

Model checking for action abstraction [★]

Harald Fecher and Michael Huth

Imperial College London, United Kingdom
{hfecher, M.Huth}@doc.ic.ac.uk

Abstract. We endow action sets of transition systems with a partial order that expresses the degree of specialization of actions, and with an intuitive but flexible consistency predicate that constrains the extension of such orders with more specialized actions. We develop a satisfaction relation for such models and the μ -calculus. We prove that this satisfaction relation is sound for Thomsen’s extended bisimulation as our refinement notion for models, even for consistent extensions of ordered action sets. We then demonstrate how this satisfaction relation can be reduced, fairly efficiently, to classical μ -calculus model checking. These results provide formal support for change management of models and their validation (e.g. in model-centric software development), and enable verification of concrete systems with respect to properties specified for abstract actions.

1 Introduction

Transition systems and their variants are popular models of state and behavior for programs and reactive systems alike. Transitions are triples (s, a, s') specifying that action a may cause the system to change from state s to state s' . Whenever transition systems function as models we can validate these models, e.g. through property verification in the form of model checking formulas of the μ -calculus [13] over the action set pertinent to the model and property. Most popular and tool-supported branching-time temporal logics embed into the μ -calculus so the latter is an ideal target for foundational studies.

Often one wishes to specify and verify properties that are more abstract than the model itself. Consider a model for a stack-like data structure. An input action $put(n)$ puts value $n \in \mathbb{N}$ on top of the stack; output action $get(m)$ reads $m \in \mathbb{N}$ as the top value of the stack and then pops the top of the stack. A property that this model should enjoy is that, at all states, there is a finite sequence of output actions whose execution in sequence results in a state where no output actions are possible: the stack can always be emptied completely. For the above infinite action set, this property can be expressed in the μ -calculus if we extend it with infinite conjunctions and disjunctions:

$$\nu X. \left(\left(\bigwedge_{n \in \mathbb{N}} [put(n)]X \wedge [get(n)]X \right) \wedge \mu Y. \left(\left(\bigvee_{m \in \mathbb{N}} \langle get(m) \rangle Y \right) \vee \left(\bigwedge_{n \in \mathbb{N}} [get(n)]ff \right) \right) \right) \quad (1)$$

where ff denotes falsity. The formula in (1) is a greatest fixed point (νX) whose body is a conjunction. The first conjunct is a recursion and ensures that the second conjunct is

[★] This work is in part financially supported by the DFG project (FE 942/2-1) and by the UK EP-SRC project *Complete and Efficient Checks for Branching-Time Abstractions* EP/E028985/1

true at all reachable states. The second conjunct states that, at the present state, there is a finite (least fixed point μY) sequence of transitions, labeled with possibly different output actions $get(m)$, to a state at which no such output actions are possible. Representing this property in an idiom such as (1) has a number of disadvantages:

- The infinite conjunctions and disjunctions in (1) appeal to an infinite state space, making model checking hard or undecidable.
- The encoding in (1) cannot be model checked on more abstract models, e.g. for one in which get abstracts all instances of $get(n)$, we cannot verify the subformula $\langle get(0) \rangle \dots$ as any transition labeled with get could be refined to $get(1)$ instead.
- The encoding in (1) lacks flexibility and support for top down development; e.g. if we extend the stack data type with a non-destructive output action $read(n)$ for all $n \in \mathbb{N}$, the formula in (1) has to be extended to

$$\nu X.((\bigwedge_{n \in \mathbb{N}} \dots \wedge [read(n)]X) \wedge \mu Y.((\bigvee_{m \in \mathbb{N}} \dots \vee \langle read(m) \rangle Y) \vee (\bigwedge_{n \in \mathbb{N}} \dots \wedge [read(n)]\text{ff})))$$

The last two points state that property specifications have to change each time a model changes its level of abstraction. This need for change management of property specifications increases even more if a model has components expressed at different levels of abstraction.

We address all these disadvantages by using an order on the set of actions. Consider the action order in Fig. 1. It has a most abstract action *anyAction*, abstracts all finite or infinite action sets of the same kind with an action for that kind (e.g. *put* abstracts all $put(n)$), and introduces a new action *out* that is a common abstraction of the already abstract output actions *get* and *read*. For this action order we can rewrite (1) in the ordinary μ -calculus as

$$\nu X.([anyAction]X \wedge \mu Y.(\langle out \rangle Y \vee [out]\text{ff})) \tag{2}$$

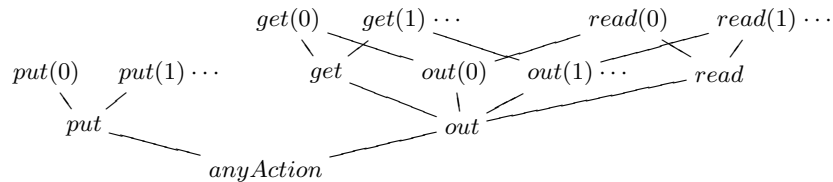


Fig. 1. An infinite partial order of actions, depicted in a Hasse diagram. Action abstraction (resp. specialization) is modeled by a decrease (resp. increase) in the order

With a compact and abstract property such as (2) at hand, we now want to be able to verify it for models that may express these actions at more concrete levels of abstraction, without having to change the formula in (2). Moreover, we want such verification to be sound for refinements of the model we check, even though the respective models and

their actions may be at different levels of abstraction. In particular more concrete models may introduce new actions that are common specializations of actions. For example, *put* and *out* of Fig. 1 may obtain a common specialization *swap* that replaces the top of the stack with another item but also outputs the value of the item at the top of the stack as it was prior to that swap. This use of common specializations is familiar from multiple inheritance if we dare to think for a moment of actions as objects.

