Checking Thorough Refinement on Modal Transition Systems Is EXPTIME-Complete

Nikola Beneš[1]*, Jan Křetínský[1]**, Kim G. Larsen[2]***, and Jiří Srba[2]†

[1] Faculty of Informatics, Masaryk Univ., Botanická 68a, 60200 Brno, Czech Republic
[2] Department of Computer Science, Aalborg Univ., Selma Lagerlöfs Vej 300, 9220 Aalborg East, Denmark

Abstract. Modal transition systems (MTS), a specification formalism introduced more than 20 years ago, has recently received a considerable attention in several different areas. Many of the fundamental questions related to MTSs have already been answered. However, the problem of the exact computational complexity of thorough refinement checking between two finite MTSs remained unsolved.

We settle down this question by showing EXPTIME-completeness of thorough refinement checking on finite MTSs. The upper-bound result relies on a novel algorithm running in single exponential time providing a direct goal-oriented way to decide thorough refinement. If the right-hand side MTS is moreover deterministic, or has a fixed size, the running time of the algorithm becomes polynomial. The lower-bound proof is achieved by reduction from the acceptance problem of alternating linear bounded automata and the problem remains EXPTIME-hard even if the left-hand side MTS is fixed.

1 Introduction

Modal transition systems (MTS) is a specification formalism which extends the standard labelled transition systems with two types of transitions, the *may* transitions that are allowed to be present in an implementation of a given modal transition system and *must* transitions that must be necessarily present in any implementation. Modal transition systems hence allow to specify both safety and liveness properties. The MTS framework was suggested more than 20 years ago by Larsen and Thomsen [14] and has recently brought a considerable attention due to several applications to e.g. component-based software development [16, 7], interface theories [20, 17], modal abstractions and program analysis [11, 12, 15] and other areas [10, 21], just to mention a few of them. A renewed interest in tool support for modal transition systems is recently also emerging [8, 9]. A

* Partially supported by the Academy of Sciences of the Czech Republic, project No. 1ET408050503.
** Partially supported by the research centre ITI, project No. 1M0545.
*** Partially supported by the VKR Center of Excellence MT-LAB.
† Partially supported by Ministry of Education of the Czech Republic, project No. MSM 0021622419.

recent overview article on the theoretical foundations of MTSs and early tool development is available in [1].

Modal transition systems were designed to support *component-based* system development via a *stepwise refinement* process where abstract specifications are gradually refined into more concrete ones until an *implementation* of the system (where the may and must transitions coincide) is obtained. One of the fundamental questions is the decidability of a *thorough refinement* relation between two specifications S and T. We say that S thoroughly refines T iff every implementation of S is also an implementation of T. While for a number of other problems, like the common implementation problem, a matching complexity lower and upper bounds were given [2, 13, 3], the question of the exact complexity of thorough refinement checking between two finite MTSs remained unanswered.

In this paper, we prove EXPTIME-completeness of thorough refinement checking between two finite MTSs. The hardness result is achieved by a reduction from the acceptance problem of alternating linear bounded automata, a well known EXPTIME-complete problem, and it improves the previously established PSPACE-hardness [2]. The main reduction idea is based on the fact that the existence of a computation step between two configurations of a Turing machine can be locally verified (one needs to consider the relationships between three tape symbols in the first configuration and the corresponding three tape symbols in the second one, see e.g. [19, Theorem 7.37]), however, a nonstandard encoding of computations of Turing machines (which is crucial for our reduction) and the addition of the alternation required a nontrivial technical treatment. Moreover, we show that the problem remains EXPTIME-hard even if the left-hand side MTS is of a constant size. Some proof ideas for the containment in EXPTIME were mentioned in [2] where the authors suggest a reduction of the refinement problem to validity checking of vectorized modal μ-calculus, which can be solved in EXPTIME—the authors in [2] admit that such a reduction relies on an unpublished popular wisdom, and they only sketch the main ideas hinting at the EXPTIME algorithm. In our paper, we describe a novel technique for deciding thorough refinement in EXPTIME. The result is achieved by a direct goal-oriented algorithm performing a least fixed-point computation, and can be easily turned into a tableau-based algorithm. As a corollary, we also get that if the right-hand side MTS is deterministic (or of a constant size), the algorithm for solving the problem runs in deterministic polynomial time.

A full version of the paper is available in [6].

2 Basic Definitions

A *modal transition system* (MTS) over an action alphabet Σ is a triple $(P, \dashrightarrow, \longrightarrow)$, where P is a set of *processes* and $\longrightarrow \subseteq \dashrightarrow \subseteq P \times \Sigma \times P$ are *must* and *may* transition relations, respectively. The class of all MTSs is denoted by $\mathcal{MTS}$. Because in MTS whenever $S \stackrel{a}{\longrightarrow} S'$ then necessarily also $S \stackrel{a}{\dashrightarrow} S'$, we adopt the convention of drawing only the must transitions $S \stackrel{a}{\longrightarrow} S'$ in such cases. An MTS is *finite* if P and Σ are finite sets.

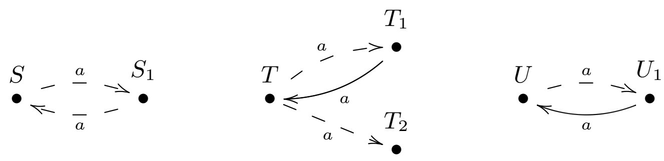


Fig. 1. $S \leq_t T$ but $S \not\leq_m T$, and $S \not\leq_t U$ and $S \not\leq_m U$

An MTS is an *implementation* if $\dashrightarrow = \longrightarrow$. The class of all implementations is denoted $i\mathcal{MTS}$ and as in implementations the must and may relations coincide, we can consider such systems as the standard *labelled transition systems*.

