions z_i can be matched on the right so that φ' is true with some transitions

$(*, u_i)$. Since choices of t_i/f_i correspond exactly to choices of x_i it only remains to set w_i true for each transition $(*, u_i)$ on the right, thus making φ true. □

Based on the idea of CNF-binding, the following propositions are proved in [3].

Proposition 17. *Modal refinement is Π_3^{p}-hard for the left-hand side in positive CNF and the right-hand side in positive DNF.*

Proposition 18. *Modal refinement is Π_2^{p}-hard even if both sides are in positive CNF.*

The last three propositions use a modification of the CNF-binding idea called *DNF-binding*. Instead of $(x_i \Rightarrow (t_i, s')) \wedge (\neg x_i \Rightarrow (f_i, s'))$ we use $(x_i \wedge (t_i, s')) \vee (\neg x_i \wedge (f_i, s'))$ to bind parameters of the left-hand side system with transitions of the right-hand side system. Details are in [3].

Proposition 19. *Modal refinement is Π_2^{p}-hard even if left-hand side is in positive DNF and right-hand side is in positive CNF.*

Proposition 20. *Modal refinement is Π_2^{p}-hard even if left-hand side is in positive DNF and right-hand side is in positive DNF.*

Proposition 21. *Modal refinement is Π_4^{p}-hard even if the left-hand side is in positive DNF.*

Although the complexity may seem discouraging in many cases, there is an important remark to make. The refinement checking may be exponential, but only in the outdegree of each state and the number of parameters, while it is polynomial in the number of states. As one may expect the outdegree and the number of parameters to be much smaller than the number of states, this means that the refinement checking may still be done in a rather efficient way. This claim is furthermore supported by the existence of efficient SAT solvers that may be employed to check the inner conditions in the modal refinement.

4 Conclusion and Future Work

We have introduced an extension of modal transition systems called PMTS for parametric systems. The formalism is general enough to capture several features missing in the other extensions, while at the same time it offers an easy to understand semantics and a natural notion of modal refinement that specializes to the well-known refinements already studied on the subclasses of PMTS. Finally, we provided a comprehensive overview of complexity of refinement checking on PMTS and its subclasses.

We believe that our formalism is a step towards a more applicable notion of specification theories based on MTS. In the future work we will study logical characterizations of the refinement relation, investigate compositional properties and focus on introducing quantitative aspects into the model in order to further increase its applicability.

Acknowledgments. We would like to thank to Sebastian Bauer for suggesting the traffic light example and for allowing us to use his figure environments.

References

1. Antonik, A., Huth, M., Larsen, K.G., Nyman, U., Wasowski, A.: 20 years of modal and mixed specifications. Bulletin of the EATCS no. 95 pp. 94–129 (2008)
2. Balcazar, J.L., Gabarró, J., Santha, M.: Deciding bisimilarity is P-complete. Formal aspects of computing 4(6 A), 638–648 (1992)
3. Beneš, N., Křetínský, J., Larsen, K.G., Møller, M.H., Srba, J.: Parametric modal transition systems. Technical report FIMU-RS-2011-03, Faculty of Informatics, Masaryk University, Brno (2011)
4. Beneš, N., Křetínský, J.: Process algebra for modal transition systemses. In: Matyska, L., Kozubek, M., Vojnar, T., Zemcík, P., Antos, D. (eds.) MEMICS. OASICS, vol. 16, pp. 9–18. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany (2010)
5. Boudol, G., Larsen, K.G.: Graphical versus logical specifications. Theor. Comput. Sci. 106(1), 3–20 (1992)
6. Fecher, H., Schmidt, H.: Comparing disjunctive modal transition systems with an one-selecting variant. J. of Logic and Alg. Program. 77(1-2), 20–39 (2008)
7. Godefroid, P., Huth, M., Jagadeesan, R.: Abstraction-based model checking using modal transition systems. In: Proc. CONCUR'01. LNCS, vol. 2154, pp. 426–440. Springer (2001)
8. Gruler, A., Leucker, M., Scheidemann, K.D.: Modeling and model checking software product lines. In: Barthe, G., de Boer, F.S. (eds.) FMOODS. Lecture Notes in Computer Science, vol. 5051, pp. 113–131. Springer (2008)
9. Huth, M., Jagadeesan, R., Schmidt, D.A.: Modal transition systems: A foundation for three-valued program analysis. In: Proc. of ESOP'01. LNCS, vol. 2028, pp. 155–169. Springer (2001)
10. Larsen, K.G., Nyman, U., Wasowski, A.: On modal refinement and consistency. In: Proc. of CONCUR'07. LNCS, vol. 4703, pp. 105–119. Springer (2007)
11. Larsen, K.G., Thomsen, B.: A modal process logic. In: LICS. pp. 203–210. IEEE Computer Society (1988)
12. Larsen, K.G., Xinxin, L.: Equation solving using modal transition systems. In: LICS. pp. 108–117. IEEE Computer Society (1990)
13. Nanz, S., Nielson, F., Nielson, H.R.: Modal abstractions of concurrent behaviour. In: Proc. of SAS'08. LNCS, vol. 5079, pp. 159–173. Springer (2008)
14. Papadimitriou, C.H.: Computational complexity. Addison-Wesley Publishing Co., Inc., Reading, MA, USA (1994)
15. Raclet, J.B., Badouel, E., Benveniste, A., Caillaud, B., Passerone, R.: Why are modalities good for interface theories? In: ACSD. pp. 119–127. IEEE (2009)
16. Sawa, Z., Jančar, P.: Behavioural equivalences on finite-state systems are PTIME-hard. Computing and informatics 24(5), 513–528 (2005)
17. Uchitel, S., Chechik, M.: Merging partial behavioural models. In: Proc. of FSE'04. pp. 43–52. ACM (2004)

Paper D:

Dual-priced modal transition systems with time durations

Nikola Beneš, Jan Křetínský, Kim G. Larsen, Mikael H. Moller, and Jiri Srba

This paper has been published in Nikolaj Bjørner and Andrei Voronkov (eds.): Logic for Programming, Artificial Intelligence, and Reasoning - 18th International Conference, LPAR-18, Mérida, Venezuela, March 11-15, 2012. Proceedings, volume 7180 of Lecture Notes in Computer Science, pages 122–137. Springer, 2012. [BKL+12]

Summary

We extend BMTS and introduce time durations of transitions. The time is chosen non-deterministically from an interval either by the implementator or by the uncontrollable environment at run time. Further, we introduce price for performing an action per unit time paid at run time and a price for implementing an action paid once at the implementation time. The latter models the hardware investment cost while the former the running cost of the resulting implementation. We show how to construct the "cheapest" implementation, i.e. an implementation with the lowest average running cost for a given maximum investment cost. To this end, we reduce our problem to a new, weighted, extension of mean-payoff games. Further, we show that deciding whether there is an implementation with both costs under given thresholds is NP-complete. However, for a certain still very general subclass we provide an optimization yielding a pseudo-polynomial algorithm.

Author's contribution: 30 %

- participating in the discussions,
- contributing, in particular, to the formulation of time durations mechanism and solution to the cheapest implementation problem using mean-payoff games,
- writing parts of the paper.

Dual-Priced Modal Transition Systems with Time Durations*

Nikola Beneš[2]** Jan Křetínský[2,3]*** Kim G. Larsen[1]
Mikael H. Møller[1] Jiří Srba[1]

[1] Aalborg University, Denmark
[2] Masaryk University, Czech Republic
[3] Technical University München, Germany

Abstract. Modal transition systems are a well-established specification formalism for a high-level modelling of component-based software systems. We present a novel extension of the formalism called modal transition systems with durations where time durations are modelled as controllable or uncontrollable intervals. We further equip the model with two kinds of quantitative aspects: each action has its own running cost per time unit, and actions may require several hardware components of different costs. We ask the question, given a fixed budget for the hardware components, what is the implementation with the cheapest long-run average reward. We give an algorithm for computing such optimal implementations via a reduction to a new extension of mean payoff games with time durations and analyse the complexity of the algorithm.

1 Introduction and Motivating Example

Modal Transition Systems (MTS) is a specification formalism [16, 2] that aims at providing a flexible and easy-to-use compositional development methodology for reactive systems. The formalism can be viewed as a fragment of a temporal logic [1, 9] that at the same time offers a behavioural compositional semantics with an intuitive notion of process refinement. The formalism of MTS is essentially a labelled transition system that distinguishes two types of labelled transitions: *must* transitions which are required in any refinement of the system, and *may* transitions that are allowed to appear in a refined system but are not required. The refinement of an MTS now essentially consists of iteratively resolving the presence or absence of may transitions in the refined process.

In a recent line of work [15, 3], the MTS framework has been extended to allow for the specification of additional constraints on quantitative aspects (e.g. time, power or memory), which are highly relevant in the area of embedded systems. In this paper we continue the pursuit of quantitative extensions of MTS

* Supported by VKR Center of Excellence MT-LAB.
** The author has been supported by Czech Grant Agency, grant no. GAP202/11/0312.
*** The author is a holder of Brno PhD Talent Financial Aid and is supported by the Czech Science Foundation, grant No. P202/10/1469.

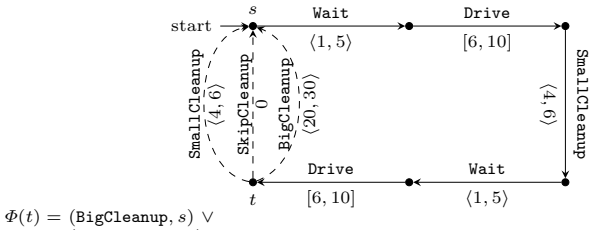
by presenting a novel extension of MTS with time durations being modelled as controllable or uncontrollable intervals. We further equip the model with two kinds of quantitative aspects: each action has its own running cost per time unit, and actions may require several hardware components of different costs. Thus, we ask the question, given a fixed budget for the investment into the hardware components, what is the implementation with the cheapest long-run average reward.

Before we give a formal definition of modal transition systems with durations (MTSD) and the dual-price scheme, and provide algorithms for computing optimal implementations, we present a small motivating example.

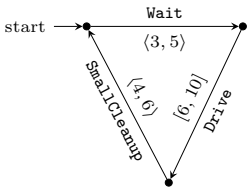
Consider the specification $\mathcal{S}$ in Figure 1a describing the work of a shuttle bus driver. He drives a bus between a hotel and the airport. First, the driver has to `Wait` for the passengers at the hotel. This can take one to five minutes. Since this behaviour is required to be present in all the implementations of this specification, it is drawn as a solid arrow and called a *must* transition. Then the driver has to `Drive` the bus to the airport (this takes six to ten minutes) where he has to do a `SmallCleanup`, then `Wait` before he can `Drive` the bus back to the hotel. When he returns he can do either a `SmallCleanup`, `BigCleanup` or `SkipCleanup` of the bus before he continues. Here we do not require a particular option to be realised in the implementations, hence we only draw the transitions as dashed arrows. As these transitions may or may not be present in the implementations, they are called *may* transitions. However, here the intention is to require at least one option be realised. Hence, we specify this using a propositional formula Φ assigned to the state t over its outgoing transitions as described in [5, 6]. After performing one of the actions, the driver starts over again. Note that next time the choice in t may differ.

Observe that there are three types of durations on the transitions. First, there are *controllable* intervals, written in angle brackets. The meaning of e.g. $\langle 1, 5\rangle$ is that in the implementation we can instruct the driver to wait for a fixed number of minutes in the range. Second, there are *uncontrollable* intervals, written in square brackets. The interval $[6, 10]$ on the `Drive` transition means that in the implementation we cannot fix any particular time and the time can vary, say, depending on the traffic and it is chosen nondeterministically by the environment. Third, the degenerated case of a single number, e.g. 0, denotes that the time taken is always constant and given by this number. In particular, a zero duration means that the transition happens instantaneously.

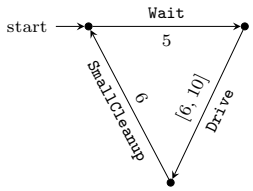
The system $\mathcal{S}_1$ is another specification, a *refinement* of $\mathcal{S}$, where we additionally specify that the driver must do a `SmallCleanup` after each `Drive`. Note that the `Wait` interval has been narrowed. The system $\mathcal{I}_1$ is an implementation of $\mathcal{S}_1$ (and actually also of $\mathcal{S}$) where all controllable time intervals have already been fully resolved to their final single values: the driver must `Wait` for 5 minutes and do the `SmallCleanup` for 6 minutes. Note that uncontrollable intervals remain unresolved in the implementations and the time is chosen by the environment each time the action is performed. This reflects the inherent uncontrollable uncertainty of the timing, e.g. of a traffic.



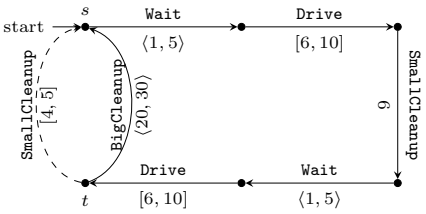
(a) Specification $\mathcal{S}$



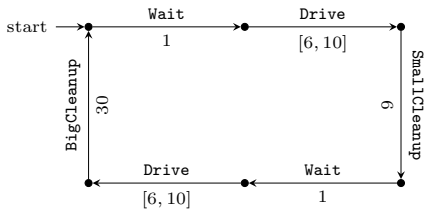
(b) Specification $\mathcal{S}_1$



(c) Implementation $\mathcal{I}_1$



(d) Specification $\mathcal{S}_2$



(e) Implementation $\mathcal{I}_2$

$a \in \Sigma$	$r(a)$
Wait	8
Drive	10
SmallCleanup	6
BigCleanup	7
SkipCleanup	0

$H = \{\texttt{VacuumCleaner}, \texttt{Sponge}\}$

$$\Psi(a) = \begin{cases} \texttt{VacuumCleaner} & \text{if } a = \texttt{BigCleanup} \\ \texttt{Sponge} \vee \texttt{VacuumCleaner} & \text{if } a = \texttt{SmallCleanup} \\ true & \text{otherwise} \end{cases}$$

$h \in H$	$i(h)$
VacuumCleaner	100
Sponge	5

(f) Price Scheme

Fig. 1: Example of Dual-Priced Modal Transition Systems with Time Durations

The system $\mathcal{S}_2$ is yet another specification and again a refinement of $\mathcal{S}$, where the driver can always do a `BigCleanup` in t and possibly there is also an alternative allowed here of a `SmallCleanup`. Notice that both `SmallCleanup` intervals have been restricted and changed to uncontrollable. This means that we give up the control over the duration of this action and if this transition is implemented, its duration will be every time chosen nondeterministically in that range. Finally, $\mathcal{I}_2$ is then an implementation of $\mathcal{S}_2$ and $\mathcal{S}$.

Furthermore, we develop a way to model cost of resources. Each action is assigned a *running price* it costs per time unit, e.g. `Drive` costs 10 each time unit it is being performed as it can be seen in the left table of Figure 1f. In addition, in order to perform an action, some hardware may be needed, e.g. a `VacuumCleaner` for the `BigCleanup` and its price is 100 as can be seen on the right. This *investment price* is paid once only.

Let us now consider the problem of finding an optimal implementation, so that we spend the least possible amount of money (e.g. the pay to the driver) per time unit while conforming to the specification $\mathcal{S}$. We call this problem *the cheapest implementation problem*. The optimal implementation is to buy a vacuum cleaner if one can afford an investment of 100 and do the `BigCleanup` every time as long as possible and `Wait` as shortly as possible. (Note that `BigCleanup` is more costly per time unit than `SmallCleanup` but lasts longer.) This is precisely implemented in $\mathcal{I}_2$ and the (worst-case) average cost per time unit is ≈ 7.97. If one cannot afford the vacuum cleaner but only a sponge, the optimal worst case long run average is then a bit worse and is implemented by doing the `SmallCleanup` as long as possible and `Wait` now as *long* as possible. This is depicted in $\mathcal{I}_1$ and the respective average cost per time unit is ≈ 8.10.

The most related work is [12] where prices are introduced into a class of interface theories and long-run average objectives are discussed. Our work omits the issue of distinguishing input and output actions. Nevertheless, compared to [12], this paper deals with the time durations, the one-shot hardware investment and, most importantly, refinement of specifications. Further, timed automata have also been extended with prices [4] and the long-run average reward has been computed in [10]. However, priced timed automata lack the hardware and any notion of refinement, too.

The paper is organized as follows. We introduce the MTS with the time durations in Section 2 and the dual-price scheme in Section 3. Section 4 presents the main results on the complexity of the cheapest implementation problem. First, we state the complexity of this problem in general and in an important special case and prove the hardness part. The algorithms proving the complexity upper bounds are presented only after introducing an extension of mean payoff games with time durations. These are needed to establish the results but are also interesting on their own as discussed in Section 4.1. Due to space limitations, some of the proofs are in the full version of the paper [7]. We conclude and give some more account on related and future work in Section 5.

2 Modal Transition Systems with Durations

In order to define MTS with durations, we first introduce the notion of *controllable* and *uncontrollable* duration intervals. A controllable interval is a pair $\langle m, n\rangle$ where $m, n \in \mathbb{N}_0$ and $m \leq n$. Similarly, an uncontrollable interval is a pair $[m, n]$ where $m, n \in \mathbb{N}_0$ and $m \leq n$. We denote the set of all controllable intervals by $\mathfrak{I}_{\mathsf{c}}$, the set of all uncontrollable intervals by $\mathfrak{I}_{\mathsf{u}}$, and the set of all intervals by $\mathfrak{I} = \mathfrak{I}_{\mathsf{c}} \cup \mathfrak{I}_{\mathsf{u}}$. We also write only m to denote the singleton interval $[m, m]$.

Singleton controllable intervals need not be handled separately as there is no semantic difference to the uncontrollable counterpart.

We can now formally define modal transition systems with durations. In what follows, $\mathcal{B}(X)$ denotes the set of propositional logic formulae over the set X of atomic propositions, where we assume the standard connectives $\wedge, \vee, \neg$.

Definition 1 (MTSD). *A* Modal Transition System with Durations *(MTSD) is a tuple* $\mathcal{S} = (S, T, D, \Phi, s_0)$ *where* S *is a set of* states *with the* initial state s_0*,* $T \subseteq S \times \Sigma \times S$ *is a set of* transitions*,* $D : T \to \mathfrak{I}$ *is a* duration interval function*, and* $\Phi : S \to \mathcal{B}(\Sigma \times S)$ *is an* obligation function*. We assume that whenever the atomic proposition* (a, t) *occurs in the Boolean formula* $\Phi(s)$ *then also* $(s, a, t) \in T$*.*

We moreover require that there is no cycle of transitions that allows for zero accumulated duration, i.e. there is no path $s_1 a_1 s_2 a_2 \cdots s_n$ *where* $(s_i, a_i, s_{i+1}) \in T$ *and* $s_n = s_1$ *such that for all* i*, the interval* $D((s_i, a_i, s_{i+1}))$ *is of the form either* $\langle 0, m\rangle$ *or* $[0, m]$ *for some* m*.*

Note that instead of the basic may and must modalities known from the classical modal transition systems (see e.g. [2]), we use arbitrary boolean formulae over the outgoing transitions of each state in the system as introduced in [6]. This provides a higher generality as the formalism is capable to describe, apart from standard modal transition systems, also more expressive formalisms like disjunctive modal transition systems [17] and transition systems with obligations [5]. See [6] for a more thorough discussion of this formalism.

In the rest of the paper, we adapt the following convention when drawing MTSDs. Whenever a state s is connected with a solid arrow labelled by a to a state s', this means that in any satisfying assignment of the Boolean formula $\Phi(s)$, the atomic proposition (a, s') is always set to true (the transition *must* be present in any refinement of the system). Should this not be the case, we use a dashed arrow instead (meaning that the corresponding transition *may* be present in a refinement of the system but it can be also left out). For example the solid edges in Figure 1a correspond to an implicitly assumed $\Phi(s) = (a, s')$ where (s, a, s') is the (only) outgoing edge from s; in this case we do not explicitly write the obligation function. The three dashed transitions in the figure are optional, though at least one of them has to be preserved during any refinement (a feature that can be modelled for example in disjunctive MTS [17]).

Remark 2. The standard notion of modal transition systems (see e.g. [2]) is obtained under the restriction that the formulae $\Phi(s)$ in any state $s \in S$ have the form $(a_1, s_1) \wedge \ldots \wedge (a_n, s_n)$ where $(s, a_1, s_1), \ldots, (s, a_n, s_n) \in T$. The edges mentioned in such formulae are exactly all must transitions; may transitions are not listed in the formula and hence can be arbitrarily set to true or false.

Let by $T(s) = \{(a, t) \mid (s, a, t) \in T\}$ denote the set of all outgoing transitions from the state $s \in S$. A modal transition system with durations is called an *implementation* if $\Phi(s) = \bigwedge T(s)$ for all $s \in S$ (every allowed transition is also

required), and $D(s, a, s') \in \mathcal{I}_{\mathsf{u}}$ for all $(s, a, s') \in T$, i.e. all intervals are uncontrollable, often singletons. Figure 1c shows an example of an implementation, while Figure 1b is not yet an implementation as it still contains the controllable intervals $\langle 3, 5\rangle$ and $\langle 4, 6\rangle$.

We now define a notion of modal refinement. In order to do that, we first need to define refinement of intervals as a binary relation $\leq \subseteq \mathfrak{I} \times \mathfrak{I}$ such that

- $\langle m', n'\rangle \leq \langle m, n\rangle$ whenever $m' \geq m$ and $n' \leq n$, and
- $[m', n'] \leq \langle m, n\rangle$ whenever $m' \geq m$ and $n' \leq n$.

Thus controllable intervals can be refined by narrowing them, at most until they become singleton intervals, or until they are changed to uncontrollable intervals. Let us denote the collection of all possible sets of outgoing transitions from a state s by $\mathrm{Tran}(s) := \{E \subseteq T(s) \mid E \models \Phi(s)\}$ where $\models$ is the classical satisfaction relation on propositional formulae assuming that E lists all true propositions.

Definition 3 (Modal Refinement). *Let $\mathcal{S}_1 = (S_1, T_1, D_1, \Phi_1, s_1)$ and $\mathcal{S}_2 = (S_2, T_2, D_2, \Phi_2, s_2)$ be two MTSDs. A binary relation $R \subseteq S_1 \times S_2$ is a* modal refinement *if for every $(s, t) \in R$ the following holds:*

$$\begin{aligned}
&\forall M \in \mathrm{Tran}(s) : \exists N \in \mathrm{Tran}(t) : \\
&\qquad \forall (a, s') \in M : \exists (a, t') \in N : D_1(s, a, s') \leq D_2(t, a, t') \ \wedge \ (s', t') \in R \text{ and} \\
&\qquad \forall (a, t') \in N : \exists (a, s') \in M : D_1(s, a, s') \leq D_2(t, a, t') \ \wedge \ (s', t') \in R \ .
\end{aligned}$$

We say that $s \in S_1$ modally refines $s' \in S_2$, denoted by $s \leq_{\mathrm{m}} s'$, if there exists a modal refinement R such that $(s, s') \in R$. We also write $\mathcal{S}_1 \leq_{\mathrm{m}} \mathcal{S}_2$ if $s_1 \leq_{\mathrm{m}} s_2$.

Intuitively, the pair (s, t) can be in the relation R if for any satisfiable instantiation of outgoing edges from s there is a satisfiable instantiation of outgoing edges from t so that they can be mutually matched, possibly with s having more refined intervals, and the resulting states are again in the relation R.

Observe that in our running example the following systems are in modal refinement: $\mathcal{I}_1 \leq_{\mathrm{m}} \mathcal{S}_1 \leq_{\mathrm{m}} \mathcal{S}$ and thus also $\mathcal{I}_1 \leq_{\mathrm{m}} \mathcal{S}$, and similarly $\mathcal{I}_2 \leq_{\mathrm{m}} \mathcal{S}_2 \leq_{\mathrm{m}} \mathcal{S}$ and thus also $\mathcal{I}_2 \leq_{\mathrm{m}} \mathcal{S}$.

The reader can verify that on the standard modal transition systems (see Remark 2) the modal refinement relation corresponds to the classical modal refinement as introduced in [16].

3 Dual-Price Scheme

In this section, we formally introduce a dual-price scheme on top of MTSD in order to model the *investment cost* (cost of hardware necessary to perform the implemented actions) and the *running cost* (weighted long-run average of running costs of actions). We therefore consider only deadlock-free implementations (every state has at least one outgoing transition) so that the long-run average reward is well defined.

Definition 4 (Dual-Price Scheme). *A* dual-price scheme *over an alphabet* Σ *is a tuple* $\mathcal{P} = (r, H, \Psi, i)$ *where*

- $r : \Sigma \to \mathbb{Z}$ *is a* running cost *function of actions per time unit,*
- H *is a finite set of available* hardware*,*
- $\Psi : \Sigma \to \mathcal{B}(H)$ *is a* hardware requirement *function, and*
- $i : H \to \mathbb{N}_0$ *is a hardware* investment cost *function.*

Hence every action is assigned its unit cost and every action can have different hardware requirements (specified as a Boolean combination of hardware components) on which it can be executed. This allows for much more variability than a possible alternative of a simple investment cost $\Sigma \to \mathbb{N}_0$. Further, observe that the running cost may be negative, meaning that execution of such an action actually gains rather than spends resources.

Let $\mathcal{I}$ be an implementation with an initial state s_0. A set $G \subseteq H$ of hardware is *sufficient* for an implementation $\mathcal{I}$, written $G \models \mathcal{I}$, if $G \models \Psi(a)$ for every action a reachable from s_0. The *investment cost* of $\mathcal{I}$ is then defined as

$$\mathsf{ic}(\mathcal{I}) = \min_{G \models \mathcal{I}} \sum_{g \in G} i(g) \ .$$

Further, a *run* of $\mathcal{I}$ is an infinite sequence $s_0 a_0 t_0 s_1 a_1 t_1 \cdots$ with $(s_i, a_i, s_{i+1}) \in T$ and $t_i \in D(s_i, a_i, s_{i+1})$. Hence, in such a run, a concrete time duration in each uncontrollable interval is selected. We denote the set of all runs of $\mathcal{I}$ by $\mathcal{R}(\mathcal{I})$. The *running cost* of an implementation $\mathcal{I}$ is the worst-case long-run average

$$\mathsf{rc}(\mathcal{I}) = \sup_{s_0 a_0 t_0 s_1 a_1 t_1 \cdots \in \mathcal{R}(\mathcal{I})} \limsup_{n \to \infty} \frac{\sum_{i=0}^{n} r(a_i) \cdot t_i}{\sum_{i=0}^{n} t_i} \ .$$

Our *cheapest-implementation problem* is now defined as follows: given an MTSD specification $\mathcal{S}$ together with a dual-price scheme over the same alphabet, and given an upper-bound max_{ic} for the investment cost, find an implementation $\mathcal{I}$ of $\mathcal{S}$ (i.e. $\mathcal{I} \leq_{\mathrm{m}} \mathcal{S}$) such that $\mathsf{ic}(\mathcal{I}) \leq max_{\mathsf{ic}}$ and for every implementation $\mathcal{I}'$ of $\mathcal{S}$ with $\mathsf{ic}(\mathcal{I}') \leq max_{\mathsf{ic}}$, we have $\mathsf{rc}(\mathcal{I}) \leq \mathsf{rc}(\mathcal{I}')$.

Further, we introduce the respective decision problem, the *implementation problem*, as follows: given an MTSD specification $\mathcal{S}$ together with a dual-price scheme, and given an upper-bound max_{ic} for the investment cost and an upper bound max_{rc} on the running cost, decide whether there is an implementation $\mathcal{I}$ of $\mathcal{S}$ such that both $\mathsf{ic}(\mathcal{I}) \leq max_{\mathsf{ic}}$ and $\mathsf{rc}(\mathcal{I}) \leq max_{\mathsf{rc}}$.

Example 5. Figure 1f depicts a dual-price scheme over the same alphabet $\Sigma = \{\texttt{Wait}, \texttt{Drive}, \texttt{SmallCleanup}, \texttt{BigCleanup}, \texttt{SkipCleanup}\}$ as of our motivating specification $\mathcal{S}$. The running cost of the implementation $\mathcal{I}_2$ is $(1 \cdot 8 + 10 \cdot 10 + 6 \cdot 6 + 1 \cdot 8 + 10 \cdot 10 + 30 \cdot 7)/(1 + 10 + 6 + 1 + 10 + 30) \approx 7.97$ as the maximum value is achieved when `Drive` (with running cost 10) takes 10 minutes. On the one hand, this is optimal for $\mathcal{S}$ and a maximum investment cost at least 100. On the other hand, if the maximum investment cost is 99 or less then the optimal implementation is depicted in $\mathcal{I}_1$ and its cost is $(5 \cdot 8 + 10 \cdot 10 + 6 \cdot 5)/(5 + 10 + 6) \approx 8.10$.

Remark 6. Note that the definition of the dual-price scheme only relies on having durations on the labelled transition systems. Hence, one could easily apply this in various other settings like in the special case of traditional MTS (with may and must transitions instead of the obligation function) or in the more general case of parametric MTS (see [6]) when equipped with durations as described above.

4 Complexity Results

In this section, we give an overview of the complexity of our problem both in general and in an important special case. We start with establishing the hardness results. The matching upper bounds and the outline of their proofs follow. When referring to the size of MTSDs and the dual-price scheme, we implicitly assume binary encoding of numbers. We start by observing that the implementation problem is NP-hard even if no hardware is involved.

Proposition 7. *The implementation problem is NP-hard even for the hardware requirement function Ψ that is constantly true for all actions.*

Proof. We shall reduce the satisfiability problem of Boolean formulae (SAT) to our problem. Let φ be a Boolean formula over the variables $x_1, \ldots, x_n$. We define a MTSD $\mathcal{S}$ over the set of actions $\Sigma = \{x_1, \ldots, x_n, *\}$ such that the running cost is $r(x_j) = 1$ for all $1 \leq j \leq n$ and $r(*) = 2$ and the duration of all actions is 1. The specification $\mathcal{S}$ has one state s and a self-loop under all elements of Σ with the obligation function $\Phi(s) = \varphi \vee (*, s)$. The reason for adding the action $*$ is to make sure that in case φ is not satisfiable then we can still have a deadlock-free, but more running-cost-expensive implementation. Now we set the hardware to $H = \emptyset$ and the hardware requirement function $\Psi(a)$ constantly true for all $a \in \Sigma$. It is easy to observe that the formula φ is satisfiable iff $\mathcal{S}$ has an implementation $\mathcal{I}$ with $\mathsf{rc}(\mathcal{I}) \leq 1$ (and $\mathsf{ic}(\mathcal{I}) = 0$). □

Note that in the proof we required Φ to be a general Boolean formula. If, for instance, we considered Φ in *positive form* (i.e. only containing $\wedge$ and $\vee$ operators and not $\neg$), the hardness would not hold. Thus on the one hand, one source of hardness is the complexity of Φ. On the other hand, even if Φ corresponds to the simplest case of an implementation (Φ is a conjunction of atomic propositions), the problem remains hard due to the hardware.

Proposition 8. *The implementation problem is NP-hard even for specifications that are already implementations.*

Proof. We reduce the NP-complete problem of vertex cover to our problem. Let (V, E) where $E \subseteq V \times V$ be a graph and $k \in \mathbb{N}$ be an integer. We ask whether there is a subset of vertices $V_k \subseteq V$ of cardinality k such that for every $(v_1, v_2) \in E$ at least $v_1 \in V_k$ or $v_2 \in V_k$. Let us construct an MTSD specification $\mathcal{S}$ with hardware $H = V$ and the investment function $i(v) = 1$ for all $v \in H$, such that $\mathcal{S}$ has only one state s and a self-loop under a single action

a that is required ($\Phi(s) = (a, s)$) and where the hardware requirement function is $\Psi(a) = \bigwedge_{(u,v)\in E}(u \vee v)$. There is now a vertex cover in (V, E) of size k iff $\mathcal{S}$ has an implementation $\mathcal{I}$ with $\mathsf{ic}(\mathcal{I}) \leq k$. Setting e.g. $D(s, a, s) = 1$ and the running cost $r(a) = 0$ establishes NP-hardness of the implementation problem where we ask for the existence of an implementation of $\mathcal{S}$ with maximum running cost 0 and maximum investment cost k.

Alternatively, we may introduce a self-loop with a new action name $a_{(u,v)}$ for every edge (u, v) in the graph such that $\Psi(a_{(u,v)}) = u \vee v$, showing NP-hardness even for the case where the hardware requirement function is a simple disjunction of hardware components. □

In the subsequent sections, we obtain the following matching upper bound which yields the following theorem.

Theorem 9. *The implementation problem is NP-complete.*

By analysing the proof of Proposition 8, it is clear that we have to restrict the hardware requirement function before we can obtain a more efficient algorithm for the implementation problem. We do so by assuming a constant number of hardware components (not part of the input). If we at the same time require the obligation function in positive form, we obtain a simpler problem as stated in the following theorem.

Theorem 10. *The implementation problem with positive obligation function and a constant number of hardware components is polynomially equivalent to mean payoff games and thus it is in NP∩coNP and solvable in pseudo-polynomial time.*

The subsequent sections are devoted to proving Theorems 9 and 10. The algorithm to solve the implementation problem first reduces the dual-priced MTSD into a mean payoff game extended with time durations and then solves this game. This new extension of mean payoff games and an algorithm to solve them is presented in Section 4.1. The translation follows in Section 4.2. Since this translation is exponential in general, Section 4.3 then shows how to translate in polynomial time with only local exponential blow-ups where negations occur. Section 4.4 then concludes and establishes the complexity bounds.

4.1 Weighted Mean Payoff Games

We extend the standard model of mean payoff games (MPG) [14] with time durations. Not only is this extension needed for our algorithm, but it is also useful for modelling by itself. Consider, for instance, energy consumption of 2kW for 10 hours and 10kW for 2 hour, both followed by 10 hours of inactivity. Obviously, although both consumptions are 20kWh per cycle, the average consumption differs: 1kW in the former case and 20/12kW in the latter one. We also allow zero durations in order to model e.g. discrete changes of states, an essential part of our algorithm. Another extension of MPGs with dual-cost was studied in [8].

Definition 11. *A* weighted mean payoff game *is* $G = (V, V_{min}, V_{max}, E, r, d)$ *where* V *is a set of vertices partitioned into* V_{min} *and* V_{max}*,* $E \subseteq V \times V$ *is a set of edges,* $r : E \to \mathbb{Z}$ *is a rate function,* $d : E \to \mathbb{N}_0$ *is a duration function.*

It is assumed that there are no deadlocks (vertices with out-degree 0) and that there are no zero-duration cycles. The game is played by two players, *min* and *max*. The play is an infinite path such that each player picks successors in his/her vertices. The value of a play $v_0 v_1 v_2 \cdots$ is defined as:

$$\nu(v_0 v_1 v_2 \cdots) = \limsup_{n \to \infty} \frac{\sum_{i=0}^{n} r(v_i, v_{i+1}) \cdot d(v_i, v_{i+1})}{\sum_{i=0}^{n} d(v_i, v_{i+1})} \ . \qquad (*)$$

Player *min* tries to minimize this value, while *max* aims at the opposite. Let $v(s)$ denote the infimum of the values *min* can guarantee if the play begins in the vertex s, no matter what the player *max* does.

Note that the standard MPGs where edges are assigned only integer weights can be seen as weighted MPGs with rates equal to weights and durations equal to 1 on all edges.

We now show how to solve weighted MPGs by reduction to standard MPGs. We first focus on the problem whether $v(s) \geq 0$ for a given vertex s. As the durations are nonnegative and there are no zero-duration cycles, the denominator of the fraction in $(*)$ will be positive starting from some n. Therefore, the following holds for every play $v_0 v_1 v_2 \ldots$ and every (large enough) n:

$$\frac{\sum_{i=0}^{n} r(v_i, v_{i+1}) \cdot d(v_i, v_{i+1})}{\sum_{i=0}^{n} d(v_i, v_{i+1})} \geq 0 \iff \frac{1}{n} \sum_{i=0}^{n} r(v_i, v_{i+1}) \cdot d(v_i, v_{i+1}) \geq 0 \ .$$

We may thus solve the question whether $v(s) \geq 0$ by transforming the weighted MPG into a standard MPG, leaving the set of vertices and edges the same and taking $w(u, v) = r(u, v) \cdot d(u, v)$ as the edge weight function. Although the value $v(s)$ may change in this reduction, its (non)negativeness does not.