The ability to specify abstract properties that capture potentially evolving action structure then fits well with model-centric software development where models may predate executable code, and so we can validate models with the same abstract property without knowing the actual action structure of the source code.

In developing this approach, we employ two orthogonal ways of model refinement – which can be applied in any order in a refinement process:

- the extension of ordered actions sets, similar in spirit to the introduction of (more) multiple inheritance; and
- a co-inductive refinement, the extended bisimulation of Thomsen in [23] for transition systems with ordered action sets.

Our contributions. In this paper we make the following contributions:

1. Develop a satisfaction relation for action-ordered transition systems and formulas of the μ -calculus. We show that this satisfaction relation is sound with respect to Thomsen’s refinement of action-ordered transition systems.
2. Show that any sensible satisfaction relation is unsound for the unconstrained extension of pairs of actions with new common specialized actions.
3. Extend partial orders of actions with a *consistency* predicate, stating which pairs of actions may obtain common specialized action, and adapt our satisfaction to ensure its soundness for new common specializations of consistent pairs of actions.
4. Give a fairly efficient reduction of this satisfaction relation to the standard one over transition systems with unordered actions and formulas of the ordinary μ -calculus, in which actions are also interpreted without any appeal to an order.

The combination of the first and third contribution above guarantees that, once a property has been verified for a model, its validity is preserved if that model then undergoes a sequence of changes, where each single change extends the action set or refines the model. The fourth contribution means we can reuse the knowledge and tool support for μ -calculus model checking over transition systems to verify properties of the action-ordered μ -calculus over action-ordered transition systems with consistency.

Outline. In Section 2 we show that extensions of ordered action sets with new action specializations poses a problem for sound model checking of abstract models. In Section 3 we present our models, their refinement, and consistent extensions of ordered action sets. A satisfaction relation for the μ -calculus over transition systems that have partial orders with consistency as action sets is motivated, defined, and proved to be sound in Section 4. A reduction of that satisfaction relation to classical μ -calculus model checking is given in Section 5. Related work is discussed in Section 6 and we conclude in Section 7.

2 Naive extension of ordered action sets is unsound

The stack-like data structure discussed in the introduction may be modeled with more abstract labels as a state machine in Fig. 2. Transitions are triples of synchronization event, guard, and side effects. For example, in states on the right hand side in Fig. 2 transition $put(m) [tt] i := i + 1$ specifies that any synchronization with a $put(m)$ action will increment the counter of the stack size as a side effect since the guard is true.

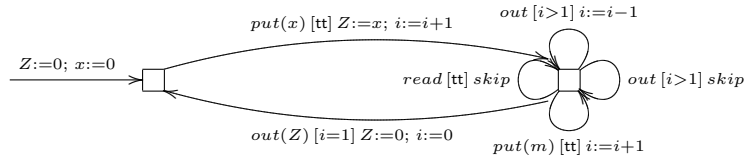


Fig. 2. Abstract model for the implementation of a stack-like data structure. Variable $i \in \mathbb{N}$ counts the stack size, and variable $Z \in \mathbb{N}$ records the value found at the bottom of the stack. The values of output actions $read$ and out are unspecified. In particular, in an implementation $read$ could return the sum of all values stored in such a non-standard stack

- Example 1.* 1. For the action order given in Fig. 3, the abstract model from Figure 2 is expected to satisfy $\langle put \rangle [out(0)] ff$, stating that it is possible to input a value such that 0 cannot be output immediately thereafter. We expect this since $out(0)$ has no further specialization in its action set, the state on the right state side with $i = 1$ and $Z = 1$ is reached after the $put(1)$ -action, and there is no transition from that state that can be specialized to $out(0)$.
2. But the same abstract model does not satisfy $\langle put \rangle [out(0)] ff$ for the order given in Fig. 1, since in a left hand state with $i = 1$ and any Z there is a transition with label $read$ that can be specialized to an action $read(0)$ that also specializes $out(0)$.

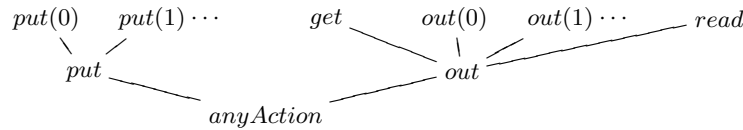


Fig. 3. Possible partial order of the set of abstract actions for the system in Fig. 2

Since the partial order in Fig. 1 is intuitively an extension of the one in Fig. 3, Example 1 suggests that soundness of satisfaction is not preserved if two actions can always obtain a common specialized action. To remedy this, we need to constrain such extensions. If we ban the introduction of new upper bounds for pairs of actions in extensions, this is simply too restrictive for modeling, validation, and code development. Yet, Example 1 mandates constraints on the introduction of new common upper bounds.

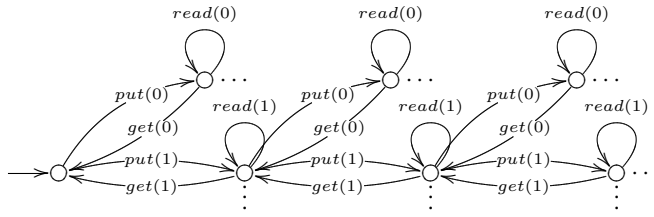


Fig. 4. A transition system realizing a full implementation of the abstract model in Fig. 2

We suggest a consistency predicate – determined by the modeler – as a flexible policy for introducing specialized actions, saying which pairs of actions a and a' can have common specialized versions. Natural requirements for such a policy are:

- pairs that already have an upper bound have to be consistent, and
- consistency of action pairs is closed under abstraction of either action.