Definition 2.1. *Let $M_1 = (P_1, \dashrightarrow_1, \longrightarrow_1)$, $M_2 = (P_2, \dashrightarrow_2, \longrightarrow_2)$ be MTSs over the same action alphabet Σ and $S \in P_1$, $T \in P_2$ be processes. We say that S* modally refines *T, written $S \leq_m T$, if there is a relation $R \subseteq P_1 \times P_2$ such that $(S, T) \in R$ and for every $(A, B) \in R$ and every $a \in \Sigma$:*

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

We often omit the indices in the transition relations and use symbols $\dashrightarrow$ and $\longrightarrow$ whenever it is clear from the context what transition system we have in mind. Note that on implementations modal refinement coincides with the classical notion of strong bisimilarity, and on modal transition systems without any must transitions it corresponds to the well-studied simulation preorder.

Example 2.2. Consider processes S and T in Fig. 1. We prove that S does not modally refine T. Indeed, there is a may-transition $S \overset{a}{\dashrightarrow} S_1$ on the left-hand side which has to be matched by entering either T_1 or T_2 on the right-hand side. However, in the first case there is a move $T_1 \overset{a}{\longrightarrow} T$ on the right-hand side which cannot be matched from S_1 as it has no must-transition under a. In the second case there is a may-transition $S_1 \overset{a}{\dashrightarrow} S$ on the left-hand side which cannot be matched by any may-transition from T_2. Hence there cannot be any relation of modal refinement containing the pair S and T, which means that $S \not\leq_m T$. Similarly, one can argue that $S \not\leq_m U$. □

We shall now observe that the modal refinement problem, i.e. the question whether a given process modally refines another given process, is tractable for finite MTSs.

Theorem 2.3. *The modal refinement problem for finite MTSs is P-complete.*

Proof. Modal refinement can be computed in polynomial time by the standard greatest fixed-point computation, similarly as in the case of strong bisimulation. P-hardness of modal refinement follows from P-hardness of bisimulation [4] (see also [18]). □

We proceed with the definition of thorough refinement, a relation that holds for two modal specification S and T iff any implementation of S is also an implementation of T.

Definition 2.4. *For a process S let us denote by $[\![S]\!] = \{I \in i\mathcal{MTS} \mid I \leq_m S\}$ the set of all implementations of S. We say that S* thoroughly refines *T, written $S \leq_t T$, if $[\![S]\!] \subseteq [\![T]\!]$.*

Clearly, if $S \leq_m T$ then also $S \leq_t T$ because the relation $\leq_m$ is transitive. The opposite implication, however, does not hold as demonstrated by the processes S and T in Fig. 1 where one can easily argue that every implementation of S is also an implementation of T. On the other hand, $S \not\leq_t U$ because a process with just a single a-transition is an implementation of S but not of U.

3 Thorough Refinement Is EXPTIME-Hard

In this section we prove that the thorough refinement relation $\leq_t$ on finite modal transition systems is EXPTIME-hard by reduction from the acceptance problem of alternating linear bounded automata.

3.1 Alternating Linear Bounded Automata

Definition 3.1. *An* alternating linear bounded automaton (ALBA) *is a tuple $\mathcal{M} = (Q, Q_\forall, Q_\exists, \Sigma, \Gamma, q_0, q_{acc}, q_{rej}, \vdash, \dashv, \delta)$ where Q is a finite set of control states partitioned into $Q_\forall$ and $Q_\exists$, universal and existential states, respectively, Σ is a finite input alphabet, $\Gamma \supseteq \Sigma$ is a finite tape alphabet, $q_0 \in Q$ is the initial control state, $q_{acc} \in Q$ is the accepting state, $q_{rej} \in Q$ is the rejecting state, $\vdash, \dashv \in \Gamma$ are the left-end and the right-end markers that cannot be overwritten or moved, and $\delta : (Q \smallsetminus \{q_{acc}, q_{rej}\}) \times \Gamma \to 2^{Q \times \Gamma \times \{L,R\}}$ is a computation step function such that for all $q, p \in Q$ if $\delta(q, \vdash) \ni (p, a, D)$ then $a = \vdash, D = R$; if $\delta(q, \dashv) \ni (p, a, D)$ then $a = \dashv, D = L$; if $\delta(q, a) \ni (p, \vdash, D)$ then $a = \vdash$; and if $\delta(q, a) \ni (p, \dashv, D)$ then $a = \dashv$.*

Remark 3.2. W.l.o.g. we assume that $\Sigma = \{a, b\}$, $\Gamma = \{a, b, \vdash, \dashv\}$, $Q \cap \Gamma = \emptyset$ and that for each $q \in Q_\forall$ and $a \in \Gamma$ it holds that $\delta(q, a)$ has exactly two elements (q_1, a_1, D_1), (q_2, a_2, D_2) where moreover $a_1 = a_2$ and $D_1 = D_2$. We fix this ordering and the successor states q_1 and q_2 are referred to as the *first* and the *second successor*, respectively. The states q_{acc}, q_{rej} have no successors.

A *configuration* of $\mathcal{M}$ is given by the state, the position of the head and the content of the tape. For technical reasons, we write it as a word over the alphabet $\Xi = Q \cup \Gamma \cup \{\vdash, \dashv, \exists, \forall, 1, 2, *\}$ (where $\exists, \forall, 1, 2, *$ are fresh symbols) in the following way. If the tape contains a word $\vdash w_1 a w_2 \dashv$, where $w_1, w_2 \in \Gamma^*$ and $a \in \Gamma$, and the head is scanning the symbol a in a state q, we write the configuration as $\vdash w_1 \alpha\beta q a w_2 \dashv$ where $\alpha\beta \in \{\exists *, \forall 1, \forall 2\}$.

The two symbols $\alpha\beta$ before the control state in every configuration are non-standard, though important for the encoding of the computations into modal transition systems to be checked for thorough refinement. Intuitively, if a control state q is preceded by $\forall 1$ then it signals that the previous configuration (in a given computation) contained a universal control state and the first successor was chosen; similarly $\forall 2$ reflects that the second successor was chosen. Finally, if the control state is preceded by $\exists *$ then the previous control state was existential and in this case we do not keep track of which successor it was, hence the symbol $*$ is used instead. The *initial configuration* for an input word w is by definition $\vdash \exists * q_0 w \dashv$.