Further, we may transform any problem of the form $v(s) \geq \lambda$ for any fixed constant λ into the above problem. Let us modify the weighted MPG as follows. Let $r'(u, v) = r(u, v) - \lambda$ and leave everything else the same. The value of a play $v_0 v_1 v_2 \cdots$ is thus changed as follows.

$$\nu'(v_0 v_1 v_2 \cdots) = \limsup_{n \to \infty} \frac{\sum_{i=0}^{n} (r(v_i, v_{i+1}) - \lambda) \cdot d(v_i, v_{i+1})}{\sum_{i=0}^{n} d(v_i, v_{i+1})} = \nu(v_0 v_1 v_2 \cdots) - \lambda$$

It is now clear that $v(s) \geq \lambda$ in the original game if and only if $v'(s) \geq 0$ in the modified game.

Furthermore, there is a one-to-one correspondence between the strategies in the original weighted MPG and the constructed MPG. Due to the two equivalences above, this correspondence preserves optimality. Therefore, there are optimal positional strategies in weighted MPGs since the same holds for standard MPGs [14]. (A strategy is positional if its decision does not depend on the current history of the play but only on the current vertex, i.e. can be described as a function $V \to V$.)

4.2 Translating Dual-priced MTSD into Weighted MPG

We first focus on the implementation problem without considering the hardware ($H = \emptyset$). We show how the implementation problem can be solved by reduction to the weighted MPGs. The first translation we present is exponential, however, we provide methods for making it smaller in the subsequent section.

We are given an MTSD $\mathcal{S} = (S, T, D, \Phi, s_0)$ and a dual-price scheme (r, H, Ψ, i) and assume that there is no state s with $\emptyset \in \mathrm{Tran}(s)$. Let us define the following auxiliary vertices that will be used to simulate the more complicated transitions of MTSD in the simpler setting of weighted MPG (by convention all singleton intervals are treated as uncontrollable).

$$T_{\mathsf{u}} = \{(s,a,t) \mid (s,a,t) \in T;\ D(s,a,t) \in \mathcal{I}_{\mathsf{u}}\}$$
$$T_{\mathsf{c}} = \{(s,a,t) \mid (s,a,t) \in T;\ D(s,a,t) \in \mathcal{I}_{\mathsf{c}}\}$$
$$T_{*} = \{(s,a,j,t) \mid (s,a,t) \in T;\ j \in D(s,a,t)\}$$

We construct the weighted mean-payoff game with $V_{min} = S \cup T_{\mathsf{c}} \cup T_{*}$, $V_{max} = 2^T \cup T_{\mathsf{u}}$ and E defined as follows:

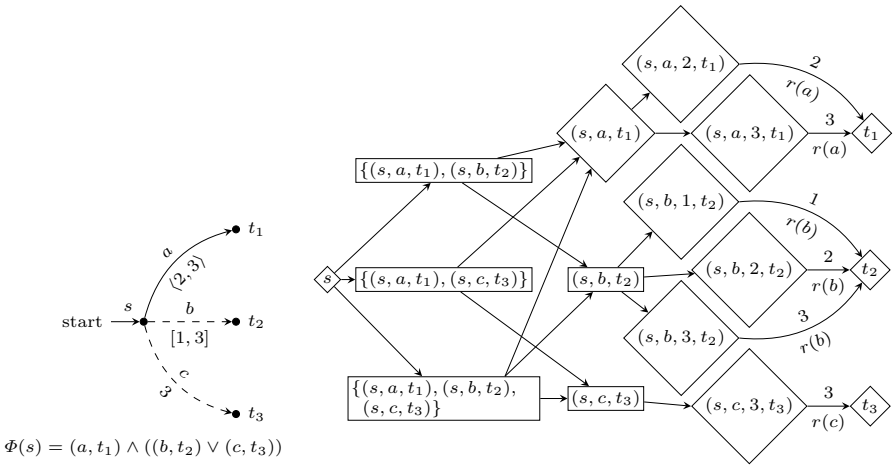
$$\begin{aligned}
(s, X) \in E \quad &\iff \exists V \in \mathrm{Tran}(s) : X = \{(s,a,t) \mid (a,t) \in V\} \\
(X, (s,a,t)) \in E \quad &\iff (s,a,t) \in X \\
((s,a,t),(s,a,j,t)) \in E \quad &\iff j \in D(s,a,t) \\
((s,a,j,t),t) \in E \quad &\text{(always)}
\end{aligned}$$

Further, $r((s,a,j,t),t) = r(a)$, $d((s,a,j,t),t) = j$ and $r(-,-) = d(-,-) = 0$ otherwise.

Example 12. In Figure 2 we show an example of how this translation to weighted MPG works. For simplicity we only translate a part of the MTSD $\mathcal{S}$ shown in Figure 2a. The resulting weighted MPG is shown in Figure 2b. The diamond shaped states belong to *min* and the squared states belong to *max*. In the vertex s, *min* chooses which outgoing transition are implemented. Only the choices satisfying $\Phi(s)$ are present in the game. Afterwards, *max* decides which transition to take. The chosen transition is then assigned by one of the players a time that it is going to take.

Notice that (s, a, t_1) is the only transition controlled by *min*, because it has a controllable interval $\langle 2, 3\rangle$. The remaining transitions with uncontrollable intervals are operated by *max* who chooses the time from these intervals. All the "auxiliary" transitions are displayed without any labels meaning their duration (and rate) is zero. Thus, only the transitions corresponding to "real" transitions in MTSDs are taken into account in the value of every play.

A strategy for *min* can now be translated into an implementation of the original MTSD in a straightforward way. The implemented transitions in s are given by $\sigma(s)$, similarly the durations of a transition (s, a, t) with a controllable interval are given by the third component of $\sigma((s, a, t))$.



(a) A part of an MTSD S (b) Resulting weighted MPG from translation of S

Fig. 2: Translating MTSD to weighted MPG

4.3 Optimizations

We now simplify the construction. The first simplification is summarized by the observation that the strategies of both players only need to choose the extremal points of the interval in vertices of the form (s, a, t).

Lemma 13. *There are optimal positional strategies for both* min *and* max *such that the choice in vertices of the form (s, a, t) is always one of the two extremal points of the interval $D(s, a, t)$.*

We may thus simplify the construction according to the previous lemma so that there are at most two outgoing edges for each state of the form (s, a, t) are as follows: $((s, a, t), (s, a, j, t)) \in E$ iff j is an extremal point of $D(s, a, t)$.

We can also optimize the expansion of $\mathrm{Tran}(s)$. So far, we have built an exponentially larger weighted MPG graph as the size of $\mathrm{Tran}(s)$ is exponential in the out-degree of s. However, we can do better if we restrict ourselves to the class of MTSD where all $\Phi(s)$ are positive boolean formulae, i.e. the only connectives are $\wedge$ and $\vee$. Instead of enumerating all valuations, we can use the syntactic tree of the formula to build a weighted MPG of polynomial size.

Let $sf(\varphi)$ denote the set of all sub-formulae of φ (including φ). Let further $S_* = \{(s, \varphi) \mid s \in S;\ \varphi \in sf(\Phi(s))\}$. The weighted MPG is constructed with

- $V_{min} = \{(s, \varphi) \in S_* \mid \varphi = \varphi_1 \vee \varphi_2$ or $(\varphi = (a, t)$ and $D(s, a, t) \in \mathcal{I}_{\mathsf{c}})\} \cup T_*$
- $V_{max} = \{(s, \varphi) \in S_* \mid \varphi = \varphi_1 \wedge \varphi_2$ or $(\varphi = (a, t)$ and $D(s, a, t) \in \mathcal{I}_{\mathsf{u}})\}$
- E is defined as follows:

$$
\begin{aligned}
&((s, \varphi_1 \wedge \varphi_2), (s, \varphi_i)) \in E && i \in \{1, 2\} \\
&((s, \varphi_1 \vee \varphi_2), (s, \varphi_i)) \in E && i \in \{1, 2\} \\
&((s, (a, t)), (s, a, j, t)) \in E && \iff j \text{ is an extremal point of } D(s, a, t) \\
&((s, a, j, t), (t, \Phi(t))) \in E && \text{(always)}
\end{aligned}
$$

– $r((s,a,j,t),(t,\Phi(t))) = r(a)$ and $r(-,-) = 0$ otherwise
– $d((s,a,j,t),(t,\Phi(t))) = j$ and $d(-,-) = 0$ otherwise.

Example 14. In Figure 3 we show the result of translating the part of an MTSD from Figure 2a. This weighted MPG is similar to the one in Figure 2b, but instead of having a vertex for each satisfying set of outgoing transitions, we now have the syntactic tree of the obligation formula for each state. Further the vertex $(s,b,2,t_2)$ is left out, due to Lemma 13. Note that the vertices $(t_1,\Phi(t_1))$, $(t_2,\Phi(t_2))$ and $(t_3,\Phi(t_3))$ are drawn as circles, because the player of these states depends on the obligation formula and the outgoing transitions.

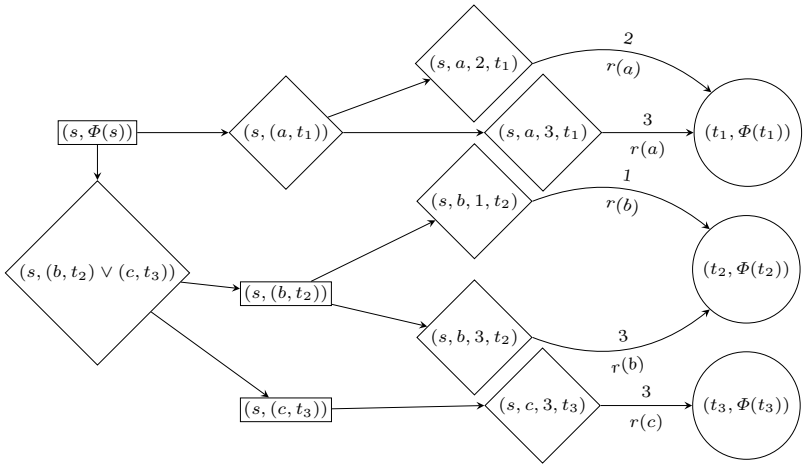


Fig. 3: Result of the improved translation of S in Figure 2a

Remark 15. Observe that one can perform this optimization even in the general case. Indeed, for those s where $\Phi(s)$ is positive we locally perform this transformation; for s with $\Phi(s)$ containing negations we stick to the original expansion. Thus, the exponential (in out-degree) blow-up occurs only locally.

Lemma 16. *Both optimized translations are correct and on MTSDs where the obligation function is positive they run in polynomial time.*

4.4 The Algorithm and its Complexity

The algorithm for our problem, given a specification $\mathcal{S}$, works as follows.

1. Nondeterministically choose hardware with the total price at most max_{ic}.
2. Create the weighted MPG out of $\mathcal{S}$.
3. Solve the weighted MPG using the reduction to MPG and any standard algorithm for MPG that finds an optimal strategy for player *min* and computes the value $v(s_0)$.
4. Transform the strategy to an implementation $\mathcal{I}$.

5. In the case of the cheapest-implementation problem return $\mathcal{I}$;
   in the case of the implementation (decision) problem return $v(s_0) \leq max_{\mathsf{rc}}$.

We can now prove the following result, finishing the proof of Theorem 9.

Proposition 17. *The implementation problem is in NP.*

Proof. We first nondeterministically guess the hardware assignment. Due to Section 4.2, we know that the desired implementation has the same states as the original MTSD and its transitions are a subset of the transitions of the original MTSD as the corresponding optimal strategies are positional. The first optimization (Section 4.3) guarantees that durations can be chosen as the extremal points of the intervals. Thus we can nondeterministically guess an optimal implementation and its durations, and verify that it satisfies the price inequality. □

Proposition 18. *The implementation problem for MTSD with positive obligation function and a constant number of hardware components is in NP ∩ coNP and solvable in pseudo-polynomial time.*

Proof. With the constant number of hardware components, we get a constant number of possible hardware configurations and we can check each configuration separately one by one. Further, by the first and the second optimization in Section 4.3, the MPG graph is of size $\mathcal{O}(|T| + |\Phi|)$. Therefore, we polynomially reduce the implementation problem to the problem of solving constantly many mean payoff games. The result follows by the existence of pseudo-polynomial algorithms for MPGs [18]. □

Further, our problem is at least as hard as solving MPGs that are clearly a special case of our problem. Hence, Theorem 10 follows.

5 Conclusion and Future Work

We have introduced a new extension of modal transition systems. The extension consists in introducing (1) variable time durations of actions and (2) pricing of actions, where we combine one-shot investment price for the hardware and cost for running it per each time unit it is active. We believe that this formalism is appropriate to modelling many types of embedded systems, where safety comes along with economical requirements.

We have solved the problem of finding the cheapest implementation w.r.t. the running cost given a maximum hardware investment we can afford, and we established the complexity of the decision problem in the general setting and in a practically relevant subcase revealing a close connection with mean payoff games.

As for the future work, apart from implementing the algorithm, one may consider two types of extensions. First, one can extend the formalism to cover the distinction between input, output and internal actions as it is usual in interface theories [12], and include even more time features, such as clocks in priced timed automata [4, 10]. Second, one may extend the criteria for synthesis of the

cheapest implementation by an additional requirement that the partial sums stay within given bounds as done in [11], or requiring the satisfaction of a temporal property as suggested in [12, 13].

References

1. Aceto, L., Fábregas, I., de Frutos-Escrig, D., Ingólfsdóttir, A., Palomino, M.: Graphical representation of covariant-contravariant modal formulae. In: EXPRESS. EPTCS, vol. 64, pp. 1–15 (2011)
2. Antonik, A., Huth, M., Larsen, K.G., Nyman, U., Wasowski, A.: 20 years of modal and mixed specifications. Bulletin of the EATCS no. 95 pp. 94–129 (2008)
3. Bauer, S.S., Fahrenberg, U., Juhl, L., Larsen, K.G., Legay, A., Thrane, C.R.: Quantitative refinement for weighted modal transition systems. In: MFCS. LNCS, vol. 6907, pp. 60–71. Springer (2011)
4. Behrmann, G., Larsen, K.G., Rasmussen, J.I.: Priced timed automata: Algorithms and applications. In: FMCO. LNCS, vol. 3657, pp. 162–182. Springer (2004)
5. Beneš, N., Křetínský, J.: Process algebra for modal transition systemses. In: MEMICS. OASICS, vol. 16, pp. 9–18. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany (2010)
6. Beneš, N., Křetínský, J., Larsen, K., Møller, M., Srba, J.: Parametric modal transition systems. In: Proceedings of ATVA'11. LNCS, vol. 6996, pp. 275–289. Springer-Verlag (2011)
7. Beneš, N., Křetínský, J., Larsen, K., Møller, M., Srba, J.: Dual-priced modal transition systems with time durations. Tech. Rep. FIMU-RS-2012-01, Faculty of Informatics MU (2012)
8. Bloem, R., Greimel, K., Henzinger, T.A., Jobstmann, B.: Synthesizing robust systems. In: Proc. of FMCAD09. pp. 85–92. IEEE (2009)
9. Boudol, G., Larsen, K.G.: Graphical versus logical specifications. Theor. Comput. Sci. 106(1), 3–20 (1992)
10. Bouyer, P., Brinksma, E., Larsen, K.G.: Optimal infinite scheduling for multi-priced timed automata. Formal Methods in System Design 32(1), 3–23 (2008)
11. Bouyer, P., Fahrenberg, U., Larsen, K.G., Markey, N., Srba, J.: Infinite runs in weighted timed automata with energy constraints. In: FORMATS. LNCS, vol. 5215, pp. 33–47. Springer (2008)
12. Chakrabarti, A., de Alfaro, L., Henzinger, T.A., Stoelinga, M.: Resource interfaces. In: Alur, R., Lee, I. (eds.) EMSOFT. Lecture Notes in Computer Science, vol. 2855, pp. 117–133. Springer (2003)
13. Chatterjee, K., Doyen, L.: Energy parity games. In: Abramsky, S., Gavoille, C., Kirchner, C., auf der Heide, F.M., Spirakis, P.G. (eds.) ICALP (2). Lecture Notes in Computer Science, vol. 6199, pp. 599–610. Springer (2010)
14. Ehrenfeucht, A., Mycielski, J.: Positional strategies for mean payoff games. International Journal of Game Theory 8, 109–113 (1979), 10.1007/BF01768705
15. Juhl, L., Larsen, K.G., Srba, J.: Introducing modal transition systems with weight intervals. Journal of Logic and Algebraic programming (2011), to appear
16. Larsen, K.G., Thomsen, B.: A modal process logic. In: LICS. pp. 203–210. IEEE Computer Society (1988)
17. Larsen, K.G., Xinxin, L.: Equation solving using modal transition systems. In: LICS. pp. 108–117. IEEE Computer Society (1990)
18. Zwick, U., Paterson, M.: The complexity of mean payoff games on graphs. Theoretical Computer Science 158, 343–359 (1996)

Paper E:

Modal process rewrite systems

Nikola Beneš and Jan Křetínský

This paper has been published in Abhik Roychoudhury and Meenakshi D'Souza (eds.): Theoretical Aspects of Computing - ICTAC 2012 - 9th International Colloquium, Bangalore, India, September 24-27, 2012. Proceedings, volume 7521 of Lecture Notes in Computer Science, pages 120–135. Springer, 2012. [BK12]

Summary

We introduce MTS with an infinite state space generated by rules for sequential and parallel compositions as defined in process rewrite systems. We investigate the induced notion of modal refinement and provide a detailed account on decidability and complexity on the subclasses corresponding to standard infinite state systems studied in literature. We show that refinement between MTS generated by sequential composition (pushdown automata) and finite MTS are decidable, whereas on MTS generated by parallel composition (Petri nets, basic parallel processes) it is undecidable even when the other system is finite. In order to achieve decidability in the case with both sides infinite, we restrict to visibly pushdown automata. Finally, we define a bisimulation relation on MTS called birefinement and show that it is decidable between finite MTS and MTS generated by any process rewrite system rules.

Author's contribution: 50 %

- participating in the discussions,
- contributing, in particular, to the definition using process rewrite systems and to undecidability proofs,
- writing Introduction and parts of the paper.

Modal Process Rewrite Systems

Nikola Beneš[1⋆] and Jan Křetínský[1,2⋆⋆]

[1] Faculty of Informatics, Masaryk University, Brno, Czech Republic
[2] Institut für Informatik, Technische Universität München, Germany
{xbenes3, jan.kretinsky}@fi.muni.cz

Abstract. We consider modal transition systems with infinite state space generated by finite sets of rules. In particular, we extend process rewrite systems to the modal setting and investigate decidability of the modal refinement relation between systems from various subclasses. Since already simulation is undecidable for most of the cases, we focus on the case where either the refined or the refining process is finite. Namely, we show decidability for pushdown automata extending the non-modal case and surprising undecidability for basic parallel processes. Further, we prove decidability when both systems are visibly pushdown automata. For the decidable cases, we also provide complexities. Finally, we discuss a notion of bisimulation over MTS.

1 Introduction

The ever increasing complexity of software systems together with their reuse call for efficient *component-based* design and verification. One of the major theoretically well founded frameworks that answer this call are *modal transition systems* (MTS) [LT88]. Their success resides in natural combination of two features. Firstly, it is the simplicity of labelled transition systems, which have proved appropriate for behavioural description of systems as well as their compositions; MTS as their extension inherit this appropriateness. Secondly, as opposed to temporal logic specifications, MTS can be easily *gradually refined* into implementations while preserving the desired behavioural properties.

MTS consist of a set of states and two transition relations. The *must* transitions prescribe which behaviour has to be present in every refinement of the system; the *may* transitions describe the behaviour that is allowed, but need not be realized in the refinements. This allows for underspecification of non-critical behaviour in the early stage of design, focusing on the main properties, verifying them and sorting out the details of the yet unimplemented non-critical behaviour later.

The formalism of MTS has proven to be useful in practice. Industrial applications are as old as [Bru97] where MTS have been used for an air-traffic system at Heathrow airport. Besides, MTS are advocated as an appropriate

⋆ The author has been supported by the Czech Science Foundation, grant No. GAP202/11/0312.
⋆⋆ The author is a holder of Brno PhD Talent Financial Aid and is supported by the Czech Science Foundation, grant No. P202/12/G061.

base for interface theories in [RBB+09] and for product line theories in [Nym08]. Further, MTS based software engineering methodology for design via merging partial descriptions of behaviour has been established in [UC04]. Moreover, the tool support is quite extensive, e.g. [BLS95,DFFU07,BML11,BČK11].

Over the years, many extensions of MTS have been proposed. While MTS can only specify whether or not a particular transition is required, some extensions equip MTS with more general abilities to describe what *combinations* of transitions are possible [LX90,FS08,BK10,BKL+11]. Further, MTS framework has also been lifted to *quantitative settings*. This includes probabilistic [CDL+10] and timed systems [ČGL93,JLS11,BFJ+11,BKL+12,DLL+10,BLPR11] with clear applications in the embedded systems design. As far as the infinite state systems are concerned, only a few more or less ad hoc extensions have been proposed, such as systems with asynchronous communication based on FIFO [BHJ10] or Petri nets [EBHH10]. In this paper, we introduce modalities into a general framework for infinite-state systems, where we study modal extensions of well-established classes of infinite-state systems.

Such a convenient unifying framework for infinite-state systems is provided by *Process rewrite systems* (PRS) [May00]. They encompass many standard models such as pushdown automata (PDA) or Petri nets (PN) as syntactic subclasses. A PRS consists of a set of rewriting rules that model computation. These rules may contain sequential and parallel composition. For example, a transition t of a Petri net with input places I_1, I_2 and output places O_1, O_2 can be described by the rule $I_1 \parallel I_2 \xrightarrow{t} O_1 \parallel O_2$. A transition of a pushdown automaton in a state s with a top stack symbol X reading a letter a resulting in changing the state to q and pushing Y to the stack can be written as $sX \xrightarrow{a} qYX$. Limiting the occurrences of parallel and sequential composition on the left and right sides of the rules yields the most common automata theoretic models. For these syntactic subclasses of PRS, see Figure 1 and a more detailed description in Section 2.

Motivation One can naturally lift PRS to the modal world by having two sets of rules, may and must rules. What is then the use of such *modal process rewrite systems* (mPRS)? Firstly, potentially infinite-state systems such as Petri nets are very popular for modelling whenever communication or synchronization between processes occurs. This is true even when they are actually bounded and thus with a finite state space.

Example 1. Consider the following may rule (we use dashed arrows to denote may rules) generating a small Petri net.

$$\mathsf{resource} \parallel \mathsf{customer} \overset{\mathsf{consume}}{\dashrightarrow} \mathsf{trash}$$

This rewrite rule implies that e.g. a process resource ∥ customer ∥ customer may be changed into trash ∥ customer. Therefore, if there is no other rule with trash on the right side a safety property is guaranteed for all implementations of this system, namely that trash can only arise if there is at least one resource and one customer. On the other hand, it is not guaranteed that trash can indeed be

produced in such a situation. This is very useful as during the design process new requirements can arise, such as necessity of adding more participants to perform this transition. For instance,

$$\mathsf{resource} \parallel \mathsf{customer} \parallel \mathsf{permit} \overset{\mathsf{consume}}{\dashrightarrow} \mathsf{trash}$$

expresses an auxiliary condition required to produce trash, namely that permit is available. Replacing the old rule with the new one is equivalent to adding an input place permit to the Petri net. In the modal transition system view, the new system *refines* the old one. Indeed, the new system is only more specific about the allowed behaviour than the old one and does not permit any previously forbidden behaviour. One can further refine the system by the one given by

$$\mathsf{resource} \parallel \mathsf{customer} \parallel \mathsf{permit} \parallel \mathsf{bribe} \overset{\mathsf{consume}}{\longrightarrow} \mathsf{trash}$$

where additional condition is imposed and now the trash-producing transition has to be available (denoted by an unbroken arrow) whenever the left hand side condition is satisfied.

Secondly, even if an original specification is finite its refinements and the final implementation might be infinite. For instance, consider a specification where permit needs to be available but is not consumed or there is an unlimited amount of permits. In an implementation, the number of permits could be limited and thus this number with no known bounds needs to be remembered in the state of the system. Similarly, consider a finite safety specification of a browser together with its implementation that due to the presence of back button requires the use of stack, and is thus a pushdown system. Further, sometimes both the specification and the implementation are infinite such as a stateless BPA specification of a stateful component implemented by a PDA.

Example 2. Consider a basic process algebra (BPA) given by rules $X \overset{(}{\longrightarrow} XX$ and $X \overset{)}{\longrightarrow} \varepsilon$ for correctly parenthesized expressions with $X \overset{a}{\dashrightarrow} X$ for all other symbols a, i.e. with no restriction on the syntax of expressions. One can easily refine this system into a PDA that accepts correct arithmetic expressions by remembering in the state whether the last symbol read was an operand or an operator.

Further, opposite to the design of correct software where an abstract verified MTS is transformed into a concrete implementation, one can consider checking correctness of software through abstracting a concrete implementation into a coarser system. The use of MTS as abstractions has been advocated e.g. in [GHJ01]. While usually overapproximations (or underapproximations) of systems are constructed and thus only purely universal (or existential) properties can be checked, [GHJ01] shows that using MTS one can check mixed formulae (arbitrarily combining universal and existential properties) and, moreover, at the same cost as checking universal properties using traditional conservative abstractions. This advantage has been investigated also in the context of systems equivalent or closely related to MTS [HJS01,DGG97,Nam03,DN04,CGLT09,GNRT10].

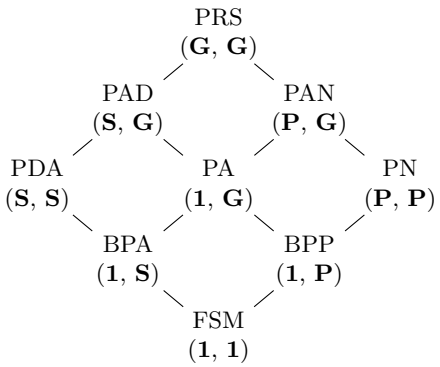


Fig. 1. PRS hierarchy

Although one is usually interested in generating finite abstractions of infinite systems, it might be interesting to consider situations where the abstract system is infinite. For instance, if one is interested in a property that is inherently non-regular such as correct parenthesizing in the previous example, the abstraction has to capture this feature. One could thus abstract the PDA from the previous example into the smaller BPA above and prove the property here using algorithms for BPA. Moreover, if one is interested in mixed properties the abstract system has to be modal. It would be useful to extend the verification algorithms for systems such as PDA to their modal versions along the lines of the generalized model checking approach [BG00,BČK11]. This is, however, beyond the scope of this paper.

Our contribution In this paper, we focus on modal infinite-state systems and decidability of the most fundamental problem here, namely deciding the refinement relation, for most common classes of systems. Since simulation is undecidable already on basic parallel processes (BPP) [Hüt94] and basic process algebras (BPA) [GH94], cf. Figure 1, the refinement as a generalization of simulation is undecidable in general. However, one can consider the case where either the refined or the refining system is finite (a finite state machine, FSM). This case is still very interesting, e.g. in the context of finite abstractions or implementations with bounded resources. [KM99] shows that while simulation remains undecidable between process algebras (PA) and FSM, it is decidable between PDA and FSM. We extend this result using methods of [KM02b] to the modal setting. Further, although simulation is decidable between PN and FSM [JM95] (in both directions), we show that surprisingly this result cannot be extended and the refinement is undecidable even for BPP and FSM in the modal setting. Although the decidability of the refinement seems quite limited now, we show that refinement is sometimes decidable even between two infinite-state systems, namely between modal extensions of visibly pushdown automata [AM04], cf. Example 2; for this, we use the methods of [Srb06]. To summarize:

– We introduce a general framework for studying modal infinite-state system, namely we lift process rewrite systems to the modal setting. This definition comes along with the appropriate notion of refinement.

– We prove un/decidability of the refinement problem for modal extensions of standard classes of infinite-state systems. Apart from trivial corollaries due to the undecidability of simulation, this amount to proving undecidability of refinement between Petri nets and FSM (on either side) and decidability between pushdown systems and FSM (again on either side). Moreover, we prove decidability for visibly PDA. For the decidable cases, we show that the complexity is the same as for checking the respective simulation in the non-modal setting. Finally, we discuss a notion of bisimulation over MTS, which we name birefinement.

Related work There are various other approaches to deal with component refinements. They range from subtyping [LW94] over Java modelling language [JP01] to interface theories close to MTS such as interface automata [dAH01]. Similarly to MTS, interface automata are behavioural interfaces for components. However, their composition works very differently. Furthermore, its notion of refinement is based on alternating simulation [AHKV98], which has been proved strictly less expressive than MTS refinement—actually coinciding on a subclass of MTS—in a paper [LNW07] that combines MTS and interface automata based on I/O automata [Lyn88]. The compositionality of this combination is further investigated in [RBB+11].

MTS can also be viewed as a fragment of mu-calculus that is "graphically representable" [BL90]. The graphical representability of a variant of alternating simulation called covariant-contravariant simulation has been recently studied in [AFdFE+11].

The PRS framework has been introduced in [May00]. Simulation on classes of PRS tends to be computationally harder than bisimilarity [KM02b]. While e.g. bisimulation between any PRS and FSM is decidable [KŘS05], simulation with FSM is undecidable already for PA (see above). Therefore, the decidability is limited to PDA and PN, and we show that refinement is even harder (undecidability for BPP). Another aspect that could help to extend the decidability is determinism. For instance, simulation between FSM and deterministic PA is decidable [KM99]. It is also the case with the abovementioned [EBHH10] where refinement over "weakly deterministic" modal Petri nets is shown decidable.

Outline of the paper In Section 2, we introduce modal process rewrite systems formally and recall the refinement preorder. In Section 3 and 4, we show undecidability and decidability results for the refinement. Section 5 concludes.

2 Refinement Problems

In this section, we introduce modal transition systems generated by process rewrite systems and define the notion of modal refinement. We start with the usual definition of MTS.

2.1 Modal Transition Systems

Definition 1 (Modal transition system). *A* modal transition system (MTS*) over an action alphabet Act is a triple* $(\mathcal{P}, \dashrightarrow, \longrightarrow)$*, where* $\mathcal{P}$ *is a set of* processes *and* $\longrightarrow \subseteq \dashrightarrow \subseteq \mathcal{P} \times Act \times \mathcal{P}$ *are* must *and* may *transition relations, respectively.*

Observe that $\mathcal{P}$ is not required to be finite. We often use letters $s, t, \ldots$ for processes of MTS. Whenever clear from the context, we refer to processes without explicitly mentioning their underlying MTS.

We proceed with the standard definition of (modal) refinement.

Definition 2 (Refinement). *Let $(\mathcal{P}_1, \dashrightarrow_1, \longrightarrow_1), (\mathcal{P}_2, \dashrightarrow_2, \longrightarrow_2)$ be MTS over the same action alphabet and $s \in \mathcal{P}_1$, $t \in \mathcal{P}_2$ be processes. We say that s* refines *t, written $s \leq_{\mathrm{m}} t$, if there is a relation $\mathcal{R} \subseteq \mathcal{P}_1 \times \mathcal{P}_2$ such that $(s, t) \in \mathcal{R}$ and for every $(p, q) \in \mathcal{R}$ and every $a \in Act$:*

1. *if $p \overset{a}{\dashrightarrow}_1 p'$ then there is a transition $q \overset{a}{\dashrightarrow}_2 q'$ s.t. $(p', q') \in \mathcal{R}$, and*
2. *if $q \overset{a}{\longrightarrow}_2 q'$ then there is a transition $p \overset{a}{\longrightarrow}_1 p'$ s.t. $(p', q') \in \mathcal{R}$.*

The ultimate goal of the refinement process is to obtain an *implementation*, i.e. an MTS with $\dashrightarrow = \longrightarrow$. Implementations can be considered as the standard labelled transition systems (LTS). Note that on implementations refinement coincides with strong bisimilarity, and on modal transition systems without any must transitions it corresponds to the simulation preorder, denoted by $\leq_{\mathrm{sim}}$. Further, refinement has a game characterization [BKLS09] similar to (bi)simulation games, which we often use in the proofs.

2.2 Modal Process Rewrite Systems

We now move our attention to infinite-state MTS generated by finite sets of rules. Let *Const* be a set of *process constants*. We define the set of process expressions $\mathcal{E}$ by the following abstract syntax:

$$E ::= \varepsilon \mid X \mid E \parallel E \mid E; E$$

where X ranges over *Const*. We often use Greek letters $\alpha, \beta, \ldots$ for elements of $\mathcal{E}$. The process expressions are considered modulo the usual structural congruence, i.e. the smallest congruence such that the operator ; is associative, $\parallel$ is associative and commutative and ε is a unit for both ; and $\parallel$. We often omit the ; operator.

Definition 3 (Modal process rewrite system). *A* process rewrite system (PRS) *is a finite relation $\Delta \subseteq (\mathcal{E} \setminus \{\varepsilon\}) \times Act \times \mathcal{E}$, elements of which are called rewrite rules. A* modal process rewrite system (mPRS) *is a tuple $(\Delta_{\mathrm{may}}, \Delta_{\mathrm{must}})$ where Δ_{may}, Δ_{must} are process rewrite systems such that $\Delta_{\mathrm{must}} \subseteq \Delta_{\mathrm{may}}$.*

An mPRS $\Delta = (\Delta_{\mathrm{may}}, \Delta_{\mathrm{must}})$ induces an MTS $\mathbb{MTS}(\Delta) = (\mathcal{E}, \dashrightarrow, \longrightarrow)$ as follows:

$$\frac{(E, a, E') \in \Delta_{\mathrm{may}}}{E \overset{a}{\dashrightarrow} E'} \qquad \frac{E \overset{a}{\dashrightarrow} E'}{E; F \overset{a}{\dashrightarrow} E'; F} \qquad \frac{E \overset{a}{\dashrightarrow} E'}{E \parallel F \overset{a}{\dashrightarrow} E' \parallel F}$$

$$\frac{(E, a, E') \in \Delta_{\mathrm{must}}}{E \overset{a}{\longrightarrow} E'} \qquad \frac{E \overset{a}{\longrightarrow} E'}{E; F \overset{a}{\longrightarrow} E'; F} \qquad \frac{E \overset{a}{\longrightarrow} E'}{E \parallel F \overset{a}{\longrightarrow} E' \parallel F}$$

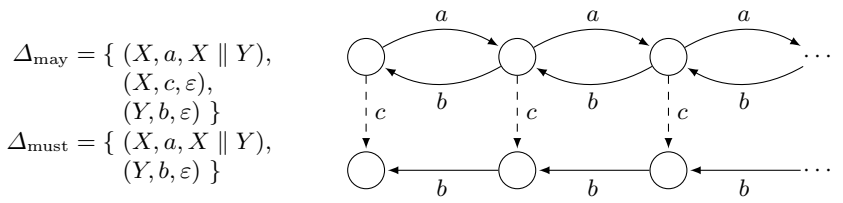


Fig. 2. An example of a mBPP and its corresponding (infinite) MTS; the dashed arrows represent *may* transitions, the unbroken arrows represent *must* transitions; as $s \xrightarrow{a} t$ implies $s \overset{a}{\dashrightarrow} t$ we omit the may transitions where must transitions are also present

We consider four distinguished classes of process expressions. Class **S** stands for expressions with no $\parallel$ (purely sequential expressions) and class **P** stands for expressions with no ; (purely parallel expressions). Further, we use **G** for the whole $\mathcal{E}$ (general expressions) and **1** for *Const* (one process constant and no operators). Now restricting the left and right sides of rules of PRS to these classes yields subclasses of PRS as depicted in Figure 1 using the standard shortcuts also introduced in Section 1. Each subclass $\mathcal{C}$ has a corresponding modal extension m$\mathcal{C}$ containing all mPRS $(\Delta_{\text{may}}, \Delta_{\text{must}})$ with both Δ_{may} and Δ_{must} in $\mathcal{C}$. For instance, mFSM correspond to the standard finite MTS and mPN are modal Petri nets as introduced in [EBHH10]. An example of an mBPP and the resulting MTS are depicted in Figure 2.

For any classes $\mathcal{C}$, $\mathcal{D}$, we define the following decision problem $m\mathcal{C} \leq_{\text{m}} m\mathcal{D}$.

> Given mPRS $\Delta_1 \in m\mathcal{C}, \Delta_2 \in m\mathcal{D}$ and process terms δ_1, δ_2 conforming to left-hand side restrictions of $\mathcal{C}, \mathcal{D}$, respectively, does $\delta_1 \leq_{\text{m}} \delta_2$ hold considering δ_1, δ_2 as processes of $\mathbb{MTS}(\Delta_1), \mathbb{MTS}(\Delta_2)$?

