FI MU

Faculty of Informatics
Masaryk University Brno

Disjunctive Modal Transition Systems and Generalized LTL Model Checking

by

Nikola Beneš
Ivana Černá
Jan Křetínský

FI MU Report Series **FIMU-RS-2010-12**

 November 2010

Publications in the FI MU Report Series are in general accessible via WWW:

`http://www.fi.muni.cz/reports/`

Further information can be obtained by contacting:

**Faculty of Informatics
Masaryk University
Botanická 68a
602 00 Brno
Czech Republic**

Disjunctive Modal Transition Systems and Generalized LTL Model Checking

Nikola Beneš*
Faculty of Informatics, Masaryk University
xbenes3@fi.muni.cz

Ivana Černá
Faculty of Informatics, Masaryk University
cerna@fi.muni.cz

Jan Křetínský
Faculty of Informatics, Masaryk University
Technische Universität München
jan.kretinsky@fi.muni.cz

October 8, 2010

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 show that refinement problems for DMTS are not harder than in the case of MTS. There are two main results in the paper. Firstly, we give a solution to the common implementation and specification problems lowering the complexity from EXPTIME to PTIME. Secondly, we identify a fundamental error made in previous attempts 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.

*The author has been supported by Czech Grant Agency grant no. GD102/09/H042.

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 [LNW07]. 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 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 [LT88, AHL+08a] 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 [Rac07], interface theories [RBB+09], or modal abstractions and program analysis [HJS01], to name just a few.

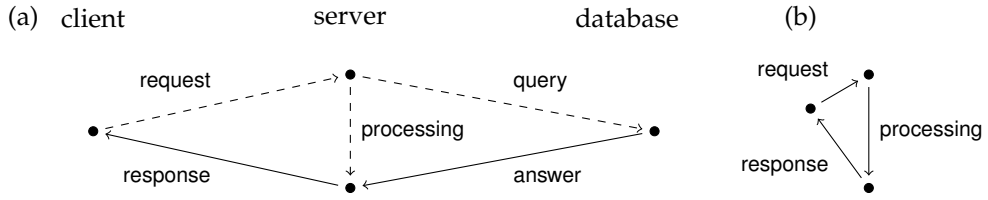

Figure 1: An example of (a) a modal transition system (b) its implementation

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. 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 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. However, MTS has no way to specify liveness in this setting. Therefore, *disjunctive modal transition systems* (DMTS) (introduced in [LX90] as solutions to process equations) are the desirable extension appropriate for specifying liveness. This has been advocated also in [FS05] 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 [FS08] where one-selecting MTS are introduced with the property that *exactly* one transition from the set must be present.

Our contribution. 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. The most important problem is deciding whether a system refines another one. The syntactically defined modal refinement (mentioned above) remains PTIME-complete. However, there is also a semantic notion of thorough refinement, which is defined by the respective sets of implementations being in inclusion. As the thorough refinement is more subtle and cannot be completely characterized by the modal refinement, it is of a higher complexity. In fact, it is EXPTIME-complete over MTS [BKLS09a], and we extend this result to DMTS. Fortunately, we show that both refinements coincide when the refined system is deterministic. In this case the complexity is only NLOGSPACE, thus allowing for easy parallelization. Therefore, using the more powerful DMTS is not more costly than using MTS.

The two main results of the paper are the following. Firstly, we solve the common implementation (CI) and, even more importantly, the common specification (CS) problems. In CI one asks whether there is an implementation that refines all specifications in a given set, i.e. whether they are consistent. In CS one computes the most general specification refining all given specifications. 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 CS. As for CI, its complexities for various MTS cases have been determined in [AHL+08b, BKLS09b]. In [JLS] CS is solved for MTS enriched with weights on transitions, however, only for the deterministic case. We show that for non-deterministic MTS there is no greatest common specification in general (there are only several maximal ones). As a matter of fact, one needs DMTS in order to express the greatest one. 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. Moreover, our construction of the common

specification is complete also with respect to the thorough refinement. Previous results on CS over DMTS [LX90] yield only an EXPTIME algorithm. Moreover, in order to decide CI using this approach one has to check for consistency, which is EXPTIME-hard. Our algorithm runs in PTIME both for CS and CI for any fixed number of specifications on input.

Secondly, 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 approach of generalized model checking of Kripke structures with partial valuations [BG00, GP09] 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 [AHL+08a] for CTL model checking MTS. Similarly, [UBC09] provides a solution to LTL model checking over deadlock-free MTS, which was implemented in the tool support for MTS [DFCU08]. However, as the second main result of this paper, we identify a fundamental 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 implementation 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 at once. MTS are perfect model for this task, as may and must transitions can capture over- and under-approximations, respectively [HJS01]. Our results thus allow for LTL model-checking of system abstractions, including counterexample generation (for refinements, see above).

Finally, we show how the model checking approach can help us getting around the fundamental problem with the composition. As there are MTSs S and T, where the

composed MTS $S \parallel T$ contains more implementations than what can be obtained by composing implementations of S and T, the composition is not complete with respect to the semantic view. Some conditions to overcome this difficulty were identified in [BKLS09b]. Here we show the general completeness of the composition with respect to the LTL formulae satisfaction.

Outline of the paper. After providing basic definitions in Section 2 and results on refinements in Section 3, the solution to the common implementation and specification problems follow in Section 4. The results on LTL model checking and its relation to the composition can be found in Section 5. Section 6 concludes and discusses future work. Due to space limitations the proofs are omitted and can be found in Appendix.

2 Definitions

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 implementations, i.e. the respective LTSs. The following definition is a slight modification of the original definition in [LX90].

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* $\nu : \mathcal{P} \to 2^{\mathrm{Ap}}$ *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 have also* $S \overset{a}{\dashrightarrow} T$*.*

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[1] 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 valuation over states.

[1]Checking consistency is an EXPTIME-complete problem. It is polynomial [LX90] 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 Appendix D.

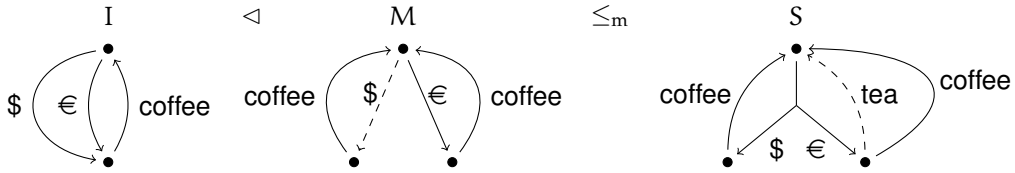

Figure 2: DMTS S, MTS M and an implementation I such that $I \lhd M \leq_m S$

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 [BK10].

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.

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 $\mathcal{T} = \{(a, T)\}$. If moreover $S \overset{a}{\dashrightarrow} T$ implies $S \xrightarrow{a} T$, then $\mathfrak{S}$ is an* implementation.

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 $S \overset{a}{\dashrightarrow} T$.*

Since in implementations the may and must transition relations coincide, implementations can be identified with labelled transition systems (with valuation).

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, since disjoint union of DMTSs is a DMTS, abusing the notation we say that I is an implementation (deterministic, MTS, etc.) meaning that the DMTS on processes reachable from I is an implementation (deterministic, MTS, etc.) as in Fig. 2. Similarly, when analyzing the complexity we assume we are given finite processes, meaning that the reachable parts of their underlying DMTSs are finite.

When refining a specification, we need to satisfy two conditions: (1) the respective refining states 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 system 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* $S \leq_m T$*, if there exists a modal refinement relation* R *with* $(S, T) \in R$*.*

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

We now define the semantics of a DMTS 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* $I \leq_m S$*. We say that* S thoroughly refines T*, denoted by* $S \leq_t T$*, if* $J \lhd S$ *implies* $J \lhd T$ *for every implementation* J*.*

3 Refinements

We investigate the relationship of the syntactic notion of modal refinement and the semantically defined notion of thorough refinement. 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. The following propositions are proved similarly as in [BKLS09b].

Proposition 3.1. *Let* S *and* T *be processes. Then* $S \leq_m T$ *implies* $S \leq_t T$*.*

Proposition 3.2. *Let* S *and* D *be processes,* D *deterministic. Then* $S \leq_t D$ *implies* $S \leq_m D$*.*

Proposition 3.3. *Deciding $\leq_m$ when restricted to the refined (i.e. right-hand-side) process being deterministic is NLOGSPACE-complete.*

Proposition 3.4. *Deciding $\leq_m$ is PTIME-complete.*

We now characterize the complexity of the thorough refinement. In order to prove the following theorem significant modifications of the approach of [BKLS09a] are needed.

Theorem 3.5. *Deciding $\leq_t$ is EXPTIME-complete.*

We show how to decide $A \not\leq_t B$ in exponential time. One might be tempted to reduce this problem to the existence of common implementation of A and the complement of B. However, (D)MTS are not closed under complementation and we have to handle the complements symbolically. We give an inductive characterization via a set $\mathsf{Avoid} \subseteq P \times 2^P$ of *avoidable pairs*. Each avoidable pair of the form $(A, \mathcal{B}) \in \mathsf{Avoid}$ means that there is an implementation of A not refining any $B \in \mathcal{B}$, thus "avoiding" all B's . In particular, $A \not\leq_t B$ if and only if there is $(A, \mathcal{B}) \in \mathsf{Avoid}$ with $B \in \mathcal{B}$.

Intuitively, every B differs from A because either A has a may transition that is not allowed in B (we put them into $\mathsf{disallowed}$), or A need not realize some must transition of B (these "forbidden" transitions are in $\mathsf{unrealized}$).

Checking the avoidability is done in two steps. Firstly, we check that none of the must transitions of A has to realize the forbidden transitions of B's. Secondly, we check that A can realize the disallowed transitions, but we need to make sure that adding these transitions does not accidentally realize any previously forbidden transitions.

Definition 3.6 (Avoidable sets)**.** *Let $(\mathcal{P}, \dashrightarrow, \longrightarrow, \nu, L)$ be a DMTS over an action alphabet Σ. The set of* avoidable pairs *is the smallest set $\mathsf{Avoid} \subseteq \mathcal{P} \times 2^{\mathcal{P}}$ such that $(A, \mathcal{B}) \in \mathsf{Avoid}$ whenever $\mathcal{B} = \emptyset$ or there are sets $\mathsf{disallowed}_a$, $\mathsf{unrealized}_a$ and $\mathsf{mustsuccA}_a \subseteq \{A' \mid A \overset{a}{\dashrightarrow} A'\}$ for every $a \in \Sigma$ such that*

- $\forall A \longrightarrow \mathcal{A}'\ \exists (a, A') \in \mathcal{A}'$ *with* $A' \in \mathsf{mustsuccA}_a$,
- $\forall B \in \mathcal{B}$ *either*
  - *(i)* $\nu(B) \neq \nu(A)$, *or*
  - *(ii) for some* $a \in \Sigma$ *we have* $B \in \mathsf{disallowed}_a$, *or*
  - *(iii) for some* $B \longrightarrow \mathcal{B}'$ *and every* $(a, B') \in \mathcal{B}'$ *we have* $B' \in \mathsf{unrealized}_a$,
- $\forall a \in \Sigma$ *both*

(1) $\forall A' \in \mathsf{mustsucc}A_a$ *we have* $(A', \mathsf{unrealized}_a) \in \mathsf{Avoid}$*, and*

(2) $\forall B \in \mathsf{disallowed}_a\ \exists A \xrightarrow{a} A_B$ *with* $(A_B, \{B' \mid B \xrightarrow{a} B'\} \cup \mathsf{unrealized}_a) \in \mathsf{Avoid}$.

Lemma 3.7. *Given a process* A *and a set* $\mathcal{B}$ *of processes, there exists an implementation* I *such that* $I \leq_m A$ *and* $I \not\leq_m B$ *for all* $B \in \mathcal{B}$ *if and only if* $(A, \mathcal{B}) \in \mathsf{Avoid}$.

The corresponding least fixed point algorithm is obvious. The complexity follows from $\mathsf{Avoid} \subseteq \mathcal{P} \times 2^{\mathcal{P}}$ and checking the condition of Definition 3.6 in exponential time.

4 Common Implementation and Specification Problems

In the following, we study the common implementation (CI) and the common specification (CS) problems. The former problem is to decide whether a given set of processes has a common implementation, the latter is to find a process that is the greatest lower bound for such set of processes w.r.t. the modal refinement.