The most restrictive change policy, for a given partial order, is to stipulate that pairs are consistent iff they have an upper bound in that partial order. We use this policy for the partial orders in Fig. 1 and 3. The partial order with consistency, depicted in Fig. 5 employs a more liberal change policy. That partial order is extended by both partial orders in Fig. 1 and 3 and the term “extension” will be formalized in Definition 2.3. Satisfaction of properties now needs to take this consistency predicate into account.

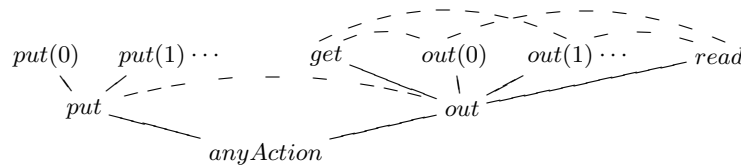


Fig. 5. A partial order of actions with a consistency predicate. Dashed bows connect pairs of actions a and a' that are consistent but have no common upper bound in this partial order already. For example, the consistency of put and out allows an extension with an action $swap$ (which may swap the top of the stack but return the value of the top prior to that swap), but action $swap$ then cannot be a specialization of get since put and get are inconsistent

Example 2. We expect property (2) not to hold in the system from Fig. 2 with respect to the order with consistency of Fig. 5: On a state on the right hand side with $i = 1$ and any Z there is a transition with label $read$ but actions $read$ and $out(0)$ could have a common specialization, since $out(0)$ and $read$ are specified to be consistent. This expectation on a satisfaction relation will be fulfilled with our formal satisfaction relation developed in Section 4.

3 Ordered actions, transition systems, and refinement

Preliminaries. $|M|$ denotes the cardinality of a set M . For a ternary relation $\rightsquigarrow \subseteq M_1 \times \text{Act} \times M_2$ we write $m_1 \overset{a}{\rightsquigarrow} m_2$ for $(m_1, a, m_2) \in \rightsquigarrow$. For $m_1 \in M_1$ the expression $m_1 \overset{a}{\rightsquigarrow}$ denotes $\{m_2 \in M_2 \mid m_1 \overset{a}{\rightsquigarrow} m_2\}$. Let $\sqsubseteq \subseteq M \times M$ be a partial order, i.e., a reflexive, antisymmetric, and transitive relation. The upper set of $X \subseteq M$ with respect to \sqsubseteq is $\uparrow_{\sqsubseteq} X = \{m \in M \mid \exists m' \in X: m' \sqsubseteq m\}$ and the lower set of X with respect to \sqsubseteq is $\downarrow_{\sqsubseteq} X = \{m \in M \mid \exists m' \in X: m \sqsubseteq m'\}$. If $X = \{m\}$, we write $\uparrow_{\sqsubseteq} m$ (resp. $\downarrow_{\sqsubseteq} m$) for $\uparrow_{\sqsubseteq} X$ (resp. $\downarrow_{\sqsubseteq} X$). For $x, y \in M$ we write $x \uparrow_{\sqsubseteq} y$ to denote that x and y have an upper bound in M : $\uparrow_{\sqsubseteq} x \cap \uparrow_{\sqsubseteq} y \neq \{\}$. The inverse relation of a binary relation $R \subseteq M \times M$ is $R^{-1} = \{(m_1, m_2) \mid (m_2, m_1) \in R\}$. For $R_1 \subseteq M_1 \times M_2$ and $R_2 \subseteq M_2 \times M_3$ let $R_1 \circ R_2$ be the relational composition $\{(m_1, m_3) \mid \exists m_2: m_1 R_1 m_2, m_2 R_2 m_3\}$.

We formalize ordered action sets and their consistency predicate.

Definition 1. A partial order with consistency is a partial order (M, \sqsubseteq) with a symmetric consistency predicate $\curvearrowright \subseteq M \times M$ that contains \uparrow_{\sqsubseteq} and is preserved by downward closure: $\curvearrowright = \curvearrowright^{-1}$, $\uparrow_{\sqsubseteq} \subseteq \curvearrowright$, and $\sqsubseteq \circ \curvearrowright \subseteq \curvearrowright$.

A partial order with consistency is illustrated in Fig. 5, where elements of $\curvearrowright \setminus \uparrow$ (consistent pairs that are not deemed consistent by \uparrow alone) are drawn as dashed bows.

Definition 2. Let $((M_i, \sqsubseteq_i), \curvearrowright_i)$, $i = 1, 2$, be partial orders with consistency. Then

1. $((M_1, \sqsubseteq_1), \curvearrowright_1)$ is finite if M_1 is finite.
2. $((M_1, \sqsubseteq_1), \curvearrowright_1)$ is discrete if $\sqsubseteq_1 = \{(m, m) \mid m \in M_1\} = \curvearrowright_1$.
3. $((M_1, \sqsubseteq_1, \curvearrowright_1))$ is an extension of $((M_2, \sqsubseteq_2), \curvearrowright_2)$ iff $M_2 \subseteq M_1$, $\sqsubseteq_2 = \sqsubseteq_1 \cap (M_2 \times M_2)$, and $\curvearrowright_1 \cap (M_2 \times M_2) \subseteq \curvearrowright_2$.

Example 3. The partial order with consistency of Fig. 1 is an extension of the one of Fig. 5 (where new actions are added and a consistency dependency is removed), but not of the one of Fig. 3.

We define transition systems as usual, but their set of actions may then be endowed with any partial order and consistency predicate.

Definition 3 (Transition system).

1. A transition system T over a (possibly infinite) set of transition labels Act is a tuple $(S, s^i, \longrightarrow)$ such that $(s \in)S$ is its set of states, $s^i \in S$ its initial state, and $\longrightarrow \subseteq S \times \text{Act} \times S$ its transition relation.
2. We call T concrete with respect to a partial order with consistency $((\text{Act}, \sqsubseteq), \curvearrowright)$ if only maximal elements of Act with respect to \sqsubseteq occur in \longrightarrow and these occurring elements are consistent with their abstractions only, i.e., $(s, a, s') \in \longrightarrow$ implies $a \in \max(\text{Act}, \sqsubseteq)$ and $\{a\} \curvearrowright = \downarrow a$ (which equals $\{a\} \cdot \uparrow$ as a is maximal).