3 Undecidability Results

In this section, we present all the negative results. As already discussed in Section 1, simulation—and thus refinement—is undecidable already on BPP [Hüt94] and BPA [GH94]. When considering the case where one of the two classes is mFSM, the undecidability holds for mPA [KM99]. Thus we are left with the problems mFSM$\leq_{\text{m}}$mPDA, mPDA$\leq_{\text{m}}$mFSM and mFSM$\leq_{\text{m}}$mPN, mPN$\leq_{\text{m}}$mFSM. On the one hand, the two former are shown decidable in Section 4 using non-modal methods for simulation of [KM02b]. On the other hand, the non-modal methods for simulation of [JM95] cannot be extended to the latter two problems. In this section, we show that (surprisingly) they are both undecidable and, moreover, even for mBPP.

Theorem 1. *The problem mBPP$\leq_{\text{m}}$mFSM is undecidable.*

Proof. We reduce the undecidable problem of simulation between two BPPs (even normed ones) to the problem mBPP$\leq_{\text{m}}$mFSM.

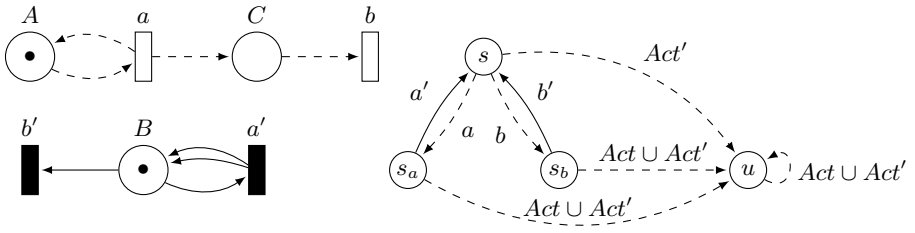


Fig. 3. $A \parallel B \leq_{\mathrm{m}} s$ where the original two BPPs are given by $A \xrightarrow{a} A \parallel C$, $C \xrightarrow{b} \varepsilon$, $B \xrightarrow{a} B \parallel B$, $B \xrightarrow{b} \varepsilon$.

Let A, B be two BPP processes with underlying PRS Δ_A and Δ_B; w.l.o.g. $\Delta_A \cap \Delta_B = \emptyset$. We transform them as follows. We rename all actions of the underlying PRS of B from a to a'. Let Act' be the set of these renamed actions and let Δ'_B be the modification of Δ_B by renaming the actions. The mBPP is defined as $(\Delta_A \cup \Delta'_B, \Delta'_B)$, i.e. the transitions of A are just may, the (modified) transitions of B are both must and may.

We then build a finite mPRS as follows. The states are $\{s, u\} \cup \{s_a \mid a \in Act\}$.

- $s \overset{a}{\dashrightarrow} s_a$ and $s \overset{a'}{\dashrightarrow} u$ for all $a \in Act$
- $s_a \xrightarrow{a'} s$ for all $a \in Act$ (with the corresponding may transition)
- $s_a \overset{x}{\dashrightarrow} u$ for all $a \in Act$ and $x \in Act \cup Act'$
- $u \overset{x}{\dashrightarrow} u$ for all $x \in Act \cup Act'$

Clearly $q \leq_{\mathrm{m}} u$ for any process q. The construction is illustrated in Figure 3.

We now show that $A \leq_{\mathrm{sim}} B$ iff $A \parallel B \leq_{\mathrm{m}} s$. In the following, α always denotes a process of Δ_A, while β denotes a process of Δ_B. Furthermore, we use the notation $\mathbb{LTS}(\Delta_A)$ to denote the LTS induced by Δ_A (similarly for Δ_B). We use the refinement game argumentation, see [BKLS09].

$\Rightarrow$: Let $\mathcal{R} = \{(\alpha \parallel \beta, s) \mid \alpha \leq_{\mathrm{sim}} \beta\}$. We show that $\mathcal{R}$ can be extended to be a modal refinement relation. Let $(\alpha \parallel \beta, s) \in \mathcal{R}$:

- If the attacker plays $\alpha \parallel \beta \overset{a'}{\dashrightarrow} \alpha \parallel \beta'$ (where $a' \in Act'$), the defender can play $s \overset{a'}{\dashrightarrow} u$ and obviously wins.
- If the attacker plays $\alpha \parallel \beta \overset{a}{\dashrightarrow} \alpha' \parallel \beta$ (where $a \in Act$), the defender has to play $s \overset{a}{\dashrightarrow} s_a$. There are two possibilities then:
  - if the attacker plays $\alpha' \parallel \beta \overset{x}{\dashrightarrow}$, the defender can play $s_a \overset{x}{\dashrightarrow} u$ and obviously wins;
  - if the attacker plays $s_a \xrightarrow{a'} s$, the defender can play $\alpha' \parallel \beta \xrightarrow{a'} \alpha' \parallel \beta'$ where β' is a process such that $\beta \xrightarrow{a} \beta'$ in $\mathbb{LTS}(\Delta_B)$ and $\alpha' \leq_{\mathrm{sim}} \beta'$. Such β' obviously exists due to $\alpha \leq_{\mathrm{sim}} \beta$. Thus $(\alpha' \parallel \beta', s) \in \mathcal{R}$.

$\Leftarrow$: We show that $\mathcal{R} := \{(\alpha, \beta) \mid \alpha \parallel \beta \leq_{\mathrm{m}} s\}$ is a simulation. Let $(\alpha, \beta) \in \mathcal{R}$:

- If $\alpha \xrightarrow{a} \alpha'$ in $\mathbb{LTS}(\Delta_A)$ then $\alpha \parallel \beta \overset{a}{\dashrightarrow} \alpha' \parallel \beta$. This has to be matched by $s \overset{a}{\dashrightarrow} s_a$. Furthermore, $s_a \xrightarrow{a'} s$ has to be matched by $\alpha' \parallel \beta \xrightarrow{a'} \alpha' \parallel \beta'$. This means that $\beta \xrightarrow{a} \beta'$ in $\mathbb{LTS}(\Delta_B)$ and that $(\alpha', \beta') \in \mathcal{R}$. □

Theorem 2. *The problem mFSM$\leq_{\mathrm{m}}$mBPP is undecidable.*

Proof. We reduce the undecidable problem of simulation between two BPPs to the problem mFSM$\leq_{\mathrm{m}}$mBPP. The proof is similar to the previous one. However, as the situation is not entirely symmetric (the requirement that $\Delta_{\mathrm{must}} \subseteq \Delta_{\mathrm{may}}$ introduces asymmetry), we need to modify the construction somewhat.

Let again A, B be two BPP processes with underlying PRS Δ_A and Δ_B; w.l.o.g. $\Delta_A \cap \Delta_B = \emptyset$. We rename all actions of Δ_B from a to a'. Let Act' be the set of these renamed actions and let Δ'_B be the modification of Δ_B. We further create a new PRS as follows:

$$\Delta_X = \{(X, a, Y) \mid a \in Act\} \cup \{(Y, x, X) \mid x \in Act \cup Act'\}$$

The mBPP is defined as $(\Delta_A \cup \Delta'_B \cup \Delta_X, \Delta_A)$, i.e. the (modified) transitions of B are just may, the transitions of A are both must and may, and the new transitions of Δ_X are may.

We then build a finite mPRS as follows. The states are $\{s, v\} \cup \{s_a \mid a \in Act\}$.

- $s \xrightarrow{a} s_a$ for all $a \in Act$ (with the corresponding may transitions)
- $s_a \dashrightarrow s$ for all $a \in Act$
- $s_a \xrightarrow{a} v$ for all $a \in Act$ (with the corresponding may transitions)
- $v \xrightarrow{a} v$ for all $a \in Act$ (with the corresponding may transitions)

The construction is illustrated in Figure 4.

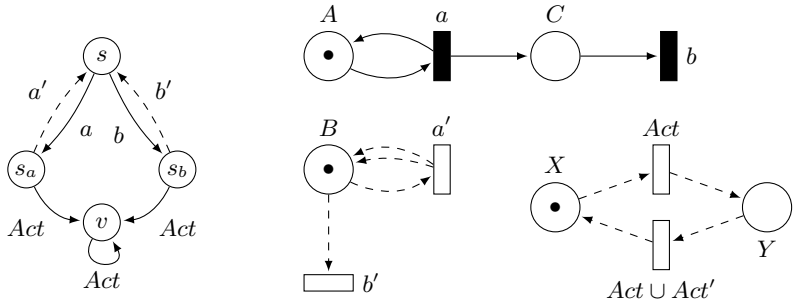


Fig. 4. $s \leq_{\mathrm{m}} A \parallel B \parallel X$ where the original two BPPs are again given by $A \xrightarrow{a} A \parallel C$, $C \xrightarrow{b} \varepsilon$, $B \xrightarrow{a} B \parallel B$, $B \xrightarrow{b} \varepsilon$.

We now show that $A \leq_{\mathrm{sim}} B$ iff $s \leq_{\mathrm{m}} A \parallel B \parallel X$. As in the previous proof, α denotes a process of Δ_A while β denotes a process of Δ_B.

We first show that $v \leq_{\mathrm{m}} \alpha \parallel \beta \parallel V$ for all $V \in \{X, Y\}$ and all processes α, β. Whenever the attacker plays a must transition of α, it is matched by $v \xrightarrow{a} v$. Whenever the attacker plays a may transition of v, it is matched either by $X \overset{a}{\dashrightarrow} Y$ or by $Y \overset{a}{\dashrightarrow} X$ (α and β are unaffected).

$\Rightarrow$: Let $\mathcal{R} = \{(s, \alpha \parallel \beta \parallel X) \mid \alpha \leq_{\mathrm{sim}} \beta\}$. We show that $\mathcal{R}$ can be extended to be a modal refinement relation. Let $(s, \alpha \parallel \beta \parallel X) \in \mathcal{R}$:

– If the attacker plays $s \overset{a}{\dashrightarrow} s_a$ then the defender can play $\alpha \parallel \beta \parallel X \overset{a}{\dashrightarrow} \alpha \parallel \beta \parallel Y$. The attacker then has two possibilities:
  - if the attacker plays $s_a \overset{a'}{\dashrightarrow} s$ then the defender can play $\alpha \parallel \beta \parallel Y \overset{a'}{\dashrightarrow} \alpha \parallel \beta \parallel X$ and the game is back in $\mathcal{R}$;
  - if the attacker plays $\alpha \parallel \beta \parallel Y \xrightarrow{a} \alpha' \parallel \beta \parallel Y$ then the defender can play $s_a \xrightarrow{a} v$ and win due to the fact above.
– If the attacker plays $\alpha \parallel \beta \parallel X \xrightarrow{a} \alpha' \parallel \beta \parallel X$ then the defender has to play $s \xrightarrow{a} s_a$. The attacker then has three possibilities:
  - if the attacker plays $\alpha' \parallel \beta \parallel X \xrightarrow{b} \alpha'' \parallel \beta \parallel X$ then the defender can play $s_a \xrightarrow{b} v$ and win due to the fact above.
  - if the attacker plays $s_a \overset{b}{\dashrightarrow} v$ then the defender can play $\alpha' \parallel \beta \parallel X \overset{b}{\dashrightarrow} \alpha' \parallel \beta \parallel Y$ and win due to the fact above.
  - if the attacker plays $s_a \overset{a'}{\dashrightarrow} s$ then the defender plays $\alpha' \parallel \beta \parallel X \overset{a'}{\dashrightarrow} \alpha' \parallel \beta' \parallel X$ where β' is a process such that $\beta \xrightarrow{a} \beta'$ in $\mathbb{LTS}(\Delta_B)$ and $\alpha' \leq_{\text{sim}} \beta'$. Such process has to exist due to $\alpha \leq_{\text{sim}} \beta$. Therefore, $(s, \alpha' \parallel \beta' \parallel X) \in \mathcal{R}$.

$\Leftarrow$: Let $\mathcal{R} = \{(\alpha, \beta) \mid s \leq_{\text{m}} \alpha \parallel \beta \parallel X\}$. We show that $\mathcal{R}$ is a simulation. Let $(\alpha, \beta) \in \mathcal{R}$.

– If $\alpha \xrightarrow{a} \alpha'$ then $\alpha \parallel \beta \parallel X \xrightarrow{a} \alpha' \parallel \beta \parallel X$. This has to be matched by $s \xrightarrow{a} s_a$. Furthermore, $s_a \overset{a'}{\dashrightarrow} s$ has to be matched by $\alpha' \parallel \beta \parallel X \overset{a'}{\dashrightarrow} \alpha' \parallel \beta' \parallel X$ (note that neither α' nor X can make an a'-transition). This means that $\beta \xrightarrow{a} \beta'$ in $\mathbb{LTS}(\Delta_B)$ and that $(\alpha', \beta') \in \mathcal{R}$. □

4 Decidability Results

We prove that the problems mFSM$\leq_{\text{m}}$mPDA and mPDA$\leq_{\text{m}}$mFSM are decidable and EXPTIME-complete like the corresponding simulation problems.

We modify the result of [KM02b] and show that, in certain classes of mPRS, refinement can be reduced to simulation. The original method introduces two translations, A and D, that transform two processes s and t into $\text{A}(s)$ and $\text{D}(t)$ in such a way that s and t are bisimilar iff $\text{A}(s) \leq_{\text{sim}} \text{D}(t)$. This approach can be modified in a straightforward way to work with modal refinement instead of bisimulation. The idea of the modification is that the part of the construction that simulates the attacker's possibility to play on the right-hand side is only done for must transitions. The modified A and D translations are then functions from MTS to LTS such that if we have two MTS processes s and t, it holds that $s \leq_{\text{m}} t$ iff $\text{A}(s) \leq_{\text{sim}} \text{D}(t)$. As these translations are only slightly changed from the original ones, we omit their definition here and refer to [BK12].

The applicability of this method is the same (modulo the modal extension) as the applicability of the original method. Both the A-translation and the D-translation preserve the following subclasses of PRS: PDA, BPA, FSM, nPDA,

nBPA and OC. Here, nPDA and nBPA are the normed variants (every process may be rewritten to ε in finite number of steps) of PDA and BPA, respectively. OC is the subclass of one-counter automata, i.e. PDA with only one stack symbol. Furthermore, the A-translation also preserves determinism.

As a direct corollary of the previous remark and the results of [KM02a], we obtain the following.

Theorem 3. *The problem mPDA$\leq_{\mathrm{m}}$mFSM is EXPTIME-complete in both ways, even if the mFSM is of a fixed size. The problem mBPA$\leq_{\mathrm{m}}$mFSM is EXPTIME-complete in both ways, but if the mFSM is of a fixed size, it is PTIME-complete.*

4.1 Visibly PDA

We have seen that the refinement relation is undecidable between any two infinite classes of the hierarchy depicted in Figure 1. However, there are other subclasses where the refinement is decidable. In this section, we show that the refinement between two modal visibly PDA is decidable.

Definition 4. *A PDA is a* visibly PDA (vPDA) *if there is a partitioning $Act = Act_c \uplus Act_r \uplus Act_i$ such that every rule $pX \xrightarrow{a} q\alpha$ satisfies the following:*

- *if $a \in Act_c$ then $|\alpha| = 2$ (call),*
- *if $a \in Act_r$ then $|\alpha| = 0$ (return),*
- *if $a \in Act_i$ then $|\alpha| = 1$ (internal).*

The modal extension (mvPDA) is straightforward; its subclass mvBPA can be defined similarly.

In order to prove decidability, we make use of the idea of [Srb06] for showing that simulation between two vPDA is decidable. We modify and simplify the method somehow, as the original method is used to prove decidability of various kinds of equivalences and preorders, while we are only considering the modal refinement.

Theorem 4. *The problem mvPDA$\leq_{\mathrm{m}}$mvPDA is decidable.*

Proof. Let $(\Delta_{\mathrm{may}}, \Delta_{\mathrm{must}})$ be a mvPDA with a stack alphabet Γ and a set of control states $\mathcal{Q}$. Let sA and tB be two processes of the mvPDA. Note that for simplicity we consider two processes of a single mvPDA. However, as a disjoint union of two mPRS is a mPRS, this also solves the case of two distinct mvPDA. Our goal is to transform the mvPDA into a PDA with a distinguished process such that this process satisfies certain μ-calculus formula if and only if $sA \leq_{\mathrm{m}} tB$.

We create a PDA Δ' with actions $Act' = \{att, def\}$, stack alphabet $\Gamma' = \mathcal{G} \times \mathcal{G}$ where $\mathcal{G} = \Gamma \cup (\Gamma \times \Gamma) \cup (\Gamma \times Act) \cup \{\varepsilon\}$, and control states $\mathcal{Q}' = \mathcal{Q} \times \mathcal{Q}$. We write Y_a instead of (Y, a) as an element of $\mathcal{G}$.

We use a (stack merging) partial mapping $[X\alpha, Y\beta] = (X, Y)[\alpha, \beta]$, $[\varepsilon, \varepsilon] = \varepsilon$. In the following, we abuse the notation of the rules, as we did in the introduction, and write e.g. $pX \overset{a}{\dashrightarrow} p'\alpha$ instead of $(pX, a, p'\alpha) \in \Delta_{\mathrm{may}}$.

The set of rules of Δ' is as follows:

– Whenever $pX \overset{a}{\dashrightarrow} p'\alpha$ then
  - $(p,q)(X,Y) \xrightarrow{att} (p',q)(\alpha, Y_a)$ for every $q \in \mathcal{Q}$ and $Y \in \Gamma$
  - $(q,p)(\beta, X_a) \xrightarrow{def} (q,p')[\beta,\alpha]$ for every $q \in \mathcal{Q}$ and $\beta \in \Gamma \times \Gamma \cup \Gamma \cup \{\varepsilon\}$
– Whenever $pX \xrightarrow{a} p'\alpha$ then
  - $(q,p)(Y,X) \xrightarrow{att} (q,p')(Y_a,\alpha)$ for every $q \in \mathcal{Q}$ and $Y \in \Gamma$
  - $(p,q)(X_a,\beta) \xrightarrow{def} (p',q)[\alpha,\beta]$ for every $q \in \mathcal{Q}$ and $\beta \in \Gamma \times \Gamma \cup \Gamma \cup \{\varepsilon\}$

Note that $[\alpha,\beta]$ and $[\beta,\alpha]$ is always well defined as $|\alpha| = |\beta|$ is guaranteed (the transition that created β has to have the same label as the transition that creates α – this is guaranteed via X_a). We conclude by the following claim whose proof can be found in [BK12].

Claim. Let φ denote an alternation-free μ-calculus formula $\nu Z.[att]\langle def\rangle Z$. Then $sA \leq_{\mathrm{m}} tB$ iff $(s,t)(A,B) \models \varphi$ □

The following theorem can be proved using complexity bounds for μ-calculus model checking, as in [Srb06].

Theorem 5. *The problem mvPDA$\leq_{\mathrm{m}}$mvPDA is EXPTIME-complete, the problem mvBPA$\leq_{\mathrm{m}}$mvBPA is PTIME-complete.*

4.2 Birefinement

Since the refinement is often undecidable, the same holds for refinement equivalence ($\leq_{\mathrm{m}} \cap \geq_{\mathrm{m}}$). Nevertheless, one can consider an even stronger relation that is still useful. We define the notion of birefinement as the modification of refinement where we require both conditions of Definition 2 to be satisfied in both directions, similarly as bisimulation can be defined as a symmetric simulation.

Definition 5 (Birefinement). *A birefinement is a symmetric refinement. We say that α birefines β ($\alpha \sim_{\mathrm{m}} \beta$) if there exists a birefinement containing (α,β).*

This notion then naturally captures the bisimilarity of modal transition systems. Furthermore, the birefinement problem on MTS can be reduced to bisimulation on LTS in the following straightforward way. Let $(\Delta_{\mathrm{may}}, \Delta_{\mathrm{must}})$ and $(\Gamma_{\mathrm{may}}, \Gamma_{\mathrm{must}})$ be two mPRS over the same action alphabet Act. We create a new action $\overline{a}$ for every $a \in Act$. We then translate the mPRS into ordinary PRS as follows. Let $\Delta = \Delta_{\mathrm{may}} \cup \{(\alpha,\overline{a},\beta) \mid (\alpha,a,\beta) \in \Delta_{\mathrm{must}}\}$ and similarly for Γ. It is then clear that if we take two processes δ of $(\Delta_{\mathrm{may}}, \Delta_{\mathrm{must}})$ and γ of $(\Gamma_{\mathrm{may}}, \Gamma_{\mathrm{must}})$ then the following holds: δ birefines γ if and only if δ and γ are bisimilar when taken as processes of Δ and Γ, respectively.

The decidability and complexity of birefinement is thus identical to that of bisimulation in the non-modal case. Therefore, we may apply the powerful result that bisimilarity between *any* PRS and FSM is decidable [KŘS05] to get the following theorem.

Theorem 6. *Birefinement between an mFSM and any mPRS is decidable.*

This is an important result since it allows us to check whether we can replace an infinite MTS with a particular finite one, which in turn may allow for checking further refinements.

Table 1. Summary of the decidability results

decidable	mFSM $\stackrel{\leq}{>}_{\mathrm{m}}$ mPDA, mvPDA $\stackrel{\leq}{>}_{\mathrm{m}}$ mvPDA, mFSM $\sim_{\mathrm{m}}$ mPRS
undecidable	mFSM $\stackrel{\leq}{>}_{\mathrm{m}}$ mBPP, mBPA $\stackrel{\leq}{>}_{\mathrm{m}}$ mBPA

5 Conclusions

We have defined a generic framework for infinite-state modal transition systems generated by finite descriptions. We investigated the corresponding notion of modal refinement on important subclasses and determined the decidability border, see Table 1. Although in some classes it is possible to extend the decidability of simulation to decidability of refinement, it is not possible always. We have shown that somewhat surprisingly the parallelism is a great obstacle for deciding the refinement relation. Therefore, the future work will concentrate on identifying conditions leading to decidability. One of the best candidates is imposing determinism, which has a remarkable effect on the complexity of the problem in the finite case [BKLS09] as well as in the only infinite case considered so far, namely modal Petri nets [EBHH10]. Further, we leave the question whether the problem becomes decidable in some cases when the refining system is an implementation open, too. Finally, it remains open to what extent can verification results on finite MTS, such as [BČK11], be extended to infinite-state MTS.

References

[AFdFE+11] L. Aceto, I. Fábregas, D. de Frutos-Escrig, A. Ingólfsdóttir, and M. Palomino. Graphical representation of covariant-contravariant modal formulae. In *EXPRESS*, pages 1–15, 2011.

[AHKV98] R. Alur, T. A. Henzinger, O. Kupferman, and M. Y. Vardi. Alternating refinement relations. In *CONCUR*, pages 163–178, 1998.

[AM04] R. Alur and P. Madhusudan. Visibly pushdown languages. In *STOC*, pages 202–211, 2004.

[BČK11] N. Beneš, I. Černá, and J. Křetínský. Modal transition systems: Composition and LTL model checking. In *ATVA*, pages 228–242, 2011.

[BFJ+11] S. S. Bauer, U. Fahrenberg, L. Juhl, K. G. Larsen, A. Legay, and C. R. Thrane. Quantitative refinement for weighted modal transition systems. In *MFCS*, volume 6907 of *LNCS*, pages 60–71. Springer, 2011.

[BG00] G. Bruns and P. Godefroid. Generalized model checking: Reasoning about partial state spaces. In *CONCUR*, pages 168–182, 2000.

[BHJ10] S. S. Bauer, R. Hennicker, and S. Janisch. Interface theories for (a)synchronously communicating modal I/O-transition systems. In *FIT*, pages 1–8, 2010.

[BK10] N. Beneš and J. Křetínský. Process algebra for modal transition systemses. In *MEMICS*, pages 9–18, 2010.

[BK12] N. Beneš and J. Křetínský. Modal process rewrite systems. Technical report FIMU-RS-2012-02, Faculty of Informatics, Masaryk University, Brno, 2012.

[BKL+11] N. Beneš, J. Křetínský, K. G. Larsen, M. H. Møller, and J. Srba. Parametric modal transition systems. In *ATVA*, pages 275–289, 2011.

[BKL+12] N. Beneš, J. Křetínský, K. G. Larsen, M. H. Møller, and J. Srba. Dual-priced modal transition systems with time durations. In *LPAR*, pages 122–137, 2012.

[BKLS09] N. Beneš, J. Křetínský, K. G. Larsen, and J. Srba. On determinism in modal transition systems. *Theor. Comput. Sci.*, 410(41):4026–4043, 2009.

[BL90] G. Boudol and K. G. Larsen. Graphical versus logical specifications. In *CAAP*, pages 57–71, 1990.

[BLPR11] N. Bertrand, A. Legay, S. Pinchinat, and J.-B. Raclet. Modal event-clock specifications for timed component-based design. *Science of Computer Programming*, To appear, 2011.

[BLS95] A. Børjesson, K. G. Larsen, and A. Skou. Generality in design and compositional verification using TAV. *Formal Methods in System Design*, 6(3):239–258, 1995.

[BML11] S. S. Bauer, P. Mayer, and A. Legay. MIO workbench: A tool for compositional design with modal input/output interfaces. In *ATVA*, pages 418–421, 2011.

[Bru97] G. Bruns. An industrial application of modal process logic. *Sci. Comput. Program.*, 29(1-2):3–22, 1997.

[CDL+10] B. Caillaud, B. Delahaye, K. G. Larsen, A. Legay, M. L. Pedersen, and A. Wasowski. Compositional design methodology with constraint markov chains. In *QEST*, pages 123–132, 2010.

[ČGL93] K. Čerāns, J. C. Godskesen, and K. G. Larsen. Timed modal specification - theory and tools. In *CAV*, pages 253–267, 1993.

[CGLT09] A. Campetelli, A. Gruler, M. Leucker, and D. Thoma. *Don't Know* for multi-valued systems. In *ATVA*, pages 289–305, 2009.

[dAH01] L. de Alfaro and T. A. Henzinger. Interface automata. In *ESEC / SIGSOFT FSE*, pages 109–120, 2001.

[DFFU07] N. D'Ippolito, D. Fischbein, H. Foster, and S. Uchitel. MTSA: Eclipse support for modal transition systems construction, analysis and elaboration. In *ETX*, pages 6–10, 2007.

[DGG97] D. Dams, R. Gerth, and O. Grumberg. Abstract interpretation of reactive systems. *ACM Trans. Program. Lang. Syst.*, 19(2):253–291, 1997.

[DLL+10] A. David, K. G. Larsen, A. Legay, U. Nyman, and A. Wasowski. ECDAR: An environment for compositional design and analysis of real time systems. In *ATVA*, pages 365–370, 2010.

[DN04] D. Dams and K. S. Namjoshi. The existence of finite abstractions for branching time model checking. In *LICS*, pages 335–344, 2004.

[EBHH10] D. Elhog-Benzina, S. Haddad, and R. Hennicker. Process refinement and asynchronous composition with modalities. In *ACSD/Petri Nets Workshops*, pages 385–401, 2010.

[FS08] H. Fecher and H. Schmidt. Comparing disjunctive modal transition systems with an one-selecting variant. *J. Log. Algebr. Program.*, 77(1-2):20–39, 2008.

[GH94] J. F. Groote and H. Hüttel. Undecidable equivalences for basic process algebra. *Inf. Comput.*, 115(2):354–371, 1994.

[GHJ01] P. Godefroid, M. Huth, and R. Jagadeesan. Abstraction-based model checking using modal transition systems. In *CONCUR*, pages 426–440, 2001.

[GNRT10] P. Godefroid, A. V. Nori, S. K. Rajamani, and S. Tetali. Compositional may-must program analysis: unleashing the power of alternation. In *POPL*, pages 43–56, 2010.

[HJS01] M. Huth, R. Jagadeesan, and D. A. Schmidt. Modal transition systems: A foundation for three-valued program analysis. In *ESOP*, pages 155–169, 2001.

[Hüt94] H. Hüttel. Undecidable equivalences for basic parallel processes. In *TACS*, pages 454–464, 1994.

[JLS11] L. Juhl, K. G. Larsen, and J. Srba. Modal transition systems with weight intervals. *Journal of Logic and Algebraic programming*, 2011. To appear.

[JM95] P. Jancar and F. Moller. Checking regular properties of petri nets. In *CONCUR*, pages 348–362, 1995.

[JP01] B. Jacobs and E. Poll. A logic for the java modeling language JML. In *FASE*, pages 284–299, 2001.

[KM99] A. Kučera and R. Mayr. Simulation preorder on simple process algebras. In *ICALP*, pages 503–512, 1999.

[KM02a] A. Kučera and R. Mayr. On the complexity of semantic equivalences for pushdown automata and BPA. In *MFCS*, volume 2420 of *Lecture Notes in Computer Science*, pages 433–445. Springer, 2002.

[KM02b] A. Kučera and R. Mayr. Why is simulation harder than bisimulation? In *CONCUR*, pages 594–610, 2002.

[KŘS05] M. Křetínský, V. Řehák, and J. Strejček. Reachability of hennessy-milner properties for weakly extended PRS. In *FSTTCS*, pages 213–224, 2005.

[LNW07] K. G. Larsen, U. Nyman, and A. Wasowski. Modal I/O automata for interface and product line theories. In *ESOP*, pages 64–79, 2007.

[LT88] K. G. Larsen and B. Thomsen. A modal process logic. In *LICS*, pages 203–210, 1988.

[LW94] B. Liskov and J. M. Wing. A behavioral notion of subtyping. *ACM Trans. Program. Lang. Syst.*, 16(6):1811–1841, 1994.

[LX90] K. G. Larsen and L. Xinxin. Equation solving using modal transition systems. In *LICS*, pages 108–117, 1990.

[Lyn88] N. Lynch. I/O automata: A model for discrete event systems. In *22nd Annual Conference on Information Sciences and Systems*, pages 29–38. Princeton University, 1988.

[May00] R. Mayr. Process rewrite systems. *Inf. Comput.*, 156(1-2):264–286, 2000.

[Nam03] K. S. Namjoshi. Abstraction for branching time properties. In *CAV*, pages 288–300, 2003.

[Nym08] U. Nyman. *Modal Transition Systems as the Basis for Interface Theories and Product Lines*. PhD thesis, Institut for Datalogi, Aalborg Universitet, 2008.

[RBB+09] J.-B. Raclet, E. Badouel, A. Benveniste, B. Caillaud, and R. Passerone. Why are modalities good for interface theories? In *ACSD*. IEEE Computer Society Press, 2009.

[RBB+11] J.-B. Raclet, E. Badouel, A. Benveniste, B.Caillaud, A. Legay, and R. Passerone. A modal interface theory for component-based design. *Fundamenta Informaticae*, 108(1-2):119–149, 2011.

[Srb06] J. Srba. Visibly pushdown automata: From language equivalence to simulation and bisimulation. In *CSL*, pages 89–103, 2006.

[UC04] S. Uchitel and M. Chechik. Merging partial behavioural models. In *SIGSOFT FSE*, pages 43–52, 2004.

Paper F:

On refinements of Boolean and parametric modal transition systems

Jan Křetínský and Salomon Sickert

This paper has been published in Zhiming Liu, Jim Woodcock, and Huibiao Zhu (eds.): Theoretical Aspects of Computing - ICTAC 2013 - 10th International Colloquium, Shanghai, China, September 4-6, 2013. Proceedings, volume 8049 of Lecture Notes in Computer Science, pages 213–23. Springer, 2013. [KS13b]

Summary

As previously advocated, BMTS and PMTS are more convenient and concise modelling frameworks than MTS. Due to conciseness the refinement problems become harder and their theoretical complexity suggests it is infeasible to check them. Therefore, we reduce the problem of modal refinement over BMTS and PMTS to a problem solvable by a QBF solver. We provide promising experimental results showing this solution scales well. Further, we extend the algorithm for thorough refinement on MTS and DMTS to BMTS and PMTS providing better complexity then via translation of these formalisms to DMTS. This also shows, together with results on modal refinement, that we can make use of the more compact representation used in the formalisms of BMTS and PMTS. Since the complexity of the thorough refinement is still too high, we also investigate the relationship between modal and thorough refinement on BMTS and PMTS and introduce approximation methods for the thorough refinement on BMTS and PMTS through the modal refinement.

Author's contribution: 50 %

- participating in the discussions,
- contributing, in particular, to setting up the research direction, formulating and proving claims on the thorough refinement and its relationship to the modal refinement,
- writing the paper except the reduction of the modal refinement to QBF queries.

On Refinements of Boolean and Parametric Modal Transition Systems

Jan Křetínský[1,2⋆] and Salomon Sickert[1⋆⋆]

[1] Institut für Informatik, Technische Universität München, Germany
[2] Faculty of Informatics, Masaryk University, Brno, Czech Republic

Abstract. We consider the extensions of modal transition systems (MTS), namely Boolean MTS and parametric MTS and we investigate the refinement problems over both classes. Firstly, we reduce the problem of modal refinement over both classes to a problem solvable by a QBF solver and provide experimental results showing our technique scales well. Secondly, we extend the algorithm for thorough refinement of MTS providing better complexity than via reductions to previously studied problems. Finally, we investigate the relationship between modal and thorough refinement on the two classes and show how the thorough refinement can be approximated by the modal refinement.

1 Introduction

Due to the ever increasing complexity of software systems and their re-use, component-based design and verification have become crucial. Therefore, having a specification formalism that supports *component-based* development and *stepwise refinement* is very useful. In such a framework, one can start from an initial specification, proceed with a series of small and successive refinements until eventually a specification is reached from which an implementation can be extracted directly. In each refinement step, we can replace a single component of the current specification with a more concrete/implementable one. The correctness of such a step should follow from the correctness of the refinement of the replaced component, so that the methodology supports *compositional* verification.

Modal transition systems (MTS) were introduced by Larsen and Thomsen [LT88] in order to obtain an operational, yet expressive and manageable specification formalism meeting the above properties. Their success resides in natural combination of two features. Firstly, the simplicity of labelled transition systems, which have proved appropriate for behavioural description of systems as well as their compositions; MTS as their extension inherit this appropriateness. Secondly, as opposed to e.g. temporal logic specifications, MTS can be easily *gradually refined* into implementations while preserving the desired behavioural

⋆ The author is partially supported by the Czech Science Foundation, project No. P202/10/1469

⋆⋆ The author is partially funded by the DFG project "Polynomial Systems on Semirings: Foundations, Algorithms, Applications"

properties. In this work, we focus on checking the refinement between MTS and also their recent extensions.

The formalism of MTS has proven to be useful in practice. Industrial applications are as old as [Bru97] where MTS have found use for an air-traffic system at Heathrow airport. Besides, MTS are advocated as an appropriate base for interface theories in [RBB+09] and for product line theories in [Nym08]. Further, MTS based software engineering methodology for design via merging partial descriptions of behaviour has been established in [UC04]. Moreover, the tool support is quite extensive, e.g. [BLS95,DFFU07,BML11,BČK11].

MTS consist of a set of states and two transition relations. The *must* transitions prescribe which behaviour has to be present in every refinement of the system; the *may* transitions describe the behaviour that is allowed, but need not be realized in the refinements. This allows for underspecification of non-critical behaviour in the early stage of design, focusing on the main properties, verifying them and sorting out the details of the yet unimplemented non-critical behaviour later.

Over the years, many extensions of MTS have been proposed. While MTS can only specify whether or not a particular transition is required, some extensions equip MTS with more general abilities to describe what *combinations* of transitions are possible. Disjunctive MTS (DMTS) [LX90] can specify that at least one of a given set of transitions is present. One selecting MTS [FS08] allow to choose exactly one of them. Boolean MTS (BMTS) [BKL+11] cover all Boolean combinations of transitions. The same holds for acceptance automata [Rac07] and Boolean formulae with states [BDF+], which both express the requirement by listing all possible sets instead of a Boolean formula. Parametric MTS (PMTS) [BKL+11] add parameters on top of it, so that we can also express persistent choices of transitions and relate possible choices in different parts of a system. This way, one can model hardware dependencies of transitions and systems with prices [BKL+12].

Our contribution In this paper, we investigate extensions of MTS with respect to two notions of refinement. The *modal refinement* is a syntactically defined notion extending on the one hand bisimulation and on the other hand simulation. Similarly to bisimulation having a counterpart in trace equivalence, here the counterpart of modal refinement is the *thorough refinement*. It is the corresponding semantically defined notion relating (by inclusion) the sets of implementations of the specifications.

We focus both on theoretical and practical complexity of the refinement problems. While modal refinement on MTS and disjunctive MTS can be decided in polynomial time, on BMTS and PMTS it is higher in the polynomial hierarchy (Π_2 and Π_4, respectively). The huge success of SAT and also QBF solvers inspired us to reduce these refinement problems to problems solvable by a QBF solver. We have also performed experiments showing that this solution scales well in the size of the system as well as in the number of parameters, while a direct naive solution is infeasible.