Theorem 4.1. *The common implementation and common specification problems are EXPTIME-complete with the number of specifications being a part of the input. The problems are PTIME-complete if the number of specifications is fixed.*

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* $A_j \xrightarrow{a} B_j$ *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 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*

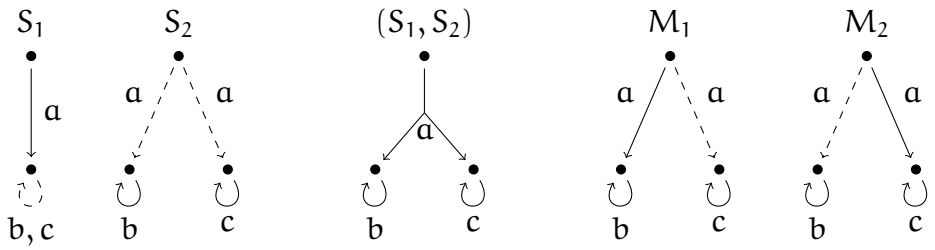

Figure 3: MTSs S_1, S_2, their greatest lower bound (S_1, S_2), and their two maximal MTS lower bounds M_1, M_2

- $\nu_{\mathsf{Con}}((A_1, \ldots, A_n)) = \nu(A_1)$,
- $(A_1, \ldots, A_n) \overset{a}{\dashrightarrow}_{\mathsf{Con}} (B_1, \ldots, B_n)$ *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* $\mathcal{B} = \{(a, (B_1, \ldots, B_n)) \mid (a, B_j) \in \mathcal{B}_j$ *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 and CS problems, 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* $S_1, \ldots, S_n$ *with regard to the* modal *as well as the* thorough *refinement*.

Note that if $S_1, \ldots, S_n$ were MTSs, $(S_1, \ldots, S_n)$ might 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. 3. However, if the MTSs are also *deterministic*, then the greatest lower bound is also a deterministic MTS [JLS].

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 be clearly 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 [AHL+08b]. The statement of Theorem 4.1 thus follows.

5 LTL Model Checking

This section discusses the model checking problem for linear temporal logic (LTL) [Pnu77] and its application on compositional verification. The following definition of state and action based LTL is equivalent to that of [CCO+04], with a slight difference in syntax.

Definition 5.1 (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*.*

Definition 5.2 (LTL semantics)**.** *Let* I *be an implementation. A run of* I *is a maximal (finite or infinite) alternating sequence of state valuations and actions* $\pi = \nu(I_0), a_1, \nu(I_1), a_2, \ldots$ *such that* $I_0 = I$ *and* $I_{i-1} \xrightarrow{a_i} I_i$ *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* i*th action of* π *as* $\ell(\pi, i) = a_i$*,* i*th state valuation of* π *as* $\pi(i) = \nu(I_i)$ *and* i*th subrun of* π *as* $\pi^i = \nu(I_i), a_i, \nu(I_{i+1}), \ldots$ *Note that these definitions only make sense when* $i < |\pi|$*. The set of all runs of* I *is denoted as* $\mathcal{R}^\infty(I)$*, the set of all infinite runs is denoted* $\mathcal{R}^\omega(I)$*.*

The semantics of LTL on runs is then defined as follows:

$$\begin{array}{lll}
\pi \models \mathbf{tt} & & \textit{always} \\
\pi \models p & \iff & p \in \pi(0) \\
\pi \models \neg\varphi & \iff & \pi \not\models \varphi \\
\pi \models \varphi \wedge \psi & \iff & \pi \models \varphi \textit{ and } \pi \models \psi \\
\pi \models \varphi \,\mathbf{U}\, \psi & \iff & \exists\, 0 \le k < |\pi| : \pi^k \models \psi \textit{ and } \forall\, 0 \le j < k : \pi^j \models \varphi \\
\pi \models \mathbf{X}\,\varphi & \iff & |\pi| > 1 \textit{ and } \pi^1 \models \varphi \\
\pi \models \mathbf{X}_a\,\varphi & \iff & |\pi| > 1,\ \ell(\pi, 0) = a \textit{ and } \pi^1 \models \varphi
\end{array}$$

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 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 no harder than the standard LTL model checking whereas deciding $\models^{\omega}_{\exists}$ is 2-EXPTIME-complete.

Theorem 5.3. *The problem of deciding* $\models^{\omega}_{\forall}$ *over DMTS is PSPACE-complete.*

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

Theorem 5.4. *The problem of deciding* $\models^{\omega}_{\exists}$ *over DMTS is 2-EXPTIME-complete.*

Sketch. We show the reduction to and from the 2-EXPTIME-complete problem of deciding existence of a winning strategy in an LTL game [PR89]. 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

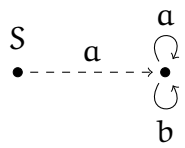

Figure 4: No deadlock-free implementation of S satisfies $\mathbf{G}\,\mathbf{X}_a\,\mathbf{tt}$

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.

The containment part of the proof constructs a Kripke structure, with states assigned to the two players, out of given DMTS. This construction bears some similarities to the construction transforming Kripke MTS into alternating tree automata in [DN05]. □

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 [AT04] 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 5.5. *The problem of deciding $\models^{\omega}_{\exists}$ over MTS is PSPACE-complete.*

Sketch. The proof is similar to the proof of Theorem 5.3, 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.

The deadlock-free approach has been studied in [UBC09] and the proposed solution was implemented in the tool MTSA [DFCU08]. 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. A simple counterexample is given in Fig. 4. Clearly, S has no deadlock-free implementation that satisfies $\mathbf{G}\,\mathbf{X}_a\,\mathbf{tt}$ (where $\mathbf{G}\,\varphi$ is the standard shorthand for $\neg(\mathbf{tt}\,\mathbf{U}\,\neg\varphi)$). Yet the method of [UBC09] as well as the tool [DFCU08] claim that such implementation exists.

Furthermore, there is no hope that the approach of [UBC09] 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 Appendix E.

Proposition 5.6. *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 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 5.7. *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.

Proposition 5.8. *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 $\mathcal{O}(|S|^{2^{\mathcal{O}(|\varphi|)}} \cdot 2^{2^{\mathcal{O}(|\varphi| \log |\varphi|)}})$. The latter upper bound is achieved by translating the LTL formula into a deterministic Rabin automaton, 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].

Table 1: Complexities of generalized LTL model checking

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

5.1 Composition

We conclude this section with an application to compositional verification. In [AHL+08a] 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 [BKLS09b]. Here we show that composition is sound and complete with respect to LTL satisfiability and validity.

We now define the straightforward extension of $\parallel$ from MTS to DMTS. Let $\Gamma \subseteq \Sigma$ be a *synchronizing alphabet*. For processes S_1 and S_2 we define the behaviour of the new composed process $S_1 \parallel S_2$ as follows. We start with the may transition relation.

- 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 $S_2 \xrightarrow{a} S_2'$.

We continue with defining the must transition relation. For a process S, let $\mathsf{Succ}(S) \subseteq 2^{\Sigma \times P}$ denote the set of all sets $\mathcal{S}$ of transitions from S such that for every $S \longrightarrow \mathcal{S}'$ there is $(a, S') \in \mathcal{S}'$ with $(a, S') \in \mathcal{S}$. The set $\mathsf{Succ}(S)$ thus consists of all possible choices of successors of S that realize all must obligations. Composing $\mathsf{Succ}(S_1)$ and $\mathsf{Succ}(S_2)$ in the same manner as may transitions above generates a new set. We denote this set $\mathsf{Succ}(S_1 \parallel S_2)$. Note that $\mathsf{Succ}(S)$ can be written equivalently in the form of a set of must transitions. Indeed, must transition relation corresponds to conjunctive normal form of the obligations while Succ corresponds to disjunctive normal form. We thus define the must transitions of the composition $S_1 \parallel S_2$ to be the corresponding counterpart to $\mathsf{Succ}(S_1 \parallel S_2)$.

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. intersection or union.

The completeness of composition with respect to LTL holds for all discussed cases: both for MTS and DMTS, both for infinite and all runs, and both universally and existentially. The results imply that although the composition is incomplete with respect to thorough refinement no new behaviour arises in the composition. In fact, the only spurious implementations are sums of legal implementations.

Theorem 5.9. *Let* S_1, S_2 *be processes,* φ *an LTL formula, 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 5.10. *Let* S_1, S_2 *be processes,* φ *an LTL formula, 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* $I_1 \parallel I_2 \models^{\star} \varphi$*.*

6 Conclusion and Future Work

We have shown that refinement problems on DMTS are not harder than for MTS and we have given a general solution to the common implementation and specification problems. We have solved the LTL model checking and synthesis problems and shown how the model checking approach helps overcoming difficulties with the 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 [AHL+08a]. Yet another modification adds weights on transitions [JLS]. 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

[AHL+08a] 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.

[AHL+08b] A. Antonik, M. Huth, K. G. Larsen, U. Nyman, and A. Wasowski. EXPTIME-complete decision problems for mixed and modal specifications. In *15th International Workshop on Expressiveness in Concurrency*, July 2008.

[AT04] R. Alur and S. La Torre. Deterministic generators and games for LTL fragments. *ACM Trans. Comput. Log.*, 5(1):1–25, 2004.

[BG00] G. Bruns and P. Godefroid. Generalized model checking: Reasoning about partial state spaces. In *CONCUR 2000*, volume 1877 of *LNCS*, pages 168–182. Springer, 2000.

[BK10] N. Beneš and J. Křetínský. Process algebra for modal transition systemses. In *Proceedings of MEMICS'10*, 2010. To appear.

[BKLS09a] N. Beneš, J. Křetínský, K. G. Larsen, and J. Srba. Checking thorough refinement on modal transition systems is EXPTIME-complete. In *Theoretical Aspects of Computing - ICTAC 2009, 6th International Colloquium. Proceedings*, volume 5684 of *LNCS*. Springer, 2009.

[BKLS09b] N. Beneš, J. Křetínský, K.G. Larsen, and J. Srba. On determinism in modal transition systems. *Theoretical Computer Science*, 410(41):4026–4043, 2009.

[CCO+04] S. Chaki, E. M. Clarke, J. Ouaknine, N. Sharygina, and N. Sinha. State/event-based software model checking. In *Proceedings of IFM'04*, volume 2999 of *LNCS*, pages 128–147. Springer-Verlag, 2004.

[DFCU08] N. D'Ippolito, D. Fischbein, M. Chechik, and S. Uchitel. MTSA: The modal transition system analyser. In *Proc. of ASE'08*, pages 475–476. IEEE, 2008.

[DN05] Dennis Dams and Kedar S. Namjoshi. Automata as abstractions. In *VMCAI*, volume 3385 of *Lecture Notes in Computer Science*, pages 216–232, 2005.

[FS05] H. Fecher and M. Steffen. Characteristic mu-calculus formulas for underspecified transition systems. *ENTCS*, 128(2):103–116, 2005.

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

[GP09] P. Godefroid and N. Piterman. LTL generalized model checking revisited. In *VMCAI*, volume 5403 of *LNCS*, pages 89–104. Springer, 2009.

[HJS01] 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.

[JLS] L. Juhl, K. G. Larsen, and J. Srba. Introducing modal transition systems with weight intervals. *Submitted*.

[LNW07] K. G. Larsen, U. Nyman, and A. Wasowski. Modeling software product lines using color-blind transition systems. *STTT*, 9(5-6):471–487, 2007.

[LT88] K. G. Larsen and B. Thomsen. A modal process logic. In *LICS*, pages 203–210. IEEE Computer Society, 1988.

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

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

[PP06] N. Piterman and A. Pnueli. Faster solution of rabin and streett games. In *In Proceedings of LICS'06*, pages 275–284. IEEE, IEEE press, 2006.

[PR89] A. Pnueli and R. Rosner. On the synthesis of an asynchronous reactive module. In *ICALP*, volume 372 of *LNSC*, pages 652–671. Springer, 1989.

[Rac07] J.-B. Raclet. Residual for component specifications. In *Proc. of the 4th International Workshop on Formal Aspects of Component Software*, 2007.

[RBB+09] 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.

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

A Appendix: Proofs from Section 3

PROPOSITION 3.1. *Let S and T be processes. Then $S \leq_m T$ implies $S \leq_t T$.*

Proof. For $I \lhd S$ we have $I \leq_m S \leq_m T$, hence $I \leq_m T$ since $\leq_m$ is obviously transitive. Thus $I \lhd T$. □

The converse of Proposition 3.1 does not hold, i.e. there are processes S, T such that $S \leq_t T$, but $\not\leq_m T$. This is already the case for MTS, see e.g. [BKLS09b]. In [BKLS09b], it is also shown that the converse holds *if the right-hand side MTS is deterministic*. We show that this is also the case with DMTS.

The gist of this result as well as the further results about complexity of deciding $\leq_t$ in the deterministic case lie in the following definition of *one-sided modal refinement* which can be shown to be equivalent to the standard modal refinement if the right-hand side process is deterministic.

Definition A.1 (One-way modal refinement)**.** *Let $(\mathcal{P}, \dashrightarrow, \longrightarrow, \nu)$ be a DMTS. Then $R \subseteq \mathcal{P} \times \mathcal{P}$ is called a* one-way 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' such that $(A', B') \in R$, and*
- *whenever $B \longrightarrow \mathcal{B}'$ then $A \longrightarrow \mathcal{A}'$ for some $\mathcal{A}'$ such that $\mathrm{act}(\mathcal{A}') \subseteq \mathrm{act}(\mathcal{B}')$ where $\mathrm{act}(\mathcal{T}) = \{a \mid (a, T) \in \mathcal{T}\}$*

We say that S one-way modally refines *T, denoted by $S \leq_m^1 T$, if there exists a one-way modal refinement relation R with $(S, T) \in R$.*

Here, the third condition is weakened. Nonetheless, $\leq_m^1$ is equivalent to $\leq_m$ if the right-hand side process is deterministic.