Example 4. Fig. 6 depicts a transition system which is not concrete for the action order from Fig. 5. This is so since, e.g., the non-maximal action *out* occurs or since, e.g., action *read* (which is consistent with another maximal element) occurs. The transition system of Fig. 4 is concrete for the order from Fig. 1.

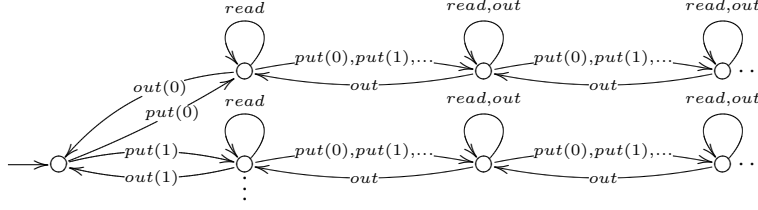


Fig. 6. The transition system corresponding to the state machine of Fig. 2

- \longrightarrow_1 : The Refuter chooses $a_1 \in \text{Act}$ and $s'_1 \in s_1$. $\xrightarrow{a_1}_1$; Verifier responds with $a_2 \in \downarrow_{\sqsubseteq} a_1$ and with $s'_2 \in s_2$. $\xrightarrow{a_2}_2$; the next configuration is (s'_1, s'_2) .
- \longrightarrow_2 : Refuter chooses $a_2 \in \text{Act}$ and $s'_2 \in s_2$. $\xrightarrow{a_2}_2$; Verifier responds with $a_1 \in \uparrow_{\sqsubseteq} a_2$ and with $s'_1 \in s_1$. $\xrightarrow{a_1}_1$; the next configuration is (s'_1, s'_2) .

Table 1. Moves of \sqsubseteq -refinement game at configuration $(s_1, s_2) \in S_1 \times S_2$. The \sqsubseteq -refinement plays are sequences of configurations generated thus

A transition system may be concrete for one action order but not for another one.

Refinement for transition systems based on ordered actions were introduced by Thomsen under the name “extended bisimulation” in [23]: To show that s refines t , one keeps the “zig-zag” nature of the bisimulation game [16] but relaxes the burden on Verifier: if Refuter moves with $(s, a, s') \in \longrightarrow$, then Verifier only has to find an abstract version a' of a and then reply with $(t, a', t') \in \longrightarrow$. Dually, if Refuter moves with some $(t, a, t') \in \longrightarrow$, Verifier needs only reply with $(s, a'', s') \in \longrightarrow$ for an action a'' that is more specialized than a , where $a' = a$ and $a'' = a$ are legal replies.

This refinement notion does not appeal to any consistency predicates and assumes that models have the same partial order of actions. This won't constrain our approach.

Definition 4 (Refinement). Let $(\text{Act}, \sqsubseteq)$ be a partial order.

- \sqsubseteq -refinement plays between a Refuter and a Verifier, for transition systems T_1 and T_2 over Act , proceed as stated in Table 1. All infinite plays are won by Verifier.
- T_1 \sqsubseteq -refines T_2 , written $T_1 \sqsubseteq_{\text{ref}} T_2$, iff Verifier has a strategy for the corresponding \sqsubseteq -refinement game between T_1 and T_2 such that Verifier wins all refinement plays started at (s_1^i, s_2^i) with her strategy.

We note that \sqsubseteq -refinement corresponds to bisimulation between transition systems whenever \sqsubseteq is discrete. The complexity of deciding refinement is bounded by the product of the respective sizes of the transition relation, assuming that there are at least as many transitions as states in models.

Example 5. The transition system in Fig. 4 is a refinement of the one in Fig. 6 with respect to the order from Fig. 1.

If the partial order $(\text{Act}, \sqsubseteq)$ is an extension of $(\text{Act}', \sqsubseteq')$, then $\sqsubseteq'_{\text{ref}}$ equals \sqsubseteq_{ref} restricted to transition systems over Act' . This is vital for our approach since it implies that refinement checks won't render conflicting results if such checks are conducted for different extensions of the systems under check.

4 Satisfaction

We present our action-ordered μ -calculus and its semantics through tree automata and games, respectively. This presentational choice simplifies proofs and anticipates future extensions of this work to fairness constraints in refinements as seen, e.g., in [6, 7].

Definition 5 (Tree automata). An alternating tree automaton A with respect to a set of actions Act is a tuple (Q, q^i, δ, Θ) such that

- $(q \in)Q$ is a finite, nonempty set of states with initial element $q^i \in Q$,
- δ is a transition relation mapping automaton states to one of the following forms, where $q \in Q$, $\tilde{Q} \subseteq Q$, and $a \in \text{Act}$: $\tilde{\vee}\tilde{Q} \mid \tilde{\wedge}\tilde{Q} \mid [a]q \mid \langle a \rangle q$ and
- $\Theta: Q \rightarrow \mathbb{N}$ is an acceptance condition whose finite image has non-empty intersection with $\{0, 1\}$. An infinite sequence of automaton states is accepted iff the maximal acceptance number occurring infinitely often in that sequence is even.

The formulas tt and ff are expressible as empty conjunctions and disjunctions in our setting (resp.) and will be used subsequently whenever convenient.