Furthermore, we extend the decision algorithm for thorough refinement checking over MTS [BKLS12] and DMTS [BČK10] to the setting of BMTS and PMTS. We show how PMTS can be translated to BMTS and BMTS can then be transformed to DMTS. As we can decide the problem on DMTS in EXPTIME, this shows decidability for BMTS and PMTS, but each of the translations is inevitably exponential. However, we show better upper bounds than doubly and triply exponential. To this end, we give also a direct algorithm for showing the problem is in NEXPTIME for BMTS and 2-EXPTIME for PMTS.

Since the thorough refinement is EXPTIME-hard for already MTS, it is harder than the modal refinement, which is in P for DMTS and in Π_4 for PMTS. Therefore, we also investigate how the thorough refinement can be approximated by the modal refinement. While underapproximation is easy, as modal refinement implies thorough refinement, overapproximation is more difficult. Here we extend our method of the deterministic hull for MTS [BKLS09] to both BMTS and PMTS. We prove that for BMTS modal and thorough refinements coincide if the refined system is deterministic, which then yields an overapproximation via the deterministic hull. Finally, in the case with PMTS, we need to overapproximate the behaviour dependent on the parameters, because the coincidence of the refinements on deterministic systems fails for PMTS.

Our contribution can be summarized as follows:

- We reduce the problem of modal refinement over BMTS and PMTS to a problem solvable by a QBF solver. We provide promising experimental results showing this solution scales well.
- We extend the algorithm for thorough refinement on MTS and DMTS to BMTS and PMTS providing better complexity then via translation of these formalisms to DMTS. This also shows (together with results on modal refinement) that we can make use of the more compact representation used in the formalisms of BMTS and PMTS.
- We investigate the relationship between modal and thorough refinement on BMTS and PMTS. We introduce approximation methods for the thorough refinement on BMTS and PMTS through the modal refinement.

Related work There are various other approaches to deal with component *refinements*. They range from subtyping [LW94] over Java modelling language [JP01] to interface theories close to MTS such as interface automata [dAH01]. Similarly to MTS, interface automata are behavioural interfaces for components. However, their composition works very differently. Furthermore, its notion of refinement is based on alternating simulation [AHKV98], which has been proved strictly less expressive than MTS refinement—actually coinciding on a subclass of MTS—in the paper [LNW07], which combines MTS and interface automata based on I/O automata [Lyn88]. The compositionality of this combination is further investigated in [RBB+11].

Further, alternatively to the design of correct software where an abstract verified MTS is transformed into a concrete implementation, one can consider checking correctness of software through *abstracting* a concrete implementation

into a coarser system. The use of MTS as abstractions has been advocated e.g. in [GHJ01]. While usually overapproximations (or underapproximations) of systems are constructed and thus only purely universal (or existential) properties can be checked, [GHJ01] shows that using MTS one can check mixed formulae (arbitrarily combining universal and existential properties) and, moreover, at the same cost as checking universal properties using traditional conservative abstractions. This advantage has been investigated also in the context of systems equivalent or closely related to MTS [HJS01,DGG97,Nam03,DN04,CGLT09,GNRT10].

MTS can also be viewed as a fragment of mu-calculus that is "graphically representable" [BL90,BDF$^+$]. The graphical representability of a variant of alternating simulation called covariant-contravariant simulation has been recently studied in [AFdFE$^+$11].

Outline of the paper In Section 2, we recall the formalism of MTS and the extensions discussed and in Section 3 the modal refinement problem is restated. We then reduce it to a QBF problem in Section 4. In Section 5, we give a solution to the thorough refinement problems. Section 6 investigates the relationship of the two refinements and how modal refinement can approximate the thorough refinement. We conclude in Section 7.

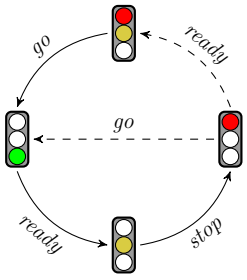
2 Modal Transition Systems and Boolean and Parametric Extensions

In this section, we introduce the studied formalisms of modal transition systems and their Boolean and parametric extensions. We first recall the standard definition of MTS:

Definition 2.1. *A* modal transition system (MTS) *over an action alphabet Σ is a triple $(S, \dashrightarrow, \longrightarrow)$, where S is a set of* states *and $\longrightarrow \subseteq \dashrightarrow \subseteq S \times \Sigma \times S$ are* must *and* may *transition relations, respectively.*

The MTS are often drawn as follows. Unbroken arrows denote the must (and underlying may) transitions while dashed arrows denote may transitions where there is no must transition.

Example 2.2. The MTS on the right is adapted from [BKL$^+$11] and models commonly used types of traffic lights. In state *green* on the left there is a must transition under *ready* to state *yellow* from which there is must transition to *red*. Here transitions to *yellowRed* and back to *green* are may transition. Intuitively, this means that any final implementation may have either one, both or none of the transitions. In contrast, the must transitions are present in all implementations.



Note that using MTS, we cannot express the set of implementations with exactly one of the transitions in

red. For that, we can use Boolean MTS [BKL$^+$11] instead, which can express not only arbitrary conjunctions and disjunctions, but also negations and thus also exclusive-or. However, in Boolean MTS it may still happen that an implementation alternates transitions to *green* and *yellowRed* between two traffic lights cycles. To make sure the choice will remain the same in the whole implementation, parametric MTS have been introduced [BKL$^+$11] extending the Boolean MTS.

Before we define the most general class - the parametric MTS - and derive other classes as special cases, we first recall the standard propositional logic. A Boolean formula over a set X of atomic propositions is given by the following abstract syntax

$$\varphi ::= \mathbf{tt} \mid x \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi$$

where x ranges over X. The set of all Boolean formulae over the set X is denoted by $\mathcal{B}(X)$. Let $\nu \subseteq X$ be a valuation, i.e. a set of variables with value true, then the satisfaction relation $\nu \models \varphi$ is given by $\nu \models \mathbf{tt}$, $\nu \models x$ iff $x \in \nu$, and the satisfaction of the remaining Boolean connectives is defined in the standard way. We also use the standard derived operators like exclusive-or $\varphi \oplus \psi := (\varphi \wedge \neg\psi) \vee (\neg\varphi \wedge \psi)$, implication $\varphi \Rightarrow \psi := \neg\varphi \vee \psi$ and equivalence $\varphi \Leftrightarrow \psi := (\neg\varphi \vee \psi) \wedge (\varphi \vee \neg\psi)$.

We can now proceed with the definition of parametric MTS. In essence, it is a labelled transition system, in which we can specify which transitions can be present depending on values of some fixed parameters.

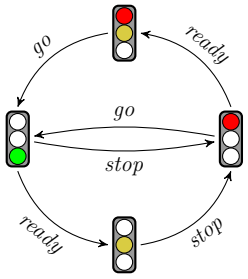
Definition 2.3. *A* parametric modal transition system (PMTS) *over an action alphabet Σ is a tuple (S, T, P, Φ) where*

- *S is a set of* states*,*
- *$T \subseteq S \times \Sigma \times S$ is a* transition relation*,*
- *P is a finite set of* parameters*, and*
- *$\Phi : S \to \mathcal{B}((\Sigma \times S) \cup P)$ is an* obligation function *over the outgoing transitions and parameters. We assume that whenever (a, t) occurs in $\Phi(s)$ then $(s, a, t) \in T$.*

A Boolean modal transition system (BMTS) *is a PMTS with the set of parameters P being empty. A* disjunctive MTS (DMTS) *is a BMTS with the obligation function in conjunctive normal form and using no negation. An* implementation *(or* labelled transition system*) is a BMTS with $\Phi(s) = \bigwedge_{(s,a,t)\in T}(a, t)$ for each $s \in S$.*

An MTS is then a BMTS with $\Phi(s)$ being a conjunction of positive literals (some of the outgoing transitions), for each $s \in S$. More precisely, $\dashrightarrow$ is the same as T, and $(s, a, t) \in \longrightarrow$ if and only if (a, t) is one of the conjuncts of $\Phi(s)$.

Example 2.4. A PMTS which captures the traffic lights used in Europe for cars and pedestrians is depicted below. Depending on the valuation of parameter *reqYellow*, we either always use the yellow light between the red and green lights, or we never do. The transition relation is depicted using unbroken arrows.



Parameters: $P = \{reqYellow\}$

Obligation function:
$\Phi(green) = ((stop, red) \oplus (ready, yellow))$
$\wedge(reqYellow \Leftrightarrow (ready, yellow))$
$\Phi(yellow) = (stop, red)$
$\Phi(red) = ((go, green) \oplus (ready, yellowRed))$
$\wedge(reqYellow \Leftrightarrow (ready, yellowRed))$
$\Phi(yellowRed) = (go, green)$

3 Modal Refinement

A fundamental advantage of MTS-based formalisms is the presence of *modal refinement* that allows for a step-wise system design (see e.g. [AHL+08]). We start with the standard definition of modal refinement for MTS and then discuss extensions to BMTS and PMTS.

Definition 3.1 (MTS Modal Refinement). *For states s_0 and t_0 of MTS $(S_1, \longrightarrow_1, \dashrightarrow_1)$ and $(S_2, \longrightarrow_2, \dashrightarrow_2)$, respectively, we say that s_0* modally refines *t_0, written $s_0 \leq_{\mathrm{m}} t_0$, if (s_0, t_0) is contained in a relation $R \subseteq S_1 \times S_2$ satisfying for every $(s,t) \in R$ and every $a \in \Sigma$:*

1. *if $s \overset{a}{\dashrightarrow}_1 s'$ then there is a transition $t \overset{a}{\dashrightarrow}_2 t'$ with $(s', t') \in R$, and*
2. *if $t \overset{a}{\longrightarrow}_2 t'$ then there is a transition $s \overset{a}{\longrightarrow}_1 s'$ with $(s', t') \in R$.*

Intuitively, $s \leq_{\mathrm{m}} t$ iff whatever s can do is allowed by t and whatever t requires can be done by s. Thus s is a refinement of t, or t is an abstraction of s. Furthermore, an *implementation of* s is a state i of an implementation (labelled transition system) with $i \leq_{\mathrm{m}} s$.

In [BKL+11], the modal refinement has been extended to PMTS (and thus BMTS) so that it coincides with the standard definition in the MTS case. We first recall the definition for BMTS. To this end, we set the following notation. Let (S, T, P, Φ) be a PMTS and $\nu \subseteq P$ be a valuation. For $s \in S$, we write $T(s) = \{(a,t) \mid (s,a,t) \in T\}$ and denote by

$$\mathrm{Tran}_\nu(s) = \{E \subseteq T(s) \mid E \cup \nu \models \Phi(s)\}$$

the set of all admissible sets of transitions from s under the fixed truth values of the parameters. In the case of BMTS, we often write Tran instead of $\mathrm{Tran}_\emptyset$.

Definition 3.2 (BMTS Modal Refinement). *For states s_0 and t_0 of BMTS $(S_1, T_1, \emptyset, \Phi_1)$ and $(S_2, T_2, \emptyset, \Phi_2)$, respectively, we say that s_0* modally refines *t_0, written $s_0 \leq_{\mathrm{m}} t_0$, if (s_0, t_0) is contained in a relation $R \subseteq S_1 \times S_2$ satisfying for every $(s,t) \in R$:*

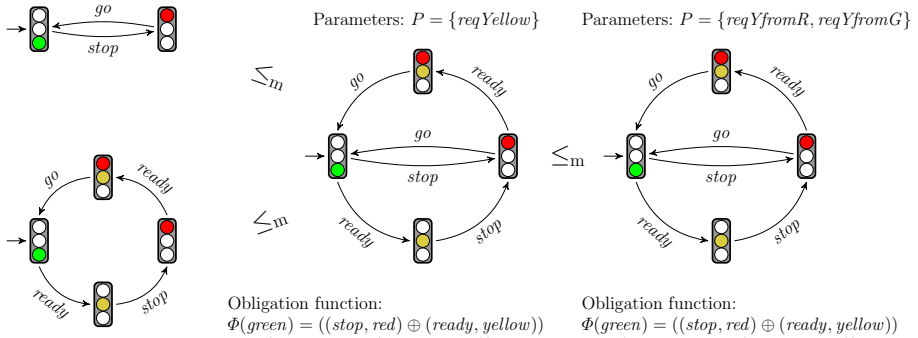
$$\forall M \in \mathrm{Tran}(s) : \exists N \in \mathrm{Tran}(t) : \ \forall (a,s') \in M : \exists (a,t') \in N : (s',t') \in R \ \wedge \\ \forall (a,t') \in N : \exists (a,s') \in M : (s',t') \in R \ .$$

For PMTS, we propose here a slightly altered definition, which corresponds more to the intuition, is closer to the semantically defined notion of thorough refinement, but still keeps the same complexity as established in [BKL$^+$11]. We use the following notation. For a PMTS $\mathcal{M} = (S, T, P, \Phi)$, a valuation $\nu \subseteq P$ of parameters induces a BMTS $\mathcal{M}^\nu = (S, T, \emptyset, \Phi')$ where each occurrence of $p \in \nu$ in Φ is replaced by $\mathbf{tt}$ and of $p \notin \nu$ by $\neg\mathbf{tt}$, i.e. $\Phi'(s) = \Phi(s)[\mathbf{tt}/p$ for $p \in \nu, \mathbf{ff}/p$ for $p \notin \nu]$ for each $s \in S$. We extend the notation to states and let s^ν denote the state of $\mathcal{M}^\nu$ corresponding to the state s of $\mathcal{M}$.

Definition 3.3 (PMTS Modal Refinement). *For states s_0 and t_0 of PMTS (S_1, T_1, P_1, Φ_1) and (S_2, T_2, P_2, Φ_2), we say that s_0* modally refines *t_0, written $s_0 \leq_\mathrm{m} t_0$, if for every $\mu \subseteq P_1$ there exists $\nu \subseteq P_2$ such that $s_0^\mu \leq_\mathrm{m} t_0^\nu$.*

Before we comment on the difference to the original definition, we illustrate the refinement on an example of [BKL$^+$11] where both definitions coincide.

Example 3.4. Consider the rightmost PMTS below. It has two parameters, namely *reqYfromG* and *reqYfromR* whose values can be set independently and it can be refined by the system in the middle of the figure having only one parameter *reqYellow*. This single parameter simply binds the two original parameters to the same value. The PMTS in the middle can be further refined into the implementations where either yellow is always used in both cases, or never at all as discussed in the previous example. Up to bisimilarity, the *green* state of this system only has the two implementations on the left.

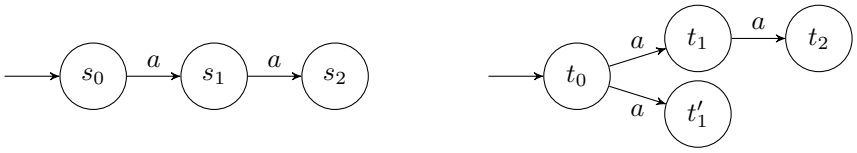


The original version of [BKL$^+$11] requires for $s_0 \leq_\mathrm{m} t_0$ to hold that there be a *fixed* $R \subseteq S_1 \times S_2$ such that for every $\mu \subseteq P_1$ there exists $\nu \subseteq P_2$ satisfying for each $(s, t) \in R$

$$\forall M \in \mathrm{Tran}_\mu(s) : \exists N \in \mathrm{Tran}_\nu(t) : \ \forall (a, s') \in M : \exists (a, t') \in N : (s', t') \in R \ \wedge \\ \forall (a, t') \in N : \exists (a, s') \in M : (s', t') \in R \ .$$

Clearly, the original definition is stronger: For any two PMTS states, if $s_0 \leq_{\mathrm{m}} t_0$ holds according to [BKL$^+$11] it also holds according to Definition 3.3. Indeed, the relation for any sets of parameters can be chosen to be the fixed relation R. On the other hand, the opposite does not hold.

Example 3.5. Consider the PMTS on the left with parameter set $\{p\}$ and obligation $\Phi(s_0) = (a, s_1),\ \Phi(s_1) = (b, s_2) \Leftrightarrow p,\ \Phi(s_2) = \mathbf{tt}$ and the PMTS on the right with parameter set $\{q\}$ and obligation $\Phi(t_0) = \big((a, t_1) \Leftrightarrow q\big) \wedge \big((a, t_1') \Leftrightarrow \neg q\big),\ \Phi(t_1) = (a, t_2),\ \Phi(t_2) = \Phi(t_1') = \mathbf{tt}$. On the one hand, according to our definition $s_0 \leq_{\mathrm{m}} t_0$, we intuitively agree this should be the case (and note they also have the same set of implementations). On the other hand, the original definition does not allow to conclude modal refinement between s_0 and t_0. The reason is that depending on the value of p, s_1 is put in the relation either with t_1 (for p being true and thus choosing q true, too) or with t_1' (for p being false and thus choosing q false, too). In contrast to the original definition, our definition allows us to pick different relations for different parameter valuations.



We propose our modification of the definition since it is more intuitive and for all considered fragments of PMTS has the same complexity as the original one. Note that both definitions coincide on BMTS. Further, on MTS they coincide with Definition 3.1 and on labelled transition systems with bisimulation.

4 Modal Refinement Checking

In this section, we show how to solve the modal refinement problem on BMTS and PMTS using QBF solvers. Although modal refinement is Π_2-complete (the second level of the polynomial hierarchy) on BMTS and Π_4-complete on PMTS (see [BKL$^+$11]), this way we obtain a solution method that is practically fast. We have implemented the approach and document its scalability with experimental results.

As mentioned, in order to decide whether modal refinement holds between two states, a reduction to a quantified boolean formula will be used. First, we recall the QBF decision problems.

Definition 4.1 (QBF_n^Q)**.** *Let Ap be a set of atomic propositions, which is partitioned into n sets with $Ap = \bigcup_{i=0}^{n} X_i$, and $\phi \in \mathcal{B}(Ap)$ a boolean formula over this set of atomic propositions. Let $Q \in \{\forall, \exists\}$ be a quantifier and $\overline{} : \{\forall \mapsto \exists, \exists \mapsto \forall\}$ a function. Then a formula*

$$QX_1\overline{Q}X_2QX_3 \ldots \widetilde{Q}X_n\phi \quad \text{with } \widetilde{Q} = \begin{cases} Q & \text{if } n \text{ is odd} \\ \overline{Q} & \text{if } n \text{ is even} \end{cases}$$

is an instance of QBF_n^Q if it is satisfiable.

Satisfiability means that if e.g. $Q = \exists$ there is some partial valuation for the atomic propositions in X_1, such that for all partial valuations for the elements of X_2, there is another partial valuation for the propositions of X_3 and so on up to X_n, such that ϕ is satisfied by the union of all partial valuations. It is well known that these problems are complete for the polynomial hierarchy: For each $i \geq 1$, $QBF_i^{\exists}$ is Σ_i-complete and $QBF_i^{\forall}$ is Π_i-complete.

4.1 Construction for BMTS

Due to the completeness of QBF problems and the results of [BKL$^+$11], it is possible to polynomially reduce modal refinement on BMTS to $QBF_2^{\forall}$. However, we would then have to perform a fixpoint computation to compute the refinement relation causing numerous invocations of the external QBF solver. Additionally this approach is not applicable in the PMTS case, hence we reduce modal refinement to $QBF_3^{\exists}$.

Let $s \in S_1$ and $t \in S_2$ be processes of two arbitrary BMTSs $\mathcal{M}_1 = (S_1, T_1, \emptyset, \Phi_1)$ and $\mathcal{M}_2 = (S_2, T_2, \emptyset, \Phi_2)$. Furthermore let

$$Ap = \underbrace{(S_1 \times S_2)}_{X_R} \uplus \underbrace{T_1}_{X_{T1}} \uplus \underbrace{(S_1 \times T_2)}_{X_{T2}}$$

be a set of atomic propositions. The intended meaning is that $(u, v) \in X_R$ is assigned $\mathbf{tt}$ if and only if it is also contained in the modal refinement relation R. Further, X_{T1} and X_{T2} are used to talk about the transitions. The prefix S_1 is attached to the set T_2 because $N \in \mathrm{Tran}(t)$ with $t \in S_2$ must be chosen independently for different states of S_1. This enables us to move the $\exists$ quantification.

We now construct a formula $\Psi_{s,t} \in \mathcal{B}(Ap)$ satisfying

$$s \leq_m t \quad \textit{iff} \quad \exists X_R \forall X_{T1} \exists X_{T2} \Psi_{s,t} \in QBF_3^{\exists} \tag{1}$$

To this end, we shall use a macro $\psi_{u,v}$ capturing the condition which has to be satisfied by any element $(u, v) \in R$. Furthermore, we ensure that (s, t) is assigned $\mathbf{tt}$ by every satisfying assignment for the formula by placing it directly in the conjunction:

$$\Psi_{s,t} = (s, t) \wedge \bigwedge_{(u,v) \in X_R} \big((u, v) \Rightarrow \psi_{u,v}\big) \tag{2}$$

It remains to define the macro $\psi_{u,v}$. We start with the modal refinement condition as a blueprint:

$$\begin{aligned} \forall M \in \mathrm{Tran}(u) : \exists N \in \mathrm{Tran}(v) : \ & \forall (a, u') \in M : \exists (a, v') \in N : (u', v') \in R \ \wedge \\ & \forall (a, v') \in N : \exists (a, u') \in M : (u', v') \in R \ . \end{aligned}$$

As M and N are subsets of $T_1(u)$ and $T_2(v)$, respectively, and are finite, the inner quantifiers can be expanded causing only a polynomial growth of the formula size (see [KS13]). Further, Tran sets are replaced by the original definition

and the outer quantifiers are moved in front of $\Psi_{s,t}$. As the state obligations are defined over a different set of atomic propositions ($\Phi(v) \in \mathcal{B}((\Sigma \times S) \cup P) \not\subseteq \mathcal{B}(Ap)$), a family of mapping functions π_p is introduced.

$$\begin{aligned} \pi_p : \mathcal{B}(\Sigma \times S) &\to \mathcal{B}(Ap) \\ \mathbf{tt} &\mapsto \mathbf{tt} \\ (a, x) &\mapsto (p, a, x) \quad \textit{with } a \in \Sigma,\ x \in S \\ \neg\varphi &\mapsto \neg\, \pi_p(\varphi) \\ \varphi_1 \wedge \varphi_2 &\mapsto \pi_p(\varphi_1) \wedge \pi_p(\varphi_2) \\ \varphi_1 \vee \varphi_2 &\mapsto \pi_p(\varphi_1) \vee \pi_p(\varphi_2) \end{aligned} \tag{3}$$

Applying these steps to the blueprint yields the following result:

$$\psi_{u,v} = \pi_u\left(\Phi_1\left(u\right)\right) \Rightarrow \pi_{u,v}\left(\Phi_2\left(v\right)\right) \wedge \varphi_{u,v} \tag{4}$$

$$\begin{aligned} \varphi_{u,v} = &\bigwedge_{\substack{u^* \in X_{T1} \\ u^* = (u,a,u')}} \left(u^* \Rightarrow \bigvee_{\substack{v^* \in X_{T2} \\ v^* = (u,v,a,v')}} \left(v^* \wedge (u', v')\right)\right) \\ &\wedge \bigwedge_{\substack{v^* \in X_{T2} \\ v^* = (u,v,a,v')}} \left(v^* \Rightarrow \bigvee_{\substack{u^* \in X_{T1} \\ u^* = (u,a,u')}} \left(u^* \wedge (u', v')\right)\right) \end{aligned} \tag{5}$$

Theorem 4.2. *For states s, t of a BMTS, we have*

$$s \leq_m t \quad \textit{iff} \quad \exists X_R \forall X_{T1} \exists X_{T2} \Psi_{s,t} \in QBF_3^{\exists}$$

Due to space constraints, the technical proof can be found in [KS13].

4.2 Construction for PMTS

We now reduce the modal refinement on PMTS to $QBF_4^{\forall}$, which now corresponds directly to the complexity established in [BKL+11]. Nevertheless, due to the first existential quantification in $\forall\exists\forall\exists$ alternation sequence, we can still guess the refinement relation using the QBF solver rather than compute the lengthy fixpoint computation.

In the PMTS case, we have to find for all parameter valuations for the system of s a valuation for the system of t, such that there exists a modal refinement relation containing (s, t). We simply choose universally a valuation for the parameters of the left system (the underlying system of s) and then existentially for the right system (the underlying system of t). Prior to checking modal refinement, the valuations are fixed, so the PMTS becomes a BMTS. This is accomplished by extending Ap with P_1 and P_2 and adding the necessary quantifiers to the formula. Thus we obtain the following:

Theorem 4.3. *For states s, t of a PMTS, we have*

$$s \leq_m t \quad \textit{iff} \quad \forall P_1 \exists P_2 \exists X_R \forall X_{T1} \exists X_{T2} \Psi_{s,t} \in QBF_4^{\forall}$$

4.3 Experimental Results

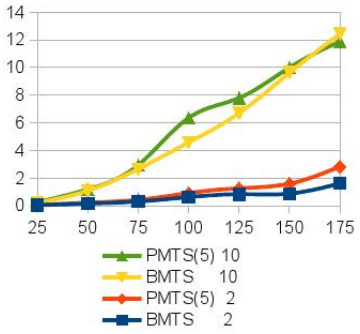
We now show how our method performs in practice. We implemented the reduction and linked it to the QBF solver Quantor. In order to evaluate whether our solution scales, we generate random samples of MTS, disjunctive MTS, Boolean MTS and parametric MTS with different numbers of parameters (as displayed in tables below in parenthesis). For each type of system and the number of reachable states (25 to 200 as displayed in columns), we generate several pairs of systems and compute the average time to check modal refinement between them. While the systems in Table 1 are generated independently, the refining systems in Table 2 are refinements of the abstract systems.

We show several sets of experiments. In Table 1, we consider (1) systems with alphabet of size 2 and all states with branching degree 2, and (2) systems with alphabet of size 10 and all states with branching degree 10. Further, in Table 2, we consider systems with alphabet of size 2 and all states with branching degree 5. Here we first consider the systems as above, i.e. with edges generated randomly so that they create a tree and with some additional "noise" edges thus making the branching degree constant. Second, we consider systems where we have different "clusters", each of which is interconnected with many edges. Each of these clusters has a couple of "interface" states, which are used to connect to other clusters. We use this class of systems to model system descriptions with more organic structure.

The entries in the tables are average running times in seconds. The standard deviation in our experiments was around 30-60%. The experiments were run on an Intel Core 2 Duo CPU P9600 2.66GHz with 3.8 GB RAM using Java 1.7. For more details, see [KS13].

Table 1. Experimental results: systems over alphabet of size 2 with branching degree 2 in the upper part, and systems over alphabet of size 10 with branching degree 10 in the lower part

	25	50	75	100	125	150	175	200
MTS	0.03	0.15	0.29	0.86	0.87	0.96	1.88	2.48
DMTS	0.04	0.22	0.39	0.91	1.13	1.34	2.61	3.19
BMTS	0.03	0.15	0.30	0.62	0.83	0.87	1.61	2.17
PMTS(1)	0.03	0.20	0.37	0.84	0.97	1.23	2.44	3.15
PMTS(5)	0.04	0.22	0.42	0.91	1.26	1.59	2.83	3.66
MTS	0.18	0.84	2.12	3.88	5.63	7.64	10.30	14.18
DMTS	0.44	2.23	5.31	8.59	10.13	14.14	13.96	66.92
BMTS	0.21	1.08	2.65	4.58	6.70	9.63	12.44	17.06
PMTS(1)	0.26	1.12	2.74	4.57	7.58	10.31	11.26	16.41
PMTS(5)	0.25	1.17	2.94	6.36	7.80	10.01	11.90	36.51



On the one hand, observe that the number of parameters does not play any major role in the running time. The running times on PMTS with 5 parameters are very close to BMTS, i.e. PMTS with zero parameters, as can be seen in the graph. Therefore, the greatest theoretical complexity threat—the num-

Table 2. Experimental results: systems over alphabet of size 2 with branching degree 5; systems with random structure in the upper part, and systems with organic structure in the lower part; the refining system is identical to the abstract system, besides a stronger obligation

	25	50	75	100	125	150	175	200
BMTS	0.32	1.57	3.46	7.18	10.24	15.18	20.6	27.05
PMTS(1)	0.34	1.57	3.21	8.25	12.46	19.88	24.53	31.01
PMTS(5)	0.33	1.65	4.48	8.21	13.14	21.5	20.55	25.82
BMTS	0.01	0.03	0.18	0.22	0.3	0.48	0.73	1.02
PMTS(1)	0.01	0.07	0.14	0.22	0.43	0.43	0.72	0.83
PMTS(5)	0.01	0.05	0.1	0.17	0.31	0.43	0.88	1.39

ber of parameters allowing in general only for searching all exponentially many combinations—is in practice eliminated by the use of QBF solvers.

On the other hand, observe that the running time is more affected by the level of non-determinism. For branching degree 10 over a 10-letter alphabet, there are more likely to be more outgoing transitions under the same letter than in the case with branching degree 2 over a 2-letter alphabet, but still less than for branching degree 5 over a 2-letter alphabet. However, the level of non-determinism is often quite low [BKLS09], hence this dependency does not pose a serious problem in practice. Further, even this most difficult setting with a high level of non-determinism allows for fast analysis if systems with natural organic structure are considered, cf. upper and lower part of Table 2.

A more serious problem stems from our use of Java. With sizes around 200, the running times often get considerably longer, as the automatic memory management takes its toll. However, this problem should diminish in a garbage-collection-free setting.

5 Thorough Refinement Checking

While modal refinement has been defined syntactically, there is also a corresponding notion defined semantically. The semantics of a state s of a PMTS is the set of its implementations $[\![s]\!] := \{i \mid i \text{ is an implementation and } i \leq_{\mathrm{m}} s\}$.

Definition 5.1 (Thorough Refinement). *For states s_0 and t_0 of PMTS, we say that s_0* thoroughly refines *t_0, written $s_0 \leq_{\mathrm{t}} t_0$, if $[\![s_0]\!] \subseteq [\![t_0]\!]$.*

5.1 Transforming PMTS to BMTS and DMTS

The thorough refinement problem is EXPTIME-complete for MTS [BKLS12] and also for DMTS [BČK11] (for proof, see [BČK10]). First, we show how to transform PMTS to BMTS and DMTS and thus reduce our problems to the already solved one.

For a PMTS, we define a system where we can use any valuation of the parameters:

Definition 5.2. *For a PMTS $\mathcal{M} = (S, T, P, \Phi)$ with initial state s_0, we define a BMTS called* de-parameterization $\mathcal{M}^B = (\{s_0^B\} \cup S \times 2^P, T', \emptyset, \Phi')$ *with initial state s_0^B and*

- $T' = \{(s_0^B, a, (s, \nu)) \mid (s_0, a, s) \in T, \nu \subseteq P\} \cup \{((s, \nu), a, (s', \nu) \mid (s, a, s') \in T\}$,
- $\Phi'(s_0^B) = \bigoplus_{\nu \subseteq P} \Phi(s_0)[\mathbf{tt}/p \text{ for } p \in \nu, \mathbf{ff}/p \text{ for } p \notin \nu, (s, \nu)/s]$,
- $\Phi'\big((s, \nu)\big) = \Phi(s)[\mathbf{tt}/p \text{ for } p \in \nu, \mathbf{ff}/p \text{ for } p \notin \nu, (s, \nu)/s]$.

The de-parameterization is a BMTS having exactly all the implementations of the PMTS and only one (trivial) valuation.

Proposition 5.3. *Let s_0 be a PMTS state. Then $[\![s_0]\!] = [\![s_0^B]\!]$ and $s_0 \leq_{\mathrm{m}} s_0^B$.*

Proof. For any parameter valuation ν we match it with $\emptyset$ and the modal refinement is achieved in the copy with ν fixed in the second component. Clearly, any implementation of s_0^B corresponds to a particular parameter valuation and thus also to an implementation of s_0. □

Remark 5.4. The price we have to pay is a blowup exponential in $|P|$. This is, however, inevitable. Indeed, consider a PMTS $(\{s_0, s_1, s_2\}, \{(s_0, p, s_1), (s_1, p, s_2) \mid p \in P\}, P, \{s_0, s_1 \mapsto \bigwedge_{p \in P}(p, s) \Leftrightarrow p, s_2 \mapsto \mathbf{tt}\})$. Then in every equivalent BMTS we need to remember the transitions of the first step so that we can repeat exactly these in the following step. Since there are exponentially many possibilities, the result follows.

Further, similarly to Boolean formulae with states in [BDF$^+$], we can transform every BMTS to a DMTS.

Definition 5.5. *For a BMTS $\mathcal{M} = (S, T, \emptyset, \Phi)$ with initial state s_0, we define a DMTS called* de-negation $\mathcal{M}^D = (S', T', \emptyset, \Phi')$

- $S' = \{M \in \mathrm{Tran}(s) \mid s \in S\}$,
- $\Phi'(M) = \bigwedge_{(a,s') \in M} \bigvee_{M' \in \mathrm{Tran}(s')} (a, M')$,

and T' minimal such that for each $M \in S'$ and each occurrence of (a, M') in $\Phi(M)$, we have $(M, a, M') \in T'$.

However, this DMTS needs to have more initial states in order to be equivalent to the original BMTS:

Lemma 5.6. *For a state s_0 of a BMTS, $[\![s_0]\!] = \bigcup_{M \in \mathrm{Tran}(s_0)} [\![M]\!]$ (where M are taken as states of the de-negation).*

Note that both transformations are exponential. The first one in $|P|$ and the second one in the branching degree. Therefore, their composition is still only singly exponential, yielding a state space where each state has two components: a valuation of original parameters and Tran of the original state under this valuation.

Theorem 5.7. *Thorough refinement on PMTS is in 2-EXPTIME.*

Proof. Recall that thorough refinement on DMTS is in EXPTIME. Further, note that we have reduced the PMTS and BMTS thorough refinement problems to the one on DMTS with more initial states. However, this does not pose a problem. Indeed, let s_0 and t_0 be states of a BMTS. We want to check whether $s_0 \leq_\mathrm{t} t_0$. According to [BČK10] where DMTS only have one initial state, we only need to check whether for each $M \in \mathrm{Tran}(s_0)$ we have $(M, \mathrm{Tran}(t_0)) \notin \mathit{Avoid}$ (defined in [BČK10]), which can clearly still be done in exponential time. □

5.2 Direct algorithm

We now extend the approach for MTS and DMTS to the BMTS case. Before proceeding, one needs to prune all inconsistent states, i.e. those with unsatisfiable obligation. This is standard and the details can be found in [KS13].

We define a set *Avoid*, which contains pairs consisting of one process and one set of processes. A pair is contained in the relation if there exists an implementation refining the single process, but none of the other processes. This approach is very similar to [BKLS12], but the rules for generating *Avoid* are much more complex.

Definition 5.8. (Avoid) *Let $(S, T, \emptyset, \Phi)$ be a globally consistent BMTS over the action alphabet Σ. The set of* avoiding *states of the form $(s, \mathcal{T})$, where $s \in S$ and $\mathcal{T} \subseteq S$, is the smallest set Avoid such that $(s, \mathcal{T}) \in \mathit{Avoid}$ whenever $\mathcal{T} = \emptyset$ or there exists an admissible set of transitions $M \in \mathrm{Tran}(s)$ and sets $later_{a,u,f} \subseteq S$ for every $a \in \Sigma$, $u \in S$, $f \in \bigcup_{t \in \mathcal{T}} \mathrm{Tran}(t)$ such that*

$$\begin{aligned} \forall t \in \mathcal{T} : \forall N_t \in \mathrm{Tran}(t) :\ & \exists a \in \Sigma : \\ & \exists t_a \in N_t(a) : \forall s_a \in M(a) : \forall f \in \bigcup_{t \in \mathcal{T}} \mathrm{Tran}(t) : t_a \in \mathit{later}_{a,s_a,f} \\ \vee\ & \exists s_a \in M(a) : \forall t_a \in N_t(a) : t_a \in \mathit{later}_{a,s_a,N_t} \end{aligned}$$

and

$$\forall f \in \bigcup_{t \in \mathcal{T}} \mathrm{Tran}(t) : \forall (a, s_a) \in M : (s_a, \mathit{later}_{a,s_a,f}) \in \mathit{Avoid}$$

hold.