Lemma A.2. *Let S, D be processes, D deterministic. Then $S \leq_m^1 D$ if and only if $S \leq_m D$.*

Proof. Clearly, $S \leq_m D$ implies $S \leq_m^1 D$ as every modal refinement relation is also an one-way modal refinement relation. We thus only need to show the 'only-if' part.

Let R be an one-way modal refinement relation such that $(S, D) \in R$. We create a new modal refinement relation R' as follows:

$$R' = \{(T, E) \mid (T, E) \in R;\ E \text{ is deterministic}\}$$

Clearly, $(S, D) \in R'$. We need to show that R' satisfies the conditions of Definition 2.4. Let $(T, E) \in R'$.

- Clearly, $\nu(T) = \nu(E)$ as $(T, E) \in R$.
- If $T \xrightarrow{a} T'$ then $E \xrightarrow{a} E'$ and $(T', E') \in R$ due to the conditions of Definition A.1. Thus also $(T', E') \in R'$.
- Suppose that $E \longrightarrow \mathcal{E}'$. Due to Definition A.1, we have that $T \longrightarrow \mathcal{T}'$ with $act(\mathcal{T}') \subseteq act(\mathcal{E}')$. Let now $(a, T') \in \mathcal{T}'$. Then $T \xrightarrow{a} T'$ and due to Definition A.1, $E \xrightarrow{a} E'$ and $(T', E') \in R$. But there can be only one such E' as E is deterministic. This, together with $act(\mathcal{T}') \subseteq act(\mathcal{E}')$, implies that $(a, E') \in \mathcal{E}'$. As $(T', E') \in R$, also $(T', E') \in R'$ and thus the third condition of Definition 2.4 is satisfied. □

The power of this characterization lies in having only one co-inductive condition (instead of two in the general definition). All the other conditions can be checked locally. Therefore, it can be checked more easily, the complexity is lower in this case (see below), and it guarantees that both refinements coincide.

PROPOSITION 3.2. *Let S and D be processes, D deterministic. Then $S \leq_t D$ implies $S \leq_m D$.*

Proof. Assume that $S \leq_t D$ and that D is deterministic. We define a one-way modal refinement relation R as $R = \{(T, E) \mid T \leq_t E;\ E \text{ is deterministic}\}$. We first show that if $(T, E) \in R$, $T \xrightarrow{a} T'$ and $E \xrightarrow{a} E'$, then $(T', E') \in R$. Suppose that I' is an arbitrary implementation of T'. Then there exists some implementation $I \lhd T$ such that $I \xrightarrow{a} I'$. But as $T \leq_t E$, I is also an implementation of E. Therefore, as E is deterministic, I' is an implementation of E', thus $T' \leq_t E'$. We can now check that R indeed satisfies the conditions of Definition A.1. Let $(T, E) \in R$ and thus $T \leq_t E$.

(i) Clearly, $\nu(T) = \nu(E)$.

(ii) Suppose that $T \xrightarrow{a} T'$. Then, there exists an implementation $I \lhd T$ that has an $\xrightarrow{a}$ transition. As $T \leq_t E$, I is also an implementation of E and therefore $E \xrightarrow{a} E'$ for some E'. By the previous observation, $(T', E') \in R$.

(iii) Suppose that $E \longrightarrow \mathcal{E}'$. Then, all implementations of E are forced to have an $\xrightarrow{a}$ transition for some $a \in act(\mathcal{E}')$. As $T \leq_t E$, this implies that all implementations of T have an $\xrightarrow{a}$ transition for some $a \in act(\mathcal{E}')$. Therefore, $T \longrightarrow \mathcal{T}'$ for some $\mathcal{T}'$ such that $act(\mathcal{T}') \subseteq act(\mathcal{E}')$.

We have thus proved that $S \leq_m^1 D$. Due to Lemma A.2, $S \leq_m D$. □

PROPOSITION 3.3. *Deciding $\leq_m$ when restricted to the refined (i.e. right-hand-side) process being deterministic is NLOGSPACE-complete.*

In order to prove the above proposition, let S be an arbitrary process and let D be a deterministic one. We show that the problem of deciding $S \not\leq_m D$ is in NLOGSPACE by reduction to the NLOGSPACE-complete problem of graph reachability. This poses no problem, as the NLOGSPACE complexity class is closed under complement. The reduction relies on the fact that there is only one co-inductive condition in the characterization of the one-way modal refinement; and on Lemma A.2. That is, we actually show the problem of deciding $S \not\leq_m^1 D$.

The graph is constructed in the following way. The nodes of the graph are all pairs (T, E) where T is a process reachable from S and E is a process reachable from D. There are four kinds of nodes, three of them considered *marked*. The marked nodes have no outgoing edges.

(i) Nodes (T, E) such that $\nu(T) \neq \nu(E)$ are *marked*.

(ii) Nodes (T, E) such that $T \overset{a}{\dashrightarrow}$ and $E \overset{a}{\not\dashrightarrow}$ for some action a are *marked*.

(iii) Nodes (T, E) such that $E \longrightarrow \mathcal{E}'$ and there is no $\mathcal{T}'$ such that $T \longrightarrow \mathcal{T}'$ with $\mathrm{act}(\mathcal{T}') \subseteq \mathrm{act}(\mathcal{E}')$ are *marked*.

(iv) Nodes (T, E) which do not satisfy any of the conditions (i)–(iii) are *unmarked* and there is an edge from (T, E) to (T', E') whenever $T \overset{a}{\dashrightarrow} T'$ and $E \overset{a}{\dashrightarrow} E'$ for some action a.

An example illustrating the reduction is given in Fig. 5. The following lemma proves the correctness of the reduction and thus finishes the proof.

Lemma A.3. *We have $S \not\leq_m D$ if and only if a marked node is reachable from the node (S, D).*

Proof. For the if case, we first observe that whenever (T, E) is marked then $T \not\leq_m^1 E$ and thus $T \not\leq_m E$. This is clear from the construction. We then suppose that there is a marked node reachable from (S, D), i.e. there exists a path $(S, D) = (T_0, E_0), (T_1, E_1), \ldots, (T_n, E_n)$ where (T_n, E_n) is marked. Suppose that $S = T_0 \leq_m E_0 = D$. Then also $T_1 \leq_m E_1$ due to the second requirement of Definition 2.4 (the modal refinement relation definition). By

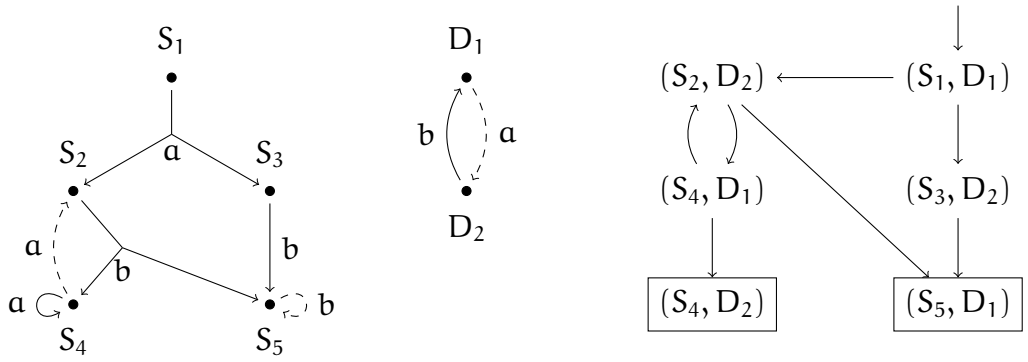

Figure 5: An example of two DMTSs and the corresponding graph reachable from (S_1, D_1) (marked nodes are in a box)

induction, also $T_2 \leq_m E_2, \ldots, T_n \leq_m E_n$. However, that last statement cannot be true, as (T_n, E_n) is marked.

For the only if case, suppose that no marked nodes are reachable from (S, D). We show a relation R that satisfies the conditions of Definition A.1. The relation R is defined as

$$R = \{(T, E) \mid (T, E) \text{ is reachable from } (S, D) \text{ in the graph}\}.$$

Clearly, $(S, D) \in R$. Now suppose that $(T, E) \in R$. That $\nu(T) = \nu(E)$ is straightforward. If $T \overset{a}{\dashrightarrow} T'$ then also, as (T, E) is unmarked, $E \overset{a}{\dashrightarrow} E'$ and moreover, $(T', E') \in R$ due to the definition of the graph. For the other condition, suppose that $E \longrightarrow \mathcal{E}'$. Then, again because (T, E) is unmarked, also $T \longrightarrow \mathcal{T}'$ for some $\mathcal{T}'$ such that $\text{act}(\mathcal{T}') \subseteq \text{act}(\mathcal{E}')$. Therefore, $S \leq_m^1 D$ and due to Lemma A.2, $S \leq_m D$. □

Since modal refinement is defined co-inductively, it can be computed in P by the standard greatest fixed-point computation, similarly as in the case of strong bisimulation. We thus obtain the following proposition.

PROPOSITION 3.4. *Deciding $\leq_m$ is PTIME-complete.*

LEMMA 3.7. *Given a process A and a set $\mathcal{B}$ of processes, there exists an implementation I such that $I \leq_m A$ and $I \not\leq_m B$ for all $B \in \mathcal{B}$ if and only if $(A, \mathcal{B}) \in \mathsf{Avoid}$.*

Proof. We prove both directions by induction.

'If' part (soundness of the construction). As Avoid is defined as the smallest set, let Avoid_0, Avoid_1, Avoid_2, ... denote the nondecreasing sequence of sets according to in which round the elements (avoidable pairs) where added to Avoid. So Avoid_0 contains

exactly all the avoidable pairs $(A, \emptyset)$, Avoid_1 contains Avoid_0 and all the avoidable pairs that were added to Avoid_0 in one iteration of the definition, etc.

We prove by induction on n that whenever $(A, \mathcal{B}) \in \mathsf{Avoid}_n$ then there exists an implementation I such that $I \lhd A$ and $I \not\lhd B$ for all $B \in \mathcal{B}$.

The base case $n = 0$ is trivial. For the induction step assume that $(A, \mathcal{B}) \in \mathsf{Avoid}_{n+1}$. Then for every $a \in \Sigma$, we have sets $\mathsf{mustsuccA}_a$, $\mathsf{disallowed}_a$ and $\mathsf{unrealized}_a$ with the properties from Definition 3.6. From the clause (1), it follows from the induction hypothesis that for every $A' \in \mathsf{mustsuccA}_a$ there exists $I_{A'}$ such that $I_{A'} \lhd A'$ and $I_{A'} \not\lhd B'$ for all $B' \in \mathsf{unrealized}_a$. Similarly, from the clause (2), it follows from the induction hypothesis that for every $B \in \mathsf{disallowed}_a$ there exists $A \overset{a}{\dashrightarrow} A_B$ and an implementation I_{A_B} such that $I_{A_B} \lhd A_B$ and $I_{A_B} \not\lhd B'$ for all B' with $B \overset{a}{\dashrightarrow} B'$ or $B' \in \mathsf{unrealized}_a$.

Now we define an implementation I witnessing the correctness of $(A, \mathcal{B}) \in \mathsf{Avoid}$. Let I be defined by $\nu(I) = \nu(A)$ and the following transitions for each $a \in \Sigma$:

1. for every $A' \in \mathsf{mustsuccA}_a$ we have $I \overset{a}{\longrightarrow} I_{A'}$, and
2. for every $B \in \mathsf{disallowed}_a$ we have $I \overset{a}{\longrightarrow} I_{A_B}$.

It is straightforward to prove that $I \lhd A$ and $I \not\lhd B$ for all $B \in \mathcal{B}$.

Let us first establish $I \lhd A$. Clearly, $\nu(I) = \nu(A)$. Assume that $A \longrightarrow \mathcal{A}'$, then there is $(a, A') \in \mathcal{A}'$ with $A' \in \mathsf{mustsuccA}_a$. Hence $I \overset{a}{\longrightarrow} I_{A'}$ will provide the match. For the other direction, first let $I \overset{a}{\dashrightarrow} I_{A'}$. Then there is $A' \in \mathsf{mustsuccA}_a$ hence $A \overset{a}{\dashrightarrow} A'$ provides the match. Second, let $I \overset{a}{\dashrightarrow} I_{A_B}$. Then $A \overset{a}{\dashrightarrow} A_B$ provides the match.

To see that $I \not\lhd B$ for $B \in \mathcal{B}$, we distinguish three cases. Either (i) $\nu(B) \neq \nu(A)$ and hence $\nu(B) \neq \nu(I)$, or (ii) $B \in \mathsf{disallowed}_a$ for some a, then $I \overset{a}{\dashrightarrow} I_{A_B}$ cannot be matched by any $B \overset{a}{\dashrightarrow} B'$, or (iii) there is $B \longrightarrow \mathcal{B}'$ and for all $a \in \Sigma$ and $(a, B') \in \mathcal{B}'$ we have $B' \in \mathsf{unrealized}_a$, thus $B \longrightarrow \mathcal{B}'$ can be matched by neither $I \overset{a}{\longrightarrow} I_{A'}$ nor $I \overset{a}{\longrightarrow} I_{A_B}$.