An intuitive semantics for the diamond ($\langle a \rangle$) and box ($[a]$) modalities, where labels are ordered by a partial order with consistency $((\text{Act}, \sqsubseteq), \curvearrowright)$, is a possibly infinite disjunction and conjunction (resp.):

$$\langle a \rangle q' \equiv \bigvee \{ \langle a' \rangle q' \mid a \sqsubseteq a' \} \quad [a]q' \equiv \bigwedge \{ \llbracket a' \rrbracket q \mid a \curvearrowright a' \} \quad (3)$$

where $\langle a' \rangle$ and $\llbracket a' \rrbracket$ are the diamond and box modalities defined in the classical, unordered, setting for a' (resp.). Quantification over actions is implicit within actions. Consequently, properties such as $\forall n \in \mathbb{N}: \text{AG}(\text{put}(n) \rightarrow \text{AFget}(n))$ have no finite representation in our approach to action abstraction.

These intuitive equations would indeed be formal equivalences if we were to extend the μ -calculus with infinite conjunctions and disjunctions. Note that $a \sqsubseteq a'$ implies $a' \curvearrowright a$ and so the box modality $[a]\phi$ will imply the diamond modality $\langle a \rangle\phi$ on transition systems that are serial for each action a . We emphasize that formulas $[a]\phi$ and $\neg\langle a \rangle\neg\phi$ are *not* equivalent for our satisfaction relation over action-ordered transition systems in general. However, for any partial order with consistency that contains a these formulas turn out to be equivalent if the transition system is *concrete*.

The constraint that the image of Θ contains 0 or 1 is a convenience so that the dual of the dual automaton of A is A again.

Definition 6 (Dual automaton). The dual automaton of an automaton A , written A^d , is $(Q, q^i, \delta^d, \Theta^d)$, where

$$\forall q: \Theta^d(q) = \begin{cases} \Theta(q) + 1 & \text{if } 0 \in \Theta(Q) \\ \Theta(q) - 1 & \text{otherwise} \end{cases}$$

and δ^d swaps $\tilde{\wedge}$ with $\tilde{\vee}$, and swaps $\langle a \rangle$ with $[a]$.

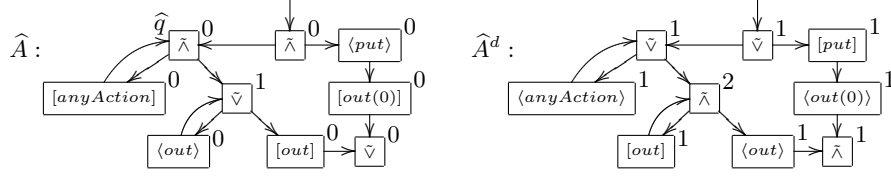


Fig. 7. Left: alternating tree automaton for the conjunction of the formula in (2) and $\langle put \rangle[out(0)]\text{ff}$. Right: its dual automaton. Accepting values and some state names are depicted to the right (resp. left) of states

$\tilde{\nu}\tilde{Q}$: Verifier picks a q' from \tilde{Q} ; the next configuration is (s, q') .

$\tilde{\lambda}\tilde{Q}$: Refuter picks a q' from \tilde{Q} ; the next configuration is (s, q') .

$\langle a \rangle q'$: Verifier picks $a' \in \uparrow \sqsubseteq a$ and $s' \in s \xrightarrow{a'}$; the next configuration is (s', q') .

$[a]q'$: Refuter picks $a' \in \text{Act}$ with $a' \frown a$ and picks $s' \in s \xrightarrow{a'}$; the next configuration is (s', q') .

Table 2. Moves of (\sqsubseteq, \frown) -satisfaction game at configuration $(s, q) \in S \times Q$, by a case analysis on $\delta(q)$. The (\sqsubseteq, \frown) -satisfaction plays are sequences of configurations generated thus

An alternating tree automaton and its dual one are depicted in Fig. 7. Next we introduce some technical notation needed for defining our satisfaction relation. For any bounded sequence \mathbf{n} of elements in \mathbb{N} we write $\text{sup}(\mathbf{n})$ for the largest m that occurs in \mathbf{n} infinitely often. Let $\text{map}(f, \Phi)$ be the sequence obtained from the sequence Φ by applying function f to all elements of Φ in situ. We write $\Phi[2]$ for the sequence obtained from Φ by projecting to the second coordinate of each configuration.

We can now define satisfaction formally.

Definition 7. Let $((\text{Act}, \sqsubseteq), \frown)$ be a partial order with consistency.

- Finite (\sqsubseteq, \frown) -satisfaction plays for transition system T over Act and alternating tree automaton A over Act have the rules as stated in Table 2. An infinite play Φ is a win for Verifier iff $\text{sup}(\text{map}(\Theta, \Phi[2]))$ is even; otherwise it is won by Refuter.
- T (\sqsubseteq, \frown) -satisfies A , written $T \models_{\sqsubseteq} A$, iff Verifier has a strategy for the corresponding satisfaction game between T and A such that Verifier wins all satisfaction plays started at (s^i, q^i) with her strategy.

The decision problem of whether T (\sqsubseteq, \frown) -satisfies A is in $\text{UP} \cap \text{coUP}$ [12], as the rules for the (\sqsubseteq, \frown) -satisfaction game specify a parity game. If $((\text{Act}, \sqsubseteq), \frown)$ is discrete, (\sqsubseteq, \frown) -satisfaction corresponds to the classical μ -calculus satisfaction [13]. Note that at a state $\langle a \rangle q'$ Verifier has to pick a more or equally specialized action, whereas at a $[a]q'$ state Refuter may pick any action that is consistent to a with respect to \frown , i.e., Refuter may pick any action a' that may have a common specialization with a . This semantics of the box modality is reasonable and even necessary since such labels a' may be subsequently specialized to some a'' with $a \sqsubseteq a''$ and so need to be under the scope of $[a]q'$.