Lemma 5.9. *Given processes $s, t_1, t_2 \dots t_n$ of some finite, global-consistent BMTS, there exists an implementation I such that $I \leq_\mathrm{m} s$ and $I \not\leq_\mathrm{m} t_i$ for all $i \in [1, n]$ if $(s, \{t_1, t_2 \dots t_n\}) \in \mathit{Avoid}$.*

Theorem 5.10. *Thorough refinement checking on BMTS is in NEXPTIME.*

Proof. For deciding $s \leq_\mathrm{t} t$ the *Avoid* relation has to be computed, whose size grows exponentially with the size of the underlying system. Moreover, in each step of adding a new element is added to *Avoid*, the sets $\mathit{later}_{a,s,f}$ need to be guessed. □

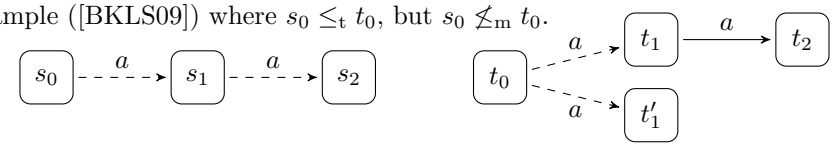
6 Thorough vs. Modal Refinement

In this section, we discuss the relationship of the two refinements. This missing proofs can be found in [KS13]. Firstly, the modal refinement is a sound approximation to the thorough refinement.

Proposition 6.1. *Let s_0 and t_0 be states of PMTS. If $s_0 \leq_{\mathrm{m}} t_0$ then $s_0 \leq_{\mathrm{t}} t_0$.*

Proof. For any $i \in [\![s_0]\!]$, we have $i \leq_{\mathrm{m}} s_0$ and due to transitivity of $\leq_{\mathrm{m}}$, $i \leq_{\mathrm{m}} s_0 \leq_{\mathrm{m}} t_0$ implies $i \leq_{\mathrm{m}} t_0$, hence $i \in [\![t_0]\!]$. □

The converse does not hold even for MTS as shown in the following classical example ([BKLS09]) where $s_0 \leq_{\mathrm{t}} t_0$, but $s_0 \not\leq_{\mathrm{m}} t_0$.



However, provided the refined MTS is deterministic, the approximation is also complete [BKLS09]. This holds also for BMTS and is very useful as deterministic system often appear in practice [BKLS09] and checking modal refinement is computationally easier than the thorough refinement. Formally, a PMTS (S, T, P, Φ) is called deterministic if for every $(s, a, t), (s, a, t') \in T$ we have $t = t'$.

Proposition 6.2. *Let s_0 be a PMTS state and t_0 a deterministic BMTS state. If $s_0 \leq_{\mathrm{t}} t_0$ then $s_0 \leq_{\mathrm{m}} t_0$.*

However, the completeness fails if the refined system is deterministic but with parameters:

Example 6.3. Consider a BMTS $(\{s_0, s_1\}, \{s_0, a, s_1\}, \emptyset, \{s_0 \mapsto \mathbf{tt}, s_1 \mapsto \mathbf{tt}\})$ and a deterministic PMTS $(\{t_0, t_1\}, \{(t_0, a, t_1)\}, \{p\}, \{t_0 \mapsto a \Leftrightarrow p, t \mapsto \mathbf{tt}\})$ below. Obviously $[\![s_0]\!] = [\![t_0]\!]$ contains the implementations with no transitions or one step a-transitions. Although $s_0 \leq_{\mathrm{t}} t_0$, we do not have $s_0 \leq_{\mathrm{m}} t_0$ as we cannot match with any valuation of p.

$\Phi(t_0) = \ a \Leftrightarrow p$

Corollary 6.4. *There is a state s_0 of a PMTS and a state t_0 of a deterministic PMTS such that $s_0 \leq_{\mathrm{t}} t_0$ but $s_0 \not\leq_{\mathrm{m}} t_0$.*

In the previous example, we lacked the option to match a system with different parameter valuations at once. However, the de-parameterization introduced earlier is non-deterministic even if the original system was deterministic. Hence the modal refinement is not guaranteed to coincide with the thorough refinement. In [BKLS09], we defined the notion of deterministic hull, the best deterministic overapproximation of a system. The construction on may transitions was the standard powerset construction and a must transition was created if all states of a macrostate had one. Here we extend this notion to PMTS, which allows to over- and under-approximate the thorough refinement by the modal refinement.

Definition 6.5. *For a PMTS $\mathcal{M} = (S, T, P, \Phi)$ with initial state s_0, we define a PMTS called* deterministic hull *$\mathcal{D}(\mathcal{M}) = (2^S \setminus \emptyset, T', P, \Phi')$ with initial state $\mathcal{D}(s_0) := \{s_0\}$ and*

- *$T' = \{(X, a, X_a)\}$ where X_a denotes all a-successors of elements of X, i.e. $X_a = \{s' \mid \exists s \in X : (s, a, s') \in T)\}$,*
- *$\Phi'(X)$ is such that $\mathrm{Tran}(X) = \bigcup_{s \in X} \mathrm{Tran}(s)[(a, X_a)/(a, s)$ for every $a, s]$.*

Proposition 6.6. *For a PMTS state s_0, $\mathcal{D}(s_0)$ is deterministic and $s_0 \leq_{\mathrm{m}} \mathcal{D}(s_0)$.*

We now show the minimality of the deterministic hull.

Proposition 6.7. *Let s_0 be a PMTS state. Then*

- *for every deterministic PMTS state t_0, if $s_0 \leq_{\mathrm{m}} t_0$ then $\mathcal{D}(s_0) \leq_{\mathrm{m}} t_0$;*
- *for every deterministic BMTS state t_0, if $s_0 \leq_{\mathrm{t}} t_0$ then $\mathcal{D}(s_0) \leq_{\mathrm{m}} t_0$.*

The next transformation allows for removing the parameters without introducing non-determinism.

Definition 6.8. *For a PMTS $\mathcal{M} = (S, T, P, \Phi)$ with initial state s_0, we define a BMTS called* parameter-free hull *$\mathcal{P}(\mathcal{M}) = (S, T, \emptyset, \Phi')$ with initial state $\mathcal{P}(s_0) := s_0$ and*

$$\Phi'(s) = \bigvee_{\nu \subseteq P} \Phi(s)[\mathbf{tt}/p \text{ for } p \in \nu, \mathbf{ff}/p \text{ for } p \notin \nu]$$

Lemma 6.9. *For a PMTS state s_0, $s_0 \leq_{\mathrm{m}} s_0^B \leq_{\mathrm{m}} \mathcal{P}(s_0)$.*

The parameter-free deterministic hull now plays the rôle of the deterministic hull for MTS.

Corollary 6.10. *For PMTS states s_0 and t_0, if $s_0 \leq_{\mathrm{t}} t_0$ then $s_0 \leq_{\mathrm{m}} \mathcal{P}(\mathcal{D}(t_0))$.*

Proof. Since $s_0 \leq_{\mathrm{t}} t_0$, we also have $s_0 \leq_{\mathrm{t}} \mathcal{D}(t_0)$ by Propositions 6.6 and 6.1. Therefore, $s_0 \leq_{\mathrm{t}} \mathcal{P}(\mathcal{D}(t_0))$ by Proposition 6.9 and thus $s_0 \leq_{\mathrm{m}} \mathcal{P}(\mathcal{D}(t_0))$ by Proposition 6.2. □

7 Conclusions

We have investigated both modal and thorough refinement on boolean and parametric extension of modal transition systems. Apart from results summarized in the table below, we have shown a practical way to compute modal refinement and use it for approximating thorough refinement. Closing the complexity gap for thorough refinement, i.e. obtaining matching lower bounds or improving our algorithm remains as an open question.

	MTS	BMTS	PMTS
$\leq_t \in$	EXPTIME	NEXPTIME	2-EXPTIME
$s \leq_t t$, t deterministic	$\leq_m = \leq_t$	$\leq_m = \leq_t$	$\leq_m \neq \leq_t$

References

[AFdFE+11] L. Aceto, I. Fábregas, D. de Frutos-Escrig, A. Ingólfsdóttir, and M. Palomino. Graphical representation of covariant-contravariant modal formulae. In *EXPRESS*, pages 1–15, 2011.

[AHKV98] R. Alur, T. A. Henzinger, O. Kupferman, and M. Y. Vardi. Alternating refinement relations. In *CONCUR*, pages 163–178, 1998.

[AHL+08] A. Antonik, M. Huth, K. G. Larsen, U. Nyman, and A. Wasowski. 20 years of modal and mixed specifications. *Bulletin of the EATCS no. 95*, pages 94–129, 2008.

[BČK10] N. Beneš, I. Černá, and J. Křetínský. Disjunctive modal transition systems and generalized LTL model checking. Technical report FIMU-RS-2010-12, Faculty of Informatics, Masaryk University, Brno, 2010.

[BČK11] N. Beneš, I. Černá, and J. Křetínský. Modal transition systems: Composition and LTL model checking. In *ATVA*, pages 228–242, 2011.

[BDF+] N. Beneš, B. Delahaye, U. Fahrenberg, J. Křetínský, and A. Legay. Hennessy-milner logic with maximal fixed points as a specification theory. Submitted.

[BKL+11] N. Beneš, J. Křetínský, K. G. Larsen, M. H. Møller, and J. Srba. Parametric modal transition systems. In *ATVA*, pages 275–289, 2011.

[BKL+12] N. Beneš, J. Křetínský, K. G. Larsen, M. H. Møller, and J. Srba. Dual-priced modal transition systems with time durations. In *LPAR*, pages 122–137, 2012.

[BKLS09] N. Beneš, J. Křetínský, K. G. Larsen, and J. Srba. On determinism in modal transition systems. *Theor. Comput. Sci.*, 410(41):4026–4043, 2009.

[BKLS12] N. Beneš, J. Křetínský, K. G. Larsen, and J. Srba. Exptime-completeness of thorough refinement on modal transition systems. *Inf. Comput.*, 218:54–68, 2012.

[BL90] G. Boudol and K. G. Larsen. Graphical versus logical specifications. In *CAAP*, pages 57–71, 1990.

[BLS95] A. Børjesson, K. G. Larsen, and A. Skou. Generality in design and compositional verification using TAV. *Formal Methods in System Design*, 6(3):239–258, 1995.

[BML11] S. S. Bauer, P. Mayer, and A. Legay. MIO workbench: A tool for compositional design with modal input/output interfaces. In *ATVA*, pages 418–421, 2011.

[Bru97] G. Bruns. An industrial application of modal process logic. *Sci. Comput. Program.*, 29(1-2):3–22, 1997.

[CGLT09] A. Campetelli, A. Gruler, M. Leucker, and D. Thoma. *Don't Know* for multi-valued systems. In *ATVA*, pages 289–305, 2009.

[dAH01] L. de Alfaro and T. A. Henzinger. Interface automata. In *ESEC / SIGSOFT FSE*, pages 109–120, 2001.

[DFFU07] N. D'Ippolito, D. Fischbein, H. Foster, and S. Uchitel. MTSA: Eclipse support for modal transition systems construction, analysis and elaboration. In *ETX*, pages 6–10, 2007.

[DGG97] D. Dams, R. Gerth, and O. Grumberg. Abstract interpretation of reactive systems. *ACM Trans. Program. Lang. Syst.*, 19(2):253–291, 1997.

[DN04] D. Dams and K. S. Namjoshi. The existence of finite abstractions for branching time model checking. In *LICS*, pages 335–344, 2004.

[FS08] H. Fecher and H. Schmidt. Comparing disjunctive modal transition systems with an one-selecting variant. *J. Log. Algebr. Program.*, 77(1-2):20–39, 2008.

[GHJ01] P. Godefroid, M. Huth, and R. Jagadeesan. Abstraction-based model checking using modal transition systems. In *CONCUR*, pages 426–440, 2001.

[GNRT10] P. Godefroid, A. V. Nori, S. K. Rajamani, and S. Tetali. Compositional may-must program analysis: unleashing the power of alternation. In *POPL*, pages 43–56, 2010.

[HJS01] M. Huth, R. Jagadeesan, and D. A. Schmidt. Modal transition systems: A foundation for three-valued program analysis. In *ESOP*, pages 155–169, 2001.

[JP01] B. Jacobs and E. Poll. A logic for the java modeling language JML. In *FASE*, pages 284–299, 2001.

[KS13] J. Křetínský and S. Sickert. On refinements of boolean and parametric modal transition systems. *CoRR*, abs/1304.5278, 2013.

[LNW07] K. G. Larsen, U. Nyman, and A. Wasowski. Modal I/O automata for interface and product line theories. In *ESOP*, pages 64–79, 2007.

[LT88] K. G. Larsen and B. Thomsen. A modal process logic. In *LICS*, pages 203–210, 1988.

[LW94] B. Liskov and J. M. Wing. A behavioral notion of subtyping. *ACM Trans. Program. Lang. Syst.*, 16(6):1811–1841, 1994.

[LX90] K. G. Larsen and L. Xinxin. Equation solving using modal transition systems. In *LICS*, pages 108–117, 1990.

[Lyn88] N. Lynch. I/O automata: A model for discrete event systems. In *22nd Annual Conference on Information Sciences and Systems*, pages 29–38. Princeton University, 1988.

[Nam03] K. S. Namjoshi. Abstraction for branching time properties. In *CAV*, pages 288–300, 2003.

[Nym08] U. Nyman. *Modal Transition Systems as the Basis for Interface Theories and Product Lines*. PhD thesis, Aalborg Universitet, 2008.

[Rac07] J.-B. Raclet. *Quotient de spécifications pour la réutilisation de composants*. PhD thesis, Université de Rennes I, december 2007. (In French).

[RBB+09] J.-B. Raclet, E. Badouel, A. Benveniste, B. Caillaud, and R. Passerone. Why are modalities good for interface theories? In *ACSD*. IEEE Computer Society Press, 2009.

[RBB+11] J.-B. Raclet, E. Badouel, A. Benveniste, B.Caillaud, A. Legay, and R. Passerone. A modal interface theory for component-based design. *Fundamenta Informaticae*, 108(1-2):119–149, 2011.

[UC04] S. Uchitel and M. Chechik. Merging partial behavioural models. In *SIGSOFT FSE*, pages 43–52, 2004.

Paper G:

Hennessy-Milner logic with greatest fixed points as a complete behavioural specification theory

Nikola Beneš, Benoît Delahaye, Uli Fahrenberg, Jan Křetínský, and Axel Legay

This paper has been published in Pedro R. D'Argenio and Hernán Melgratti (eds.): Proceedings of Concurrency Theory, 24th International Conference, CONCUR 2013, Buenos Aires, Argentina, August 27-30, 2013. Proceedings, volume 8052 of Lecture Notes in Computer Science, pages 76–90. Springer, 2013. [BDF+13]

Some of the results have been published in a Bachelor's thesis [Sic12] supervised by the author.

Summary

We unify the behavioural and logical approach to specification and verification in the setting of MTS extensions and Hennessy-Milner logic with recursion. To this end, we provide translations between BMTS (or DMTS with more initial states) and ν-calculus (or Hennessy-Milner logic with greatest fix-points). Further, we also show this formalism is a complete specification theory equipped with all necessary operations, which, moreover, exhibit nice algebraic properties. The technical core is the construction of the quotient for (generally non-deterministic) BMTS. This is the first quotient for non-deterministic modal specifications. Moreover, we also give a construction of the quotient for MTS, which is exponentially more efficient. This is a vast improvement on the previous construction for deterministic MTS defined using simple syntactic rules.

Author's contribution: 30 %

- participating in the discussions,
- contributing, in particular, to the translations and the construction of the quotient,
- writing parts of the paper.

Hennessy-Milner Logic with Greatest Fixed Points as a Complete Behavioural Specification Theory

Nikola Beneš[1⋆], Benoît Delahaye[2], Uli Fahrenberg[2], Jan Křetínský[1,3⋆⋆], and Axel Legay[2]

[1] Masaryk University, Brno, Czech Republic
[2] Irisa / INRIA Rennes, France
[3] Technische Universität München, Germany

Abstract. There are two fundamentally different approaches to specifying and verifying properties of systems. The *logical* approach makes use of specifications given as formulae of temporal or modal logics and relies on efficient model checking algorithms; the *behavioural* approach exploits various equivalence or refinement checking methods, provided the specifications are given in the same formalism as implementations. In this paper we provide translations between the logical formalism of Hennessy-Milner logic with greatest fixed points and the behavioural formalism of disjunctive modal transition systems. We also introduce a new operation of quotient for the above equivalent formalisms, which is adjoint to structural composition and allows synthesis of missing specifications from partial implementations. This is a substantial generalisation of the quotient for deterministic modal transition systems defined in earlier papers.

1 Introduction

There are two fundamentally different approaches to specifying and verifying properties of systems. Firstly, the *logical* approach makes use of specifications given as formulae of temporal or modal logics and relies on efficient model checking algorithms. Secondly, the *behavioural* approach exploits various equivalence or refinement checking methods, provided the specifications are given in the same formalism as implementations.

In this paper, we discuss different formalisms and their relationship. As an example, let us consider labelled transition systems and the property that *"at all time points after executing* request*, no* idle *nor further requests but only* work *is allowed until* grant *is executed"*. The property can be written in *e.g.* CTL [14] as

$$\mathrm{AG}(\mathsf{request} \Rightarrow \mathrm{AX}(\mathsf{work}\ \mathrm{AW}\ \mathsf{grant}))$$

⋆ The author has been supported by the Czech Science Foundation grant No. GAP202/11/0312.
⋆⋆ The author is partially supported by the Czech Science Foundation, project No. P202/10/1469.

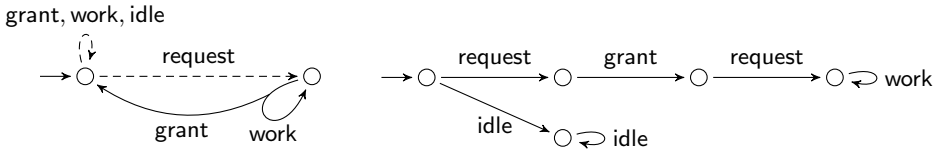


Fig. 1. DMTS specification corresponding to AG(request ⇒ AX(work AW grant)), and its implementation

or as a recursive system of equations in Hennessy-Milner logic [29] as

$$X = [\mathsf{grant}, \mathsf{idle}, \mathsf{work}]X \wedge [\mathsf{request}]Y$$
$$Y = (\langle \mathsf{work} \rangle Y \vee \langle \mathsf{grant} \rangle X) \wedge [\mathsf{idle}, \mathsf{request}]\mathbf{ff}$$

where the solution is given by the greatest fixed point.

As formulae of modal logics can be difficult to read, some people prefer automata-based behavioural specifications to logical ones. One such behavioural specification formalism is the one of disjunctive modal transition systems (DMTS) [26]. Fig. 1 (left) displays a specification of our example property as a DMTS. Here the dashed arrows indicate that the transitions *may or may not* be present, while branching of the solid arrow indicates that at least one of the branches *must* be present. An example of a labelled transition system that *satisfies* our logical specifications and *implements* the behavioural one is also given in Fig. 1.

The alternative between logical and behavioural specifications is not only a question of preference. Logical specification formalisms put a powerful logical language at the disposal of the user, and the logical approach to model checking [14, 34] has seen a lot of success and tool implementations. Automata-based specifications [12, 27], on the other hand, have a focus on *compositional* and *incremental* design in which logical specifications are somewhat lacking, with the trade-off of generally being less expressive than logics.

To be more precise, automata-based specifications are, by design, compositional in the sense that they support structural *composition* of specifications and, in most cases, its adjoint, *quotient*. This is useful, even necessary, in practical verification, as it means that (1) it is possible to infer properties of a system from the specifications of its components, and (2) the problem of correctness for a system can be decomposed into verification problems for its components. We refer to [28] for a detailed account on composition and decomposition.

It is thus desirable to be able to translate specifications from the logical realm into behavioural formalisms, and *vice versa* from behavioural formalisms to logic-based specifications. This is, then, the first contribution of this paper: we show that Hennessy-Milner logic with greatest fixed points (νHML) and DMTS (with several initial states) are equally expressive, and we provide translations forth and back. For doing this, we introduce an auxiliary intermediate formalism NAA (a nondeterministic extension of acceptance automata [22, 35]) which is equivalent in expressiveness to both νHML and DMTS.

We also discuss other desirable features of specification formalisms, namely structural composition and quotient. As an example, consider a specification S

of the final system to be constructed and T either an already implemented component or a specification of a service to be used. The task is to construct the most general specification of the rest of the system to be implemented, in such a way that when composed with any implementation of T, it conforms with the specification S. This specification is exactly the quotient S/T.

Contribution Firstly, we show that the formalisms of νHML, NAA and DMTS have the same expressive power, and provide the respective translations. As a result, the established connection allows for a graphical representation of νHML as DMTS. This extends the graphical representability of HML without fixed points as modal transition systems [10, 27]. In some sense this is optimal, as due to the alternation of least and greatest fixed points, there seems to be no hope that the whole μ-calculus could be drawn in a similarly simple way.

Secondly, we show that there are natural operations of conjunction and disjunction for NAA which mimic the ones of νHML. As we work with multiple initial states, disjunction is readily defined, and conjunction extends the one for DMTS [6]. Thirdly, we introduce structural composition on NAA. For simplicity we assume CSP-style synchronisation of labels, but the construction can easily be generalised to other types of label synchronisation.

Finally, we provide a solution to the open problem of the general quotient. We extend the quotient constructions for deterministic modal transition systems (MTS) and acceptance automata [35] to define the quotient for the full class of (possibly nondeterministic) NAA. We also provide a more efficient procedure for (possibly nondeterministic) MTS. These constructions are the technically most demanding parts of the paper.

With the operations of structural composition and quotient, NAA, and hence also DMTS and νHML, are fully compositional behavioural specification theories and form a *commutative residuated lattice* [21, 39] up to equivalence. This makes a rich algebraic theory available for compositional reasoning about specifications. Most of the constructions we introduce are implemented in a prototype tool [8]. Due to space constraints, some of the proofs had to be omitted from the paper and can be found in [3].

Related work Hennessy-Milner logic with recursion [29] is a popular logical specification formalism which has the same expressive power as μ-calculus [25]. It is obtained from Hennessy-Milner logic (HML) [23] by introducing variables and greatest and least fixed points. Hennessy-Milner logic with *greatest* fixed points (νHML) is equivalent to ν-calculus, *i.e.* μ-calculus with greatest fixed points only.

DMTS have been proposed as solutions to algebraic process equations in [26] and further investigated also as a specification formalism [6, 28]. The DMTS formalism is a member of the modal transition systems (MTS) family and as such has also received attention recently. The MTS formalisms have proven to be useful in practice. Industrial applications started as early as [11] where MTS have been used for an air-traffic system at Heathrow airport. Besides, MTS classes are advocated as an appropriate base for interface theories in [36] and for product

line theories in [31]. Further, an MTS based software engineering methodology for design via merging partial descriptions of behaviour has been established in [38] and methods for supervisory control of MTS shown in [15]. Tool support is quite extensive, *e.g.* [2, 6, 9, 16].

Over the years, many extensions of MTS have been proposed. While MTS can only specify whether or not a particular transition is required, some extensions equip MTS with more general abilities to describe what *combinations* of transitions are possible. These include DMTS [26], 1-MTS [17] allowing to express exclusive disjunction, OTS [4] capable of expressing positive Boolean combinations, and Boolean MTS [5] covering all Boolean combinations. The last one is closely related to our NAA, the acceptance automata of [22, 35], as well as hybrid modal logic [7, 33].

Larsen has shown in [27] that any finite MTS is equivalent to a HML formula (without recursion or fixed points), the *characteristic formula* of the given MTS. Conversely, Boudol and Larsen show in [10] that any consistent and *prime* HML formula is equivalent to a MTS. Here we extend these results to νHML formulae, and show that any such formula is equivalent to a DMTS, solving a problem left open in [26]. Hence νHML supports full compositionality and decomposition in the sense of [28]. This finishes some of the work started in [10, 27, 28].

Quotients are related to *decomposition* of processes and properties, an issue which has received considerable attention through the years. In [26], a solution to bisimulation $C(X) \sim P$ for a given process P and context C is provided (as a DMTS). This solves the quotienting problem P/C for the special case where both P and C are processes. This is extended in [30] to the setting where the context C can have several holes and $C(X_1, \ldots, X_n)$ must satisfy a property Q of νHML. However, C remains to be a process context, not a specification context. Our *specification* context allows for arbitrary specifications, representing infinite sets of processes and process equations. Another extension uses infinite conjunctions [19], but similarly to the other approaches, generates partial specifications from an overall specification and a given set of processes. This is subsumed by a general quotient.

Quotient operators, or *guarantee* or *multiplicative implication* as they are called there, are also well-known from various logical formalisms. Indeed, the algebraic properties of our parallel composition $\|$ and quotient / resemble closely those of multiplicative conjunction & and implication $\multimap$ in *linear logic* [20], and of spatial conjunction and implication in *spatial logic* [13] and *separation logic* [32, 37]. For these and other logics, proof systems have been developed which allow one to reason about expressions containing these operators.

In spatial and separation logic, & and $\multimap$ (or the operators corresponding to these linear-logic symbols) are first-class operators on par with the other logical operators, and their semantics are defined as certain sets of processes. In contrast, for NAA and hence, via the translations, also for νHML, $\|$ and / are *derived* operators, and we provide constructions to reduce any expression which contains them, to one which does not. This is important from the perspective of reuse of components and useful in industrial applications.

2 Specification Formalisms

In this section, we define the specification formalisms νHML, DMTS and NAA and show that they are equivalent.

For the rest of the paper, we fix a finite alphabet Σ. In each of the formalisms, the semantics of a specification is a set of implementations, in our case always a set of *labelled transition systems* (LTS) over Σ, *i.e.* structures $(S, s^0, \longrightarrow)$ consisting of a set S of *states*, an initial state $s^0 \in S$, and a *transition relation* $\longrightarrow \subseteq S \times \Sigma \times S$. We assume that the transition relation of LTS is always *image-finite*, i.e. that for every $a \in \Sigma$ and $s \in S$ the set $\{s' \in S \mid s \xrightarrow{a} s'\}$ is finite.

2.1 Hennessy-Milner Logic with Greatest Fixed Points

We recap the syntax and semantics of HML with variables developed in [29]. A *HML formula* ϕ over a set X of variables is given by the abstract syntax $\phi ::= \mathbf{tt} \mid \mathbf{ff} \mid x \mid \phi \wedge \phi \mid \phi \vee \phi \mid \langle a \rangle \phi \mid [a]\phi$, where x ranges over X and a over Σ. The set of such formulae is denoted $\mathcal{H}(X)$. Notice that instead of including fixed point operators in the logic, we choose to use declarations with a greatest fixed point semantics, as explained below.

A *declaration* is a mapping $\Delta : X \to \mathcal{H}(X)$. We shall give a greatest fixed point semantics to declarations. Let $(S, s^0, \longrightarrow)$ be an LTS, then an *assignment* is a mapping $\sigma : X \to 2^S$. The set of assignments forms a complete lattice with $\sigma_1 \sqsubseteq \sigma_2$ iff $\sigma_1(x) \subseteq \sigma_2(x)$ for all $x \in X$ and $\big(\bigsqcup_{i \in I} \sigma_i\big)(x) = \bigcup_{i \in I} \sigma_i(x)$.

The semantics of a formula is a subset of S, given relative to an assignment σ, defined as follows: $(\!|\mathbf{tt}|\!)\sigma = S$, $(\!|\mathbf{ff}|\!)\sigma = \emptyset$, $(\!|x|\!)\sigma = \sigma(x)$, $(\!|\phi \wedge \psi|\!)\sigma = (\!|\phi|\!)\sigma \cap (\!|\psi|\!)\sigma$, $(\!|\phi \vee \psi|\!)\sigma = (\!|\phi|\!)\sigma \cup (\!|\psi|\!)\sigma$, $(\!|\langle a \rangle \phi|\!)\sigma = \{s \in S \mid \exists s \xrightarrow{a} s' : s' \in (\!|\phi|\!)\sigma\}$, and $(\!|[a]\phi|\!)\sigma = \{s \in S \mid \forall s \xrightarrow{a} s' : s' \in (\!|\phi|\!)\sigma\}$. The semantics of a declaration Δ is then the assignment defined by $(\!|\Delta|\!) = \bigsqcup\{\sigma : X \to 2^S \mid \forall x \in X : \sigma(x) \subseteq (\!|\Delta(x)|\!)\sigma\}$: the greatest (pre)fixed point of Δ.

An *initialised* HML declaration, or *νHML formula*, is a structure (X, X^0, Δ), with $X^0 \subseteq X$ finite sets of variables and $\Delta : X \to \mathcal{H}(X)$ a declaration. We say that an LTS $(S, s^0, \longrightarrow)$ *implements* (or models) the formula, and write $S \models \Delta$, if it holds that there is $x^0 \in X^0$ such that $s^0 \in (\!|\Delta|\!)(x^0)$. We write $[\![\Delta]\!]$ for the set of implementations (models) of a νHML formula Δ.

2.2 Disjunctive Modal Transition Systems

A DMTS is essentially a labelled transition system (LTS) with two types of transitions, *may* transitions which indicate that implementations are permitted to implement the specified behaviour, and *must* transitions which proclaim that any implementation is required to implement the specified behaviour. Additionally, *must* transitions may be *disjunctive*, in the sense that they can require that *at least one* out of a number of specified behaviours must be implemented. We now recall the syntax and semantics of DMTS as introduced in [26]. We modify

the syntax slightly to permit multiple initial states and, in the spirit of later work [6,18], ensure that all required behaviour is also allowed:

A *disjunctive modal transition system* (DMTS) over the alphabet Σ is a structure $(S, S^0, \dashrightarrow, \longrightarrow)$ consisting of a set of *states* S, a finite subset $S^0 \subseteq S$ of *initial states*, a *may*-transition relation $\dashrightarrow \subseteq S \times \Sigma \times S$, and a *disjunctive must*-transition relation $\longrightarrow \subseteq S \times 2^{\Sigma \times S}$. It is assumed that for all $(s, N) \in \longrightarrow$ and all $(a, t) \in N$, $(s, a, t) \in \dashrightarrow$. We usually write $s \overset{a}{\dashrightarrow} t$ instead of $(s, a, t) \in \dashrightarrow$ and $s \longrightarrow N$ instead of $(s, N) \in \longrightarrow$. We also assume that the may transition relation is image-finite. Note that the two assumptions imply that $\longrightarrow \subseteq S \times 2^{\Sigma \times S}_{\mathrm{Fin}}$ where 2^X_{Fin} denotes the set of all finite subsets of X.

A DMTS $(S, S^0, \dashrightarrow, \longrightarrow)$ is an *implementation* if $S^0 = \{s^0\}$ is a singleton and $\longrightarrow = \{(s, \{(a, t)\} \mid s \overset{a}{\dashrightarrow} t\}$, hence if N is a singleton for each $s \longrightarrow N$ and there are no superfluous may-transitions. Thus DMTS implementations are precisely LTS.

We proceed to define the semantics of DMTS. First, a relation $R \subseteq S_1 \times S_2$ is a *modal refinement* between DMTS $(S_1, S^0_1, \dashrightarrow_1, \longrightarrow_1)$ and $(S_2, S^0_2, \dashrightarrow_2, \longrightarrow_2)$ if it holds for all $(s_1, s_2) \in R$ that

- for all $s_1 \overset{a}{\dashrightarrow} t_1$ there is $s_2 \overset{a}{\dashrightarrow} t_2$ for some $t_2 \in S_2$ with $(t_1, t_2) \in R$, and
- for all $s_2 \longrightarrow N_2$ there is $s_1 \longrightarrow N_1$ such that for each $(a, t_1) \in N_1$ there is $(a, t_2) \in N_2$ with $(t_1, t_2) \in R$.

Such a modal refinement is *initialised* if it is the case that, for each $s^0_1 \in S^0_1$, there is $s^0_2 \in S^0_2$ for which $(s^0_1, s^0_2) \in R$. In that case, we say that S_1 refines S_2 and write $S_1 \leq_{\mathrm{m}} S_2$. We write $S_1 \equiv_{\mathrm{m}} S_2$ if $S_1 \leq_{\mathrm{m}} S_2$ and $S_2 \leq_{\mathrm{m}} S_1$.

We say that an LTS I *implements* a DMTS S if $I \leq_{\mathrm{m}} S$ and write $[\![S]\!]$ for the set of implementations of S. Notice that the notions of implementation and modal refinement agree, capturing the essence of DMTS as a *specification theory*: A DMTS may be *gradually* refined, until an LTS, in which all behaviour is fully specified, is obtained.

For DMTS S_1, S_2 we say that S_1 *thoroughly* refines S_2, and write $S_1 \leq_{\mathrm{t}} S_2$, if $[\![S_1]\!] \subseteq [\![S_2]\!]$. We write $S_1 \equiv_{\mathrm{t}} S_2$ if $S_1 \leq_{\mathrm{t}} S_2$ and $S_2 \leq_{\mathrm{t}} S_1$. By transitivity, $S_1 \leq_{\mathrm{m}} S_2$ implies $S_1 \leq_{\mathrm{t}} S_2$.

Example 1. Figs. 2 and 3 show examples of important basic properties expressed both as νHML formulae, NAA (see below) and DMTS. For DMTS, may transitions are drawn as dashed arrows and disjunctive must transitions as branching arrows. States with a short incoming arrow are initial (the DMTS in Fig. 3 has *two* initial states).

$$X = \langle a \rangle \mathrm{tt} \wedge [a]X \wedge [b]X$$

$(\{s_0\}, \{s_0\}, \mathrm{Tran})$
$\mathrm{Tran}(s_0) = \{\{(a, s_0)\}, \{(a, s_0), (b, s_0)\}\}$

Fig. 2. νHML formula, NAA and DMTS for the invariance property "*there is always an 'a' transition available*", with $\Sigma = \{a, b\}$

$$X = \langle b\rangle \mathtt{tt} \vee \big(\langle a\rangle \mathtt{tt} \wedge [a]X \wedge [b]X \wedge [c]X\big)$$

$(\{s_0, s_1\}, \{s_0\}, \mathrm{Tran})$
$\mathrm{Tran}(s_0) = \big\{\{(b, s_1)\}, \{(b, s_1), (a, s_1)\}, \{(b, s_1), (c, s_1)\},$
$\{(b, s_1), (a, s_1), (c, s_1))\}, \{(a, s_0)\}, \{(a, s_0), (c, s_0)\}\big\}$
$\mathrm{Tran}(s_1) = 2^{\{s_1\}\times\{a,b,c\}}$

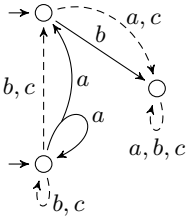
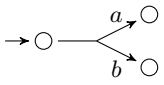


Fig. 3. νHML formula, NAA and DMTS for the ("weak until") property "*there is always an 'a' transition available, until a 'b' transition becomes enabled*", with $\Sigma = \{a, b, c\}$

Modal Transition Systems An interesting subclass of DMTS are *modal transition systems* (MTS) [27]. A DMTS $(S, S^0, \dashrightarrow, \longrightarrow)$ is said to be a MTS if (1) $S^0 = \{s^0\}$ is a singleton, (2) for every $s \longrightarrow N$, the set N is a singleton. Hence, for each transition, we specify whether it must, may, or must not be present; no disjunctions can be expressed. It is easy to see that MTS are less expressive than DMTS, *i.e.* there are DMTS S for which no MTS S' exists so that $[\![S]\!] = [\![S']\!]$. One example is provided on the right. Here any implementation must have an a or a b transition from the initial state, but then any MTS which permits all such implementations will also allow implementations without any transition from the initial state.



2.3 NAA

We now define NAA, the nondeterministic extension to the formalism of acceptance automata [35]. We shall use this formalism to bridge the gap between νHML and DMTS. A *nondeterministic acceptance automaton* over the alphabet Σ is a structure (S, S^0, Tran) where S and S^0 are the states and initial states as previously, and $\mathrm{Tran} : S \to 2^{2^{\Sigma\times S}_{\mathrm{Fin}}}$ assigns admissible transition sets.

A NAA (S, S^0, Tran) is an *implementation* if $S^0 = \{s^0\}$ is a singleton and $\mathrm{Tran}(s) = \{M\}$ is a singleton for every $s \in S$; clearly, NAA implementations are precisely LTS. We also define the *inconsistent NAA* to be $\bot = (\emptyset, \emptyset, \emptyset)$ and the *universal NAA* by $\top = (\{s\}, \{s\}, 2^{2^{\Sigma\times\{s\}}})$.