'*Only-if*' part (completeness of the construction). Let us define $I \lhd^n T$ if either $n = 0$ or (i) whenever $I \overset{a}{\dashrightarrow} I'$ then $T \overset{a}{\dashrightarrow} T'$ with $I' \lhd^{n-1} T'$ and (ii) whenever $T \longrightarrow \mathcal{T}'$ then $I \overset{a}{\longrightarrow} I'$ with $I' \lhd^{n-1} T'$ and $(a, T') \in \mathcal{T}'$. Hence the relation $\lhd^n$ is a natural generalization of the classical bisimulation approximants to modal refinement of DMTS by an implementation, and clearly (on finite DMTS) we have that $I \lhd T$ iff $I \lhd^n T$ for all n.

We prove by induction on n that whenever there exists an implementation I such that $I \lhd A$ and $I \not\lhd^n B$ for all $B \in \mathcal{B}$ then $(A, \mathcal{B}) \in \mathsf{Avoid}$.

As for the base case $n = 0$, it holds that $\nu(B) \neq \nu(A)$, i.e. the condition (i), for all $B \in \mathcal{B}$. Hence the choice $\mathsf{disallowed}_a = \mathsf{unrealized}_a = \emptyset$ and $\mathsf{mustsuccA}_a = \{A' \mid A \xrightarrow{a} A'\}$ for all $a \in \Sigma$ yields $(A, \mathcal{B}) \in \mathsf{Avoid}$.

For the induction step assume for some I that $I \lhd A$ and $I \not\trianglelefteq^{n+1} B$ for all $B \in \mathcal{B}$. We define the following sets of processes.

- $\mathsf{Cond}_{(i)} = \{B \in \mathcal{B} \mid \nu(B) = \nu(A)\}$,
- $\mathsf{Cond}_{(ii)}(a) = \{B \in \mathcal{B} \mid \exists I \xrightarrow{a} I' \, \forall B \xrightarrow{a} B' \; I' \not\trianglelefteq^{n} B'\}$, for every $a \in \Sigma$,
- $\mathsf{Cond}_{(iii)} = \{B \in \mathcal{B} \mid \exists B \longrightarrow \mathcal{B}'_B \; \forall (a, B') \in \mathcal{B}'_B \; \forall I \xrightarrow{a} I' \; I' \not\trianglelefteq^{n} B'\}$.

Note that $\mathsf{Cond}_{(i)} \cup \bigcup_{a \in \Sigma} \mathsf{Cond}_{(ii)}(a) \cup \mathsf{Cond}_{(iii)} = \mathcal{B}$. We set $\mathsf{disallowed}_a = \mathsf{Cond}_{(ii)}(a)$ and $\mathsf{unrealized}_a = \{B' \mid \exists B \in \mathsf{Cond}_{(iii)} \; \exists (a, B') \in \mathcal{B}'_B\}$ which satisfies the second condition of Definition 3.6. To satisfy the first condition, we set $\mathsf{mustsuccA}_a = \{A' \mid A \longrightarrow \mathcal{A}' \ni (a, A'), I \xrightarrow{a} I' \lhd A'\}$. It is now straightforward to check the third condition.

As for clause (1), for every $A' \in \mathsf{mustsuccA}_a$ there is $I \xrightarrow{a} I'$ where $I' \lhd A'$ and by definition of $\mathsf{Cond}_{(iii)}$ also $I' \not\trianglelefteq B'$ for $B' \in \mathsf{unrealized}_a$. By induction hypothesis $(A', \mathsf{unrealized}_a) \in \mathsf{Avoid}$. As for clause (2), for every $B \in \mathsf{disallowed}_a$ we have $I \xrightarrow{a} I'$ (hence $I' \lhd A'$ for some $A \xrightarrow{a} A'$) with $I' \not\trianglelefteq^{n} B'$ for all $B \xrightarrow{a} B'$ by definition of $\mathsf{Cond}_{(ii)}(a)$. Since also $I' \not\trianglelefteq^{n} B' \in \mathsf{unrealized}_a$ using definition of $\mathsf{Cond}_{(iii)}$, induction hypothesis guarantees $(A', \{B' \mid B \xrightarrow{a} B'\} \cup \mathsf{unrealized}_a) \in \mathsf{Avoid}$.

It follows that the sets $\mathsf{disallowed}_a$, $\mathsf{unrealized}_a$, and $\mathsf{mustsuccA}_a$ provide the evidence required by the definition to conclude that $(A, \mathcal{B}) \in \mathsf{Avoid}$. □

B Appendix: Proofs from Section 4

The following lemmata state the correctness of the construction given in Section 4.

Lemma B.1. *Let $S_1, \ldots, S_n$ have a common implementation. Then there is a consistency relation containing $(S_1, \ldots, S_n)$.*

Proof. We define $C = \{(A_1, \ldots, A_n) \mid \exists I : I \lhd A_1, \ldots, I \lhd A_n\}$ and prove that C is a consistency relation. The first condition of Definition 4.2 is clearly satisfied. To show the validity of the second condition, suppose that we have $(A_1, \ldots, A_n) \in C$ and that there is some i such that $A_i \longrightarrow \mathcal{B}_i$. We know that there is some I that is a common implementation of $A_1, \ldots, A_n$. Due to the must transition of A_i, there has to be some

$(a, B_i) \in \mathcal{B}_i$ and some J such that $I \xrightarrow{a} J$ with $J \lhd B_i$. However, as $I \lhd A_j$ for all j, there have to be B_j for all $j \neq i$ such that $A_j \xdashrightarrow{a} B_j$ with $J \lhd B_j$. Thus $(B_1, \ldots, B_n) \in C$ as J is their common implementation. This is then exactly the second condition from Definition 4.2. This means that C is a consistency relation. □

Lemma B.2. *Let* $T, S_1, \ldots, S_n$ *be processes such that* $T \leq_m S_i$ *for all* i. *Then* $(S_1, \ldots, S_n) \in \mathsf{Con}$ *and* $T \leq_m (S_1, \ldots, S_n)$.

Proof. The fact that $(S_1, \ldots, S_n) \in \mathsf{Con}$ comes from Lemma B.1 and the fact that in our setting, every process has an implementation. We then define the relation R as $R = \{(U, (A_1, \ldots, A_n)) \mid (A_1, \ldots, A_n) \in \mathsf{Con} \text{ and } \forall i : U \leq_m A_i\}$ and prove that R is a modal refinement relation.

- If $U \xdashrightarrow{a} V$ then due to the fact that $U \leq_m A_i$ we have that $A_i \xdashrightarrow{a} B_i$ and $V \leq_m B_i$ for all i. This means that $(B_1, \ldots, B_n) \in \mathsf{Con}$ due to Lemma B.1. Thus also $(A_1, \ldots, A_n) \xdashrightarrow{a}_{\mathsf{Con}} (B_1, \ldots, B_n)$ and $(V, (B_1, \ldots, B_n)) \in R$.
- If $(A_1, \ldots, A_n) \longrightarrow_{\mathsf{Con}} \mathcal{B}$ then there is some j such that $A_j \longrightarrow \mathcal{B}_j$ and the relation between $\mathcal{B}$ and $\mathcal{B}_j$ is as in Definition 4.3. As $U \leq_m A_j$ there has to be some $\mathcal{V}$ such that $U \longrightarrow \mathcal{V}$ and for all $(a, V) \in \mathcal{V}$ there exists some $(a, B_j) \in \mathcal{B}_j$ such that $V \leq_m B_j$. Let $(a, V) \in \mathcal{V}$ be arbitrary. We need to show the existence of $(a, (B_1, \ldots, B_n)) \in \mathcal{B}$ such that $(V, (B_1, \ldots, B_n)) \in R$. But as $U \xdashrightarrow{a} V$ and $U \leq_m A_i$ for all i, there have to exist B_i with $A_i \xdashrightarrow{a} B_i$ and $V \leq_m B_i$ for all $i \neq j$. Clearly then $(a, (B_1, \ldots, B_n)) \in \mathcal{B}$ and $(V, (B_1, \ldots, B_n)) \in R$. □

Lemma B.3. *Let* $(S_1, \ldots, S_n)$ *be a process of* $\mathfrak{S}_{\mathsf{Con}}$. *Then* $(S_1, \ldots, S_n) \leq_m S_i$ *for all* i.

Proof. Let i be arbitrary. We define a relation R_i and prove that it is a modal refinement relation. We take the obvious choice of $R_i = \{((A_1, \ldots, A_n), A_i) \mid (A_1, \ldots, A_n) \in \mathsf{Con}\}$. We now prove the conditions of Definition 2.4.

- $\nu_{\mathsf{Con}}((A_1, \ldots, A_n)) = \nu(A_1) = \nu(A_i)$ by Definition 4.2.
- If $(A_1, \ldots, A_n) \xdashrightarrow{a}_{\mathsf{Con}} (B_1, \ldots, B_n)$ then $A_i \xdashrightarrow{a} B_i$ by Definition 4.3 and $((B_1, \ldots, B_n), B_i) \in R_i$ by definition of R_i.
- If $A_i \longrightarrow \mathcal{B}_i$ then $(A_1, \ldots, A_n) \longrightarrow_{\mathsf{Con}} \mathcal{B}$ where $\mathcal{B}$ is defined as in Definition 4.3. Clearly, for all $(a, (B_1, \ldots, B_n)) \in \mathcal{B}$ there is some $(a, B_j) \in \mathcal{B}_j$ due to the definition of $\mathcal{B}$. □

The theorems 4.4 and 4.5 now follow as straightforward corollaries to the three lemmata.

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 $(S_1, \ldots, S_n) \in \mathsf{Con}$.*

Proof. The "only-if" part is the statement of Lemma B.1. The "if" part follows from Lemma B.3 and the fact that our definition of DMTS guarantees that every DMTS has an implementation. □

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 for all i, $I \lhd S_i$. Therefore, $(S_1, \ldots, S_n)$ as a process of $\mathfrak{S}_{\mathsf{Con}}$ is the greatest lower bound of $S_1, \ldots, S_n$ with regard to the* modal *as well as the* thorough *refinement.*

Proof. We prove the first statement of the theorem. All implementations of $(S_1, \ldots, S_n)$ are also implementations of all S_i due to Lemma B.3 and Proposition 3.1. Let now $I \lhd S_i$ for all i. Then $I \leq_m S_i$ for all i and due to Lemma B.2, $I \lhd (S_1, \ldots, S_n)$. Thus $I \lhd (S_1, \ldots, S_n)$ if and only if $I \lhd S_i$ for all i. This also proves that $(S_1, \ldots, S_n)$ is the greatest lower bound of all S_i with regard to the *thorough* refinement.

The fact that $(S_1, \ldots, S_n)$ is the greatest lower bound of all S_i with regard to the *modal* refinement is again a direct corollary to Lemmata B.3 and B.2. □

C Appendix: Proofs from Section 5

C.1 Derived LTL Operators

In the following, we use several derived operators. For the sake of better readability, we present their definitions in this subsection.

$$\varphi \vee \psi = \neg(\neg\varphi \wedge \neg\psi) \qquad \varphi \Rightarrow \psi = \neg\varphi \vee \psi$$

$$\mathbf{F}\,\varphi = \mathbf{tt}\,\mathbf{U}\,\varphi \qquad \mathbf{G}\,\varphi = \neg\,\mathbf{F}\neg\varphi$$

$$\mathbf{X}_A\,\varphi = \bigvee_{a \in A} \mathbf{X}_a\,\varphi \qquad \mathbf{X}_{\neg a}\,\varphi = \mathbf{X}_{\Sigma\setminus\{a\}}\,\varphi$$

Note that we measure the size of the formula as the size of the directed acyclic graph (*dag*) representation of the formula. Thus, any formula written using the derived operators is linear in the size of equivalent (unpacked) formula using only the original operators of Definition 5.1.

C.2 Proof of Theorem 5.3

THEOREM 5.3. *The problem of deciding $\models^{\omega}_{\forall}$ over DMTS is PSPACE-complete.*

Proof. First, note that PSPACE-hardness follows from PSPACE-hardness of ordinary LTL model checking. Let now S be a process of a DMTS, φ be an LTL formula. Let us define I_{max} as the implementation of S that is created by promoting all may transitions of S into must transitions. That is, we can see I_{max} as the same process as S, only instead of having $(\mathcal{P}, \dashrightarrow, \longrightarrow, \nu)$ as the underlying DMTS, I_{max} has $(\mathcal{P}, \dashrightarrow, \dashrightarrow, \nu)$. We can then make the observation that for all $I \lhd S$, $\mathcal{R}^{\omega}(I) \subseteq \mathcal{R}^{\omega}(I_{max})$. Thus, $I_{max} \models^{\omega} \varphi \iff \forall I \lhd S : I \models^{\omega} \varphi$. The problem whether $S \models^{\omega}_{\forall} \varphi$ can be thus solved by running the classical LTL model checking algorithm on the may structure of S. This not only gives us the containment in PSPACE, but also proves that deciding $\models^{\omega}_{\forall}$ for DMTS can be done in time linear in the size of the DMTS's may relation and exponential in the size of the formula. □

C.3 Proof of Theorem 5.4

THEOREM 5.4. *The problem of deciding $\models^{\omega}_{\exists}$ over DMTS is 2-EXPTIME-complete.*

We prove the theorem by showing both reductions between our problem and the problem of finding a winning strategy in an LTL game, which is known to be 2-EXPTIME-complete [PR89].

