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.

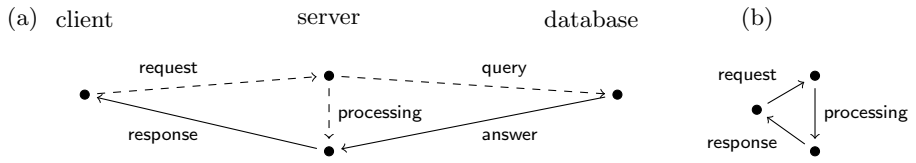

Fig. 1. An example of (a) a modal transition system (b) its implementation

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.

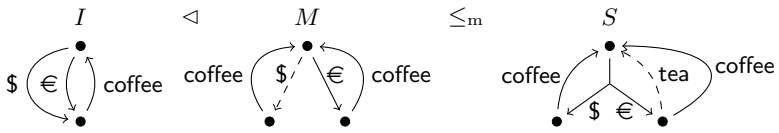

Fig. 2. An implementation I, a process M of an MTS, and a process S of a DMTS such that $I \lhd M \leq_{\mathrm{m}} S$

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].

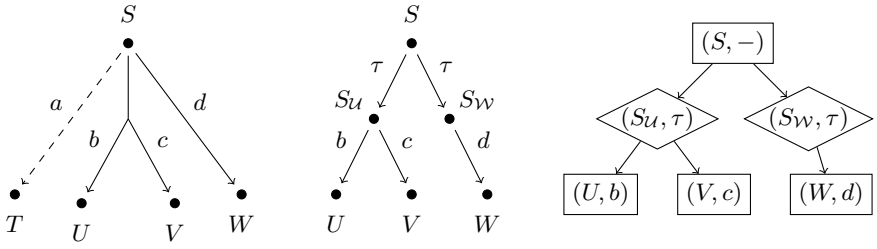

Fig. 3. Transformation from DMTS into a two player game

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.

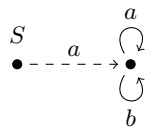

Fig. 4. No deadlock-free implementation of S satisfies $\mathbf{G}\,\mathbf{X}_a\,\mathbf{tt}$

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 $\overrightarrow{\text{MoTraS}}$, our prototype tool available

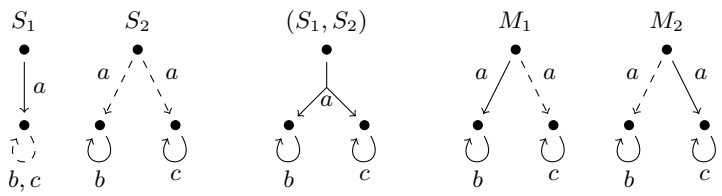

Fig. 5. MTSs S_1, S_2, their greatest lower bound (S_1, S_2), and their two maximal MTS lower bounds M_1, M_2

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)