A relation $R \subseteq S_1 \times S_2$ is a *modal refinement* between NAA $(S_1, S_1^0, \mathrm{Tran}_1)$, $(S_2, S_2^0, \mathrm{Tran}_2)$ if it holds for all $(s_1, s_2) \in R$ and all $M_1 \in \mathrm{Tran}_1(s_1)$ that there exists $M_2 \in \mathrm{Tran}_2(s_2)$ such that

- $\forall (a, t_1) \in M_1 : \exists (a, t_2) \in M_2 : (t_1, t_2) \in R$,
- $\forall (a, t_2) \in M_2 : \exists (a, t_1) \in M_1 : (t_1, t_2) \in R$.

We define and use the notions of initialised modal refinement, $\leq_{\mathrm{m}}$, $\equiv_{\mathrm{m}}$, implementation, $\leq_{\mathrm{t}}$, and $\equiv_{\mathrm{t}}$ the same way as for DMTS.

Proposition 2. *The class of NAA is preordered by modal refinement $\leq_{\mathrm{m}}$, with bottom element $\bot$ and top element $\top$.*

Note that as implementations of all our three formalisms νHML, DMTS and NAA are LTS, it makes sense to use thorough refinement $\leq_{\mathrm{t}}$ and equivalence $\equiv_{\mathrm{t}}$ *across* formalisms, so that we *e.g.* can write $S \leq_{\mathrm{t}} \Delta$ for a NAA S and a νHML formula Δ.

2.4 Equivalences

We proceed to show that νHML, DMTS and NAA are equally expressive:

Theorem 3. *For any set $\mathcal{S}$ of LTS, the following are equivalent:*

1. *There exists a νHML formula Δ with $[\![\Delta]\!] = \mathcal{S}$.*
2. *There exists a finite NAA S with $[\![S]\!] = \mathcal{S}$.*
3. *There exists a finite DMTS S with $[\![S]\!] = \mathcal{S}$.*

Furthermore, the latter two statements are equivalent even if we drop the finiteness constraints.

Note that we could drop the finiteness assumption about the set of variables of νHML formulae, while retaining the fact that $\Delta(x)$ is a finite HML formula. The result of Theorem 3 could then be extended with the statement that these possibly infinite νHML formulae are equivalent to general DMTS/NAA.

For a DMTS $S = (S, S^0, \dashrightarrow, \longrightarrow)$, let $\mathrm{Tran}(s) = \{M \subseteq \Sigma \times S \mid \exists N : s \longrightarrow N, N \subseteq M; \forall (a,t) \in M : s \overset{a}{\dashrightarrow} t\}$ and define the NAA $dn(S) = (S, S^0, \mathrm{Tran})$.

Conversely, for an NAA (S, S^0, Tran), define the DMTS $nd(S) = (T, T^0, \dashrightarrow, \longrightarrow)$ as follows:

- $T = \{M \in \mathrm{Tran}(s) \mid s \in S\}$, $T^0 = \{M \in \mathrm{Tran}(s^0) \mid s^0 \in S^0\}$,
- $\longrightarrow = \{(M, \{(a, M') \mid M' \in \mathrm{Tran}(s')\} \mid (a, s') \in M\}$,
- $\dashrightarrow = \{(t, a, t') \mid t \in T, \exists (t, N) \in \longrightarrow : (a, t') \in N\}$.

Note that both *nd* and *dn* preserve finiteness. Both translation are exponential in their respective arguments.

Lemma 4. *For every DMTS S, $S \equiv_{\mathrm{t}} dn(S)$. For every NAA S, $S \equiv_{\mathrm{t}} nd(S)$.*

For a set of pairs of actions and states M we use M_a to denote the set $\{s \mid (a,s) \in M\}$. Let (S, S^0, Tran) be a finite NAA and let $s \in S$, we then define

$$\Delta_{\mathrm{Tran}}(s) = \bigvee_{M \in \mathrm{Tran}(s)} \Big(\bigwedge_{(a,t) \in M} \langle a \rangle t \wedge \bigwedge_{a \in \Sigma} [a] \Big(\bigvee_{u \in M_a} u \Big) \Big)$$

We then define the νHML formula $nh(S) = (S, S^0, \Delta_{\mathrm{Tran}})$. Notice that variables in $nh(S)$ are states of S.

Lemma 5. *For all NAA S, $S \equiv_{\mathrm{t}} nh(S)$.*

Our translation from νHML to DMTS is based on the constructions in [10]. First, we need a variant of a disjunctive normal form for HML formulae:

Lemma 6. *For any νHML formula (X_1, X_1^0, Δ_1), there exists another formula (X_2, X_2^0, Δ_2) with $[\![\Delta_1]\!] = [\![\Delta_2]\!]$ and such that any formula $\Delta_2(x)$, for $x \in X_2$, is $\mathbf{tt}$ or of the form $\Delta_2(x) = \bigvee_{i\in I} \left(\bigwedge_{j\in J_i} \langle a_{ij}\rangle x_{ij} \wedge \bigwedge_{a\in\Sigma} [a] y_{i,a} \right)$ for finite (possibly empty) index sets I and J_i, $i \in I$, and all $x_{ij}, y_{i,a} \in X_2$. Additionally we can assume that for all $i \in I$, $j \in J_i$, $a \in \Sigma$, $a_{ij} = a$ implies $[\![x_{ij}]\!] \subseteq [\![y_{i,a}]\!]$.*

Let now (X, X^0, Δ) be a νHML formula in the form introduced above, then we define a DMTS $hd(\Delta) = (S, S^0, \dashrightarrow, \longrightarrow)$ as follows:

- $S = \{(x,k) \mid x \in X, \Delta(x) = \bigvee_{i\in I} \phi_i, k \in I \neq \emptyset\} \cup \{\bot, \top\}$,
- $S^0 = \{(x^0, k) \mid x^0 \in X^0\}$.
- For each $(x,k) \in S$ with $\Delta(x) = \bigvee_{i\in I}(\bigwedge_{j\in J_i} \langle a_{ij}\rangle x_{ij} \wedge \bigwedge_{a\in\Sigma}[a]y_{i,a})$ and $I \neq \emptyset$,
  - for each $j \in J_i$, let $\mathrm{Must}_j(x,k) = \{(a_{ij}, (x_{ij}, i')) \in \Sigma \times S\}$,
  - for each $a \in \Sigma$, let $\mathrm{May}_a(x,k) = \{(x', i') \in S \mid [\![x']\!] \subseteq [\![y_{i,a}]\!]\}$.
- Let $\dashrightarrow = \{(s, a, s') \mid s \in S, a \in \Sigma, s' \in \mathrm{May}_a(s)\} \cup \{(\top, a, \top) \mid a \in \Sigma\}$ and $\longrightarrow = \{(s, \mathrm{Must}_j(s)) \mid s = (x,i) \in S, j \in J_i\} \cup \{(\bot, \emptyset)\}$.

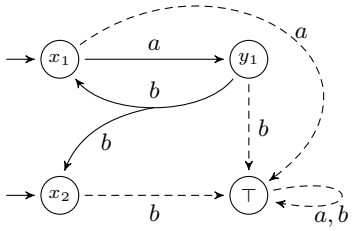
Lemma 7. *For all νHML formulae Δ, $\Delta \equiv_{\mathrm{t}} hd(\Delta)$.*

Further, we remark that the overall translation from DMTS to νHML is quadratic and in the other direction inevitably exponential.

Example 8. Consider the νHML formula $X = (\langle a\rangle(\langle b\rangle X \wedge [a]\mathbf{ff}) \wedge [b]\mathbf{ff}) \vee [a]\mathbf{ff}$. Changing the formula into the normal form of Lemma 6 introduces a new variable Y as illustrated below; X remains the sole initial variable. The translation hd then gives a DMTS with two initial states (the inconsistent state $\bot$ and redundant may transitions such as $x_1 \overset{a}{\dashrightarrow} x_2$, $x_2 \overset{b}{\dashrightarrow} x_1$, etc. have been omitted):

$$X = (\overbrace{\langle a\rangle Y \wedge [a]\mathbf{tt} \wedge [b]\mathbf{ff}}^{x_1}) \vee (\underbrace{[a]\mathbf{ff} \wedge [b]\mathbf{tt}}_{x_2})$$

$$Y = \underbrace{\langle b\rangle X \wedge [a]\mathbf{ff} \wedge [b]\mathbf{tt}}_{y_1}$$



3 Specification Theory

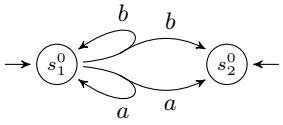
In this section, we introduce operations of conjunction, disjunction, structural composition and quotient for NAA, DMTS and νHML. Together, these operations yield a *complete specification theory* in the sense of [1], which allows for compositional design and verification using both logical and structural operations. We remark that conjunction and disjunction are straightforward for logical formalisms such as νHML, whereas structural composition is more readily defined on behavioural formalisms such as (D)MTS. For the mixed formalism of NAA, disjunction is trivial as we permit multiple initial states, but conjunction requires some work. Note that our construction of conjunction works for nondeterministic systems in contrast to all the work in this area except for [6, 26].

3.1 Disjunction

The disjunction of NAA $S_1 = (S_1, S_1^0, \mathrm{Tran}_1)$ and $S_2 = (S_2, S_2^0, \mathrm{Tran}_2)$ is $S_1 \vee S_2 = (S_1 \cup S_2, S_1^0 \cup S_2^0, \mathrm{Tran}_1 \cup \mathrm{Tran}_2)$. Similarly, the disjunction of two DMTS $S_1 = (S_1, S_1^0, \dashrightarrow_1, \longrightarrow_1)$ and $S_2 = (S_2, S_2^0, \dashrightarrow_2, \longrightarrow_2)$ is $S_1 \vee S_2 = (S_1 \cup S_2, S_1^0 \cup S_2^0, \dashrightarrow_1 \cup \dashrightarrow_2, \longrightarrow_1 \cup \longrightarrow_2)$. It follows that disjunction respects the translation mappings *dn* and *nd* from the previous section.

Theorem 9. *Let S_1, S_2, S_3 be NAA or DMTS. Then $[\![S_1 \vee S_2]\!] = [\![S_1]\!] \cup [\![S_2]\!]$. Further, $S_1 \vee S_2 \leq_{\mathrm{m}} S_3$ iff $S_1 \leq_{\mathrm{m}} S_3$ and $S_2 \leq_{\mathrm{m}} S_3$.*

We point out one important distinction between NAA and DMTS: NAA with a *single* initial state are equally expressive as general NAA, while for DMTS, this is not the case. The example on the right shows a DMTS $(S, S^0, \dashrightarrow, \longrightarrow)$, with $S = S^0 = \{s_1^0, s_2^0\}$, $s_1^0 \longrightarrow \{(a, s_1^0), (a, s_2^0)\}$ and $s_1^0 \longrightarrow \{(b, s_1^0), (b, s_2^0)\}$ (and the corresponding may-transitions). Two initial states are necessary for capturing $[\![S]\!]$.



Lemma 10. *For any NAA S there is a NAA $T = (T, T^0, \Psi)$ with $T^0 = \{t^0\}$ a singleton and $S \equiv_{\mathrm{m}} T$.*

3.2 Conjunction

Conjunction for DMTS is an extension of the construction from [6] for multiple initial states. Given two DMTS $(S_1, S_1^0, \dashrightarrow_1, \longrightarrow_1)$, $(S_2, S_2^0, \dashrightarrow_2, \longrightarrow_2)$, we define $S_1 \wedge S_2 = (S, S^0, \dashrightarrow, \longrightarrow)$ with $S = S_1 \times S_2$, $S^0 = S_1^0 \times S_2^0$, and

- $(s_1, s_2) \overset{a}{\dashrightarrow} (t_1, t_2)$ iff $s_1 \overset{a}{\dashrightarrow}_1 t_1$ and $s_2 \overset{a}{\dashrightarrow}_2 t_2$,
- for all $s_1 \longrightarrow N_1$, $(s_1, s_2) \longrightarrow \{(a, (t_1, t_2)) \mid (a, t_1) \in N_1, (s_1, s_2) \overset{a}{\dashrightarrow} (t_1, t_2)\}$,
- for all $s_2 \longrightarrow N_2$, $(s_1, s_2) \longrightarrow \{(a, (t_1, t_2)) \mid (a, t_2) \in N_2, (s_1, s_2) \overset{a}{\dashrightarrow} (t_1, t_2)\}$.

To define conjunction for NAA, we need auxiliary projection functions $\pi_i : \Sigma \times S_1 \times S_2 \to \Sigma \times S_i$. These are defined by

$$\pi_1(M) = \{(a, s_1) \mid \exists s_2 \in S_2 : (a, s_1, s_2) \in M\}$$
$$\pi_2(M) = \{(a, s_2) \mid \exists s_1 \in S_1 : (a, s_1, s_2) \in M\}$$

Given NAA $(S_1, S_1^0, \mathrm{Tran}_1)$, $(S_2, S_2^0, \mathrm{Tran}_2)$, define $S_1 \wedge S_2 = (S, S^0, \mathrm{Tran})$, with $S = S_1 \times S_2$, $S^0 = S_1^0 \times S_2^0$ and $\mathrm{Tran}((s_1, s_2)) = \{M \subseteq \Sigma \times S_1 \times S_2 \mid \pi_1(M) \in \mathrm{Tran}_1(s_1), \pi_2(M) \in \mathrm{Tran}_2(s_2)\}$.

Lemma 11. *For DMTS S_1, S_2, $dn(S_1 \wedge S_2) = dn(S_1) \wedge dn(S_2)$.*

For the translation from NAA to DMTS, $nd(S_1 \wedge S_2) = nd(S_1) \wedge nd(S_2)$ does not necessarily hold, as the translation changes the state space. However, Theorem 12 below will ensure that $nd(S_1 \wedge S_2) \equiv_{\mathrm{t}} nd(S_1) \wedge nd(S_2)$.

Theorem 12. *Let S_1, S_2, S_3 be NAA or DMTS. Then $[\![S_1 \wedge S_2]\!] = [\![S_1]\!] \cap [\![S_2]\!]$. Further, $S_1 \leq_{\mathrm{m}} S_2 \wedge S_3$ iff $S_1 \leq_{\mathrm{m}} S_2$ and $S_1 \leq_{\mathrm{m}} S_3$.*

Theorem 13. *With operations $\wedge$ and $\vee$, the sets of DMTS and NAA form bounded distributive lattices up to $\equiv_{\mathrm{m}}$.*

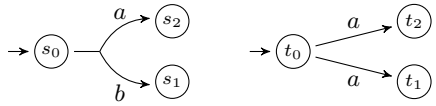
3.3 Structural Composition

We define structural composition for NAA. For NAA $S_1 = (S_1, S_1^0, \mathrm{Tran}_1)$, $S_2 = (S_2, S_2^0, \mathrm{Tran}_2)$, we define $S_1 \| S_2 = (S, S^0, \mathrm{Tran})$ with $S = S_1 \times S_2$, $S^0 = S_1^0 \times S_2^0$, and for all $(s_1, s_2) \in S$, $\mathrm{Tran}((s_1, s_2)) = \{M_1 \| M_2 \mid M_1 \in \mathrm{Tran}_1(s_1), M_2 \in \mathrm{Tran}_2(s_2)\}$, where $M_1 \| M_2 = \{(a, (t_1, t_2)) \mid (a, t_1) \in M_1, (a, t_2) \in M_2\}$.

Lemma 14. *Up to $\equiv_{\mathrm{m}}$, the operator $\|$ on NAA is associative and commutative, distributes over $\vee$, and has unit U, where U is the LTS $(\{s\}, s, \longrightarrow)$ with $s \xrightarrow{a} s$ for all $a \in \Sigma$.*

Theorem 15. *For all NAA S_1, S_2, S_3, S_4, $S_1 \leq_{\mathrm{m}} S_3$ and $S_2 \leq_{\mathrm{m}} S_4$ imply $S_1 \| S_2 \leq_{\mathrm{m}} S_3 \| S_4$.*

We remark that structural composition on MTS [27] coincides with our NAA composition, so that for MTS S_1, S_2, $dn(S_1) \| dn(S_2) = dn(S_1 \| S_2)$. On the other hand, structural composition for DMTS (with single initial states) as defined in [6] is *weaker* than NAA composition, *i.e.* for DMTS S_1, S_2, and denoting by $\|'$ the composition from [6], only $dn(S_1) \| dn(S_2) \leq_{\mathrm{t}} dn(S_1 \|' S_2)$ holds. Consider for example the DMTS S and S' in the figure below. When considering their NAA composition, the initial state is the pair (s_0, t_0) with $\mathrm{Tran}((s_0, t_0)) = \{\emptyset, \{(a, (s_2, t_1)), (a, (s_2, t_2))\}$. Since this constraint cannot be represented as a disjunctive must, there is no DMTS with a single initial state which can represent the NAA composition precisely.



Hence the DMTS composition of [6] is a DMTS over-approximation of the NAA composition, and translating from DMTS to NAA before composing (and back again) will generally give a tighter specification. However, as noted already in [24], MTS composition itself is an over-approximation, in the sense that there will generally be implementations $I \in [\![S_1 \| S_2]\!]$ which cannot be written $I = I_1 \| I_2$ for $I_1 \in [\![S_1]\!]$ and $I_2 \in [\![S_2]\!]$; the same is the case for NAA and DMTS.

3.4 Quotient

We now present one of the central contributions of this paper, the construction of quotient. The quotient S/T is to be the most general specification that, when composed with T, refines S. In other words, it must satisfy the property that for all specifications X, $X \leq_{\mathrm{m}} S/T$ iff $X \parallel T \leq_{\mathrm{m}} S$. Quotient has been defined for deterministic MTS and for deterministic acceptance automata in [35]; here we extend it to the nondeterministic case (*i.e.* NAA). The construction incurs an exponential blow-up, which however is local and depends on the degree of nondeterminism. We also provide a quotient construction for nondeterministic MTS; this is useful because MTS encodings for NAA can be very compact.

Let $(S, S^0, \mathrm{Tran}_S)$, $(T, T^0, \mathrm{Tran}_T)$ be two NAA. We define the quotient $S/T = (Q, \{q^0\}, \mathrm{Tran}_Q)$. Let $Q = 2^{S\times T}_{\mathrm{Fin}}$ and $q^0 = \{(s^0, t^0) \mid s^0 \in S^0, t^0 \in T^0\}$. States in Q will be written $\{s_1/t_1, \ldots, s_n/t_n\}$ instead of $\{(s_1, t_1), \ldots, (s_n, t_n)\}$.

In the following, we use the notation $x \in\in z$ as a shortcut for the fact that there exists y with $x \in y \in z$. We first define $\mathrm{Tran}_Q(\emptyset) = 2^{\Sigma\times\{\emptyset\}}$. This means that the empty set of pairs is the universal state $\top$. Now let $q = \{s_1/t_1, \ldots, s_n/t_n\} \in Q$. We first define the auxiliary set of possible transitions $pt(q)$ as follows. For $x \in S \cup T$, let $\alpha(x) = \{a \in \Sigma \mid \exists y : (a, y) \in\in \mathrm{Tran}(x)\}$ and $\gamma(q) = \bigcap_i \big(\alpha(s_i) \cup (\Sigma \setminus \alpha(t_i))\big)$. Let further $\pi_a(X) = \{x \mid (a, x) \in X\}$.

Let now $a \in \gamma(q)$. For all $i \in \{1, \ldots, n\}$, let $\{t_{i,1}, \ldots, t_{i,m_i}\} = \pi_a(\bigcup \mathrm{Tran}_T(t_i))$ be the possible next states from t_i after an a-transition, and define

$$pt_a(q) = \big\{\{s_{i,j}/t_{i,j} \mid i \in \{1, \ldots, n\}, j \in \{1, \ldots, m_i\}\} \mid \\ \forall i \in \{1, \ldots, n\} : \forall j \in \{1, \ldots, m_i\} : (a, s_{i,j}) \in\in \mathrm{Tran}_S(s_i)\big\}$$

and $pt(q) = \bigcup_{a\in\Sigma}(\{a\} \times pt_a(q))$. Hence $pt_a(q)$ contains sets of possible next quotient states after an a-transition, each obtained by combining the $t_{i,j}$ with some permutation of possible next a-states in S. We then define

$$\mathrm{Tran}_Q(q) = \{X \subseteq pt(q) \mid \forall i : \forall Y \in \mathrm{Tran}_T(t_i) : X \triangleright Y \in \mathrm{Tran}_S(s_i)\},$$

where the operator $\triangleright$ is defined by $\{s_1/t_1, \ldots, s_k/t_k\} \triangleright t_\ell = s_\ell$ and $X \triangleright Y = \{(a, x \triangleright y) \mid (a, x) \in X, (a, y) \in Y\}$. Hence $\mathrm{Tran}_Q(q)$ contains all sets of (possible) transitions which are compatible with all t_i in the sense that (the projection of) their parallel composition with any set $Y \in \mathrm{Tran}_T(t_i)$ is in $\mathrm{Tran}_S(s_i)$.

Theorem 16. *For all NAA S, T and X, $X\|T \leq_{\mathrm{m}} S$ iff $X \leq_{\mathrm{m}} S/T$.*

Theorem 17. *With operations $\wedge$, $\vee$, $\|$ and $/$, the set of NAA forms a commutative residuated lattice up to $\equiv_{\mathrm{m}}$.*

This theorem makes clear the relation of NAA to linear logic [20]: except for completeness of the lattice induced by $\wedge$ and $\vee$ (*cf.* Theorem 13), NAA form a *commutative unital Girard quantale* [40], the standard algebraic setting for linear logic. Completeness of the lattice can be obtained by allowing infinite conjunctions and disjunctions (and infinite NAA).

3.5 Quotient for MTS

We now give a quotient algorithm for the important special case of MTS, which results in a much more compact quotient than the NAA construction in the previous section. However, MTS are not closed under quotient; *cf.* [28, Thm. 5.5]. We show that the quotient of two MTS will generally be a DMTS.

Let $(S, s^0, \dashrightarrow_S, \longrightarrow_S)$ and $(T, t^0, \dashrightarrow_T, \longrightarrow_T)$ be nondeterministic MTS. We define the quotient $S/T = (Q, \{q^0\}, \dashrightarrow_Q, \longrightarrow_Q)$. We let $Q = 2^{S\times T}_{\mathrm{Fin}}$ as before, and $q^0 = \{(s^0, t^0)\}$. The state $\emptyset \in Q$ is again universal, so we define $\emptyset \overset{a}{\dashrightarrow} \emptyset$ for all $a \in \Sigma$. There are no must transitions from $\emptyset$.

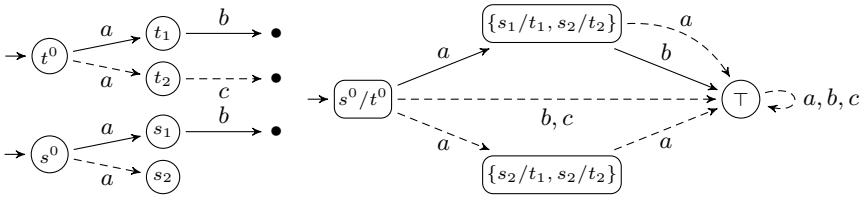


Fig. 4. Two nondeterministic MTS and their quotient

Let $\alpha(s)$, $\gamma(q)$ be as in the previous section. For convenience, we work with sets $\mathrm{May}_a(s)$, for $a \in \Sigma$ and states s, instead of may transitions, *i.e.* we have $\mathrm{May}_a(s) = \{t \mid s \dashrightarrow^a t\}$.

Let $q = \{s_1/t_1, \ldots, s_n/t_n\} \in Q$ and $a \in \Sigma$. First we define the may transitions. If $a \in \gamma(q)$ then for each $i \in \{1, \ldots, n\}$, write $\mathrm{May}_a(t_i) = \{t_{i,1}, \ldots, t_{i,m_i}\}$, and define

$$\mathrm{May}_a(q) = \big\{\{s_{i,j}/t_{i,j} \mid i \in \{1,\ldots,n\}, j \in \{1,\ldots,m_i\}\} \mid \\ \forall i \in \{1,\ldots,n\} : \forall j \in \{1,\ldots,m_i\} : s_{i,j} \in \mathrm{May}_a(s_i)\big\}.$$

For the (disjunctive) must-transitions, we let, for every $s_i \xrightarrow{a} s'$,

$$q \longrightarrow \{(a, M) \in \{a\} \times \mathrm{May}_a(q) \mid \exists t' : s'/t' \in M,\ t_i \xrightarrow{a} t'\}.$$

Example 18. We illustrate the construction on an example. Let S and T be the MTS in the left part of Fig. 4. We construct S/T; the end result is displayed in the right part of the figure.

First we construct the may-successors of s^0/t^0. Under b and c there are no constraints, hence we go to $\top$. For a, we have all permutations of assignments of successors of s to successors of t, namely $\{s_1/t_1, s_1/t_2\}$, $\{s_1/t_1, s_2/t_2\}$, $\{s_2/t_1, s_1/t_2\}$ and $\{s_2/t_1, s_2/t_2\}$. Since there is a must-transition from s (to s_1), we create a disjunctive must-transition to all successors that can be used to yield a must-transition when composed with the must-transition from t to t_1. These are all successors where t_1 is mapped to s_1, hence the first two. However, $\{s_1/t_1, s_1/t_2\}$ will turn out inconsistent, as it requires to refine s_1 by a composition with t_2. As t_2 has no must under b, the composition has none either, hence the must of s_1 can never be matched. As a result, after pruning, the disjunctive must from $\{s^0/t^0\}$ leads only to $\{s_1/t_1, s_2/t_2\}$. Further, $\{s_2/t_1, s_1/t_2\}$ is inconsistent for the same reason, so that we only have one other may-transition under a from $\{s^0/t^0\}$.

Now $\{s_1/t_1, s_2/t_2\}$ is obliged to have a must under b so that it refines s_1 when composed with t_1, but cannot have any c in order to match s_2 when composed with t_2. Similarly, $\{s_2/t_1, s_2/t_2\}$ has neither c nor b. One can easily verify that $T \| (S/T) \equiv_{\mathrm{m}} S$ in this case.

Note that the constructions may create inconsistent states, which have no implementation. In order to get a consistent system, it needs to be pruned. This is standard and the details can be found in [3]. The pruning can be done in polynomial time.

Theorem 19. *For all MTS S, T and X, $X \leq_{\mathrm{m}} S/T$ iff $T \| X \leq_{\mathrm{m}} S$.*

4 Conclusion and Future Work

In this paper we have introduced a general specification framework whose basis consists of three different but equally expressive formalisms: one of a graphical behavioural kind (DMTS), one logic-based (νHML) and one an intermediate language between the former two (NAA). We have shown that the framework possesses a rich algebraic structure that includes logical (conjunction, disjunction) and structural operations (parallel composition and quotient). Moreover, the construction of the quotient solves an open problem in the area of MTS. As for future work, we hope to establish the exact complexity of the quotient constructions. We conjecture that the exponential blow-up of the construction is in general unavoidable.

References

1. S.S. Bauer, A. David, R. Hennicker, K.G. Larsen, A. Legay, U. Nyman, and A. Wasowski. Moving from specifications to contracts in component-based design. In *FASE*, pages 43–58, 2012.
2. S.S. Bauer, P. Mayer, and A. Legay. MIO workbench: A tool for compositional design with modal input/output interfaces. In *ATVA*, pages 418–421, 2011.
3. N. Beneš, B. Delahaye, U. Fahrenberg, J. Křetínský, and A. Legay. Hennessy-Milner logic with greatest fixed points as a complete behavioural specification theory. *CoRR*, abs/1306.0741, 2013.
4. N. Beneš and J. Křetínský. Process algebra for modal transition systemses. In *MEMICS*, pages 9–18, 2010.
5. N. Beneš, J. Křetínský, K. G. Larsen, M. H. Møller, and J. Srba. Parametric modal transition systems. In *ATVA*, pages 275–289, 2011.
6. N. Beneš, I. Černá, and J. Křetínský. Modal transition systems: Composition and LTL model checking. In *ATVA*, pages 228–242, 2011.
7. P. Blackburn. Representation, reasoning, and relational structures: a hybrid logic manifesto. *Logic J. IGPL*, 8(3):339–365, 2000.
8. BMoTras. http://delahaye.benoit.free.fr/BMoTraS.tar.
9. A. Børjesson, K.G. Larsen, and A. Skou. Generality in design and compositional verification using TAV. *Formal Meth. Syst. Design*, 6(3):239–258, 1995.
10. G. Boudol and K.G. Larsen. Graphical versus logical specifications. *Theor. Comput. Sci.*, 106(1):3–20, 1992.
11. G. Bruns. An industrial application of modal process logic. *Sci. Comput. Program.*, 29(1-2):3–22, 1997.
12. G. Bruns and P. Godefroid. Model checking partial state spaces with 3-valued temporal logics. In *CAV*, pages 274–287, 1999.
13. L. Caires and L. Cardelli. A spatial logic for concurrency (part I). *Inf. Comput.*, 186(2):194–235, 2003.
14. E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs*, pages 52–71, 1981.
15. P. Darondeau, J. Dubreil, and H. Marchand. Supervisory control for modal specifications of services. In *WODES*, pages 428–435, 2010.

16. N. D'Ippolito, D. Fischbein, H. Foster, and S. Uchitel. MTSA: Eclipse support for modal transition systems construction, analysis and elaboration. In *ETX*, pages 6–10, 2007.
17. H. Fecher and H. Schmidt. Comparing disjunctive modal transition systems with an one-selecting variant. *J. Logic Algebr. Program.*, 77(1-2):20–39, 2008.
18. H. Fecher and M. Steffen. Characteristic mu-calculus formulas for underspecified transition systems. *Electr. Notes Theor. Comput. Sci.*, 128(2):103–116, 2005.
19. W. Fokkink, R.J. van Glabbeek, and P. de Wind. Compositionality of Hennessy-Milner logic by structural operational semantics. *Theor. Comput. Sci.*, 354(3):421–440, 2006.
20. Jean-Yves Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987.
21. J.B. Hart, L. Rafter, and C. Tsinakis. The structure of commutative residuated lattices. *Internat. J. Algebra Comput.*, 12(4):509–524, 2002.
22. M. Hennessy. Acceptance trees. *J. ACM*, 32(4):896–928, 1985.
23. M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *J. ACM*, 32(1):137–161, 1985.
24. H. Hüttel and K.G. Larsen. The use of static constructs in a modal process logic. In *Logic at Botik*, pages 163–180, 1989.
25. D. Kozen. Results on the propositional mu-calculus. *Theor. Comput. Sci.*, 27:333–354, 1983.
26. K. G. Larsen and Liu X. Equation solving using modal transition systems. In *LICS*, pages 108–117, 1990.
27. K.G. Larsen. Modal specifications. In *Automatic Verification Methods for Finite State Systems*, pages 232–246, 1989.
28. K.G. Larsen. Ideal specification formalism = expressivity + compositionality + decidability + testability + ... In *CONCUR*, pages 33–56, 1990.
29. K.G. Larsen. Proof systems for satisfiability in Hennessy-Milner logic with recursion. *Theor. Comput. Sci.*, 72:265–288, 1990.
30. K.G. Larsen and Liu X. Compositionality through an operational semantics of contexts. In *ICALP*, pages 526–539, 1990.
31. U. Nyman. *Modal Transition Systems as the Basis for Interface Theories and Product Lines*. PhD thesis, Institut for Datalogi, Aalborg Universitet, 2008.
32. P.W. O'Hearn, J.C. Reynolds, and H. Yang. Local reasoning about programs that alter data structures. In *CSL*, pages 1–19, 2001.
33. A.N. Prior. *Papers on Time and Tense*. Oxford: Clarendon Press, 1968.
34. J.-P. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *Symp. Program.*, pages 337–351, 1982.
35. J.-B. Raclet. Residual for component specifications. *Electr. Notes Theor. Comput. Sci.*, 215:93–110, 2008.
36. J.-B. Raclet, E. Badouel, A. Benveniste, B. Caillaud, and R. Passerone. Why are modalities good for interface theories? In *ACSD*, pages 119–127, 2009.
37. J.C. Reynolds. Separation logic: A logic for shared mutable data structures. In *LICS*, pages 55–74, 2002.
38. S. Uchitel and M. Chechik. Merging partial behavioural models. In *SIGSOFT FSE*, pages 43–52, 2004.
39. M. Ward and R.P. Dilworth. Residuated lattices. *Trans. AMS*, 45(3):335–354, 1939.
40. D.N. Yetter. Quantales and (noncommutative) linear logic. *J. Symb. Log.*, 55(1):41–64, 1990.

Paper H:

MoTraS: A tool for modal transition systems and their extensions

Jan Křetínský and Salomon Sickert

This tool paper has been published in Dang Van Hung and Mizuhito Ogawa, editors. Automated Technology for Verification and Analysis - 11th International Symposium, ATVA 2013, Hanoi, Vietnam, October 15-18, 2013. Proceedings, volume 8172 of Lecture Notes in Computer Science, pages 487–491. Springer, 2013. [KS13a]

Summary

We present a tool called $\overrightarrow{\text{MoTraS}}$ for MTS and DMTS and partially BMTS and PMTS extensions. It implements the operations supporting non-deterministic systems. We also implement various refinement decision procedures (fix-point computation as well as reduction to QBF queries), hull operations (deterministic, parameter-free) and model checking (for LTL). The functionality is accessible both in graphical and command-line interface.

Author's contribution: 50 %

- participating in the discussions,
- contributing, in particular, to setting up the research direction and designing the functionality of the tool,
- writing the paper.

MoTraS: A Tool for Modal Transition Systems and Their Extensions*

Jan Křetínský[1,2] and Salomon Sickert[1]

[1] Institut für Informatik, Technische Universität München, Germany
[2] Faculty of Informatics, Masaryk University, Brno, Czech Republic

Abstract. We present a tool for modal transition systems (MTS), disjunctive MTS and further extensions of MTS supporting also non-deterministic systems. We provide the operations required from specification theories as well as some additional support such as deterministic hull, LTL model checking etc. The tool comes with both graphical and command line interface.

1 Introduction

Due to the ever increasing complexity of software systems and their reuse, component-based design and verification have become crucial. Therefore, having a specification formalism that supports *component-based* development and *stepwise refinement* is very useful. In such a framework, one can start from an initial specification, proceed with a series of small and successive refinements until eventually a specification is reached from which an implementation can be extracted directly. *Modal transition systems* (MTS) [11] is a successful specification formalism satisfying the above requirements.

The formalism of MTS has proven to be useful in practice. Industrial applications are as old as [5] where MTS have been used for an air-traffic system at Heathrow airport. Besides, MTS are advocated as an appropriate base for interface theories in [16] and for product line theories in [13]. Further, MTS based software engineering methodology for design via merging partial descriptions of behaviour has been established in [17].

MTS consist of a set of states and two transition relations. The *must* transitions prescribe which behaviour has to be present in every refinement of the system; the *may* transitions describe the behaviour that is allowed, but need not be realized in the refinements. Over the years, many extensions of MTS have been proposed. While MTS can only specify whether or not a particular transition is required, some extensions equip MTS with more general abilities to describe what *combinations* of transitions are possible. Disjunctive MTS (DMTS) [12, 3] can specify that at least one of a given set of transitions is present. One selecting

* Jan Křetínský is partially supported by the Czech Science Foundation, project No. P202/12/G061, and Salomon Sickert is partially supported by the DFG project "Polynomial Systems on Semirings: Foundations, Algorithms, Applications"

MTS [8] allow to choose exactly one of them. Boolean MTS (BMTS) [4] and expressively equivalent acceptance automata [14] cover all Boolean combinations of transitions. Parametric MTS (PMTS) [4] add parameters on top of it, so that we can also express persistent choices of transitions and relate possible choices in different parts of a system. This way, one can also model hardware dependencies of transitions and systems with prices.

The tool support is so far limited to basic MTS and, moreover, partially limited to deterministic systems. The currently available tools are `MTSA` (Modal transition system analyzer) [7] and `MIO` (MIO Workbench) [1]. While `MTSA` is a tool for MTS, `MIO` is a tool for modal I/O automata (MIOA) [10, 15], which combine MTS and interface automata based on I/O automata. Although MIOA have three types of may and must transitions (input, output, and internal), if we restrict to say only input transitions, the refinement works the same as for MTS, and some other operations, too. Further, there are also tools for loosely related formalisms of I/O automata (with no modalities) such as `ECDAR` (Environment for Compositional Design and Analysis of Real Time Systems) [6], which supports their timed extension.

In this paper, we present a tool for MTS and DMTS together with partial support of BMTS and PMTS as described below. In the following sections, we describe MoTraS and compare it to the existing tools both with respect to functionality and experimentally. The tool can be downloaded and additional materials found at `http://www.model.in.tum.de/~kretinsk/motras.html`

2 Functionality

MoTraS comes not only with a graphical user interface, but as opposed to other mentioned tools also with a command line interface, which allows for batch processing. The Netbeans-based GUI offers all the standard components such as a canvas for drawing systems, windows for editing their properties, algorithms menu, possibility to view more systems at once etc. Both the GUI and the independent algorithms package, which contains all data-structures, algorithms and the CLI, are written in Java.