An LTL game consists of a finite Kripke structure whose states are partitioned between two players, the Protagonist and the Antagonist, and an LTL formula φ. A play consist of starting in a designated initial state and moving according to whichever player owns the current state. If the play is infinite, it thus forces an infinite run in the LTS and we can whether the play satisfies φ. The goal of the game is to determine whether the Protagonist has a strategy that ensures satisfaction of φ. If the play is finite, the Protagonist wins.

Thus, an LTL game may be seen as a special kind of DMTS over unary action alphabet, where 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. For every process A corresponding to a state owned by the Antagonist, we set $A \longrightarrow \{A'\}$ if and only if $A \dashrightarrow A'$. For every process B corresponding to a state owned by the Protagonist, we set $B \longrightarrow \{B' \mid B \dashrightarrow B'\}$. The reduction from LTL games to the problem

of deciding $\models^{\omega}_{\exists}$ over DMTS is thus straightforward. As an implementation of the above DMTS has no choice in processes corresponding to states of the Antagonist, strategies of the Protagonist correspond to implementations and vice versa. Thus, deciding $\models^{\omega}_{\exists}$ is 2-EXPTIME-hard.

To show the containment in 2-EXPTIME we modify the DMTS into the above form. We proceed in two steps. Firstly, since we are interested in $\models^{\omega}_{\exists}$, we may safely remove all may transitions that are not backed up by some must transition, i.e. we are not forced to realize them. Secondly, we transform the DMTS into a DMTS of the above form with the same implementations up to some auxiliary actions. These dummy actions can be taken care of easily. Thirdly, we transform this new DMTS into a game with only *state* atomic propositions. The label of a transition will be encoded into the valuation of its target state.

First step: We remove every $S \overset{a}{\dashrightarrow} T$ if there is no $S \longrightarrow \mathcal{T}$ with $(a, T) \in \mathcal{T}$. We thus get a new DMTS. Let S' be the process of the new DMTS that corresponds to S in the original one. The following lemma states the correctness of this step.

Lemma C.1. *There exists* $I \lhd S$ *with* $I \models^{\omega} \varphi$ *iff there exists* $I' \lhd S'$ *with* $I' \models^{\omega} \varphi$.

Proof. Clearly, $S' \leq_t S$ and thus the 'if' part of the lemma holds. For the other direction, let us take $I \lhd S$ such that $I \models^{\omega} \varphi$. We then create I' by eliminating transitions from I that are not allowed in S'. Clearly, I' only has less infinite runs than I and thus $I' \models^{\omega} \varphi$. □

Second step: Let $\mathfrak{S} = (\mathcal{P}, \dashrightarrow, \longrightarrow, \nu)$ be a DMTS with all may transitions backed up by some must transition. We define a new DMTS $\mathfrak{S}_\tau = (\mathcal{P}_\tau, \dashrightarrow_\tau, \longrightarrow_\tau, \nu_\tau)$ as follows. Every must transition is changed into a must transition leading to exactly one new state with one must transition spanning all original may successors. Formally,

- $P_\tau = \{S_\tau \mid S \in \mathcal{P}\} \cup \{(S, \mathcal{U}) \mid S \in \mathcal{P}, S \longrightarrow \mathcal{U}\}$
- for every $S \in \mathcal{P}$ and $S \longrightarrow \mathcal{U}$ we set
  - $S_\tau \overset{\tau}{\longrightarrow}_\tau (S, \mathcal{U})$, and $\nu_\tau(S_\tau) = \nu(S)$,
  - for every $(a, U) \in \mathcal{U}$ we set $(S, \mathcal{U}) \overset{a}{\dashrightarrow}_\tau U_\tau$,
    $(S, \mathcal{U}) \longrightarrow_\tau \{(a, U_\tau) \mid (S, \mathcal{U}) \overset{a}{\dashrightarrow}_\tau U_\tau\}$ and $\nu_\tau((S, \mathcal{U})) = \nu(S)$.

To handle the dummy τ steps we adapt the formula. For a formula φ, we define $T(\varphi)$ as the result of the following substitution performed on φ:

- $T(p) = p$
- $T(\neg\varphi) = \neg T(\varphi)$
- $T(\varphi \wedge \psi) = T(\varphi) \wedge T(\psi)$
- $T(\varphi \,\mathbf{U}\, \psi) = T(\varphi) \,\mathbf{U}\, T(\psi)$
- $T(\mathbf{X}\,\varphi) = \mathbf{X}_\tau \,\mathbf{X}\, T(\varphi) \vee \mathbf{X}_{\neg\tau}\, T(\varphi)$
- $T(\mathbf{X}_a\,\varphi) = \mathbf{X}_\tau \,\mathbf{X}_a\, T(\varphi) \vee \mathbf{X}_a\, T(\varphi)$

Recall (from C.1) that we measure the size of the formula as the size of its *dag* representation. In that sense, this transformation is linear in the size of φ.

Also note that the transformation preserves various LTL fragments, such as LTL(**X**), LTL(**F**), LTL(**G**), LTL(**U**), etc.

Lemma C.2. *Let $\pi = \nu_0, a_1, \nu_1, a_2, \ldots$ be an infinite run such that $a_i \neq \tau$ for all i, let $\sigma = \nu_0, \tau, \nu_0, a_1, \nu_1, \tau, \nu_1, a_2, \ldots$ be an infinite run created from π by inserting τ steps not changing state valuations at every odd position. Let φ be a formula not containing $\mathbf{X}_\tau$. Then $\pi \models \varphi$ iff $\sigma \models T(\varphi)$.*

Proof. We actually prove a stronger statement, namely that for all m, $\pi^m \models \varphi$ iff $\sigma^{2m} \models T(\varphi)$ iff $\sigma^{2m+1} \models T(\varphi)$. The proof is done by induction on the formula φ. The interesting cases are that of **U**, **X** and $\mathbf{X}_a$.

- Let $\pi^m \models \varphi \,\mathbf{U}\, \psi$. Then there is some k such that $\pi^{k+m} \models \psi$ and for all $j < k$, $\pi^{k+j} \models \varphi$. Due to the induction hypothesis, $\sigma^{2k+2m} \models T(\psi)$ and for all $j < k$, $\sigma^{2j+2m} \models T(\varphi)$ and $\sigma^{2j+2m+1} \models T(\varphi)$. Therefore, $\sigma^{2m} \models T(\varphi \,\mathbf{U}\, \psi)$ and $\sigma^{2m+1} \models T(\varphi \,\mathbf{U}\, \psi)$.
- Let $\sigma^{2m} \models T(\varphi \,\mathbf{U}\, \psi)$. Then there is some k such that $\sigma^{2m+k} \models T(\psi)$ and for all $j < k$, $\sigma^{2m+j} \models T(\varphi)$. Due to the induction hypothesis, we may assume that k is even. (If it were odd we could take $k-1$ instead of k.) This means that $\pi^{m+k/2} \models \psi$ and for all $l < k/2$, $\pi^{m+l} \models \varphi$. Thus, $\pi^m \models \varphi \,\mathbf{U}\, \psi$.
- The case with $\sigma^{2m+1} \models T(\varphi \,\mathbf{U}\, \psi)$ is similar to the previous item; we may assume that k is odd in this case.
- Let $\pi^m \models \mathbf{X}_a\,\varphi$. This means that $\pi^m = \nu_m, a, \nu_{m+1}, \ldots$ and thus $\sigma^{2m} = \nu_m, \tau, \nu_m, a, \nu_{m+1}, \ldots$ Therefore $\sigma^{2m} \models \mathbf{X}_\tau \,\mathbf{X}_a\, T(\varphi)$ and $\sigma^{2m+1} \models \mathbf{X}_a\, T(\varphi)$. Both σ^{2m} and σ^{2m+1} satisfy $T(\mathbf{X}_a\,\varphi)$.

- Let $\sigma^{2m} \models T(\mathbf{X}_a\,\varphi) = \mathbf{X}_\tau\,\mathbf{X}_a\,T(\varphi) \vee \mathbf{X}_a\,T(\varphi)$. Clearly, as σ^{2m} always starts with a τ action, it cannot satisfy $\mathbf{X}_a\,T(\varphi)$ for $a \neq \tau$ (this holds because of the premise in the lemma). Therefore, $\sigma^{2m} \models \mathbf{X}_\tau\,\mathbf{X}_a\,T(\varphi)$. This means that $\sigma^{2m} = \nu_m, \tau, \nu_m, a, \nu_{m+1}, \ldots$ and $\sigma^{2m+2} \models T(\varphi)$. Due to the induction hypothesis and the construction of σ, $\pi^m \models \mathbf{X}_a\,\varphi$.

- The case with σ^{2m+1} is again similar to the previous one. Clearly σ^{2m+1} cannot satisfy $\mathbf{X}_\tau\,\mathbf{X}_a\,T(\varphi)$ due to the fact that $a_{m+1} \neq \tau$ and has to satisfy $\mathbf{X}_a\,T(\varphi)$.

- The $\mathbf{X}$ operator can be dealt with similarly to the previous three items. □

Lemma C.3. *There exists* $I \lhd S$ *with* $I \models^\omega \varphi$ *iff there exists* $I_\tau \lhd S_\tau$ *with* $I_\tau \models^\omega T(\varphi)$.

Proof. Suppose that $I \lhd S$ and $I \models^\omega \varphi$. We create I_τ by a modification of I, inserting a τ action between every two successive actions. For every run π of I there now exists a run σ of I_τ with the structure as in Lemma C.2 and due to the same lemma $I_\tau \models^\omega T(\varphi)$. The fact that $I_\tau \lhd S_\tau$ is clear from the construction.

On the other hand, suppose that $I_\tau \lhd S_\tau$ and $I_\tau \models^\omega T(\varphi)$. Due to the construction of S_τ, for every process reachable from I_τ either all or none of its outgoing transitions are labelled with τ. Furthermore, there are no τ loops. We create I from I_τ by the following modification: For every J_τ reachable from I_τ, whenever $J_\tau \xrightarrow{\tau} K_\tau \xrightarrow{a} L_\tau$, then $J \xrightarrow{a} L$. The runs of I are thus the runs of I_τ with all τ transitions removed. Again, due to Lemma C.2, $I \models^\omega \varphi$, and that $I \lhd S$ is clear from the construction. □

Third step: We now modify the DMTS into a DMTS with only one action "•". The set of processes is changed from P into $P \times \Sigma$. Whenever $S \overset{a}{\dashrightarrow} T$ in the original DMTS, then $(S, x) \overset{\bullet}{\dashrightarrow} (T, a)$ for all x; similarly for must transitions. The valuation is then $\nu((S, a)) = \nu(S) \cup \{a\}$.

We then change the LTL formula such that all subformulae of the form $\mathbf{X}_a\,\varphi$ are changed to $\mathbf{X}(a \wedge \varphi)$. We thus get a new formula $\mathrm{act}(\varphi)$.

Lemma C.4. *Let* $\pi = \nu_0, a_1, \nu_1, a_2, \ldots$ *and* $\sigma = \nu_0 \cup \{x\}, \bullet, \nu_1 \cup \{a_1\}, \bullet, \ldots$ *where* x *is an arbitrary action. Then* $\pi \models \varphi$ *if and only if* $\sigma \models \mathrm{act}(\varphi)$.

Proof. The proof is done by induction on the formula. The only interesting case here is that of $\mathbf{X}_a\,\varphi$.

Let $\pi \models \mathbf{X}_a\,\varphi$. Then $\ell(\pi, 1) = a$ and $\pi^1 \models \varphi$ (i.e. $\pi = \nu_0, a, \nu_1, a_2, \ldots$); this means that $\sigma^1 \models a$ and $\sigma^1 \models \mathrm{act}(\varphi)$ (i.e. $\sigma = \nu_0 \cup \{x\}, \bullet, \nu_1 \cup \{a\}, \bullet, \ldots$). Therefore, $\sigma \models \mathbf{X}(a \wedge \mathrm{act}(\varphi))$. The other direction is similar. □

Lemma C.5. *There exists* $I \lhd S$ *with* $I \models^{\omega} \varphi$ *iff there exists* $I' \lhd (S, x)$ *with* $I' \models^{\omega} act(\varphi)$ *where* x *is an arbitrary action.*

Proof. Let $I \lhd S$ with $I \models^{\omega} \varphi$. We then create $I' = (I, x)$ from I using the same construction we created (S, x) from S. Clearly $I' \lhd (S, x)$ and for every run π of I there is a run σ of I' of the form described in Lemma C.4. Due to the same lemma, $I' \models^{\omega} act(\varphi)$.