A transition system may satisfy neither an alternating tree automaton nor its dual:

Example 6. The transition system in Fig. 6 satisfies the automaton $\widehat{A}_{\hat{q}}$, which is \widehat{A} but with initial state \hat{q} . But it does not satisfy the automata \widehat{A} from Fig. 7 nor its dual for the order of Fig. 5. For the disjunct $\langle put \rangle [out(0)]ff$ we saw in Section 2 that this is false. The disjunct $[put] \langle out(0) \rangle tt$ is not satisfied since, after $put(1)$ no $out(0)$ is guaranteed, the transition labeled with $read$ can be specialized to $read(0)$ only.

Satisfaction checks are inherently three-valued. A fourth truth value for “inconsistency”, as found in Belnap’s four-valued bilattice [2], is not required.

Proposition 1. *For any transition system T and alternating tree automata A over the partial order with consistency $((Act, \sqsubseteq), \neg)$ as actions, T cannot (\sqsubseteq, \neg) -satisfy A as well as its dual: never do $T \models_{\sqsubseteq} A$ and $T \models_{\sqsubseteq} A^d$ hold at the same time.*

We prove that order extensions provide sound extensions of satisfaction and that our notion of satisfaction is closed under all ordered refinements. Both are essential meta-properties for our modeling and validation framework for systems at varying abstraction levels. In particular, we secure soundness for any finite sequence of refinement steps where each refinement step extends the action order or refines the transition system.

Theorem 1. *1. If T_1, T_2 are transition systems over Act , and A an automaton over Act such that $T_1 (\sqsubseteq, \neg)$ -refines T_2 , and $T_2 (\sqsubseteq, \neg)$ -satisfies A , then transition system $T_1 (\sqsubseteq, \neg)$ -satisfies A .*
2. Let $((Act, \sqsubseteq), \neg)$ be an extension of $((Act', \sqsubseteq'), \neg')$. Then
(a) $\models_{\sqsubseteq'}$ implies (i.e. is contained in) \models_{\sqsubseteq} for all transition systems over Act' .
(b) if T_1 is a transition system over Act , T_2 a transition system over Act' , and A an automaton over Act' such that $T_1 \sqsubseteq$ -refines T_2 and $T_2 (\sqsubseteq', \neg')$ -satisfies A , then $T_1 (\sqsubseteq, \neg)$ -satisfies A .

We next illustrate this soundness of order extensions and model refinement.

Example 7. Let $((Act, \sqsubseteq), \neg)$ be the partial order with (most restrictive) consistency from Fig. 1. Let $((Act', \sqsubseteq'), \neg')$ be the partial order with consistency from Fig. 5. The former is an extension of the latter as seen in Example 3. Let T_1 be the transition system over Act given in Fig. 4. Let T_2 be the transition system over Act' , and therefore also over Act , given in Figure 6. Let A be the automaton on the left in Fig. 7, except that the initial state is \hat{q} . Then $T_1 \sqsubseteq$ -refines T_2 by Example 5. Also, $T_2 (\sqsubseteq', \neg')$ -satisfies A by Example 6. So by Theorem 1(2) we know that $T_1 (\sqsubseteq, \neg)$ -satisfies A .

The usage of a consistency predicate, operative in Theorem 1(2), is not only sufficient but also necessary as already illustrated in Section 2.

5 Reduction

In this section we show that our satisfaction relation for ordered actions reduces to the usual one for transition systems and the μ -calculus. This enables the reuse of existing theory, algorithms, and tool support. Let T be a transitions system and A an automata over the same partial order with consistency $((Act, \sqsubseteq), \neg)$. The reduction of their satisfaction check will be done in several stages, each illustrated with the transition system from Fig. 6, the automata \widehat{A} from Fig. 7, and the action order with consistency of Fig. 5.

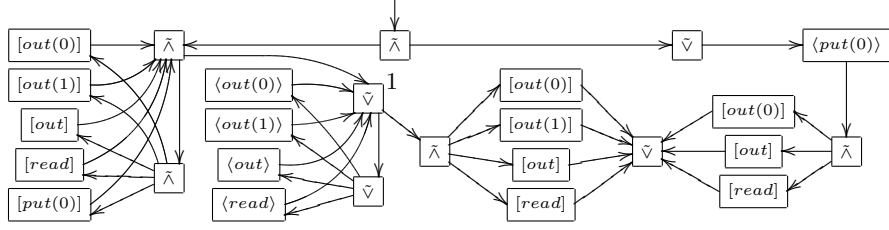


Fig. 10. The discretization of the automata \hat{A} from Fig. 7 with respect to the transition system of Fig. 9 and the order of Fig. 8. All acceptance values 0 are omitted

be replacing each action a occurring in A with $[a]_{\sqsubseteq}$, which is $\{a\}$. Then we have

$$T' \models_{\sqsubseteq'} A' \iff T \models_{\sqsubseteq} A \quad (5)$$

Derived automaton. The model check $T' \models_{\sqsubseteq'} A'$ is now over a *finite* partial order with consistency. Guided by (3) we exploit this finiteness to convert A' into a discrete automaton A'' such that the classical model check $T' \models A''$ captures the ordered model check $T' \models_{\sqsubseteq'} A'$. In doing so, we optimize this discretization of A' by slicing it with respect to the subset $\text{Act}_{T'}$ of Act' of those actions that occur in T' . This is done by letting any quantifier automata state $\langle A \rangle q$ (resp. $[A]q$) become a \tilde{v} -state (resp. $\tilde{\lambda}$ -state) that then points to newly added states (A', q, \diamond) (resp. (A', q, \square)), where A' ranges over all states from $\text{Act}_{T'}$ that are \sqsubseteq' -above (resp. \sim' -consistent with) A . Such newly added states (A', q, \diamond) (resp. (A', q, \square)) in turn point to q via $\langle A' \rangle$ (resp. via $[A']$). Acceptance values remain unaffected by this optimization. New states get acceptance value 0.