As to the available algorithms, MoTraS supports all operations required for complete specification theories [2] and more. This includes modal refinement checking, parallel composition (for quotient see below), conjunction (or merge) and the related consistency checking and maximal implementation generation, deterministic hull and generalized LTL model checking. This functionality comes for MTS as well as more general DMTS and in all cases also *non-deterministic* systems are supported (in particular, the algorithm for conjunction is now considerably more complex [3]). In contrast, `MTSA` supports only modal refinement, parallel composition and consistency. It also offers a model checking procedure, which is, unfortunately, fundamentally flawed. This has been shown in [3] from where we adopt the corrected implementation. `MIO` offers modal refinement, the MIOA parallel composition and conjunction for deterministic systems. On the top, it also offers quotient for deterministic systems. As there are no algorithms

for the quotient of non-deterministic MTS, DMTS and BMTS and this question is a subject of current research, we only implement the quotient for deterministic systems as `MIO` does. Note that both `MTSA` and `MIO` can only handle modal systems, not their disjunctive extension. MoTraS supports DMTS as they have more expressive power, and as opposed to (non-deterministic) MTS are rich enough to express solutions to process equations [12] (hence a specification of a missing component in a system can be computed) and are closed under all operations, in particular conjunction (which is necessary for merging views on a system).

Further, on the top of this functionality for MTS and DMTS, we also provide an implementation of a new method for modal refinement checking of BMTS and PMTS [9]. While modal refinement on MTS and DMTS can be decided in polynomial time, on BMTS and PMTS it is higher in the polynomial hierarchy (Π_2 and Π_4, respectively). The new method, however, reduces the refinement problem to a problem directly and efficiently solvable by a QBF solver. Already the preliminary results of [9] show that this solution scales well in the size of the system as well as in the number of parameters, while a direct naive solution is infeasible.

Moreover, we also implement the deterministic hull and the parameter-free hull for BMTS and PMTS, which we recently proposed [9]. This allows to both over- and under-approximate the EXPTIME-complete thorough refinement using the fast modal refinement, now even for the most general class of PMTS.

In order to make the tool easily extensible, we introduced a file format *xmts*, which facilitates textual representation of different extensions of modal transition systems. The description of the format can be found on the web page of the tool. The table below summarizes the functionality: ✓indicates a MoTraS implementation; for the other tools, the name indicates an implementation; "det." denotes a functionality limited to deterministic systems.

Operation	MTS			DMTS	BMTS	PMTS
Parallel composition	`MTSA`	`MIO`(MIAO)	✓	✓		
Consistency	`MTSA`(of 2 systems)	`MIO`(det.)	✓	✓		
Conjunction		`MIO`(det.)	✓	✓		
Quotient (det.)		`MIO`	✓	×		
Generalized LTL	`MTSA`(incorrect)		✓	✓		
Det./Par. hull			✓	✓	✓	✓
Refinement	`MTSA`	`MIO`	✓	✓	✓	✓

3 Experimental Results

We briefly compare the performance of the MTS tools on the algorithm that each of them implements, namely modal refinement of MTS. We compare `MoTraS` and `MTSA` on systems with 500 states, see the table on the right (computational times in seconds). We consider systems with different sizes

Size	Structure:	`MTSA`	`MoTraS`
Alphabet 2,	Monolithic:	4.57	2.23
branching 5	Clustered:	0.34	0.04
Alphabet 2,	Monolithic:	6.62	8.75
branching 10	Clustered:	5.99	6.73
Alphabet 10,	Monolithic:	1.46	0.50
branching 5	Clustered:	1.54	0.01

of alphabet and branching degrees. We further consider monolithic systems where the transitions are evenly distributed, and systems with several clusters mutually connected with only a few edges. We do not include results for `MIO` here as there are stack overflows already for systems with <150 states, and for systems with 100 states the time is—despite `MIO` statistics reporting 0 seconds—actually more than 3 seconds. See the webpage for more details and more experiments.

Although our results are already more than competitive, there are several ways how to further optimize them. The algorithms are implemented using fixed-point-iteration and waiting-queue skeleton classes, which allows for an easy introduction of multi-threading to all algorithms. Due to the independence of elements processing we conjecture the speed up factor will be very close to the number of cores used.

For the QBF-based modal refinement some additional steps were taken to reduce the memory footprint, such as storing the generated formulae in negation-normal-form and using the Tseitin encoding to limit the growth of the formulae while transforming it into CNF, which is required for the QBF solver. An interesting task for the future work is to introduce combined modal refinement checker, which uses the standard modal refinement checker to prune the initial relation before the QBF-based checker is called.

References

1. S. S. Bauer, P. Mayer, and A. Legay. MIO workbench: A tool for compositional design with modal input/output interfaces. In *ATVA*, pages 418–421, 2011.
2. S.S. Bauer, A. David, R. Hennicker, K.G. Larsen, A. Legay, U. Nyman, and A. Wasowski. Moving from specifications to contracts in component-based design. In *FASE*, pages 43–58, 2012.
3. N. Beneš, I. Cerná, and J. Křetínský. Modal transition systems: Composition and LTL model checking. In *ATVA*, pages 228–242, 2011.
4. N. Beneš, J. Křetínský, K. G. Larsen, M. H. Møller, and J. Srba. Parametric modal transition systems. In *ATVA*, pages 275–289, 2011.
5. G. Bruns. An industrial application of modal process logic. *Sci. Comput. Program.*, 29(1-2):3–22, 1997.
6. A. David, K. G. Larsen, A. Legay, U. Nyman, and A. Wasowski. ECDAR: An environment for compositional design and analysis of real time systems. In *ATVA*, pages 365–370, 2010.
7. N. D'Ippolito, D. Fischbein, H. Foster, and S. Uchitel. MTSA: Eclipse support for modal transition systems construction, analysis and elaboration. In *ETX*, pages 6–10, 2007.
8. H. Fecher and H. Schmidt. Comparing disjunctive modal transition systems with an one-selecting variant. *J. Log. Algebr. Program.*, 77(1-2):20–39, 2008.
9. J. Křetínský and S. Sickert. On refinements of Boolean and parametric modal transition systems. In *ICTAC*, pages 213–230, 2013.
10. K. G. Larsen, U. Nyman, and A. Wasowski. Modal I/O automata for interface and product line theories. In *ESOP*, pages 64–79, 2007.
11. K. G. Larsen and B. Thomsen. A modal process logic. In *LICS*, pages 203–210, 1988.

12. K. G. Larsen and L. Xinxin. Equation solving using modal transition systems. In *LICS*, pages 108–117, 1990.
13. U. Nyman. *Modal Transition Systems as the Basis for Interface Theories and Product Lines*. PhD thesis, Institut for Datalogi, Aalborg Universitet, 2008.
14. J.-B. Raclet. *Quotient de spécifications pour la réutilisation de composants*. PhD thesis, Université de Rennes I, december 2007. (In French).
15. J.-B. Raclet, E. Badouel, A. Benveniste, B.Caillaud, A. Legay, and R. Passerone. A modal interface theory for component-based design. *Fundamenta Informaticae*, 108(1-2):119–149, 2011.
16. J.-B. Raclet, E. Badouel, A. Benveniste, B. Caillaud, and R. Passerone. Why are modalities good for interface theories? In *ACSD*, 2009.
17. S. Uchitel and M. Chechik. Merging partial behavioural models. In *SIGSOFT FSE*, pages 43–52, 2004.

Paper I:

From LTL to deterministic automata: A Safraless compositional approach

Javier Esparza and Jan Křetínský

This paper has been accepted at CAV 2014. [EK14]

Summary

As opposed to our previous work on translating fragments of LTL to deterministic automata [KE12, GKE12, KLG13, Kře13], here we provide a translation for the whole LTL. As the target formalism we use deterministic generalized Rabin automata, which allow not only for more efficient probabilistic model checking, but also for more efficient LTL game solving [CGK13] and thus also DMTS implementation synthesis. The resulting automaton is composed of automata for each **G**-subformula, which can be separately optimized allowing for compositional optimization. According to the experimental results, the resulting automata are smaller than those produced by standard methods. The differences are in orders of magnitude for more complex, but even still short formulae. Moreover, although the translation works for the whole LTL, the results achieved are as good as of the previous direct translations for fragments of LTL.

Author's contribution: 50 %

- participating in the discussions,
- designing the translation,
- writing the translation procedure and minor parts of the paper.

From LTL to Deterministic Automata: A Safraless Compositional Approach

Javier Esparza and Jan Křetínský*

Institut für Informatik, Technische Universität München, Germany
IST Austria

Abstract. We present a new algorithm to construct a (generalized) deterministic Rabin automaton for an LTL formula φ. The automaton is the product of a master automaton and an array of slave automata, one for each **G**-subformula of φ. The slave automaton for $\mathbf{G}\psi$ is in charge of recognizing whether $\mathbf{FG}\psi$ holds. As opposed to standard determinization procedures, the states of all our automata have a clear logical structure, which allows for various optimizations. Our construction subsumes former algorithms for fragments of LTL. Experimental results show improvement in the sizes of the resulting automata compared to existing methods.

1 Introduction

Linear temporal logic (LTL) is the most popular specification language for linear-time properties. In the automata-theoretic approach to LTL verification, formulae are translated into ω-automata, and the product of these automata with the system is analyzed. Therefore, generating small ω-automata is crucial for the efficiency of the approach.

In quantitative probabilistic verification, LTL formulae need to be translated into *deterministic* ω-automata [BK08,CGK13]. Until recently, this required to proceed in two steps: first translate the formula into a non-deterministic Büchi automaton (NBA), and then apply Safra's construction [Saf88], or improvements on it [Pit06,Sch09] to transform the NBA into a deterministic automaton (usually a Rabin automaton, or DRA). This is also the approach adopted in `PRISM` [KNP11], a leading probabilistic model checker, which reimplements the optimized Safra's construction of `ltl2dstar` [Kle].

In [KE12] we presented an algorithm that *directly* constructs a generalized DRA (GDRA) for the fragment of LTL containing only the temporal operators **F** and **G**. The GDRA can be either (1) degeneralized into a standard DRA, or (2) used directly in the probabilistic verification process [CGK13]. In both cases we get much smaller automata for many formulae. For instance, the standard

* This research was funded in part by the European Research Council (ERC) under grant agreement 267989 (QUAREM) and by the Austrian Science Fund (FWF) project S11402-N23 (RiSE). The author is on leave from Faculty of Informatics, Masaryk University, Czech Republic, and partially supported by the Czech Science Foundation, grant No. P202/12/G061.

approach translates a conjunction of three fairness constraints into an automaton with over a million states, while the algorithm of [KE12] yields a GDRA with one single state (when acceptance is defined on transitions), and a DRA with 462 states. In [GKE12,KLG13] our approach was extended to larger fragments of LTL containing the **X** operator and restricted appearances of **U**, but a general algorithm remained elusive.

In this paper we present a novel approach able to handle full LTL, and even the alternation-free linear-time μ-calculus. The approach is *compositional*: the automaton is obtained as a parallel composition of automata for different parts of the formula, running in lockstep[1]. More specifically, the automaton is the parallel composition of a *master automaton* and an array of *slave automata*, one for each **G**-subformula of the original formula, say φ. Intuitively, the master monitors the formula that remains to be fulfilled (for example, if $\varphi = (\neg a \wedge \mathbf{X}a) \vee \mathbf{XXG}a$, then the remaining formula after $\emptyset\{a\}$ is $\mathbf{tt}$, and after $\{a\}$ it is $\mathbf{XG}a$), and takes care of checking safety and reachability properties. The slave for a subformula $\mathbf{G}\psi$ of φ checks whether $\mathbf{G}\psi$ *eventually* holds, i.e., whether $\mathbf{FG}\psi$ holds. It also monitors the formula that remains to be fulfilled, but only partially: more precisely, it does not monitor any **G**-subformula of ψ, as other slaves are responsible for them. For instance, if $\psi = a \wedge \mathbf{G}b \wedge \mathbf{G}c$, then the slave for $\mathbf{G}\psi$ only checks that eventually a always holds, and "delegates" checking $\mathbf{FG}b$ and $\mathbf{FG}c$ to other slaves. Further, and crucially, the slave may provide the information that not only $\mathbf{FG}\psi$, but a stronger formula holds; the master needs this to decide that, for instance, not only $\mathbf{FG}\varphi$ but even $\mathbf{XG}\varphi$ holds.

The acceptance condition of the parallel composition of master and slaves is a disjunction over all possible subsets of **G**-subformulas, and all possible stronger formulas the slaves can check. The parallel composition accepts a word with the disjunct corresponding to the subset of formulas which hold in it.

The paper is organized incrementally. In Section 3 we show how to construct a DRA for a formula $\mathbf{FG}\varphi$, where φ has no occurrence of **G**. This gives the DRA for a bottom-level slave. Section 4 constructs a DRA for an arbitrary formula $\mathbf{FG}\varphi$, which gives the DRA for a general slave, in charge of a formula that possible has **G**-subformulas. Finally, Section 5 constructs a DRA for arbitrary formulas by introducing the master and its parallel composition with the slaves. Full proofs can be found in [EK14].

Related work A comparison of LTL translators into deterministic ω-automata can be found in [BKS13]. Safra's construction with optimizations described in [KB07] is implemented in the tool ltl2dstar [Kle], and reimplemenetd in the probabilistic model checker PRISM [KNP11]. There are many constructions translating LTL to NBA [Cou99,DGV99,EH00,SB00,GO01,GL02,Fri03,BKRS12,DL13]. The one recommended by ltl2dstar and used in PRISM is LTL2BA [GO01].

[1] We could also speak of a product of automata, but the operational view behind the term parallel composition helps to convey the intuition.

2 Linear Temporal Logic

In this paper, $\mathbb{N}$ denotes the set of natural numbers including zero. "For almost every $i \in \mathbb{N}$" means for all but finitely many $i \in \mathbb{N}$.

This section recalls the notion of linear temporal logic (LTL). We consider the negation normal form and we have the future operator explicitly in the syntax:

Definition 1 (LTL Syntax). *The formulae of the linear temporal logic (LTL) are given by the following syntax:*

$$\varphi ::= \mathbf{tt} \mid \mathbf{ff} \mid a \mid \neg a \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathbf{X}\varphi \mid \mathbf{F}\varphi \mid \mathbf{G}\varphi \mid \varphi \mathbf{U} \varphi$$

over a finite fixed set Ap of atomic propositions.

Definition 2 (Words and LTL Semantics). *Let $w \in (2^{Ap})^\omega$ be a word. The ith letter of w is denoted $w[i]$, i.e. $w = w[0]w[1]\cdots$. We write w_{ij} for the finite word $w[i]w[i+1]\cdots w[j]$, and $w_{i\infty}$ or just w_i for the suffix $w[i]w[i+1]\cdots$.*

The semantics of a formula on a word w is defined inductively as follows:

$$\begin{array}{lll} w \models \mathbf{tt} & & \\ w \not\models \mathbf{ff} & & \\ w \models a & \iff & a \in w[0] \\ w \models \neg a & \iff & a \notin w[0] \\ w \models \varphi \wedge \psi & \iff & w \models \varphi \text{ and } w \models \psi \\ w \models \varphi \vee \psi & \iff & w \models \varphi \text{ or } w \models \psi \end{array} \qquad \begin{array}{lll} w \models \mathbf{X}\varphi & \iff & w_1 \models \varphi \\ w \models \mathbf{F}\varphi & \iff & \exists\, k \in \mathbb{N} : w_k \models \varphi \\ w \models \mathbf{G}\varphi & \iff & \forall\, k \in \mathbb{N} : w_k \models \varphi \\ w \models \varphi \mathbf{U} \psi & \iff & \exists\, k \in \mathbb{N} : w_k \models \psi \text{ and} \\ & & \forall\, 0 \leq j < k : w_j \models \varphi \end{array}$$

Definition 3 (Propositional implication). *Given two formulae φ and ψ, we say that φ* propositionally implies *ψ, denoted by $\varphi \models_p \psi$, if we can prove $\varphi \models \psi$ using only the axioms of propositional logic. We say that φ and ψ are* propositionally equivalent*, denoted by $\varphi \equiv_p \psi$, if φ and ψ propositionally imply each other.*

Remark 4. We consider formulae up to propositional equivalence, i.e., $\varphi = \psi$ means that φ and ψ are propositionally equivalent. Sometimes (when there is risk of confusion) we explicitly write $\equiv_p$ instead of $=$.

2.1 The formula $af(\varphi, w)$

Given a formula φ and a finite word w, we define a formula $af(\varphi, w)$, read "φ after w". Intuitively, it is the formula that any infinite continuation w' must satisfy for ww' to satisfy φ.

Definition 5. *Let φ be a formula and $\nu \in 2^{Ap}$. We define the formula $af(\varphi, \nu)$ as follows:*

$$\begin{array}{ll} af(\mathbf{tt}, \nu) & = \mathbf{tt} \\ af(\mathbf{ff}, \nu) & = \mathbf{ff} \\ af(a, \nu) & = \begin{cases} \mathbf{tt} \text{ if } a \in \nu \\ \mathbf{ff} \text{ if } a \notin \nu \end{cases} \\ af(\neg a, \nu) & = \neg af(a, \nu) \\ af(\varphi \wedge \psi, \nu) & = af(\varphi, \nu) \wedge af(\psi, \nu) \\ af(\varphi \vee \psi, \nu) & = af(\varphi, \nu) \vee af(\psi, \nu) \end{array} \qquad \begin{array}{ll} af(\mathbf{X}\varphi, \nu) & = \varphi \\ af(\mathbf{G}\varphi, \nu) & = af(\varphi, \nu) \wedge \mathbf{G}\varphi \\ af(\mathbf{F}\varphi, \nu) & = af(\varphi, \nu) \vee \mathbf{F}\varphi \\ af(\varphi \mathbf{U} \psi, \nu) & = af(\psi, \nu) \vee (af(\varphi, \nu) \wedge \varphi \mathbf{U} \psi) \end{array}$$

We extend the definition to finite words as follows: $af(\varphi, \epsilon) = \varphi$ *and* $af(\varphi, \nu w) = af(af(\varphi, \nu), w)$*. Finally, we define* $Reach(\varphi) = \{af(\varphi, w) \mid w \in (2^{Ap})^*\}$.

Example 6. Let $Ap = \{a\}$, and, for the sake of readability, let $\alpha = \{a\}$ and $\beta = \emptyset$ be the two letters of 2^{Ap}. Consider the formula $\varphi = (\mathbf{X}\alpha)\ \mathbf{U}\ (\beta \wedge \mathbf{X}\beta)$. We have $af(\varphi, \alpha) = \alpha \wedge \varphi$, $af(\varphi, \beta) = \beta \vee (\alpha \wedge \varphi) \equiv_p \beta \vee \varphi$, and $Reach(\varphi) = \{\varphi, \alpha \wedge \varphi, \beta \vee \varphi, \mathbf{tt}, \mathbf{ff}\}$.

Lemma 7. *Let* φ *be a formula, and let* $ww' \in (2^{Ap})^\omega$ *be an arbitrary word. Then* $ww' \models \varphi$ *iff* $w' \models af(\varphi, w)$.

Proof. Straightforward induction on the length of w. □

3 DRAs for simple FG-formulae

We start with formulae $\mathbf{FG}\varphi$ where φ is $\mathbf{G}$-free, i.e., contains no occurrence of $\mathbf{G}$. The main building block of our paper is a procedure to construct a DRA recognizing $\mathsf{L}(\mathbf{FG}\varphi)$. (Notice that even the formula $\mathbf{FG}a$ has no deterministic Büchi automaton.) We proceed in two steps. First we introduce Mojmir automata and construct a Mojmir automaton that clearly recognizes $\mathsf{L}(\mathbf{FG}\varphi)$. We then show how to transform Mojmir automata into equivalent DRAs.

A Mojmir automaton[2] is a deterministic automaton that, at each step, puts a fresh token in the initial state, and moves all older tokens according to the transition function. The automaton accepts if all but finitely many tokens eventually reach an accepting state.

Definition 8. *A Mojmir automaton* $\mathcal{M}$ *over an alphabet* Σ *is a tuple* (Q, i, δ, F)*, where* Q *is a set of states,* $i \in Q$ *is the initial state,* $\delta\colon Q \times \Sigma \to Q$ *is a transition function, and* $F \subseteq Q$ *is a set of accepting states satisfying* $\delta(F, \Sigma) \subseteq F$*, i.e., states reachable from final states are also final.*

The run *of* $\mathcal{M}$ *over a word* $w[0]w[1]\cdots \in (2^{Ap})^\omega$ *is the infinite sequence* $(q_0^0)(q_0^1, q_1^1)(q_0^2, q_1^2, q_2^2)\cdots$ *such that*

$$q_{token}^{step} = \begin{cases} i & \text{if } token = step, \\ \delta(q_{token}^{step-1}, w[step-1]) & \text{if } token < step \end{cases}$$

A run is accepting if for almost every token $\in \mathbb{N}$ *there exists* $step \geq token$ *such that* $q_{token}^{step} \in F$.

Notice that if two tokens reach the same state at the same time point, then from this moment on they "travel together".

The Mojmir automaton for a formula φ has formulae as states. The automaton is constructed so that, when running on a word w, the i-th token "tracks" the formula that must hold for w_i to satisfy φ. That is, after j steps the i-th token is on the formula $af(\varphi, w_{ij})$. There is only one accepting state here, namely the one propositionally equivalent to $\mathbf{tt}$. Therefore, if the i-th token reaches an accepting state, then w_i satisfies φ.

[2] Named in honour of Mojmír Křetínský, father of one of the authors

Definition 9. *Let φ be a* **G***-free formula. The Mojmir automaton for φ is* $\mathcal{M}(\varphi) = (\mathit{Reach}(\varphi), \varphi, \mathit{af}, \{\mathbf{tt}\})$.

Example 10. Figure 1 on the left shows the Mojmir automaton for the formula $\varphi = (\mathbf{X}\alpha)\ \mathbf{U}\ (\beta \wedge \mathbf{X}\beta)$ of Example 6.

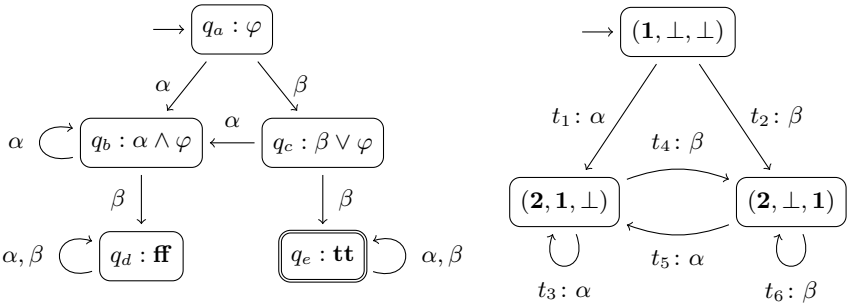


Fig. 1. A Mojmir automaton for $\varphi = (\mathbf{X}\alpha)\ \mathbf{U}\ (\beta \wedge \mathbf{X}\beta)$ and its corresponding DRA.

Since $\mathcal{M}(\varphi)$ accepts iff almost every token eventually reaches an accepting state, $\mathcal{M}(\varphi)$ accepts a word w iff $w \models \mathbf{FG}\varphi$.

Lemma 11. *Let φ be a* **G***-free formula and let w be a word. Then $w \models \varphi$ iff $\mathit{af}(\varphi, w_{0i}) = \mathbf{tt}$ for some $i \in \mathbb{N}$.*

Theorem 12. *Let φ be a* **G***-free formula. Then $\mathsf{L}(\mathcal{M}(\varphi)) = \mathsf{L}(\mathbf{FG}\varphi)$.*

3.1 From Mojmir automata to DRAs

Given a Mojmir automaton $\mathcal{M} = (Q, i, \delta, F)$ we construct an equivalent DRA. We illustrate all steps on the Mojmir automaton on the left of Figure 1. It is convenient to use shorthands q_a to q_e for state names as shown in the figure.

We label tokens with their dates of birth (token i is the token born at "day" i). Initially there is only one token, token 0, placed on the initial state i. If, say, $\delta(i, \nu) = q$, then after $\mathcal{M}$ reads ν token 0 moves to q, and token 1 appears on i.

A state of a Mojmir automaton is called a *sink* if it is not the initial state and all its outgoing transitions are self-loops. For instance, states q_d and q_e are the sinks of the automaton on the left of Figure 1. We define a *configuration* of $\mathcal{M}$ as a mapping $C \colon Q \setminus S \to 2^{\mathbb{N}}$, where S is the set of sinks and $C(q)$ is the set of (dates of birth of the) tokens that are currently at state q. Notice that we do not keep track of tokens in sinks.

We extend the transition function to configurations: $\delta(C)$ is the configuration obtained by moving all tokens of C according to δ. Let us represent a configuration C of our example by the vector $(C(q_a), C(q_b), C(q_c))$. For instance, we have $\delta((\{1\}, \emptyset, \{0\}), \alpha)) = (\{2\}, \{0, 1\}, \emptyset)$. We represent a run as an infinite sequence of configurations starting at $(\{0\}, \emptyset, \ldots, \emptyset)$. Then the run

$$(q_a) \xrightarrow{\beta} (q_c, q_a) \xrightarrow{\alpha} (q_b, q_b, q_a) \xrightarrow{\beta} (q_d, q_d, q_c, q_a) \cdots$$

is represented by

$$(0,\emptyset,\emptyset) \xrightarrow{\beta} (1,\emptyset,0) \xrightarrow{\alpha} (2,\{0,1\},\emptyset) \xrightarrow{\beta} (3,\emptyset,2)\cdots$$

where for readability we identify the singleton $\{n\}$ and the number n.

We now provide a finite abstraction of configurations. A *ranking* of Q is a partial function $r\colon Q \to \{\mathbf{1},\ldots,|\mathbf{Q}|\}$ that assigns to some states q a *rank*, and and satisfies the following conditions: (1) the initial state is ranked (i.e., $r(i)$ is defined), and sinks are unranked (i.e., if q is a sink, then $r(q)$ is undefined); (2) distinct ranked states have distinct ranks; and (3) if some state has rank $\mathbf{j}$, then some state has rank $\mathbf{k}$ for every $\mathbf{1} \leq \mathbf{k} \leq \mathbf{j}$. For $\mathbf{i} < \mathbf{j}$, we say that $\mathbf{i}$ is *older than* $\mathbf{j}$. The *abstraction* of a configuration C is the ranking $\alpha[C]$ defined as follows for every non-sink q:

- If $C(q) = \emptyset$, then q is unranked.
- If $C(q) \neq \emptyset$, then let $x_q = \min\{C(q)\}$ be the oldest token in $C(q)$ (remember that tokens are labeled by their date of birth; older tokens have smaller dates of birth). We call x_q the *senior token* of state q, and $\{x_q \in \mathbb{N} \mid q \in Q\}$ the set of *senior tokens*. We define $\alpha[C](q)$ as the *seniority rank* of x_q: if x_q is the oldest senior token, then $\alpha[C](q) = 1$; if it is the second oldest, then $\alpha[C](q) = 2$, and so on.

For instance, the senior tokens of $(2,\{0,1\},\emptyset)$ are 2 and 0, and so $\alpha(2,\{0,1\},\emptyset) = (\mathbf{2},\mathbf{1},\perp)$ (recall that sinks are unranked). Notice that there are only finitely many rankings, and so only finitely many abstract configurations. The transition function δ can be lifted to a transition function δ' on abstract configurations by defining $\delta'(\alpha[C],\nu) = \alpha[\delta(C,\nu)]$. It is easy to see that $\delta'(\alpha[C],\nu)$ can be computed directly from $\alpha[C]$ (even if C is not known). We describe how, and at the same time illustrate the construction on the abstract configuration $(\mathbf{3},\mathbf{2},\mathbf{1})$ of our running example and $\nu = \alpha$.

(i) Move the senior tokens according to δ. (Tokens with ranks $\mathbf{1},\mathbf{2},\mathbf{3}$ all move to q_b.)
(ii) If a state holds more than one token, keep only the most senior token. (Only the token with rank $\mathbf{1}$ survives.)
(iii) Recompute the seniority ranks of the remaining tokens. (In this case unnecessary; if after step (ii) we also had a token with rank $\mathbf{3}$ on state, say q_c, then its rank would be upgraded to $\mathbf{2}$.)
(iv) If there is no token on the initial state, add one with the next lowest seniority rank. (Add a token to q_a of rank $\mathbf{2}$.)

Example 13. Figure 1 shows on the right the transition system generated by the function δ' starting at the abstract configuration $(\mathbf{1},\perp,\perp)$.

It is useful to think of tokens as companies that can buy other companies: at step (2), the senior company buys all junior companies; they all get the rank of the senior company, and from this moment on travel around the automaton together with the senior company. So, at every moment in time, every token in a

non-sink state has a rank (the rank of its senior token). The rank of a token can age as it moves along the run, for two different reasons: its senior token can be bought by another senior token of an older rank, or all tokens of an older rank reach a sink. However, ranks can never get younger.

Further, observe that in any run, the tokens that never reach any sink eventually get the oldest ranks, i.e., ranks $\mathbf{1}$ to $\mathbf{i}-\mathbf{1}$ for some $i \geq 1$. We call these tokens *squatters*. Each squatter either enters the set of accepting states (and stays there by assumption on Mojmir automata) or never visits any accepting state. Now, consider a run in which almost every token succeeds. Squatters that never visit accepting states eventually stop buying other tokens, because otherwise infinitely many tokens would travel with them, and thus infinitely many tokens would never reach final states. So the run satisfies these conditions:

(1) Only finitely many tokens reach a non-accepting sink ("fail").
(2) There is a rank $\mathbf{i}$ such that
  (2.1) tokens of rank older than $\mathbf{i}$ buy other tokens in non-accepting states only finitely often, and
  (2.2) infinitely many tokens of rank $\mathbf{i}$ reach an accepting state ("succeed").

Conversely, we prove that if infinitely many tokens never succeed, then (1) or (2) does not hold. If infinitely many tokens fail, then (1) does not hold. If only finitely many tokens fail, but infinitely many tokens squat in non-accepting non-sinks, then (2) does not hold. Indeed, since the number of states is finite, infinitely many squatters get bought in non-accepting states and, since ranks can only improve, their ranks eventually stabilize. Let $\mathbf{j}-\mathbf{1}$ be the youngest rank such that infinitely many tokens stabilize with that rank. Then the squatters are exactly the tokens of ranks $\mathbf{1},\ldots,\mathbf{j}-\mathbf{1}$, and infinitely many tokens of rank $\mathbf{j}$ reach (accepting) sinks. But then (2.2) is violated for every $\mathbf{i} < \mathbf{j}$, and (2.1) is violated for every $\mathbf{i} \geq \mathbf{j}$ as, by the pigeonhole principle, there is a squatter (with rank older than $\mathbf{j}$) residing in non-accepting states and buying infintely many tokens.

So the runs in which almost every token succeeds are exactly those satisfying (1) and (2). We define a Rabin automaton having rankings as states, and accepting exactly these runs. We use a Rabin condition with pairs of sets of transitions, instead of states.[3] Let *fail* be the set of transitions that move a token into a non-accepting sink. Further, for every rank $\mathbf{j}$ let $\mathit{succeed}(\mathbf{j})$ be the set of transitions that move a token of rank $\mathbf{j}$ into an accepting state, and $\mathit{buy}(\mathbf{j})$ the set of transitions that move a token of rank older than $\mathbf{j}$ and another token into the same non-accepting state, causing one of the two to buy the other.

Example 14. Let us determine the accepting pairs of the DRA on the right of Figure 1. Since the Mojmir automaton has three non-sink states, states can have ranks $\mathbf{1},\mathbf{2},\mathbf{3}$, and so we there are three Rabin pairs. We have $\mathit{fail} = \{t_4\}$, $\mathit{buy}(\mathbf{1}) = \emptyset$, $\mathit{succeed}(\mathbf{1}) = \{t_6\}$, and $\mathit{buy}(\mathbf{2}) = \mathit{buy}(\mathbf{3}) = \{t_3, t_5\}$, $\mathit{succeed}(\mathbf{2}) = \mathit{succeed}(\mathbf{3}) = \emptyset$.

[3] It is straightforward to give an equivalent automaton with a condition on states, but transitions are better for us.

Definition 15. *Let $\mathcal{M} = (Q, i, \delta, F)$ be a Mojmir automaton with a set S of sinks. The deterministic Rabin automaton $\mathcal{R}(\mathcal{M}) = (Q_\mathcal{R}, i_\mathcal{R}, \delta_\mathcal{R}, \bigvee_{i=1}^{|Q|} P_i)$ is defined as follows:*

- *$Q_\mathcal{R}$ is the set of rankings $r\colon Q \to \{1, \ldots, |Q|\}$;*
- *$i_\mathcal{R}$ is the ranking defined only at the initial state i (and so $i_\mathcal{R}(i) = \mathbf{1}$);*
- *$\delta_\mathcal{R}(r, \nu) = \alpha[\delta(r, \nu)]$ for every ranking r and letter ν;*
- *$P_j = (\mathit{fail} \cup \mathit{buy}(\mathbf{j}), \mathit{succeed}(\mathbf{j}))$, where*

$$\begin{aligned}
\mathit{fail} &= \{(r, \nu, s) \in \delta_\mathcal{R} \mid \exists q \in Q : r(q) \in \mathbb{N} \wedge \delta(q, \nu) \in S \setminus F\} \\
\mathit{succeed}(\mathbf{j}) &= \{(r, \nu, s) \in \delta_\mathcal{R} \mid \exists q \in Q : r(q) = \mathbf{j} \wedge \delta(q, \nu) \in F\} \\
\mathit{buy}(\mathbf{j}) &= \{(r, \nu, s) \in \delta_\mathcal{R} \mid \exists q, q' \in Q : r(q) < \mathbf{j} \wedge r(q') \in \mathbb{N} \\
&\qquad\qquad \wedge \delta(q, \nu) = \delta(q', \nu) \notin F\}
\end{aligned}$$

We say that a word $w \in \mathsf{L}(\mathcal{R}(\mathcal{M}))$ is accepted at rank $\mathbf{j}$ *if P_j is the accepting pair in the run of $\mathcal{R}(\mathcal{M})$ on w with smallest index. The rank at which w is accepted is denoted by $rk(w)$.*

By the discussion above, we have

Theorem 16. *For every Mojmir automaton $\mathcal{M}$: $\mathsf{L}(\mathcal{M}) = \mathsf{L}(\mathcal{R}(\mathcal{M}))$.*

3.2 The Automaton $\mathcal{R}(\varphi)$

Given a **G**-free formula φ, we define $\mathcal{R}(\varphi) = \mathcal{R}(\mathcal{M}(\varphi))$. By Theorem 12 and Theorem 16, we have $\mathsf{L}(\mathcal{R}(\varphi)) = \mathsf{L}(\mathbf{FG}\varphi)$.

If w is accepted by $\mathcal{R}(\varphi)$ at rank $rk(w)$, then we not only know that w satisfies $\mathbf{FG}\varphi$. In order to explain exactly what else we know, we need the following definition.

Definition 17. *Let $\delta_\mathcal{R}$ be the transition function of the DRA $\mathcal{R}(\varphi)$ and let $w \in \mathsf{L}(\varphi)$ be a word. For every $j \in \mathbb{N}$, we denote by $\mathcal{F}(w_{0j})$ the conjunction of the formulae of rank younger than or equal to $rk(w)$ at the state $\delta_\mathcal{R}(i_\mathcal{R}, w_{0j})$.*

Intuitively, we also know that w_j satisfies $\mathcal{F}(w_{0j})$ for almost every index $j \in \mathbb{N}$. We set out to prove this. If $w \models \mathbf{FG}\varphi$, there is a smallest index $\mathit{ind}(w, \varphi)$ at which φ "starts to hold". For every index $j \geq \mathit{ind}(w, \varphi)$, we have $w_j \models \bigwedge_{k=\mathit{ind}(w,\varphi)}^{j} \mathit{af}(\varphi, w_{kj})$. Intuitively, this formula is the conjunction of the formulae "tracked" by the tokens of $\mathcal{M}(\varphi)$ born on days $\mathit{ind}(w, \varphi), \mathit{ind}(w, \varphi)+1, \ldots, j$. These are the "true" tokens of $\mathcal{M}(\varphi)$, that is, those that eventually reach an accepting state. We get:

Lemma 18. *Let φ be a **G**-free formula and let $w \in \mathsf{L}(\mathcal{R}(\varphi))$. Then*

(1) $\mathcal{F}(w_{0j}) \equiv \bigwedge_{k=\mathit{ind}(w,\varphi)}^{j} \mathit{af}(\varphi, w_{kj})$ for almost every $j \in \mathbb{N}$; and
(2) $w_j \models \mathcal{F}(w_{0j})$ for almost every $j \in \mathbb{N}$.