On the other hand, let $I' \lhd (S, x)$ with $I' \models^{\omega} act(\varphi)$. We then create I by reversing the previous construction. That is, for every J', K' reachable from I' such that $a \in \nu(K')$ and $J' \longrightarrow K'$ we set $J \xrightarrow{a} K$, $\nu(J) = \nu(J') \setminus \Sigma$, $\nu(K) = \nu(K') \setminus \Sigma$. Again, $I \lhd S$ and for every run σ of I' we have a run π of I with the relationship as in Lemma C.4. □

We now have a DMTS that can be easily changed into an LTL game. We build the Kripke structure from the may structure of the DMTS, the states of the Kripke structure being the processes of the DMTS. Whenever a process had only singleton must transitions, we assign the corresponding state to the Antagonist. Whenever a process had only one must transition to more that one target, we assign the corresponding state to the Protagonist.

Clearly, a strategy for the Protagonist in such a game induces an implementation of the original DMTS and vice versa. A strategy is winning for formula φ if and only if its induced implementation satisfies φ. We have thus shown that the problem of deciding $\models^{\omega}_{\exists}$ is in 2-EXPTIME. Moreover, it is known that the strategy for the Protagonist requires finite history, at most doubly exponential in the size of the formula and linear in the size of the Kripke structure. This means that we can synthesise a finite implementation of a given finite DMTS satisfying φ.

C.4 Proof of Theorem 5.5

THEOREM 5.5. *The problem of deciding* $\models^{\omega}_{\exists}$ *over MTS is PSPACE-complete.*

Proof. The proof is similar to the proof of Theorem 5.3, only instead of I_{max} we now define I_{min} to be the implementation of S that is created by removing all may transitions without corresponding must transitions. (Recall that in MTS must transitions are singletons and the must transition corresponding to $S \xrightarrow{a} T$ is $S \xrightarrow{a} T$, or, $S \longrightarrow (a, T)$.) Clearly for all $I \lhd S$, $\mathcal{R}^{\omega}(I_{min}) \subseteq \mathcal{R}^{\omega}(I)$, and thus $I_{min} \models^{\omega} \varphi \iff \exists I \lhd S : I \models^{\omega} \varphi$. □

C.5 Proof of Proposition 5.6

PROPOSITION 5.6. *The problem of deciding the existence of a deadlock-free implementation of a given MTS satisfying a given LTL formula, is 2-EXPTIME-complete.*

To prove the containment in 2-EXPTIME, we show the existence of a DMTS whose implementations are exactly the deadlock-free implementations of a given MTS. The containment in 2-EXPTIME then follows from Theorem 5.4.

To prove the 2-EXPTIME-hardness, we show a reduction from finding a winning strategy in LTL games into deciding the existence of a deadlock-free implementation.

Let now $(\mathcal{P}, \dashrightarrow, \longrightarrow, \nu)$ be an MTS, $S \in \mathcal{P}$. Our first task is to create a DMTS S_D such that implementations of S_D are exactly the deadlock-free implementations of S. We first need an auxiliary definition of deadlock-free processes. Those will be the processes from $\mathcal{P}$ that have a deadlock-free implementation. Such processes can be characterised as follows. Let DF be the maximal subset of P satisfying the following:

- whenever $T \not\dashrightarrow$ then $T \notin DF$,
- if $T \xrightarrow{a} U$ for some a and $U \notin DF$ then $T \notin DF$, and
- if for all a and U such that $T \overset{a}{\dashrightarrow} U$, $U \notin DF$, then $T \notin DF$.

Clearly, DF exactly captures the deadlock-free processes and can be computed in linear time. We now define the DMTS $(\mathcal{P}_D, \dashrightarrow_D, \longrightarrow_D, \nu_D)$ as follows:

- $P_D = \{T_D \mid T \in DF\}$.
- $\nu_D(T_D) = \nu(T)$.
- Whenever $T \overset{a}{\dashrightarrow} U$ and $T, U \in DF$, then $T_D \overset{a}{\dashrightarrow}_D U_D$.
- Whenever $T \xrightarrow{a} U$ and $T \in DF$ (thus also $U \in DF$ from the construction of DF), then $T_D \longrightarrow_D \{(a, U_D)\}$.
- Whenever $T \in DF$ has no must-successors, then $T_D \longrightarrow_D \{(a, U_D) \mid T_D \overset{a}{\dashrightarrow}_D U_D\}$.

The following lemma is straightforward.

Lemma C.6. *Let S be an MTS, S_D a DMTS as constructed above. Then for all I, I is a deadlock-free implementation of S if and only if $I \lhd S_D$.*

We can now come to the hardness part of the proof. This is done similarly to the hardness part of the proof in Section C.3. An LTL game can be transformed into a MTS

over unary action alphabet, where 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. For every process A corresponding to a state owned by the Antagonist, we set $A \longrightarrow A'$ if and only if $A \dashrightarrow A'$. Processes corresponding to states owned by the Protagonist have no must transitions. Now, the strategies of the Protagonist correspond to deadlock-free implementations and vice versa. Thus follows the statement of the proposition.

(A small note: The above statement about strategies of Protagonist corresponding to deadlock-free implementations does not hold if the Kripke structure of the LTL game has deadlocks, i.e. states with no outgoing transitions. However, every such LTL game can be easily modified into a game with no deadlocks.)

C.6 Proof of Theorem 5.7

THEOREM 5.7. *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.*

We first show that the problem of finding a winning strategy in an LTL game can be reduced to the problem of deciding $\models^{\infty}_{\exists}$ over MTS (the hardness for DMTS then easily follows). The LTL game can be seen as a MTS. The processes are the states of the Kripke structure, the may structure is given by the transitions in the Kripke structure. For every process A corresponding to a state belonging to the Antagonist, we set $A \xrightarrow{a} A'$ if and only if $A \overset{a}{\dashrightarrow} A'$. Every process corresponding to a state belonging to the Protagonist has no outgoing must transitions. We then change the formula φ into $\mathbf{GXtt} \Rightarrow \varphi$. Clearly, there exists an implementation satisfying this formula if and only if there exists a winning strategy in the LTL game.

We can now show the reduction from $\models^{\infty}_{\exists}$ over DMTS into $\models^{\omega}_{\exists}$ over DMTS, thus showing the containment in 2-EXPTIME (the containment for MTS easily follows). In the following we assume that e is a new action, $e \notin \Sigma$. Let S be a process. We modify its underlying DMTS as follows. For each process T reachable from S such that $T \not\dashrightarrow$ we add a may transition $T \overset{e}{\dashrightarrow} T_e$ where T_e is a new process such that $T_e \overset{e}{\dashrightarrow} T_e$ and $\nu(T_e) = \nu(T)$.

We then change the formula φ inductively into $e(\varphi)$ as follows:

- $e(p) = p$
- $e(\neg\varphi) = \neg e(\varphi)$

- $e(\varphi \wedge \psi) = e(\varphi) \wedge e(\psi)$
- $e(\varphi \mathbf{U} \psi) = e(\varphi) \mathbf{U} e(\psi)$
- $e(\mathbf{X} \varphi) = \mathbf{X}_{\neg e} e(\varphi)$
- $e(\mathbf{X}_a \varphi) = \mathbf{X}_a e(\varphi)$

We also define the extension of π, denoted as $\mathrm{ext}(\pi)$ as identical to π if $|\pi| = \infty$, otherwise $\mathrm{ext}(\pi) = \pi(e\nu)^\omega$ where ν is the last state valuation on π.

We now have to prove several lemmata.

Lemma C.7. *Let π be a run not containing action e, let $\sigma = \mathrm{ext}(\pi)$. Let φ be an LTL formula that does not contain $\mathbf{X}_e$. Then $\pi \models \varphi$ iff $\sigma \models e(\varphi)$.*

Proof. The proof is done by induction. The cases of p, $\neg\varphi$ and $\varphi \wedge \psi$ are straightforward. The other cases are the following:

- $\pi \models \varphi \mathbf{U} \psi$. Then $\pi^k \models \psi$ and $|\pi| > k$. Thus also $\sigma^k \models e(\psi)$ and for all $j < k$, $\sigma^j \models e(\varphi)$ due to the induction hypothesis. Therefore, $\sigma \models e(\varphi \mathbf{U} \psi)$.
- $\sigma \models e(\varphi \mathbf{U} \psi)$. Then $\sigma^k \models e(\psi)$. We need to show that $k < |\pi|$. This is trivial if π is infinite, so let us suppose that π is finite. But then $\sigma^i = \sigma^{|\pi|-1}$ for all $i \geq |\pi|$, so we may assume that $k < |\pi|$ w.l.o.g. Using the induction hypothesis, we get that $\pi^k \models \psi$ and for all $j < k$, $\pi^j \models \varphi$. Thus, $\pi \models \varphi \mathbf{U} \psi$.
- $\pi \models \mathbf{X} \varphi$. Then $|\pi| > 1$ and thus $\ell(\pi, 0)$ is well defined and not equal to e. Therefore, $\sigma \models \mathbf{X}_{\neg e} e(\varphi)$.
- $\sigma \models e(\mathbf{X} \varphi) = \mathbf{X}_{\neg e} e(\varphi)$. Then $\ell(\sigma, 0) \neq e$ and thus $|\pi| > 1$. Clearly then $\pi \models \mathbf{X} \varphi$.
- $\mathbf{X}_a$ is dealt with similarly. □

The following lemma states that the above modification preserves all implementations.

Lemma C.8. *Let S be a process, S' a modification of S as stated previously. Then for each $I \lhd S$ there exists some $I' \lhd S'$ such that $\mathrm{ext}(\mathcal{R}^\infty(I)) = \mathcal{R}^\omega(I')$.*

Proof. We take I and modify it as follows. Whenever there is a deadlock J reachable from I, we add a new transition $J \xrightarrow{e} J$. Clearly, as originally $J \lhd T$, it also holds that $J \lhd T$ in the new DMTS. □

Unfortunately, the converse is not true, the new process S' may have more implementations that just those from S extended. We can, however, prove two weaker statements that suffice for $\models^{\infty}_{\exists}$ and $\models^{\infty}_{\forall}$, respectively.

Lemma C.9. *Let S be a process, S' a modification of S as stated previously, and let $I' \lhd S'$. Then there exists an implementation $I \lhd S$ such that $\mathrm{ext}(\mathcal{R}^{\infty}(I)) \subseteq \mathcal{R}^{\omega}(I')$.*

Proof. We simply remove all e transitions from reachable processes of I' and obtain thus a new implementation I. The statement of the lemma is then straightforward. □

Lemma C.10. *Let S be a process, S' a modification of S as stated previously, let $I' \lhd S'$ and let $\sigma \in \mathcal{R}^{\omega}(I')$. Then there exists and implementation I of S such that $\sigma = \mathrm{ext}(\pi)$ and $\pi \in \mathcal{R}^{\infty}(I)$.*

Proof. If σ does not contain e, then we remove all e transitions from reachable processes of I' and thus obtain I. If σ contains e, we take π as the prefix of σ before first e. We then build I by taking π and adding transitions as needed. Again, the statement is then straightforward. □

The statement of the theorem is now a corollary to all previous lemmata. We already proved the 2-EXPTIME-hardness of $\models^{\infty}_{\exists}$ for MTS. The following corollary states the containment of $\models^{\infty}_{\exists}$ for DMTS, thus establishing 2-EXPTIME-completeness of $\models^{\infty}_{\exists}$ for both MTS and DMTS.

Corollary C.11. *The problem of deciding $\models^{\infty}_{\exists}$ over DMTS is in 2-EXPTIME.*

Proof. Let S be a DMTS, φ an LTL formula. Let S' be the modification of S as described earlier. Suppose that there exists $I \lhd S$ with $I \models^{\infty} \varphi$. Due to Lemma C.8 there exists $I' \lhd S'$ with $\mathcal{R}^{\omega}(I') = \mathrm{ext}(\mathcal{R}^{\infty}(I))$. This means that $I' \models^{\infty} e(\varphi)$ due to Lemma C.7.

On the other hand, let $I' \lhd S'$ with $I' \models^{\infty} e(\varphi)$. Due to Lemma C.9, there exists $I \lhd S$ with $\mathrm{ext}(\mathcal{R}^{\infty}(I)) \subseteq \mathcal{R}^{\omega}(I')$. This means that $I \models \varphi$ again due to Lemma C.7.

We thus have a reduction from deciding $\models^{\infty}_{\exists}$ to deciding $\models^{\omega}_{\exists}$. Therefore, deciding $\models^{\infty}_{\exists}$ is in 2-EXPTIME. □

The last corollary establishes the containment of $\models^{\infty}_{\forall}$ for DMTS in PSPACE. The PSPACE-hardness follows straightforwardly from the PSPACE-hardness of ordinary LTL model checking.