Depending on the present control state, every configuration is called either *universal*, *existential*, *accepting* or *rejecting*.

A *step of computation* is a relation $\rightarrow$ between configurations defined as follows (where $w_1, w_2 \in \Gamma^*$, $\alpha\beta \in \{\forall 1, \forall 2, \exists *\}$, $a, b, c \in \Gamma$, $i \in \{1, 2\}$, and $w_1 a w_2$ and $w_1 c a w_2$ both begin with $\vdash$ and end with $\dashv$):

- $w_1 \alpha\beta q a w_2 \rightarrow w_1 b \forall i p w_2$
  if $\delta(q, a) \ni (p, b, R)$, $q \in Q_\forall$ and (p, b, R) is the i'th successor,
- $w_1 \alpha\beta q a w_2 \rightarrow w_1 b \exists * p w_2$
  if $\delta(q, a) \ni (p, b, R)$ and $q \in Q_\exists$,
- $w_1 c \alpha\beta q a w_2 \rightarrow w_1 \forall i p c b w_2$
  if $\delta(q, a) \ni (p, b, L)$, $q \in Q_\forall$ and (p, b, L) is the i'th successor, and
- $w_1 c \alpha\beta q a w_2 \rightarrow w_1 \exists * p c b w_2$
  if $\delta(q, a) \ni (p, b, L)$ and $q \in Q_\exists$.

Note that for an input w of length n all reachable configurations are of length $n + 5$. A standard result is that one can efficiently compute the set $Comp \subseteq \Xi^{10}$ of all compatible 10-tuples such that for each sequence $C = c_1 c_2 \cdots c_k$ of configurations $c_1, c_2, \ldots, c_k$, with the length of the first configuration being $l = |c_1| = n + 5$, we have $c_1 \rightarrow c_2 \rightarrow \cdots \rightarrow c_k$ iff for all i, $0 \leq i \leq (k-1)l - 5$,

$$(C(i+1), C(i+2), C(i+3), C(i+4), C(i+5),\\ C(i+1+l), C(i+2+l), C(i+3+l), C(i+4+l), C(i+5+l)) \in Comp\ .$$

A *computation tree* for $\mathcal{M}$ on an input $w \in \Sigma^*$ is a tree T satisfying the following: the root of T is (labeled by) the initial configuration, and whenever N is a node of T labeled by a configuration c then the following holds:

- if c is accepting or rejecting then N is a leaf;
- if c is existential then N has one child labeled by some d such that $c \rightarrow d$;
- if c is universal then N has two children labelled by the first and the second successor of c, respectively.

Without loss of generality, we shall assume from now on that any computation tree for $\mathcal{M}$ on an input w is finite (see e.g. [19, page 198]) and that every accepting configuration contains at least four other symbols following after the state q_{acc}.

We say that $\mathcal{M}$ *accepts* w iff there is a (finite) computation tree for $\mathcal{M}$ on w with all leaves labelled with accepting configurations. The following fact is well known (see e.g. [19]).

Proposition 3.3. *Given an ALBA M and a word w, the problem whether M accepts w is EXPTIME-complete.*

3.2 Encoding of Configurations and Computation Trees

In this subsection we shall discuss the particular encoding techniques necessary for showing the lower bound. For technical convenience we will consider only tree encodings and so we first introduce the notion of tree-thorough refinement.

Definition 3.4. *Let Tree denote the class of all MTSs with their graphs being trees. We say that a process S* tree-thoroughly refines *a process T, denoted by $S \leq_{tt} T$, if $[\![S]\!] \cap \mathit{Tree} \subseteq [\![T]\!] \cap \mathit{Tree}$.*

Lemma 3.5. *For any two processes S and T, $S \leq_{tt} T$ iff $S \leq_t T$.*

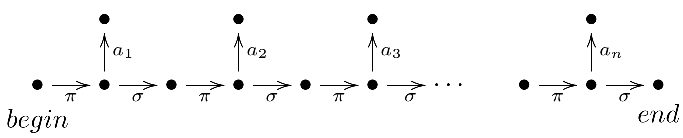
Proof. The if case is trivial. For the only if case, we define an *unfold* $U(S)$ of a process S over an MTS $M = (P, \dashrightarrow, \longrightarrow)$ with an alphabet Σ to be a process S over an MTS $U(M) = (P^*, \dashrightarrow_U, \longrightarrow_U)$ over the same alphabet and where P^* is the set of all finite sequences over the symbols from P. The transition relations are defined as follows: for all $a \in \Sigma$, $T, R \in P$ and $\alpha \in P^*$, whenever $T \overset{a}{\dashrightarrow} R$ then $\alpha T \overset{a}{\dashrightarrow}_U \alpha T R$, and whenever $T \overset{a}{\longrightarrow} R$ then $\alpha T \overset{a}{\longrightarrow}_U \alpha T R$. Since the transitions in $U(S)$ depend only on the last symbol, we can easily see that $U(S) \leq_m S$ and $S \leq_m U(S)$ for every process S.

Let I be now an implementation of S. Its unfold $U(I)$ is also an implementation of S by $U(I) \leq_m I \leq_m S$ and the transitivity of $\leq_m$. By our assumption that $S \leq_{tt} T$ and the fact that $U(I)$ is a tree, we get that $U(I)$ is also an implementation of T. Finally, $I \leq_m U(I) \leq_m T$ and the transitivity of $\leq_m$ allow us to conclude that I is an implementation of T. $\square$

Let $\mathcal{M} = (Q, Q_\forall, Q_\exists, \Sigma, \Gamma, q_0, q_{acc}, q_{rej}, \vdash, \dashv, \delta)$ be an ALBA and $w \in \Sigma^*$ an input word of length n. We shall construct (in polynomial time) modal transition systems L and R such that $\mathcal{M}$ accepts w iff $L \not\leq_{tt} R$. The system L will encode (almost) all trees beginning with the initial configuration, while the implementations of R encode only the incorrect or rejecting computation trees.