Formally, the $\text{Act}_{T'}$ -discretization of A' with respect to $((\text{Act}', \sqsubseteq'), \sim')$ is the alternating tree automaton $A'' = (Q' \cup Q^{dis}, q^i, \delta'', \Theta'')$ where

$$\begin{aligned} - Q^{dis} &= \{(A', q, \diamond) \mid A' \in \text{Act}_{T'} \ \& \ \exists A \in \downarrow_{\sqsubseteq'} A', \tilde{q} \in Q' : \delta'(\tilde{q}) = \langle A \rangle q\} \cup \\ &\quad \{(A', q, \square) \mid A' \in \text{Act}_{T'} \ \& \ \exists A \in \text{Act}', \tilde{q} \in Q' : A \sim' A' \ \& \ \delta'(\tilde{q}) = [A]q\} \\ - \delta''(\tilde{q}) &= \begin{cases} \tilde{v}\{(A', q, \diamond) \mid A' \in \text{Act}_{T'} \cap \uparrow_{\sqsubseteq'} A\} & \text{if } \tilde{q} \in Q' \ \& \ \delta'(\tilde{q}) = \langle A \rangle q \\ \tilde{\lambda}\{(A', q, \square) \mid A' \in \text{Act}_{T'} \ \& \ A \sim' A'\} & \text{if } \tilde{q} \in Q' \ \& \ \delta'(\tilde{q}) = [A]q \\ \langle A' \rangle q & \text{if } \tilde{q} = (A', q, \diamond) \in Q^{dis} \\ [A']q & \text{if } \tilde{q} = (A', q, \square) \in Q^{dis} \\ \delta'(\tilde{q}) & \text{otherwise} \end{cases} \\ - \Theta''(\tilde{q}) &= \begin{cases} \Theta'(\tilde{q}) & \text{if } \tilde{q} \in Q' \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

The discretization automata of our running example is illustrated in Fig. 10.

Theorem 2. For the reduction described in this section we have

$$T' \models A'' \iff T' \models_{\sqsubseteq'} A' \iff T \models_{\sqsubseteq} A \quad (6)$$

where \models denotes the ordinary satisfaction relation between transition systems and alternating tree automata [24]. Thus, model checking property A on T with respect to the partial order with consistency $((\text{Act}, \sqsubseteq), \sim)$ reduces to classical model checking of property A'' on transition system T' .

The complexity of this reduction is as follows: the size of the derived order is $|\text{Act}'| \leq \min\{4^{|\text{Act}_A|} + |\text{Act}_A|, |\text{Act}|\}$. The size of the transition system does not increase. The size of the automaton is $O(\max\{m_1, m_2\} \cdot |Q|)$ where m_1 is the size of the largest filter $\uparrow_{\sqsubseteq'} a$ with $a \in \text{Act}_{T'}$, and m_2 the size of the largest set of actions in $\text{Act}_{T'}$ whose elements are all consistent to the same singleton element in $\text{Act}_{T'}$.

6 Related work

Having a finite and compact (e.g. symbolic) description of concrete and abstract models enables automated abstraction and refinement methods (e.g. predicate abstraction). If our transition systems are represented in such form, e.g. in a kind of state machine, we would like to express the derived transition system defined in Section 5 in the same formalism. For state machines this can be achieved by extending their modeling language with an operator for persistent choice [8].

In our treatment of consistency in action sets pairs with upper bounds *must* be consistent, and pairs that are in the consistency predicate *may* be consistent. This is similar to the interpretation of may- and must-transitions in modal transition systems [17]. It is different to that interpretation, though, it that our box and diamond modalities are not always duals whereas this is the case for modal transition systems. So our setting is similar to that of intuitionistic modal logic [22] where, additionally, propositional logic operators cannot be interdefined.

Lattice automata [14] map propositions and transitions to elements of a lattice, and lattice elements are derived for computation paths through the algebraic operations on the lattice. In our approach, elements of a partial order are annotated with subformulas and no algebraic or lattice structure of elements is present or used. Our consistency predicate ensures that we reason about an entire set of partial orders in a compact and incremental fashion. Our approach is different to that of multi-valued model checking [4] for essentially the same reasons. Latticed simulation [15] can be considered as the simulation version of the refinement [23], presented here. The reduction of multi-valued model checking to the ordinary one [3] exploits the representation theorem for finite distributive lattices, structures we do not have in our approach.

The order on actions sets defined in this paper and our talk of “abstracting” and “specializing” actions suggests an alternative presentation of such orders through the use of abstract interpretation [5]. For any action order, one can obtain a Galois connection between the set of bounded lower sets of that action order and the powerset of maximal actions in that action order. The details of that construction are given in [11], but in a completely different setting.

Let us dare to equate actions with objects and action orders with the transitive sub-type relationship between classes in an object-oriented language with inheritance, interfaces and abstract classes. Then common lower bounds of actions are similar to interfaces, and common upper bounds of actions are similar to multiple inheritance. These similarities may be helpful in understanding our approach.

The abstraction of infinitely many actions $a(n)$ to a single abstract action a may suggest that our work has connections to parameterized model checking [25] and data independence techniques [19]. In parameterized systems, parameters represent size in-

stances of finite-state modules (e.g. the number of layers in a bus architecture). Invisible auxiliary assertions and counting abstractions are two methods for addressing the undecidability of model checking that occurs for most of such systems. In our approach actions such as *put* are like local parameters but they can't function as global ones since their disjunctive interpretation cannot be moved to the front of formulas that contain recursion. As for data independence, this usually requires a polymorphic treatment of data variables whereas actions such as *put* are evaluated over models that may specialize this action and thereby expose implementation details for that data type.