Corollary C.12. *The problem of deciding $\models^{\infty}_{\forall}$ over DMTS is in PSPACE.*

Proof. Let S be a DMTS, φ an LTL formula, let S' be the modification of S as described earlier. Suppose that for all $I \lhd S$, $I \models^{\infty} \varphi$. Take an arbitrary $I' \lhd S'$ such that $\mathcal{R}^{\omega}(I') \neq \emptyset$ (such I' has to exist due to the construction) and an arbitrary $\sigma \in \mathcal{R}^{\omega}(I')$. Due to Lemma C.10, there exists some implementation I and a run $\pi \in \mathcal{R}^{\infty}(I)$ such that $\sigma = \mathrm{ext}(\pi)$. But as $I \models^{\infty} \varphi$, also $\pi \models \varphi$ and thus $\sigma \models e(\varphi)$ due to Lemma C.7.

On the other hand, suppose that for all $I' \lhd S'$, $I' \models^{\omega} e(\varphi)$. Take an arbitrary $I \lhd S$. Due to Lemma C.8, there exists some $I' \lhd S'$ such that $\mathcal{R}^{\omega}(I') = \mathrm{ext}(\mathcal{R}^{\infty}(I))$. Therefore, $I \models^{\infty} \varphi$ due to Lemma C.7.

We thus have a reduction from deciding $\models^{\infty}_{\forall}$ to deciding $\models^{\omega}_{\forall}$. Therefore, deciding $\models^{\infty}_{\forall}$ is in PSPACE. □

This concludes the proof of Theorem 5.7.

C.7 Proof of Proposition 5.8

An LTL formula φ is called *purely state-based* if it does not contain the $\mathbf{X}_a$ operator. A formula is called *purely action-based* if it does not contain atomic propositions. (Recall that **tt** is a part of the syntax, so eliminating atomic propositions does make sense, as we can still use $\mathbf{X}_a$ **tt**.)

PROPOSITION 5.8. *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 proof of the proposition is done by reduction from the general problem for LTL into that for state- or action-based LTL. The first reduction (encoding actions into states) has already been done as a part of the proof of Theorem 5.4, see Lemma C.4 and preceding construction.

We can thus continue with the second reduction, encoding state valuations into actions. Note that the straightforward reduction from states into actions (i.e. simply moving the whole valuation onto outgoing transitions) is not a polynomial one, as then $|\Sigma| \geq |2^{\mathrm{Ap}}|$. We thus present a different reduction.

Let $(\mathcal{P}, \dashrightarrow, \longrightarrow, \nu)$ be the original DMTS. Let Ap be the finite set of atomic propositions with an arbitrary ordering, i.e. $\mathrm{Ap} = \{p_1, p_2, \ldots, p_n\}$. Also, assume that 0, 1, and $\triangleright$ are new symbols not in Σ. The new DMTS is then $(\mathcal{P} \times \{0, \ldots, n, \triangleright\}, \dashrightarrow, \longrightarrow, \nu')$ over an alphabet $\Sigma' = \Sigma \cup \{0, 1, \triangleright\}$, where

- $\nu'(X) = \emptyset$ for all X,

- $(S, \triangleright) \xrightarrow{\triangleright} (S, 0)$ for all $S \in \mathcal{P}$,
- $(S, i) \xrightarrow{1} (S, i+1)$ for all $S \in \mathcal{P}$ such that $p_{i+1} \in \nu(S)$,
- $(S, i) \xrightarrow{0} (S, i+1)$ for all $S \in \mathcal{P}$ such that $p_{i+1} \notin \nu(S)$,
- $(S, n) \dashrightarrow^{a} (T, 0)$ whenever $S \dashrightarrow^{a} T$, and
- $(S, n) \longrightarrow \{(a, (T, 0)) \mid (a, T) \in \mathcal{T}\}$ for all $\mathcal{T}$ such that $S \longrightarrow \mathcal{T}$.

Let now $\Gamma = \Sigma \cup \{\triangleright\} = \Sigma' \setminus \{0, 1\}$. For every LTL formula φ, we define a new formula $A(\varphi)$ inductively as follows:

- $A(p_i) = \mathbf{X}^i \mathbf{X}_1 \mathbf{tt}$
- $A(\neg\varphi) = \neg A(\varphi)$
- $A(\varphi \wedge \psi) = A(\varphi) \wedge A(\psi)$
- $A(\varphi \mathbf{U} \psi) = (\mathbf{X}_\Gamma \mathbf{tt} \Rightarrow A(\varphi)) \mathbf{U} (\mathbf{X}_\Gamma \mathbf{tt} \wedge A(\psi))$
- $A(\mathbf{X} \varphi) = \mathbf{X}^{n+1} A(\varphi)$
- $A(\mathbf{X}_a \varphi) = \mathbf{X}^{n+1}(\mathbf{X}_a \mathbf{tt} \wedge A(\varphi))$

Here, we use the notation $\mathbf{X}^1 \varphi = \mathbf{X} \varphi$, $\mathbf{X}^{k+1} \varphi = \mathbf{X}^k \mathbf{X} \varphi$ for $k > 0$.

Let π be an infinite run $\nu_0, a_1, \nu_1, a_2, \ldots$. We create a run σ as follows (all valuations are $\emptyset$ and are omitted):

$$\sigma = \triangleright, [p_1 \in \nu_0], [p_2 \in \nu_0], \ldots, [p_n \in \nu_0], a_1, [p_1 \in \nu_1], [p_2 \in \nu_1], \ldots, [p_n \in \nu_1], a_2, \ldots$$

where $[p_i \in \nu_j] = 1$ if $p_i \in \nu_j$, and 0 otherwise.

Lemma C.13. *Let π, σ be as just stated. Let φ be an LTL formula, $A(\varphi)$ its modification as in previous. Then $\pi \models \varphi$ iff $\sigma \models A(\varphi)$.*

Proof. We first observe that $\sigma^s \models \mathbf{X}_\Gamma \mathbf{tt}$ iff s is a multiple of $(n+1)$ as symbols from Γ only appear on σ every $(n+1)$ steps.

We then prove that $\pi^m \models \varphi$ iff $\sigma^{m(n+1)} \models A(\varphi)$ by induction.

- $\pi^m \models p_i$ iff $\ell(\sigma, m(n+1) + i + 1) = 1$, and that holds iff $\sigma^{m(n+1)} \models A(p_i)$.
- The cases of $\neg\varphi$ and $\varphi \wedge \psi$ are evident.

- $\pi^m \models \varphi \,\mathbf{U}\, \psi$. Then there is some k with $\pi^{m+k} \models \psi$ and for all $j < k$, $\pi^{m+j} \models \varphi$. Due to the induction hypothesis, $\sigma^{(m+k)(n+1)} \models A(\psi)$ and for all $j < k$, $\sigma^{(m+j)(n+1)} \models A(\varphi)$. Clearly, $\sigma^{(m+k)(n+1)} \models \mathbf{X}_\Gamma\, \mathbf{tt} \wedge A(\psi)$ due to the earlier observation. That $\sigma^l \models \mathbf{X}_\Gamma\, \mathbf{tt} \Rightarrow A(\varphi)$ for all $m(n+1) \leq l < (m+k)(n+1)$ also follows from the earlier observation, as for all multiples of $(n+1)$ from $m(n+1)$ to $(m+k-1)(n+1)$, $A(\varphi)$ holds and for all other subruns, $\mathbf{X}_\Gamma\, \mathbf{tt}$ does not hold, thus the implication holds trivially. Altogether $\sigma^{m(n+1)} \models (\mathbf{X}_\Gamma\, \mathbf{tt} \Rightarrow A(\varphi))\, \mathbf{U}\, (\mathbf{X}_\Gamma\, \mathbf{tt} \wedge A(\psi)) = A(\varphi \,\mathbf{U}\, \psi)$.

- $\sigma^{m(n+1)} \models A(\varphi \,\mathbf{U}\, \psi) = (\mathbf{X}_\Gamma\, \mathbf{tt} \Rightarrow A(\varphi))\, \mathbf{U}\, (\mathbf{X}_\Gamma\, \mathbf{tt} \wedge A(\psi))$. Then there is some k with $\sigma^{m(n+1)+k} \models (\mathbf{X}_\Gamma\, \mathbf{tt} \wedge A(\psi))$ and for all $j < k$, $\sigma^{m(n+1)+j} \models (\mathbf{X}_\Gamma\, \mathbf{tt} \Rightarrow A(\varphi))$. Clearly k has to be a multiple of $(n+1)$ as otherwise $\mathbf{X}_\Gamma\, \mathbf{tt}$ would not hold for $\sigma^{m(n+1)+k}$. Therefore $k = h(n+1)$ and $\pi^{m+h} \models \psi$. Clearly also $\pi^{m+l} \models \varphi$ for all $l < h$ as in $\sigma^{(m+l)(n+1)}$, $\mathbf{X}_\Gamma\, \mathbf{tt}$ holds and thus also $A(\varphi)$ has to hold.

- $\pi^m \models \mathbf{X}\, \varphi$ iff $\pi^{m+1} \models \varphi$ iff $\sigma^{(m+1)(n+1)} \models A(\varphi)$ iff $\sigma^{m(n+1)} \models \mathbf{X}^{n+1}\, A(\varphi)$.

- $\pi^m \models \mathbf{X}_a\, \varphi$ iff $\pi^{m+1} \models \varphi$ and $\ell(\pi, m+1) = a$ iff $\sigma^{(m+1)(n+1)} \models A(\varphi)$ and $\ell(\sigma, (m+1)(n+1)) = a$ iff $\sigma^{m(n+1)} \models \mathbf{X}^{n+1}\, A(\varphi)$ and $\sigma^{m(n+1)} \models \mathbf{X}^{n+1}\, \mathbf{X}_a\, \mathbf{tt}$ iff $\sigma^{m(n+1)} \models \mathbf{X}^{n+1}(\mathbf{X}_a\, \mathbf{tt} \wedge A(\varphi))$. □

We now need to prove that the construction preserves implementations.

Lemma C.14. *Let S be a process, $(S, \triangleright)$ a new process as in previous. Then the implementations of S and $(S, \triangleright)$ have the same runs, modulo the transformation above. Formally, for all $I \lhd S$ there exists $(I, \triangleright) \lhd (S, \triangleright)$ such that every run π of I corresponds to a run σ of $(I, \triangleright)$ as constructed above, and similarly, every run σ of $(I, \triangleright)$ corresponds to a run π of I; and for all $(I, \triangleright) \lhd (S, \triangleright)$ exists $I \lhd S$ with the same property.*

Proof. Let $I \lhd S$. We can then apply the transformation to I and thus straightforwardly obtain an implementation of $(S, \triangleright)$.

The other direction is more involved, as an implementation I of $(S, \triangleright)$ may branch arbitrarily before coming into a process that refines (S, n). Nonetheless, all these branches have to be the same. We may thus modify I into I' as follows: $I' \xrightarrow{b_1} I'_1 \xrightarrow{b_2} \cdots \xrightarrow{b_n} I'_n$ where I'_n is the sum of all implementations of I reachable in exactly n steps. We may then apply the transformation in reverse direction and the lemma easily follows. □

This concludes the proof of Proposition 5.8.

C.8 Proofs from Section 5.1

Recall that a composition has a may transition under a synchronizing action (an action from Γ) if both components have one, and a may transition under a non-synchronizing action (an action from $\Sigma \setminus \Gamma$) if at least one component has one. The must transition relation forces exactly the options induced by composing possible implementations.

The following lemma proves the soundness of the composition.

Lemma C.15. *Let* $I_1 \lhd S_1$ *and* $I_2 \lhd S_2$. *Then* $I_1 \parallel I_2 \lhd S_1 \parallel S_2$.

Proof. This lemma essentially amounts to completeness of the may transition relation and soundness of the must transition relation in the composition.

We prove that $R = \{(J_1 \parallel J_2, T_1 \parallel T_2) \mid J_1 \leq_m T_1,\ J_2 \leq_m T_2\}$ is a modal refinement relation. Let $(J_1 \parallel J_2, T_1 \parallel T_2) \in R$. Clearly $\nu(J_1 \parallel J_2) = \nu(T_1 \parallel T_2)$.

Firstly, let $J_1 \parallel J_2 \xrightarrow{a} J_1' \parallel J_2'$. We distinguish the following cases.

- If $a \in \Gamma$ then $J_1 \xrightarrow{a} J_1'$ and $J_2 \xrightarrow{a} J_2'$. By definition of R, $T_1 \xrightarrow{a} T_1'$ with $J_1' \leq_m T_1'$ and $T_2 \xrightarrow{a} T_2'$ with $J_2' \leq_m T_2'$. Hence $T_1 \parallel T_2 \xrightarrow{a} T_1' \parallel T_2'$ with $(J_1' \parallel J_2', T_1' \parallel T_2') \in R$.
- If $a \in \Gamma,\ J_1 \xrightarrow{a} J_1',\ J_2 = J_2'$ then $T_1 \xrightarrow{a} T_1'$ with $J_1' \leq_m T_1'$. Hence $T_1 \parallel T_2 \xrightarrow{a} T_1' \parallel T_2$ with $(J_1' \parallel J_2, T_1' \parallel T_2) \in R$.
- The case $a \in \Gamma,\ J_1 = J_1',\ J_2 \xrightarrow{a} J_2'$ follows from symmetry.