Configurations, i.e. sequences of letters from Ξ, are not encoded straightforwardly as sequences of actions (the reason why this naive encoding does not work is explained later on in Remark 3.12). Instead we have to use two auxiliary actions π a σ. The intended implementations of L and R will alternate between the actions π and σ on a linear path, while the symbols in the encoded configuration are present as side-branches on the path.

Formally, a sequence $a_1 a_2 a_3 \cdots a_n \in \Xi^*$ is encoded as



and denoted by $\mathrm{code}(a_1 a_2 \cdots a_n)$.

We now describe how to transform computation trees into their corresponding implementations. We simply concatenate the subsequent codes of configurations in the computation tree such that the end node of the previous configuration is merged with the begin node of the successor configuration. Whenever there is a (universal) branching in the tree, we do not branch in the corresponding implementation at its beginning but we wait until we reach the occurrence of $\forall$. The branching happens exactly before the symbols 1 or 2 that follow after $\forall$. This occurs in the same place on the tape in both of the configurations due to the assumption that the first and the second successor move simultaneously either to the left or to the right, and write the same symbol (see Remark 3.2). A formal definition of the encoding of computation trees into implementations follows.

Definition 3.6 (Encoding computation trees into implementations). *Let $\mathcal{T}$ be a (finite) computation tree. We define its tree implementation* $\text{code}(\mathcal{T})$ *inductively as follows:*

- *if $\mathcal{T}$ is a leaf labelled with a configuration c then* $\text{code}(\mathcal{T}) = \text{code}(c)$*;*
- *if the root of $\mathcal{T}$ is labelled by an existential configuration c with a tree $\mathcal{T}'$ being its child, then* $\text{code}(\mathcal{T})$ *is rooted in the begin node of* $\text{code}(c)$*, followed by* $\text{code}(\mathcal{T}')$ *where the end node of* $\text{code}(c)$ *and the begin node of* $\text{code}(\mathcal{T}')$ *are identified;*
- *if the root of $\mathcal{T}$ is labelled by a universal configuration c with two children $d_1 \ldots \forall 1 \ldots d_n^1$ and $d_1 \ldots \forall 2 \ldots d_n^2$ that are roots of the subtrees $\mathcal{T}_1$ and $\mathcal{T}_2$, respectively, then* $\text{code}(\mathcal{T})$ *is rooted in the begin node of* $\text{code}(c)$*, followed by two subtrees* $\text{code}(\mathcal{T}_1)$ *and* $\text{code}(\mathcal{T}_2)$ *where the nodes in* $\text{code}(d_1 \ldots \forall)$ *of the initial part of* $\text{code}(\mathcal{T}_1)$ *are identified with the corresponding nodes in the initial part of* $\text{code}(\mathcal{T}_2)$ *(note that by Remark 3.2 this prefix is common in both subtrees), and finally the end node of* $\text{code}(c)$ *is identified with now the common begin node of both subtrees.*

Fig. 2 illustrates this definition on a part of a computation tree, where the first configuration $c_1 \ldots c_n$ is universal and has two successor configurations $d_1 \ldots \forall 1 \ldots d_n^1$ and $d_1 \ldots \forall 2 \ldots d_n^2$.

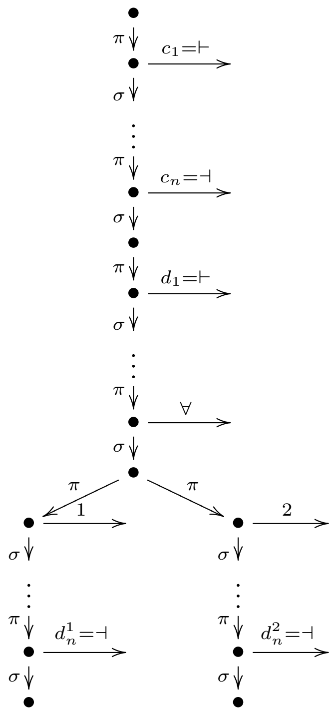


Fig. 2. Comp. Tree Encoding

3.3 The Reduction—Part 1

We now proceed with the reduction. As mentioned earlier, our aim is to construct for a given ALBA $\mathcal{M}$ and a string w two modal transition systems L and R such that $L \not\leq_{tt} R$ iff $\mathcal{M}$ accepts w. Implementations of L will include all (also

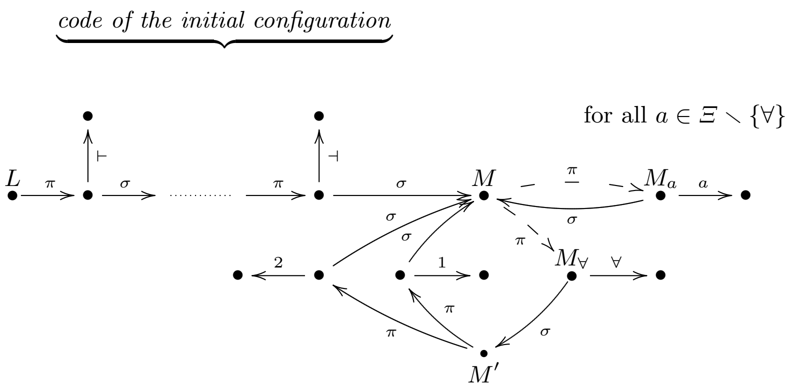


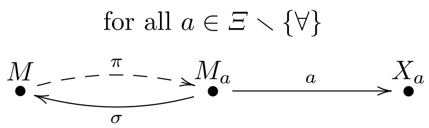
Fig. 3. Full specification of the process L

incorrect) possible computation trees. We only require that they start with the encoding of the initial configuration and do not "cheat" in the universal branching (i.e. after the encoding of every symbol $\forall$ there must follow a branching such that at least one of the branches encodes the symbol 1 and at least another one encodes the symbol 2).