We note that (weak) bisimulation [21, 20] is not a suitable refinement notion for ordered-labeled models, since abstract labels cannot be refined into concrete ones.

Our approach is not connected to action refinement [9, 1, 10]. In the latter an action is being replaced by a unique process (possibly another action name) whereas, in our setting, an action can be replaced by any number of concrete actions that are not yet specified and where such replacements may differ from instance to instance.

Finally, our approach can be extended or adapted to other abstract models of systems, e.g. to the aforementioned modal transition systems and disjunctive modal transition systems [18]. Such extensions are routine matters and so not discussed here.

7 Conclusion

We considered transition systems as models of implementations and μ -calculus formulas as validation properties of such models (and so of their implementations). We then sought notions of model refinement and a satisfaction relation between models and properties that can, at the same time, accommodate abstraction and the incremental specialization of actions in models. We discovered the extended bisimulation in [23] as a suitable candidate for refinement of models. We then saw that a naive satisfaction relation cannot be sound for extending ordered action sets with novel specialized actions. This led us to endow ordered action sets with a robust consistency predicate that provides sufficient constraints to such extensions so that a satisfaction relation that takes action orders into account is indeed sound for refinement of models *and* for extension of action orders. This consistency notion is flexible enough to be potentially useful for model-driven development and validation. Finally, we demonstrated that this ordered satisfaction problem can be, fairly efficiently, reduced to a standard model checking problem for the μ -calculus and so to the use of standard tools.

Acknowledgments. We thank the anonymous referees and Nir Piterman for their comments which helped with improving the presentation of this paper.

References

1. L. Aceto. *Action refinement in process algebras*. Cambridge University Press, 1992.
2. N. D. Belnap. A useful four-valued logic. In J. M. Dunn and G. Epstein, editors, *Modern Uses of Multiple-Valued Logic*, pages 8–37. D. Reidel, Dordrecht, 1977.
3. G. Bruns and P. Godefroid. Generalized model checking: Reasoning about partial state spaces. In Proc. of *CONCUR'00*, volume 1877 of *LNCS*, pages 168–182. Springer, 2000.

4. M. Chechik, B. Devereux, A. Gurfinkel, and S. Easterbrook. Multi-Valued Symbolic Model-Checking. *ACM Transactions on Software Engineering and Methodology*, 12:1–38, 2003.
5. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. of POPL'77*, pages 238–252. ACM Press, New York, 1977.
6. D. Dams and K. S. Namjoshi. The existence of finite abstractions for branching time model checking. In *Proc. of LICS'04*, pages 335–344. IEEE Computer Society Press, 2004.
7. H. Fecher and M. Huth. Ranked predicate abstraction for branching time: Complete, incremental, and precise. In *Proc. of ATVA'06*, volume 4218 of *LNCS*, pages 322–336. Springer, 2006.
8. H. Fecher, M. Huth, H. Schmidt and J. Schönborn. Refinement sensitive formal semantics of state machines with persistent choice. In *Proc. of AVoCS'07*, to appear in ENCTS.
9. R. v. Glabbeek and U. Goltz. Refinement of actions and equivalence notions for concurrent systems. *Acta Informatica*, 37:229–327, 2001.
10. R. Gorrieri and A. Rensink. Action refinement. In J. A. Bergstra, A. Ponse, and S. A. Smolka, editors, *Handbook of Process Algebra*, pages 1047–1147. North-Holland, 2001.
11. M. Huth. Labelled transition systems as a Stone space. *Logical Methods in Computer Science* 1(1):1–28, 2005.
12. M. Jurdzinski. Deciding the winner in parity games is in $UP \cap co-UP$. *Inf. Process. Lett.*, 68(3):119–124, 1998.
13. D. Kozen. Results on the propositional μ -calculus. *Theor. Comput. Sci.*, 27:333–354, 1983.
14. O. Kupferman and Y. Lustig. Lattice Automata. In *Proc. of VMCAI'07*, volume 4349 of *LNCS*, pages 199 – 213. Springer, 2007.
15. O. Kupferman and Y. Lustig. Latticed simulation relations and games. To appear in *Proc. of ATVA'07*.
16. M. Lange and C. Stirling. Model checking games for branching time logics. *J. Log. Comput.*, 12(4):623–639, 2002.
17. Larsen, K. G. and B. Thomsen. *A modal process logic*. In *Proc. of LICS'88*, pp. 203–210, IEEE Computer Society Press.
18. Larsen, K. G. and L. Xinxin. Equation solving using modal transition systems. In *Proc. of LICS'90*, pp. 108–117, IEEE Computer Society Press.
19. Ranko Lazic. *A semantic study of Data Independence with Applications to Model Checking*, DPhil thesis, Oxford University Computing Laboratory, April 1999.
20. R. Milner. *A Calculus for Communicating Systems*, volume 92 of *LNCS*. Springer, 1980.
21. D. Park. Concurrency and automata on infinite sequences. In *Proc. of 5th GI Conference*, volume 104 of *LNCS*, pages 167–183. Springer, 1981.
22. G. Plotkin and C. Stirling. A framework for intuitionistic modal logics. In *Theoretical aspects of reasoning about knowledge*, Monterey, 1986.
23. B. Thomsen. An extended bisimulation induced by a preorder on actions. Master's thesis, Aalborg University Centre, 1987.
24. Th. Wilke. Alternating tree automata, parity games, and modal μ -calculus. In *Bull. Soc. Math. Belg.* 8: 359–391, 2001.
25. L. Zuck and A. Pnueli. Model Checking and Abstraction to the Aid of Parameterized Systems (a survey). In *Computer Languages, Systems, and Structures* 30(3-4): 139–169, 2004.