Secondly, let $T_1 \parallel T_2 \longrightarrow \mathcal{T}'$. By the definition of $\mathsf{Succ}(T_1 \parallel T_2)$ there are either $a \in \Gamma,\ (a, T_1' \parallel T_2') \in \mathcal{T}'$ with $J_i \xrightarrow{a} J_i' \leq_m T_i'$ for $i = 1, 2$, or $a \in \Sigma \setminus \Gamma,\ (a, T_1' \parallel T_2') \in \mathcal{T}'$ with $J_1 \xrightarrow{a} J_1' \leq_m T_1'$ and $T_2' = T_2$, or symmetrically. In all cases $J_1 \parallel J_2 \xrightarrow{a} J_1' \parallel J_2'$ with $(J_1' \parallel J_2, T_1' \parallel T_2') \in R$. □

We now turn to completeness of the composition.

THEOREM 5.9. *Let* S_1, S_2 *be processes,* φ *an LTL formula, and* $\star \in \{\omega, \infty\}$. *Then* $S_1 \parallel S_2 \models^\star_\forall \varphi$ *if and only if for all* $I_1 \lhd S_1$ *and* $I_2 \lhd S_2$ *it holds* $I_1 \parallel I_2 \models^\star \varphi$.

Proof. *'Only-if'* part follows directly from Lemma C.15.
'If' part essentially amounts to proving the soundness of the may transition relation in the composition. Firstly, we define an implementation that captures all possibilities where may transitions are/are not realized. Formally, for every process S and $\mathcal{S}' \in$

$\mathsf{Succ}(S)$, let us define an implementation $(S, \mathcal{S}')$ and set $(S, \mathcal{S}') \xrightarrow{a} (S', \mathcal{S}'')$ whenever $(a, S') \in \mathcal{S}'$ and $\mathcal{S}'' \in \mathsf{Succ}(S')$. We will prove that for every $I \lhd S_1 \parallel S_2$ we have

$$\mathcal{R}^\infty(I) \subseteq \mathcal{R}^\infty((S_1, \bigcup \mathsf{Succ}(S_1)) \parallel (S_2, \bigcup \mathsf{Succ}(S_2))) \cup \{\nu(S_1 \parallel S_2)\}. \tag{1}$$

As (1) implies also $\mathcal{R}^\omega(I) \subseteq \mathcal{R}^\omega((S_1, \bigcup \mathsf{Succ}(S_1)) \parallel (S_2, \bigcup \mathsf{Succ}(S_2)))$, this concludes the proof for $\star = \omega$. Further, if $\{\nu(S_1 \parallel S_2)\} \in \mathcal{R}^\infty(I)$ then $\{\nu(S_1 \parallel S_2)\} = \mathcal{R}^\infty(I)$ and there is no must transition leading from $S_1 \parallel S_2$ and thus there are $I_1 \lhd S_1$ and $I_2 \lhd S_2$ with $\mathcal{R}^\infty(I_1 \parallel I_2) = \{\nu(S_1 \parallel S_2)\}$ which concludes the proof for $\star = \infty$.

In order to prove the trace inclusion (1), observe that considered as labelled transition systems $(S_1 \parallel S_2, \bigcup \mathsf{Succ}(S_1 \parallel S_2))$ is bisimilar to $(S_1, \bigcup \mathsf{Succ}(S_1)) \parallel (S_2, \bigcup \mathsf{Succ}(S_2))$, therefore their sets of runs are equal. In addition, $\mathcal{R}^\infty(I) \subseteq \mathcal{R}^\infty((S_1 \parallel S_2, \mathcal{S}'))$ for some $\mathcal{S}' \in \mathsf{Succ}(S_1 \parallel S_2)$. If $\mathcal{S}' \neq \emptyset$ then $\mathcal{R}^\infty((S_1 \parallel S_2, \mathcal{S}')) \subseteq \mathcal{R}^\infty((S_1 \parallel S_2, \bigcup \mathsf{Succ}(S_1 \parallel S_2)))$. And if $\mathcal{S}' = \emptyset$ then $\mathcal{R}^\infty(I) = \{\nu(S_1 \parallel S_2)\}$ which concludes the proof of (1). □

THEOREM 5.10. *Let S_1, S_2 be processes, φ an LTL formula, 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 $I_1 \parallel I_2 \models^\star \varphi$.*

Proof. *'If'* part follows directly from Lemma C.15.
'Only-if' part essentially amounts to proving the completeness of the must transition relation in the composition. Let $I \lhd S_1 \parallel S_2$, it is sufficient to find $I_1 \lhd S_1$ and $I_2 \lhd S_2$ such that $\mathcal{R}^\star(I_1 \parallel I_2) \subseteq \mathcal{R}^\star(I)$.

Since I need not be directly decomposable into two implementations, we construct an implementation $\bar{I} \lhd S_1 \parallel S_2$ such that $\mathcal{R}^\star(\bar{I}) \subseteq \mathcal{R}^\star(I)$ and then decompose $\bar{I}$. We define $\bar{I}$ to be minimal in the following sense. We have $\mathsf{Succ}(I) = \{\mathcal{I}\}$ and $\mathsf{Succ}(S_1 \parallel S_2) = \{\mathcal{S}_1, \ldots, \mathcal{S}_m\}$. Since $I \leq_m S_1 \parallel S_2$, there is $\mathcal{S}_k$ (we now fix it) such that for every $(a, S') \in \mathcal{S}_k$ there is a fixed $(a, I') \in \mathcal{I}$ with $I' \lhd S'$. Let Succ_I be the set of these fixed transitions (a, I'). This way we compute Succ_J for every implementation J and define $\bar{J} \xrightarrow{a} \bar{J'}$ if $(a, J') \in \mathsf{Succ}_J$.

Clearly, $\bar{I} \lhd S_1 \parallel S_2$ and $\mathcal{R}^\star(\bar{I}) \subseteq \mathcal{R}^\star(I)$. Moreover, $\mathsf{Succ}(\bar{I})$ contains exactly the transitions corresponding to those in Succ_I, which realize exactly the transitions from $\mathsf{Succ}(S_1 \parallel S_2)$. Therefore, we get by definition the decomposition of $\mathsf{Succ}(S_1 \parallel S_2)$ into an element from $\mathsf{Succ}(S_1)$ and an element from $\mathsf{Succ}(S_2)$, and thus a decomposition of $\bar{I}$ into I_1 and I_2 with $I_1 \lhd S_1$ and $I_2 \lhd S_2$. □

D Appendix: Complexity of the Previous Solution to CI/CS

The goal of this appendix is to give more insight into the complexity of the common implementation and specification problems solutions provided in [LX90]. Recall that the definition in [LX90] does not require the specifications to be consistent, i.e. the requirements (i) and (ii) of Definition 2.1 need not be fulfilled.

First of all, *conjunctions* of processes are introduced as follows:

$$\frac{S \longrightarrow \mathcal{S}'}{S \wedge T \longrightarrow \mathcal{S}'} \qquad \frac{T \longrightarrow \mathcal{T}'}{S \wedge T \longrightarrow \mathcal{T}'}$$

$$\frac{S \xrightarrow{a} S' \qquad T \xrightarrow{a} T'}{S \wedge T \xrightarrow{a} S' \wedge T'}$$

This leads to a solution to the common specification problem. Indeed, $S_1 \wedge \cdots \wedge S_n$ is the greatest lower bound of $S_1, \ldots, S_n$. Therefore, the complexity of the construction is the same as in our setting, i.e. it is PTIME if n is fixed and EXPTIME if n is a part of the input. However, the greatest lower bound may be vacuous, as it need not have any implementations. Therefore, a consistency check is needed to decide the common implementation problem. Checking consistency is EXPTIME-hard even for mixed transition systems [AHL+08b], which is a syntactic subclass of DMTS of [LX90]. To summarize, this yields an EXPTIME solution to CI even for the case with fixed number of specifications.

Note that in [LX90] the authors claim that checking consistency can be done in PTIME. This is only true under the assumption that all conjunctions of processes (of arbitrary arity) are already given in the input. This assumption was reasonable in [LX90] as the ultimate goal of that paper was solving process equations and the additional conjunctions arose in the preprocessing phase.

E Appendix: Errors in the Previous Attempt at LTL Model Checking over MTS

The goal of this appendix is to explain the error made in [UBC09] when dealing with generalized model checking of MTS.

The logic that is used by [UBC09] is the Fluent LTL, which bears some differences to our state and action based LTL. Nonetheless, both Fluent LTL and our LTL (as used

throughout this paper) contain the action-based LTL as a fragment. We thus only use action-based LTL in the following to explain our claim. (Note that the syntax is slightly different, e.g. [UBC09] use a where we write $\mathbf{X}_a\,\mathbf{tt}$, $\Box\,\varphi$ where we write $\mathbf{G}\,\varphi$, etc. This does not influence our claim.)

We start by quoting from [UBC09, p. 6]:

> If a property evaluates to *true* in M, it is *true* in all deadlock-free implementations of M, and if a property evaluates to *false* in M, it is *false* in all deadlock-free implementations of M. Furthermore, if a property evaluates to *maybe* in M, it is *true* in some deadlock-free implementations of M and *false* in others.
>
> [...]
>
> *Definition 16: (3-valued semantics of FLTL)* The function $\|\cdot\|^M : \mathrm{FLTL} \to \mathbf{3}$ is defined as follows:
>
> $$\begin{array}{lcl} \|\varphi\|^M = \mathbf{t} & \triangleq & \forall \pi \in \textsc{PosTr}_\infty(M) \cdot \pi \models \varphi \\ \|\varphi\|^M = \mathbf{f} & \triangleq & (\exists \pi \in \textsc{ReqTr}_\infty(M) \cdot \pi \not\models \varphi) \vee \\ & & (\forall \pi \in \textsc{PosTr}_\infty(M) \cdot \pi \not\models \varphi) \\ \|\varphi\|^M = \bot & \triangleq & \neg(\|\varphi\|^M = \mathbf{t}) \wedge \neg(\|\varphi\|^M = \mathbf{f}) \end{array}$$
>
> A formula φ is *true* in M (denoted $\|\varphi\|^M = \mathbf{t}$ or $M \models \varphi$) if every trace in $\textsc{PosTr}_\infty(M)$ satisfies φ. A formula φ is *false* in M (denoted $\|\varphi\|^M = \mathbf{f}$ or $M \not\models \varphi$) if there is a trace in $\textsc{ReqTr}_\infty(M)$ that refutes φ or if all traces in $\textsc{PosTr}_\infty(M)$ refute φ. Otherwise, a formula φ evaluates to *maybe* in M (denoted $\|\varphi\|^M = \bot$).

(In the following, whenever we write *Definition 16*, we mean the definition from [UBC09] as quoted above.)

Here, $\textsc{PosTr}_\infty(M)$ and $\textsc{ReqTr}_\infty(M)$ are earlier in the paper defined as the set of all infinite traces of M consisting of may or must transitions only, respectively. The trouble with *Definition 16* here is that it does not correspond to the intuitive meaning as explained in the paragraph preceding it. Indeed, take the example from page 14, also redrawn here as Fig. 6.

Here $\textsc{ReqTr}_\infty(M) = \emptyset$ while $\textsc{PosTr}_\infty(M) = \{a\} \cdot \{a, b\}^\omega$. Let $\varphi = \Box\, a$ (in our LTL syntax $\varphi = \mathbf{G}\,\mathbf{X}_a\,\mathbf{tt}$). Then, according to *Definition 16* in the quotation above, $\|\varphi\|^M = \bot$ as it does not hold that $\forall \pi \in \textsc{PosTr}_\infty(M) : \pi \models \varphi$ (taking $\pi = ab^\omega$), it does not hold that

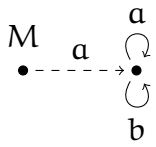

Figure 6: No deadlock-free implementation of M satisfies $\mathbf{G}\,\mathbf{X}_a\,\mathbf{tt}$

$\exists \pi \in \textsc{ReqTr}_\infty(M) : \pi \not\models \varphi$ (as $\textsc{ReqTr}_\infty(M)$ is empty), and it does not hold that $\forall \pi \in \textsc{PosTr}_\infty(M) : \pi \not\models \varphi$ (taking $\pi = a^\omega$). Thus the formula φ evaluates to *maybe* in M. This, according to the paragraph preceding *Definition 16*, should mean that there are some implementations of M that satisfy the formula and some implementations of M that refute the formula. Yet, it may be clearly verified that no deadlock-free implementation of M satisfying φ exists (as there is only one, up to bisimilarity).

The problem is more fundamental. Were *Definition 16* correct (or easily fixable), the approach of [UBC09] would give a PSPACE algorithm. However, as we know from Proposition 5.6, deciding whether there exists a deadlock-free implementation of a given MTS satisfying a given formula is 2-EXPTIME-complete, a strictly harder complexity.