As L should capture implementations corresponding to computations starting in the initial configuration, we set L to be the begin of code($\vdash\exists{*}q_0w\dashv$) and denote its end by M. After the initial configuration has been forced, we allow all possible continuations of the computation. This can be simply done by setting

$$\begin{aligned} M &\overset{\pi}{\dashrightarrow} M_a \\ M_a &\overset{\sigma}{\longrightarrow} M \\ M_a &\overset{a}{\longrightarrow} X_a \end{aligned}$$

for all letters $a \in \Xi \smallsetminus \{\forall\}$ and there are no outgoing transitions from X_a.



Finally, we add a fragment of MTS into the constructed process L which will guarantee the universal branching as mentioned above whenever the symbol $\forall$ occurs on a side-branch. The complete modal transition system L is now depicted in Fig. 3.

We shall now state some simple observations regarding tree implementations of the process L.

Proposition 3.7. *Every tree implementation I of the process L satisfies that*

1. *every branch in I is labelled by an alternating sequence of π and σ actions, beginning with the action π, and if the branch is finite then it ends either with the action σ or with an actions $a \in \Xi \smallsetminus \{\forall\}$, and*
2. *every state in I with an incoming transition under the action π has at least one outgoing transition under the action σ and at least one outgoing transition under an action $a \in \Xi$, and*
3. *whenever from any state in I there are two outgoing transitions under some $a \in \Xi$ and $b \in \Xi$ then $a = b$, and moreover no further actions are possible after taking any transition under $a \in \Xi$, and*
4. *every branch in I longer than $2(n+5)$ begins with the encoding of the initial configuration $\vdash\exists{*}q_0w\dashv$ where $n = |w|$, and*
5. *every state in I with an incoming transition under σ from a state where the action $\forall$ is enabled satisfies that every outgoing transition under π leads to a state where either the action 1 or 2 is enabled (but not both at the same time), and moreover it has at least one such transition that enables the action 1 and at least one that enables the action 2.*

Of course, not every tree implementation of the process L represents a correct computation tree of the given ALBA. Implementations of L can widely (even uncountably) branch at any point and sequences of configurations they encode on some (or all) of their branches may not be correct computations of the given ALBA. Nevertheless, the encoding of any computation tree of the given ALBA is an implementation of the processes L, as stated by the following lemma.

Lemma 3.8. *Let T be a computation tree of an ALBA $\mathcal{M}$ on an input w. Then* $\mathrm{code}(T) \leq_m L$.

Proof (Sketch). To show that the implementation $\mathrm{code}(T)$ modally refines L is rather straightforward. The implementation $\mathrm{code}(T)$ surely starts with the encoding of the initial configuration and all symbols $a \in \Xi \smallsetminus \{\forall\}$ on the side-branches in $\mathrm{code}(T)$ can be matched by entering M_a in the right-hand side process M. In case that the implementation contains a side-branch with the symbol $\forall$, the specification M will enter the state $M_\forall$ and require that two branches with labels 1 and 2 follow, however, from definition of $\mathrm{code}(T)$ this is clearly satisfied. □

3.4 The Reduction—Part 2

We now proceed with the construction of the right-hand side process R. Its implementations should be the codes of all incorrect or rejecting computation trees. To cover the notion of incorrect computation, we define a so-called bad path (see page 5 for definition of the relation *Comp*).

Definition 3.9. *A sequence*

$$c_1c_2c_3c_4c_5\,\underbrace{a_1a_2\ldots a_{n-6}a_{n-5}}_{n-5\ \textit{elements from}\ \Xi}\,d_1d_2d_3d_4d_5$$

is called a bad path *if* $(c_1, c_2, c_3, c_4, c_5, d_1, d_2, d_3, d_4, d_5) \in \Xi^{10} \smallsetminus \mathit{Comp}$.

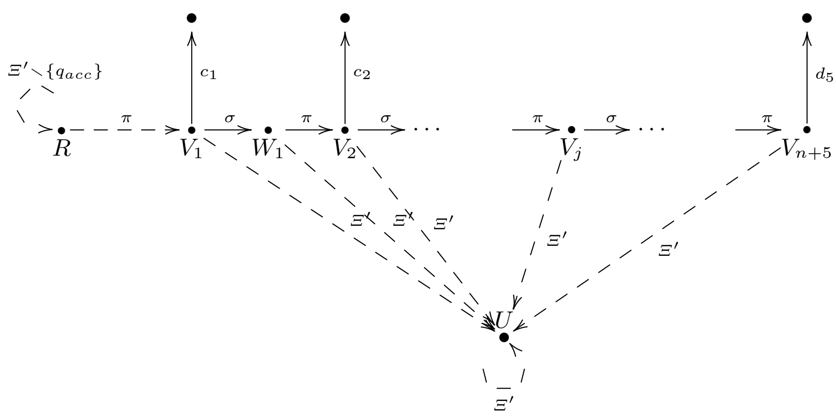


Fig. 4. A fragment of the system R for a bad path $c_1c_2c_3c_4c_5 \dots d_1d_2d_3d_4d_5$

To cover the incorrect or rejecting computations, we loop in the process R under all actions, including the auxiliary ones, except for q_{acc}. For convenience we denote $\Xi' = \Xi \cup \{\pi, \sigma\}$. For any bad path, the process R can at any time nondeterministically guess the beginning of its first quintuple, realize it, then perform $n - 5$ times a sequence of π and σ, and finally realize the second quintuple. Moreover, we have to allow all possible detours of newly created branches to end in the state U where all available actions from Ξ' are always enabled and hence the continuation of any implementation is modally refined by U. Formally, for any $(c_1, c_2, c_3, c_4, c_5, d_1, d_2, d_3, d_4, d_5) \in \Xi^{10} \smallsetminus \mathit{Comp}$ we add (disjointly) the following fragment into the process R (see also Fig. 4).