4 DRAs for arbitrary FG-formulae

We construct a DRA for an arbitrary formula **FG**-formula $\mathbf{FG}\varphi$. It suffices to construct a Mojmir automaton, and then apply the construction of Section 3.1. We show that the Mojmir automaton can be defined compositionally, as a parallel composition of Mojmir automata, one for each **G**-subformula.

Definition 19. *Given a formula φ, we denote by $\mathbb{G}(\varphi)$ the set of* **G***-subformulae of φ, i.e., the subformulae of φ of the form $\mathbf{G}\psi$.*

More precisely, for every $\mathcal{G} \subseteq \mathbb{G}(\mathbf{FG}\varphi)$ and every $\mathbf{G}\psi \in \mathcal{G}$, we construct a Mojmir automaton $\mathcal{M}(\psi, \mathcal{G})$. Automata $\mathcal{M}(\psi, \mathcal{G})$ and $\mathcal{M}(\psi, \mathcal{G}')$ for two different sets $\mathcal{G}, \mathcal{G}'$ have the same transition system, i.e., they differ only on the accepting condition. The automaton $\mathcal{M}(\psi, \mathcal{G})$ checks that $\mathbf{FG}\psi$ holds, under the assumption that $\mathbf{FG}\psi'$ holds for all the subformulae $\mathbf{G}\psi'$ of ψ that belong to $\mathcal{G}$. Circularity is avoided, because automata for ψ only rely on assumptions about proper subformulae of ψ. Loosely speaking, the Rabin automaton for $\mathbf{FG}\varphi$ is the parallel composition (or product) of the Rabin automata for the $\mathcal{M}(\psi, \mathcal{G})$ (which are independent of $\mathcal{G}$), with an acceptance condition obtained from the acceptance conditions of the $\mathcal{M}(\psi, \mathcal{G})$.

We only need to define the automaton $\mathcal{M}(\varphi, \mathcal{G})$, because the automata $\mathcal{M}(\psi, \mathcal{G})$ are defined inductively in exactly the same way. Intuitively, the automaton for $\mathcal{M}(\varphi, \mathcal{G})$ does not "track" **G**-subformulae of φ, it delegates that task to the automata for its subformulae. This is formalized with the help of the following definition.

Definition 20. *Let φ be a formula and $\nu \in 2^{Ap}$. The formula $af_{\mathbf{G}}(\varphi, \nu)$ is inductively defined as $af(\varphi, \nu)$, with only this difference:*

$$af_{\mathbf{G}}(\mathbf{G}\varphi, \nu) = \mathbf{G}\varphi \qquad \textit{(instead of } af(\mathbf{G}\varphi, \nu) = af(\varphi, \nu) \wedge \mathbf{G}\varphi\textit{).}$$

We define $Reach_{\mathbf{G}}(\varphi) = \{af_{\mathbf{G}}(\varphi, w) \mid w \in (2^{Ap})^*\}$ *(up to $\equiv_p$).*

Example 21. Let $\varphi = \psi \mathbf{U} \beta$, where $\psi = \mathbf{G}(\alpha \wedge \mathbf{X}\beta)$. We have

$$\begin{aligned} af_{\mathbf{G}}(\varphi, \alpha) &= af_{\mathbf{G}}(\psi, \alpha) \wedge \varphi \equiv_p \psi \wedge \varphi \\ af(\varphi, \alpha) &= af(\psi, \alpha) \wedge \varphi \equiv_p \beta \wedge \psi \wedge \varphi \end{aligned}$$

Definition 22. *Let φ be a formula and let $\mathcal{G} \subseteq \mathbb{G}(\varphi)$. The Mojmir automaton of φ with respect to $\mathcal{G}$ is the quadruple $\mathcal{M}(\varphi, \mathcal{G}) = (Reach_{\mathbf{G}}(\varphi), \varphi, af_{\mathbf{G}}, F_{\mathcal{G}})$, where $F_{\mathcal{G}}$ contains the formulae $\varphi' \in Reach_{\mathbf{G}}(\varphi)$ propositionally implied by $\mathcal{G}$, i.e. the formulae satisfying $\bigwedge_{\mathbf{G}\psi \in \mathcal{G}} \mathbf{G}\psi \models_p \varphi'$.*

Observe that only the set of accepting states of $\mathcal{M}(\varphi, \mathcal{G})$ depends on $\mathcal{G}$. The following lemma shows that states reachable from final states are also final.

Lemma 23. *Let φ be a formula and let $\mathcal{G} \subseteq \mathbb{G}(\varphi)$. For every $\varphi' \in Reach_{\mathbf{G}}(\varphi)$, if $\bigwedge_{\mathbf{G}\psi \in \mathcal{G}} \mathbf{G}\psi \models_p \varphi'$ then $\bigwedge_{\mathbf{G}\psi \in \mathcal{G}} \mathbf{G}\psi \models_p af_{\mathbf{G}}(\varphi', \nu)$ for every $\nu \in 2^{Ap}$.*

Proof. Follows easily from the definition of $\models_p$ and $af_{\mathbf{G}}(\mathbf{G}\psi) = \mathbf{G}\psi$.

Example 24. Let $\varphi = (\mathbf{G}\psi)\mathbf{U}\beta$, where $\psi = \alpha \wedge \mathbf{X}\beta$. We have $\mathbb{G}(\varphi) = \{\mathbf{G}\psi\}$, and so two automata $\mathcal{M}(\varphi, \emptyset)$ and $\mathcal{M}(\varphi, \{\mathbf{G}\psi\})$, whose common transition system is shown on the left of Figure 2. We have one single automaton $\mathcal{M}(\psi, \emptyset)$, shown on the right of the figure. A formula φ' is an accepting state of $\mathcal{M}(\varphi, \emptyset)$ if $\mathbf{tt} \models_p \varphi'$; and so the only accepting state of the automaton on the right is $\mathbf{tt}$. On the other hand, $\mathcal{M}(\varphi, \{\mathbf{G}\psi\})$ has both $\mathbf{G}\psi$ and $\mathbf{tt}$ as accepting states, but the only accepting state of $\mathcal{M}(\varphi, \emptyset)$ is $\mathbf{tt}$.

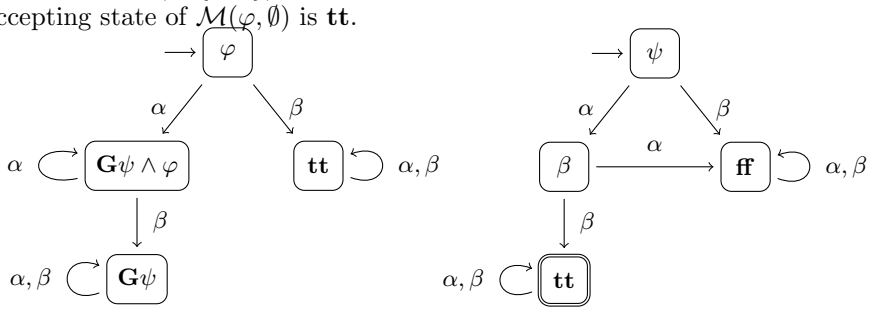


Fig. 2. Mojmir automata for $\varphi = (\mathbf{G}\psi)\ \mathbf{U}\beta$, where $\psi = \alpha \wedge \mathbf{X}\beta$.

Theorem 25. *Let φ be a formula and let w be a word. Then $w \models \mathbf{FG}\varphi$ iff there is $\mathcal{G} \subseteq \mathbb{G}(\varphi)$ such that (1) $w \in \mathsf{L}(\mathcal{M}(\varphi, \mathcal{G}))$, and (2) $w \models \mathbf{FG}\psi$ for every $\mathbf{G}\psi \in \mathcal{G}$.*

Using induction on the structure of $\mathbf{G}$-subformulae we obtain:

Theorem 26. *Let φ be a formula and let w be a word. Then $w \models \mathbf{FG}\varphi$ iff there is $\mathcal{G} \subseteq \mathbb{G}(\mathbf{FG}\varphi)$ such that $w \in \mathsf{L}(\mathcal{M}(\psi, \mathcal{G}))$ for every $\mathbf{G}\psi \in \mathcal{G}$.*

4.1 The Product Automaton

Theorem 26 allows us to construct a Rabin automaton for an arbitrary formula of the form $\mathbf{FG}\varphi$. For every $\mathbf{G}\psi \in \mathbb{G}(\mathbf{FG}\varphi)$ and every $\mathcal{G} \subseteq \mathbb{G}(\mathbf{FG}\varphi)$ let $\mathcal{R}(\psi, \mathcal{G}) = (Q_\psi, i_\psi, \delta_\psi, Acc_\psi^{\mathcal{G}})$ be the Rabin automaton obtained by applying Definition 15 to the Mojmir automaton $\mathcal{M}(\psi, \mathcal{G})$. Since $Q_\psi, i_\psi, \delta_\psi$ do not depend on $\mathcal{G}$, we define the product automaton $\mathcal{P}(\varphi)$ as

$$\mathcal{P}(\varphi) = \left(\prod_{\mathbf{G}\psi \in \mathbb{G}(\varphi)} Q_\psi,\ \prod_{\mathbf{G}\psi \in \mathbb{G}(\varphi)} \{i_\psi\},\ \prod_{\mathbf{G}\psi \in \mathbb{G}(\varphi)} \delta_\psi,\ \bigvee_{\mathcal{G} \subseteq \mathbb{G}\varphi} \bigwedge_{\mathbf{G}\psi \in \mathbb{G}(\varphi)} Acc_\psi^{\mathcal{G}} \right)$$

Since each of the $Acc_\psi^{\mathcal{G}}$ is a Rabin condition, we obtain a generalized Rabin condition. This automaton can then be transformed into an equivalent Rabin automaton [KE12]. However, as shown in [CGK13], for many applications it is better to keep it in this form. By Theorem 26 we immediately get:

Theorem 27. *Let φ be a formula and let w be a word. Then $w \models \mathbf{FG}\varphi$ iff there is $\mathcal{G} \subseteq \mathbb{G}(\mathbf{FG}\varphi)$ such that $w \in L(\mathcal{P}(\varphi))$.*

5 DRAs for Arbitrary Formulae

In order to explain the last step of our procedure, consider the following example.

Example 28. Let $\varphi = b \wedge \mathbf{X}b \wedge \mathbf{G}\psi$, where $\psi = a \wedge \mathbf{X}(b\mathbf{U}c)$ and let $Ap = \{a, b, c\}$. The Mojmir automaton $\mathcal{M}(\psi)$ is shown in the middle of Figure 3. Its corresponding Rabin automaton $\mathcal{R}(\psi)$ is shown on the right, where the state $(\mathbf{i}, \mathbf{j})$ indicates that ψ has rank $\mathbf{i}$ and $b\mathbf{U}c$ has rank $\mathbf{j}$. We have $\mathit{fail} = \{t_1, t_5, t_6, t_7, t_8\}$, $\mathit{buy}(\mathbf{1}) = \emptyset$, $\mathit{succeed}(\mathbf{1}) = \{t_4, t_7\}$ and $\mathit{buy}(\mathbf{2}) = \{t_3\}$, $\mathit{succeed}(\mathbf{2}) = \emptyset$.

Both $\mathcal{M}(\psi)$ and $\mathcal{R}(\psi)$ recognize $\mathsf{L}(\mathbf{FG}\psi)$, but not $\mathsf{L}(\mathbf{G}\psi)$. In particular, even though any word whose first letter does not contain a can be immediately rejected, $\mathcal{M}(\psi)$ fails to capture this. This is a general problem of Mojmir automata: they can never "reject (or accept) in finite time" because the acceptance condition refers to an infinite number of tokens.

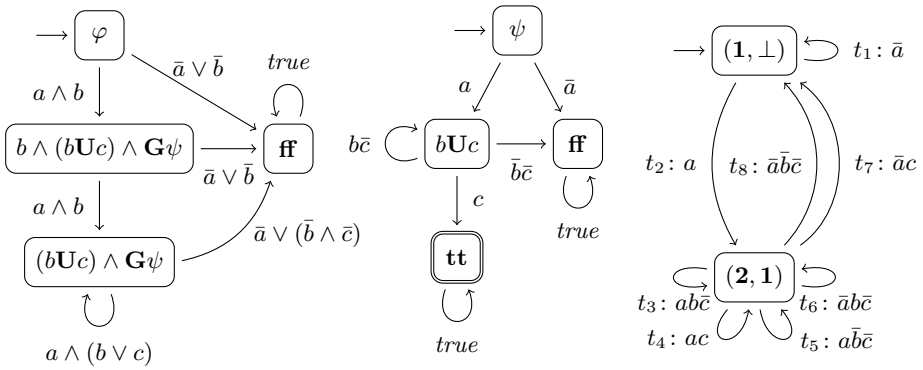


Fig. 3. Automata $\mathcal{T}(\varphi)$, $\mathcal{M}(\psi)$, and $\mathcal{R}(\psi)$ for $\varphi = b \wedge \mathbf{X}b \wedge \mathbf{G}\psi$ and $\psi = a \wedge \mathbf{X}(b\mathbf{U}c)$.

5.1 Master Transition System

The "accept/reject in finite time" problem can be solved with the help of the *master transition system* (an automaton without an accepting condition).

Definition 29. *Let φ be a formula. The* master transition system *for φ is the tuple $\mathcal{T}(\varphi) = (\mathit{Reach}(\varphi), \varphi, \mathit{af})$.*

The master transition system for the formula of Example 28 is shown on the left of Figure 3. Whenever we enter state **ff**, we have $\mathit{af}(\varphi, w) = \mathbf{ff}$ for the word w read so far, and so the run is not accepting.

Consider now the word $w = \{a, b, c\}^\omega$, which clearly satisfies φ. How do master $\mathcal{T}(\varphi)$ and slave $\mathcal{M}(\psi)$ decide together that $w \models \varphi$ holds? Intuitively, $\mathcal{M}(\psi)$ accepts, and tells the master that $w \models \mathbf{FG}\psi$ holds. The master reaches

the state $(b \,\mathbf{U}\, c) \wedge \mathbf{G}\psi$ and stays there forever. Since she knows that $\mathbf{FG}\psi$ holds, the master deduces that $w \models \varphi$ holds if $w \models \mathbf{FG}(b \,\mathbf{U}\, c)$. But where can it get this information from?

At this point the master resorts to Lemma 18: the slave $\mathcal{M}(\psi)$ (or, more precisely, its Rabin automaton $\mathcal{R}(\psi)$) not only tells the master that w satisfies $\mathbf{FG}\psi$, but also at which rank, and so that w_j satisfies $\mathcal{F}(w_{0j})$ for almost every $j \in \mathbb{N}$. In our example, during the run $w = \{a, b, c\}^\omega$, all tokens flow down the path $a \wedge \mathbf{X}(b \,\mathbf{U}\, c) \xrightarrow{a} b \,\mathbf{U}\, c \xrightarrow{c} \mathbf{tt}$ "in lockstep". No token buys any other, and all tokens of rank $\mathbf{1}$ succeed. The corresponding run of $\mathcal{R}(\psi)$ executes the sequence $t_2 t_4^\omega$ of transitions, stays in $(\mathbf{2}, \mathbf{1})$ forever, and accepts at rank $\mathbf{1}$. So we have $\mathcal{F}(w_{0j}) = (b \,\mathbf{U}\, c) \wedge \psi$ for every $j \geq 0$, and therefore the slave tells the master that $w_j \models (b \,\mathbf{U}\, c)$ for almost every $j \in \mathbb{N}$.

So in this example the information required by the master is precisely the additional information supplied by $\mathcal{M}(\psi)$. The next theorem shows that this is always the case.

Theorem 30. *Let φ be a formula and let w be a word. Let $\mathcal{G}$ be the set of formulae $\mathbf{G}\psi \in \mathbb{G}(\varphi)$ such that $w \models \mathbf{FG}\psi$. We have $w \models \varphi$ iff for almost every $i \in \mathbb{N}$:*

$$\bigwedge_{\mathbf{G}\psi \in \mathcal{G}} \big(\mathbf{G}\psi \ \wedge\ \mathcal{F}(\psi, w_{0i})\big) \models_p af(\varphi, w_{0i}) \ .$$

The automaton recognizing φ is a product of the automaton $\mathcal{P}(\varphi)$ defined in Section 4.1, and $\mathcal{T}(\varphi)$. The run of $\mathcal{P}(\varphi)$ of a word w determines the set $\mathcal{G} \subseteq \mathbb{G}(\varphi)$ such that $w \models \mathbf{FG}\psi$ iff $\psi \in \mathcal{G}$. Moreover, each component of $\mathcal{P}(\varphi)$ accepts at a certain rank, and this determines the formula $\mathcal{F}(\psi, w_{0i})$ for every $i \geq 0$ (it suffices to look at the state reached by the component of $\mathcal{P}(\varphi)$ in charge of the formula ψ). By Theorem 30, it remains to check whether eventually

$$\bigwedge_{\mathbf{G}\psi \in \mathcal{G}} \big(\mathbf{G}\psi \ \wedge\ \mathcal{F}(\psi, w_{0i})\big) \models_p af(\varphi, w_{0i})$$

holds. This is done with the help of $\mathcal{T}(\varphi)$, which "tracks" $af(\varphi, w_{0i})$. To check the property, we turn the accepting condition into a disjunction not only on the possible $\mathcal{G} \subseteq \mathbb{G}(\varphi)$, but also on the possible rankings that assign to each formula $\mathbf{G}\psi \in \mathcal{G}$ a rank. This corresponds to letting the product guess which $\mathbf{G}$-subformulae will hold, and at which rank they will be accepted. The slaves check the guess, and the master checks that it eventually only visits states implied by the guess.

5.2 The GDRA $\mathcal{A}(\varphi)$

We can now formally define the final automaton $\mathcal{A}(\varphi)$ recognizing φ. Let $\mathcal{P}(\varphi) = (Q_\mathcal{P}, i_\mathcal{P}, \delta_\mathcal{P}, Acc_\mathcal{P})$ be the product automaton described in Section 4.1, and let $\mathcal{T}(\varphi) = (Reach(\varphi), \varphi, af)$. We let

$$\mathcal{A}(\varphi) = (Reach(\varphi) \times Q_P, (\varphi, i_\mathcal{P}), af \times \delta_P, Acc)$$

where the accepting condition Acc is defined top-down as follows:

– Acc is a disjunction containing a disjunct $Acc_\pi^{\mathcal{G}}$ for each pair $(\mathcal{G}, \pi)$, where $\mathcal{G} \subseteq \mathbb{G}(\varphi)$ and π is a mapping assigning to each $\psi \in \mathcal{G}$ a rank, i.e., a number between $\mathbf{1}$ and the number of Rabin pairs of $\mathcal{R}(\varphi, \mathcal{G})$.
– The disjunct $Acc_\pi^{\mathcal{G}}$ is a conjunction of the form $Acc_\pi^{\mathcal{G}} = M_\pi^{\mathcal{G}} \wedge \bigwedge_{\psi \in \mathcal{G}} Acc_\pi(\psi)$.
– Condition $Acc_\pi(\psi)$ states that $\mathcal{R}(\psi, \mathcal{G})$ accepts with rank $\pi(\psi)$ for every $\psi \in \mathcal{G}$. It is therefore a Rabin condition with only one Rabin pair.
– Condition $M_\pi^{\mathcal{G}}$ states that $\mathcal{A}(\varphi)$ eventually stays within a subset F of states defined as follows. Let $(\varphi', r_{\psi_1}, \ldots, r_{\psi_k}) \in Reach(\varphi) \times Q_P$, where r_ψ is a ranking of the formulae of $Reach_{\mathbf{G}}(\psi)$ for every $\mathbf{G}\psi \in \mathbb{G}(\varphi)$, and let $\mathcal{F}(r_\psi)$ be the conjunction of the states of $\mathcal{R}(\psi)$ to which r_ψ assigns rank $\pi(\psi)$ or higher. Then

$$(\varphi', r_{\psi_1}, \ldots, r_{\psi_k}) \in F \quad \text{iff} \bigwedge_{\mathbf{G}\psi \in \mathcal{G}} \mathbf{G}\psi \wedge \mathcal{F}(r_\psi) \models_p \varphi' \ .$$

Notice that $M_\pi^{\mathcal{G}}$ is a co-Büchi condition, and so a Rabin condition with only one pair.

Theorem 31. *For any LTL formula φ, $\mathsf{L}(\mathcal{A}(\varphi)) = \mathsf{L}(\varphi)$.*

6 The Alternation-Free Linear-Time μ-calculus

The linear-time μ-calculus is a linear-time logic with the same expressive power as Büchi automata and DRAs (see e.g. [Var88,Dam92]. It extends propositional logic with the next operator $\mathbf{X}$, and least and greatest fixpoints. This section is addressed to readers familiar with this logic. We take as syntax

$$\varphi ::= \mathbf{tt} \mid \mathbf{ff} \mid a \mid \neg a \mid y \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathbf{X}\varphi \mid \mu x.\varphi \mid \nu x.\varphi$$

where y ranges over a set of variables. We assume that if $\sigma y.\varphi$ and $\sigma z.\psi$ are distinct subformulae of a formula, then y and z are also distinct. A formula is *alternation-free* if for every subformula $\mu y.\varphi$ ($\nu y.\varphi$) no path of the syntax tree leading from μy (νy) to y contains an occurrence of νz (μz) for some variable z. For instance, $\mu y.(a \vee \mu z.(y \vee \mathbf{X}z)$ is alternation-free, but $\nu y.\mu z((a \wedge y) \vee \mathbf{X}z)$ is not. It is well known that the alternation-free fragment is strictly more expressive than LTL and strictly less expressive than the full linear-time μ-calculus. In particular, the property "a holds at every even moment" is not expressible in LTL, but corresponds to $\nu y.(a \wedge \mathbf{XX}y)$.

Our technique extends to the alternation-free linear-time μ-calculus. We have refrained from presenting it for this more general logic because it is less well known and formulae are more difficult to read. We only need to change the definition of the functions af and $af_{\mathbf{G}}$. For the common part of the syntax (everything but the fixpoint formulae) the definition is identical. For the rest we define

$$af(\mu y.\varphi, \nu) = af(\varphi, \nu) \vee \mu y.\varphi \qquad af_{\mathbf{G}}(\mu y.\varphi, \nu) = af_{\mathbf{G}}(\varphi, \nu) \vee \mu y.\varphi$$
$$af(\nu y.\varphi, \nu) = af(\varphi, \nu) \wedge \nu y.\varphi \qquad af_{\mathbf{G}}(\nu y.\varphi, \nu) = \nu y.\varphi$$

The automaton $\mathcal{A}(\varphi)$ is a product of automata, one for every ν-subformula of φ, and a master transition system. Our constructions can be reused, and the proofs require only technical changes in the structural inductions.

7 Experimental results

We compare the performance of the following tools and methods:

(T1) ltl2dstar [Kle] implements and optimizes [KB07] Safra's construction [Saf88]. It uses LTL2BA [GO01] to obtain the non-deterministic Büchi automata (NBA) first. Other translators to NBA may also be used, such as Spot [DL13] or LTL3BA [BKRS12] and in some cases may yield better results (see [BKS13] for comparison thereof), but LTL2BA is recommended by ltl2dstar and is used this way in PRISM [KNP11].

(T2) Rabinizer [GKE12] and Rabinizer 2 [KLG13] implement a direct construction based on [KE12] for fragments LTL($\mathbf{F}, \mathbf{G}$) and LTL$_{\setminus \mathbf{GU}}$, respectively. The latter is used only on formulae not in LTL($\mathbf{F}, \mathbf{G}$).

(T3) LTL3DRA [BBKS13] which implements a construction via alternating automata, which is "inspired by [KE12]" (quoted from [BBKS13]) and performs several optimizations.

(T4) Our new construction. Notice that we produce a state space with a logical structure, which permits many optimizations; for instance, one could incorporate the suspension optimization of LTL3BA [BBDL$^+$13]. However, in our prototype implementation we use only the following optimization: In each state we only keep track of the slaves for formulae ψ that are still "relevant" for the master's state φ, i.e. $\varphi[\psi/\mathbf{tt}] \not\equiv_p \varphi[\psi/\mathbf{ff}]$. For instance, after reading $\emptyset$ in $\mathbf{GF}a \vee (b \wedge \mathbf{GF}c)$, it is no longer interesting to track if c occurs infinitely often.

Table 1 compares these four tools. For T1 and T2 we produce DRAs (although Rabinizer 2 can also produce GDRAs). For T3 and T4 we produce GDRAs with transition acceptance (tGDRAs), which can be directly used for probabilistic model checking without blow-up [CGK13]. The table shows experimental results on four sets of formulae (see the four parts of the table)

1. Formulae of the LTL($\mathbf{F}, \mathbf{G}$) fragment taken from (i) BEEM (BEnchmarks for Explicit Model checkers) [Pel07] and from [SB00] on which ltl2dstar was originally tested [KB06] (see [EK14]); and (ii) fairness-like formulae. All the formulae were used already in [KE12,BBKS13]. Our method usually achieves the same results as the optimized LTL3DRA, outperforming the first two approaches.
2. Formulae of LTL$_{\setminus \mathbf{GU}}$ taken from [KLG13] and [EH00]. They illustrate the problems of the standard approach to handle (i) $\mathbf{X}$ operators inside the scope of other temporal operators and (ii) conjunctions of liveness properties.
3. Some further formulae illustrating teh same phenomenon.
4. Some complex LTL formulae expressing "after Q until R" properties, taken from Spec Pattern [DAC99] (available at [spe]) .

Formula	T1	T2	T3	T4
$\mathbf{FG}a \vee \mathbf{GF}b$	4	4	1	1
$(\mathbf{FG}a \vee \mathbf{GF}b) \wedge (\mathbf{FG}c \vee \mathbf{GF}d)$	11324	18	1	1
$\bigwedge_{i=1}^{3}(\mathbf{GF}a_i \rightarrow \mathbf{GF}b_i)$	1 304 706	462	1	1
$(\bigwedge_{i=1}^{5} \mathbf{GF}a_i) \rightarrow \mathbf{GF}b$	?	64	1	1
$\bigwedge_{i=1}^{2}(\mathbf{GF}a_i \rightarrow \mathbf{GF}a_{i+1})$	572	11	1	1
$\bigwedge_{i=1}^{3}(\mathbf{GF}a_i \rightarrow \mathbf{GF}a_{i+1})$	290 046	52	1	1
$(\mathbf{X}(\mathbf{G}r \vee r\mathbf{U}(r \wedge s\mathbf{U}p)))\mathbf{U}(\mathbf{G}r \vee r\mathbf{U}(r \wedge s))$	18	9	8	8
$p\mathbf{U}(q \wedge \mathbf{X}(r \wedge (\mathbf{F}(s \wedge \mathbf{X}(\mathbf{F}(t \wedge \mathbf{X}(\mathbf{F}(u \wedge \mathbf{XF}v)))))))$	9	13	13	13
$(\mathbf{GF}(a \wedge \mathbf{XX}b) \vee \mathbf{FG}b) \wedge \mathbf{FG}(c \vee (\mathbf{X}a \wedge \mathbf{XX}b))$	353	73	–	12
$\mathbf{GF}(\mathbf{XXX}a \wedge \mathbf{XXXX}b) \wedge \mathbf{GF}(b \vee \mathbf{X}c) \wedge \mathbf{GF}(c \wedge \mathbf{XX}a)$	2127	169	–	16
$(\mathbf{GF}a \vee \mathbf{FG}b) \wedge (\mathbf{GF}c \vee \mathbf{FG}(d \vee \mathbf{X}e))$	18176	80	–	2
$(\mathbf{GF}(a \wedge \mathbf{XX}c) \vee \mathbf{FG}b) \wedge (\mathbf{GF}c \vee \mathbf{FG}(d \vee \mathbf{X}a \wedge \mathbf{XX}b))$	?	142	–	12
$a\mathbf{U}b \wedge (\mathbf{GF}a \vee \mathbf{FG}b) \wedge (\mathbf{GF}c \vee \mathbf{FG}d) \vee$ $\vee a\mathbf{U}c \wedge (\mathbf{GF}a \vee \mathbf{FG}d) \wedge (\mathbf{GF}c \vee \mathbf{FG}b)$	?	210	8	7
$\mathbf{FG}((a \wedge \mathbf{XX}b \wedge \mathbf{GF}b)\mathbf{U}(\mathbf{G}(\mathbf{XX}!c \vee \mathbf{XX}(a \wedge b))))$	2053	–	–	11
$\mathbf{G}(\mathbf{F}!a \wedge \mathbf{F}(b \wedge \mathbf{X}!c) \wedge \mathbf{GF}(a\mathbf{U}d)) \wedge \mathbf{GF}((\mathbf{X}d)\mathbf{U}(b \vee \mathbf{G}c))$	283	–	–	7
φ_{35} : 2 cause-1 effect precedence chain	6	–	–	6
φ_{40} : 1 cause-2 effect precedence chain	314	–	–	32
φ_{45} : 2 stimulus-1 response chain	1450	–	–	78
φ_{50} : 1 stimulus-2 response chain	28	–	–	23

Table 1. Some experimental results

All automata were constructed within a few seconds, with the exception of the larger automata generated by ltl2dstar: the automata over ten thousand states took several minutes each, and the automaton for $\bigwedge_{i=1}^{3}(\mathbf{GF}a_i \rightarrow \mathbf{GF}b_i)$ more than a day. Except for this formula, the timeout was set to five minutes and denoted by ?; not applicability of the tool to the formula is denoted by −. Additional details and more experimental results can be found in [EK14].

8 Conclusions

We have presented the first direct translation from LTL formulae to deterministic Rabin automata able to handle arbitrary formulae. We exploit the structure of the formula to compute the automaton in a compositional way, as a parallel composition of a master automaton and a number of slaves, one for each **G**-subformula. The construction generalizes previous ones for LTL fragments [KE12,GKE12,KLG13].

We have conducted a detailed experimental comparison. Our construction outperforms two-step approaches that first translate the formula into a Büchi automaton and then apply Safra's construction. Moreover, despite handling full LTL, it is at least as efficient as previous constructions for fragments. Finally, we produce a (often much smaller) generalized Rabin automaton, which can be

directly used for verification, without a further translation into a standard Rabin automaton.

The compositional approach opens the door to many possible optimizations. Since slave automata are typically very small, we can aggressively try to optimize them, knowing that each reduced state in one slave potentially leads to large savings in the final number of states of the product. So far we have only implemented the simplest optimizations, and we think there is still much room for improvement.

References

[BBDL+13] Tomáš Babiak, Thomas Badie, Alexandre Duret-Lutz, Mojmír Křetínský, and Jan Strejček. Compositional approach to suspension and other improvements to LTL translation. In *SPIN*, pages 81–98, 2013.

[BBKS13] Tomáš Babiak, František Blahoudek, Mojmír Křetínský, and Jan Strejček. Effective translation of ltl to deterministic rabin automata: Beyond the (F, G)-fragment. In *ATVA*, pages 24–39, 2013.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

[BKRS12] Tomáš Babiak, Mojmír Křetínský, Vojtěch Rehák, and Jan Strejček. LTL to Büchi automata translation: Fast and more deterministic. In *TACAS*, pages 95–109, 2012.

[BKS13] František Blahoudek, Mojmír Křetínský, and Jan Strejček. Comparison of LTL to deterministic rabin automata translators. In *LPAR*, pages 164–172, 2013.

[CGK13] Krishnendu Chatterjee, Andreas Gaiser, and Jan Křetínský. Automata with generalized Rabin pairs for probabilistic model checking and LTL synthesis. In *CAV*, pages 559–575, 2013.

[Cou99] Jean-Michel Couvreur. On-the-fly verification of linear temporal logic. In *World Congress on Formal Methods*, pages 253–271, 1999.

[DAC99] Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. Patterns in property specifications for finite-state verification. In *ICSE*, pages 411–420, 1999.

[Dam92] Mads Dam. Fixed points of büchi automata. In *FSTTCS*, pages 39–50, 1992.

[DGV99] Marco Daniele, Fausto Giunchiglia, and Moshe Y. Vardi. Improved automata generation for linear temporal logic. In *CAV*, pages 249–260, 1999.

[DL13] Alexandre Duret-Lutz. Manipulating ltl formulas using spot 1.0. In *ATVA*, pages 442–445, 2013.

[EH00] Kousha Etessami and Gerard J. Holzmann. Optimizing büchi automata. In *CONCUR*, pages 153–167, 2000.

[EK14] Javier Esparza and Jan Kretínský. From LTL to deterministic automata: A Safraless compositional approach. Technical Report abs/1402.3388, arXiv.org, 2014.

[Fri03] Carsten Fritz. Constructing Büchi automata from linear temporal logic using simulation relations for alternating büchi automata. In *CIAA*, pages 35–48, 2003.

[GKE12] Andreas Gaiser, Jan Křetínský, and Javier Esparza. Rabinizer: Small deterministic automata for LTL(F,G). In *ATVA*, pages 72–76, 2012.

[GL02] Dimitra Giannakopoulou and Flavio Lerda. From states to transitions: Improving translation of LTL formulae to Büchi automata. In *FORTE*, pages 308–326, 2002.

[GO01] Paul Gastin and Denis Oddoux. Fast LTL to Büchi automata translation. In *CAV*, volume 2102 of *LNCS*, pages 53–65. Springer, 2001. Tool accessible at http://www.lsv.ens-cachan.fr/ gastin/ltl2ba/.

[KB06] Joachim Klein and Christel Baier. Experiments with deterministic *omega*-automata for formulas of linear temporal logic. *Theor. Comput. Sci.*, 363(2):182–195, 2006.

[KB07] Joachim Klein and Christel Baier. On-the-fly stuttering in the construction of deterministic *omega* -automata. In *CIAA*, volume 4783 of *LNCS*, pages 51–61. Springer, 2007.

[KE12] Jan Křetínský and Javier Esparza. Deterministic automata for the (F,G)-fragment of LTL. In *CAV*, pages 7–22, 2012.

[Kle] Joachim Klein. ltl2dstar - LTL to deterministic Streett and Rabin automata. http://www.ltl2dstar.de/.

[KLG13] Jan Křetínský and Ruslán Ledesma-Garza. Rabinizer 2: Small deterministic automata for LTL\GU. In *ATVA*, pages 446–450, 2013.

[KNP11] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *CAV*, pages 585–591, 2011.

[Pel07] Radek Pelánek. Beem: Benchmarks for explicit model checkers. In *Proc. of SPIN Workshop*, volume 4595 of *LNCS*, pages 263–267. Springer, 2007.

[Pit06] Nir Piterman. From nondeterministic Buchi and Streett automata to deterministic parity automata. In *LICS*, pages 255–264, 2006.

[Saf88] Shmuel Safra. On the complexity of ω-automata. In *FOCS*, pages 319–327. IEEE Computer Society, 1988.

[SB00] Fabio Somenzi and Roderick Bloem. Efficient Büchi automata from LTL formulae. In *CAV*, volume 1855 of *LNCS*, pages 248–263. Springer, 2000.

[Sch09] Sven Schewe. Tighter bounds for the determinisation of büchi automata. In *FOSSACS*, pages 167–181, 2009.

[spe] Spec Patterns. Available at `http://patterns.projects.cis.ksu.edu/documentation/patterns/ltl.shtml`.

[Var88] Moshe Y. Vardi. A temporal fixpoint calculus. In *POPL*, pages 250–259, 1988.

Note on copyrights

According to the rules for publishing in LIPIcs (Leibniz International Proceedings in Informatics) with Schloss Dagstuhl Leibniz-Zentrum für Informatik GmbH, the author of the thesis is allowed to include Paper A in the thesis:

> All publication series follow the concept of OpenAccess, i.e., the articles are freely available online for the reader and the rights are retained by the author.

For more information, please see `http://www.dagstuhl.de/publikationen/`

According to the Consent to Publish in Lecture Notes in Computer Science with Springer-Verlag GmbH, the author of the thesis is allowed to include Papers B-I in the thesis:

> Author retains the right to use his/her Contribution for his/her further scientific career by including the final published paper in his/her dissertation or doctoral thesis provided acknowledgement is given to the original source of publication.

For more information, please see the copyright form electronically accessible at `ftp.springer.de/pub/tex/latex/llncs/LNCS-Springer_Copyright_Form.pdf`