$$
\begin{array}{ll}
R \overset{\pi}{\dashrightarrow} V_1 & \\
V_j \overset{\pi}{\longrightarrow} W_j \overset{\sigma}{\longrightarrow} V_{j+1} & \text{for } 1 \le j < n+5 \\
V_j \overset{c_j}{\longrightarrow} X_j & \text{for } 1 \le j \le 5 \\
V_{n+j} \overset{d_{n+j}}{\longrightarrow} X_{5+j} & \text{for } 1 \le j \le 5 \\
V_j \overset{x}{\dashrightarrow} U, W_j \overset{x}{\dashrightarrow} U, V_{n+5} \overset{x}{\dashrightarrow} U & \text{for } 1 \le j < n+5 \text{ and } x \in \Xi' \\
U \overset{x}{\dashrightarrow} U & \text{for all } x \in \Xi' \\
R \overset{x}{\dashrightarrow} R & \text{for all } x \in \Xi' \smallsetminus \{q_{acc}\}
\end{array}
$$

We also add ten new states $N_1, \dots, N_{10}$ and the following transitions: $R \overset{\pi}{\dashrightarrow} N_1 \overset{\Xi'}{\dashrightarrow} N_2 \overset{\Xi'}{\dashrightarrow} N_3 \overset{\Xi'}{\dashrightarrow} N_4 \overset{\Xi'}{\dashrightarrow} \dots \overset{\Xi'}{\dashrightarrow} N_{10}$ and $N_1 \overset{q_{acc}}{\longrightarrow} N_{10}$ where any transition labelled by Ξ' is the abbreviation for a number of transitions under all actions from Ξ'.

Remark 3.10. We do not draw these newly added states $N_1, \dots, N_{10}$ into Fig. 4 in order not to obstruct its readability. The reason why these states are added is purely technical. It is possible that there is an incorrect computation that ends with the last symbol q_{acc} but it cannot be detected by any bad path as defined in Definition 3.9 because that requires (in some situations) that there should be present at least four other subsequent symbols. By adding these new states into the process R, we guarantee that such situations where a branch in a computation tree ends in q_{acc} without at least four additional symbols will be easily matched in R by entering the state N_1. □

Lemma 3.11. *Let I be a tree implementation of L such that every occurrence of q_{acc} in I is either preceded by a code of a bad path or does not continue with the encoding of at least four other symbols. Then $I \leq_m R$.*

Proof (Sketch). All branches in I that do not contain q_{acc} can be easily matched by looping in R and all branches that contain an error (bad path) before q_{acc} appears on that branch are matched by entering the corresponding state V_1 and at some point ending in the state U which now allows an arbitrary continuation of the implementation I (including the occurrence of the state q_{acc}). □

Remark 3.12. Lemma 3.11 demonstrates the point where we need our special encoding of configurations using the alternation of π and σ actions together with side-branches to represent the symbols in the configurations. If the configurations were encoded directly as sequences of symbols on a linear path, the construction would not work. Indeed, the must path of alternating σ and π actions in the process R is necessary to ensure that the bad path entered in the left-hand side implementation I is indeed realizable. This path cannot be replaced by a linear path of must transitions containing directly the symbols of the configurations because the sequence of $n-5$ symbols in the middle of the bad sequence would require exponentially large MTS to capture all such possible sequences explicitly and the reduction would not be polynomial. □

Let us now finish the definition of the process R. Note that in ALBA even rejecting computation trees can still contain several correct computation paths ending in accepting configurations. We can only assume that during any universal branching in a rejecting tree, *at least one* of the two possible successors forms a rejecting branch. The process R must so have the possibility to discard the possibly correct computation branch in universal branching and it suffices to make sure that the computation will continue with only one of the branches.

So in order to finish the construction of R we add an additional fragment to R as depicted in Fig. 5 (it is the part below R that starts with branching to U_1 and U_2).

The construction of the process R is now finished (recall that the part of the construction going from R to the right is repeated for any bad path of the machine $\mathcal{M}$). Because the newly added part of the construction does not use any must transitions, it does not restrict the set of implementations and hence Lemma 3.11 still holds. The following two lemmas show that the added part of the construction correctly handles the universal branching.

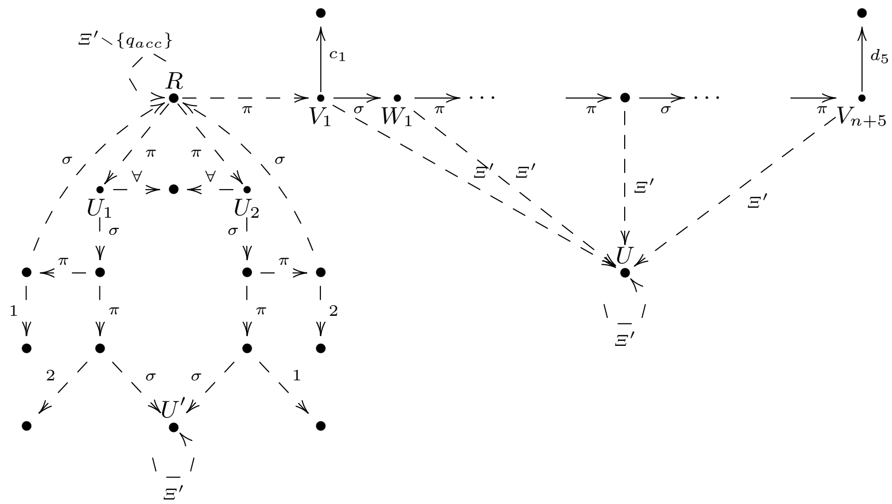


Fig. 5. Full specification of the process R

Lemma 3.13. *Let I be a tree implementation of L which is not, even after removing any of its branches, a code of any accepting computation tree of $\mathcal{M}$ on the input w. Then $I \leq_m R$.*

Proof (Sketch). We should prove that in the universal branching in I, the specification R can choose one of the two possible continuations and discard the checking of the other one. This is achieved by entering either the state U_1 or U_2 whenever the next side-branch in I contains the symbol $\forall$. From U_1 the continuation under the second successor is discarded by entering the state U' and symmetrically from U_2 the continuation under the first successor is discarded. We argued in Lemma 3.11 for the rest. $\square$

Lemma 3.14. *Let T be an accepting computation tree of an ALBA $\mathcal{M}$ on the input w. Then* $\text{code}(T) \not\leq_m R$.

Proof (Sketch). Indeed, in $\text{code}(T)$ any branch ends in a configuration containing q_{acc} and there is no error (bad path), so clearly $\text{code}(T) \not\leq_m R$. $\square$

3.5 Summary

We can now combine the facts about the constructed systems L and R.

Theorem 3.15. *An ALBA $\mathcal{M}$ accepts an input w iff $L \not\leq_t R$.*

Proof. If $\mathcal{M}$ accepts the input w then clearly it has an accepting computation tree T. By Lemma 3.8 $\text{code}(T) \leq_m L$ and by Lemma 3.14 $\text{code}(T) \not\leq_m R$. This implies that $L \not\leq_t R$.

On the other hand, if $\mathcal{M}$ does not accept w then none of the tree implementations of L represents a code of an accepting computation tree of $\mathcal{M}$ on w. By Lemma 3.13 this means that any tree I such that $I \leq_m L$ satisfies that $I \leq_m R$ and hence $L \leq_{tt} R$ which is by Lemma 3.5 equivalent to $L \leq_t R$. $\square$

Corollary 3.16. *The problem of checking thorough refinement on finite modal transition systems is EXPTIME-hard.*

In fact, we can strengthen the result by adapting the above described reduction to the situation where the left-hand side system is of a fixed size (see [6]).

Theorem 3.17. *The problem of checking thorough refinement on finite modal transition systems is EXPTIME-hard even if the left-hand side system is fixed.*

4 Thorough Refinement Is in EXPTIME

In this section we provide a direct algorithm for deciding thorough refinement between MTSs in EXPTIME. Given two processes A and B over some finite-state MTSs, the algorithm will decide if there exists an implementation I that implements A but not B, i.e. $I \leq_m A$ and $I \not\leq_m B$.

For a modal transition systems B, we introduce the syntactical notation $\overline{B}$ to denote the semantical complement of B, i.e. $I \leq_m \overline{B}$ iff $I \not\leq_m B$. Our algorithm now essentially checks for consistency (existence of a common implementation) between A and $\overline{B}$ with the outcome that they are consistent if and only if $A \not\leq_t B$.

In general, we shall check for consistency of sets of the form $\{A, \overline{B}_1, \ldots, \overline{B}_k\}$ in the sense of existence of an implementation I such that $I \leq_m A$ but $I \not\leq_m B_i$ for all $i \in \{1, \ldots, k\}$. Before the full definition is given, let us get some intuition by considering the case of consistency of a simple pair $A, \overline{B}$. During the arguments, we shall use CCS-like constructs (summation and action-prefixing) for defining implementations.

Clearly, if for some B' with $B \stackrel{a}{\longrightarrow} B'$ and for all A_i with $A \stackrel{a}{\longrightarrow} A_i$ we can find an implementation I_i implementing A_i but not B' (i.e. we demonstrate consistency between all the pairs $A_i, \overline{B'}$), we can claim consistency between A and $\overline{B}$: as a common implementation I simply take $H + \sum_i a.I_i$, where H is some arbitrary implementation of A with all a-derivatives removed.

We may also conclude consistency of A and $\overline{B}$, if for some A' with $A \stackrel{a}{\dashrightarrow} A'$, we can find an implementation I' of A', which is not an implementation of any B' where $B \stackrel{a}{\dashrightarrow} B'$. Here a common implementation would simply be $H + a.I'$ where H is an arbitrary implementation of A. However, in this case we will need to determine consistency of the set $\{A'\} \cup \{\overline{B'} \mid B \stackrel{a}{\dashrightarrow} B'\}$ which is in general not a simple pair.

Definition 4.1. *Let $M = (P, \dashrightarrow, \longrightarrow)$ be an MTS over the action alphabet Σ. The set of* consistent *sets of the form $\{A, \overline{B}_1, \ldots, \overline{B}_k\}$, where $A, B_1, \ldots, B_k \in P$, is the smallest set* Con *such that $\{A, \overline{B}_1, \ldots, \overline{B}_k\} \in \mathsf{Con}$ whenever $k = 0$ or for some $a \in \Sigma$ and some $J \subseteq \{1, \ldots, k\}$, where for all $j \in J$ there exists B'_j such that $B_j \stackrel{a}{\longrightarrow} B'_j$, we have*

1. $\{A', \overline{B'_j} \mid j \in J\} \in \mathsf{Con}$ *for all* A' *with* $A \xrightarrow{a} A'$, *and*
2. $\{A_\ell, \overline{B'_\ell} \mid B_\ell \overset{a}{\dashrightarrow} B'_\ell\} \cup \{\overline{B'_j} \mid j \in J\} \in \mathsf{Con}$ *for all* $\ell \notin J$ *and some* A_ℓ *with* $A \overset{a}{\dashrightarrow} A_\ell$.

Lemma 4.2. *Given processes* $A, B_1, \ldots, B_k$ *of some finite MTS, there exists an implementation* I *such that* $I \leq_m A$ *and* $I \not\leq_m B_i$ *for all* $i \in \{1, \ldots, k\}$ *if and only if* $\{A, \overline{B}_1, \ldots, \overline{B}_k\} \in \mathsf{Con}$.

Computing the collection of consistent sets $\{A, \overline{B}_1, \ldots, \overline{B}_k\}$ over an MTS $(P, \dashrightarrow, \longrightarrow)$ may be done as a simple (least) fixed-point computation. The running time is polynomial in the number of potential sets of the form $\{A, \overline{B}_1, \ldots, \overline{B}_k\}$ where $A, B_1, \ldots, B_k \in P$, hence it is exponential in the number of states of the underlying MTS. This gives an EXPTIME algorithm to check for thorough refinement.

Theorem 4.3. *The problem of checking thorough refinement on finite modal transition systems is decidable in EXPTIME.*

Example 4.4. Consider S and T from Fig. 1. We have already mentioned in Section 2 that $S \leq_t T$. To see this, we will attempt (and fail) to demonstrate consistency of $\{S, \overline{T}\}$ according to Definition 4.1, which essentially asks for a finite tableau to be constructed. Now, in order for $\{S, \overline{T}\}$ to be concluded consistent, we have to establish consistency of $\{S_1, \overline{T}_1, \overline{T}_2\}$— as T has no must-transitions the only choice for J is $J = \emptyset$. Now, to establish consistency of $\{S_1, \overline{T}_1, \overline{T}_2\}$ both $J = \emptyset$ and $J = \{1\}$ are possibilities. However, in both cases the requirement will be that $\{S, \overline{T}\}$ must be consistent. Given this cyclic dependency together with the minimal fixed-point definition of Con it follows that $\{S, \overline{T}\}$ is *not* consistent, and hence that $S \leq_t T$. □

Example 4.5. Consider S and U from Fig. 1. Here $S \not\leq_t U$ clearly with $I = a.0$ as a witness implementation. Let us demonstrate consistency of $\{S, \overline{U}\}$. Choosing $J = \emptyset$, this will follow from the consistency of $\{S_1, \overline{U}_1\}$. To conclude this, note that $J = \{1\}$ will leave us with the empty collection of sets—as S_1 has no must-transitions—all of which are obviously consistent. □

Note that in the case of B being deterministic, we only need to consider pairs of the form $\{A, \overline{B}\}$ for determining consistency. This results in a polynomial time algorithm (see also [5] for an alternative proof of this fact). Similarly, if the process B is of a constant size, our algorithm runs in polynomial time as well.

Corollary 4.6. *The problem of checking thorough refinement between a given finite modal transition system and a finite deterministic or fixed-size modal transition system is in P.*

To conclude, by Theorem 4.3 and Corollary 3.16 we get our main result.

Theorem 4.7. *The problem of checking thorough refinement on finite modal transition systems is EXPTIME-complete.*

References

1. A. Antonik, M. Huth, K.G. Larsen, U. Nyman, and A. Wasowski. 20 years of modal and mixed specifications. *Bulletin of the EATCS 95*, pages 94–129, 2008.
2. A. Antonik, M. Huth, K.G. Larsen, U. Nyman, and A. Wasowski. Complexity of decision problems for mixed and modal specifications. In *Proc. of FOSSACS'08*, volume 4962 of *LNCS*, pages 112–126. Springer, 2008.
3. A. Antonik, M. Huth, K.G. Larsen, U. Nyman, and A. Wasowski. EXPTIME-complete decision problems for mixed and modal specifications. In *Proc. of EXPRESS'08*, July 2008.
4. J. L. Balcazar, J. Gabarró, and M. Santha. Deciding bisimilarity is P-complete. *Formal aspects of computing*, 4(6 A):638–648, 1992.
5. N. Beneš, J. Křetínský, K.G. Larsen, and J. Srba. On determinism in modal transition systems. *Theoretical Computer Science*, 2008. To appear.
6. N. Beneš, J. Křetínský, K.G. Larsen, and J. Srba. Checking thorough refinement on modal transition systems is EXPTIME-complete. Technical report FIMU-RS-2009-03, Faculty of Informatics, Masaryk University, Brno, 2009.
7. N. Bertrand, S. Pinchinat, and J.-B. Raclet. Refinement and consistency of timed modal specifications. In *Proc. of LATA'09*, volume 5457 of *LNCS*, pages 152–163. Springer, 2009.
8. 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.
9. N. D'Ippolito, D. Fischbein, H. Foster, and S. Uchitel. MTSA: Eclipse support for modal transition systems construction, analysis and elaboration. In *Proc. of (ETX'07)*, pages 6–10. ACM, 2007.
10. 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.
11. P. Godefroid, M. Huth, and R. Jagadeesan. Abstraction-based model checking using modal transition systems. In *Proc. CONCUR'01*, volume 2154 of *LNCS*, pages 426–440. Springer, 2001.
12. 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.
13. K.G. Larsen, U. Nyman, and A. Wasowski. On modal refinement and consistency. In *Proc. of CONCUR'07*, volume 4703 of *LNCS*, pages 105–119. Springer, 2007.
14. K.G. Larsen and B. Thomsen. A modal process logic. In *Proc. of LICS'88*, pages 203–210. IEEE, 1988.
15. S. Nanz, F. Nielson, and H.R. Nielson. Modal abstractions of concurrent behaviour. In *Proc. of SAS'08*, volume 5079 of *LNCS*, pages 159–173. Springer, 2008.
16. J.-B. Raclet. Residual for component specifications. In *Proc. of the 4th International Workshop on Formal Aspects of Component Software*, 2007.
17. J.-B. Raclet, E. Badouel, A. Benveniste, B. Caillaud, and R. Passerone. Why are modalities good for interface theories? In *Proc. of ACSD'09*, 2009. To appear.
18. Z. Sawa and P. Jančar. Behavioural equivalences on finite-state systems are PTIME-hard. *Computing and informatics*, 24(5):513–528, 2005.
19. M. Sipser. *Introduction to the Theory of Computation.* Course Technology, 2006.
20. S. Uchitel and M. Chechik. Merging partial behavioural models. In *Proc. of FSE'04*, pages 43–52. ACM, 2004.
21. O. Wei, A. Gurfinkel, and M. Chechik. Mixed transition systems revisited. In *Proc. of VMCAI'09*, volume 5403 of *LNCS*, pages 349–365. Springer, 2009.