

Model checking for action abstraction

Harald Fecher¹ Michael Huth²

¹Albert-Ludwigs-Universität Freiburg, Germany

²Imperial College London, United Kingdom

San Francisco, VMCAI 2008

Motivating problem

Properties may be more abstract than the model. Example:

System: stack that communicates with the environment via input action $put(n)$ with $n \in N$ and output action $get(n)$ with $n \in N$.

Property: stack can always be completely emptied via output actions.

Motivating problem

Properties may be more abstract than the model. Example:

System: stack that communicates with the environment via input action $put(n)$ with $n \in N$ and output action $get(n)$ with $n \in N$.

Property: stack can always be completely emptied via output actions.
As μ -calculus formula:

$$\left(\bigwedge_{n \in N} [get(n)]ff \right)$$

Motivating problem

Properties may be more abstract than the model. Example:

System: stack that communicates with the environment via input action $put(n)$ with $n \in N$ and output action $get(n)$ with $n \in N$.

Property: stack can always **be completely emptied via output actions**.
As μ -calculus formula:

$$\mu Y.((\bigvee_{m \in N} \langle get(m) \rangle Y) \vee (\bigwedge_{n \in N} [get(n)]ff))$$

Motivating problem

Properties may be more abstract than the model. Example:

System: stack that communicates with the environment via input action $put(n)$ with $n \in N$ and output action $get(n)$ with $n \in N$.

Property: stack can **always** be completely emptied via output actions.
As μ -calculus formula:

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

Motivating problem

Properties may be more abstract than the model. Example:

System: stack that communicates with the environment via input action $put(n)$ with $n \in N$ and output action $get(n)$ with $n \in N$.

Property: stack can always be completely emptied via output actions.
As μ -calculus formula:

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

Formula too complex.

Motivating problem

Properties may be more abstract than the model. Example:

System: stack that communicates with the environment via input action $put(n)$ with $n \in N$ and output action $get(n)$ with $n \in N$.

Property: stack can always be completely emptied via output actions.
As μ -calculus formula:

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

Formula too complex.

Better use abstract actions:

$$\nu X. ([anyAction]X \wedge \mu Y. (\langle out \rangle Y \vee [out]ff))$$

Satisfaction for concrete systems

Satisfaction is based on the system, M , the property, ϕ , and the action-order, \sqsubseteq .

Intuition for concrete systems (having only maximal, most specialist, actions):

$$\langle a \rangle \phi \equiv \bigvee_{a' \sqsupseteq_{\max} a} \langle a' \rangle \phi \quad \text{and} \quad [a] \phi \equiv \bigwedge_{a' \sqsupseteq_{\max} a} [a'] \phi$$

Satisfaction for concrete systems

Satisfaction is based on the system, M , the property, ϕ , and the action-order, \sqsubseteq .

Intuition for concrete systems (having only maximal, most specialist, actions):

$$\langle a \rangle \phi \equiv \bigvee_{a' \sqsupseteq_{\max} a} \langle a' \rangle \phi \quad \text{and} \quad [a] \phi \equiv \bigwedge_{a' \sqsupseteq_{\max} a} [a'] \phi$$

Without property transformation:

$\langle a \rangle \phi$ holds at s iff there is a transition (s, a', s') with $a' \sqsupseteq_{\max} a$ such that s' satisfies ϕ .

$[a] \phi$ holds at s iff for any transition (s, a', s') with $a' \sqsupseteq_{\max} a$, state s' satisfies ϕ .

Satisfaction for abstract systems

Abstract systems: may contain non-maximal actions.

Satisfaction for abstract systems

Abstract systems: may contain non-maximal actions.

Intuition for concrete systems:

$$\langle a \rangle \phi \equiv \bigvee_{a' \sqsupseteq_{\max} a} \langle a' \rangle \phi \text{ and}$$

$$[a] \phi \equiv \bigwedge_{a' \sqsupseteq_{\max} a} [a'] \phi$$

Satisfaction for abstract systems

Abstract systems: may contain non-maximal actions.

Intuition for abstract systems:

$$\langle a \rangle \phi \equiv \bigvee_{a' \sqsupseteq a} \langle a' \rangle \phi \text{ and}$$

$$[a] \phi \equiv \bigwedge_{a' \sqsupseteq_{\max} a} [a'] \phi$$

Satisfaction for abstract systems

Abstract systems: may contain non-maximal actions.

Intuition for abstract systems:

$$\langle a \rangle \phi \equiv \bigvee_{a' \sqsupseteq a} \langle a' \rangle \phi \text{ and}$$

$$[a] \phi \equiv \bigwedge_{a' \text{ has common specialized action with } a} [a'] \phi$$

Satisfaction has to be preserved under refinement.

Satisfaction for abstract systems

Abstract systems: may contain non-maximal actions.

Intuition for abstract systems:

$\langle a \rangle \phi \equiv \bigvee_{a' \sqsupseteq a} \langle a' \rangle \phi$ and

$[a] \phi \equiv \bigwedge_{a' \text{ has common specialized action with } a} [a'] \phi$

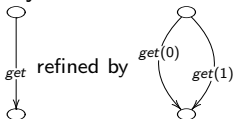
Satisfaction has to be preserved under refinement.

What exact refinement notion is adequate?

Refinement notion

Replace abstract action by

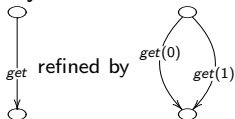
every lower maximal action?



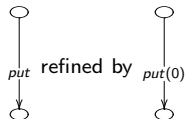
Refinement notion

Replace abstract action by

every lower maximal action?



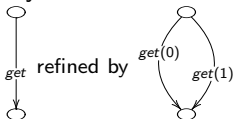
one lower maximal action?



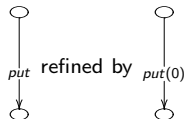
Refinement notion

Replace abstract action by

every lower maximal action?



one lower maximal action?

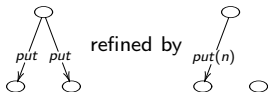


Solution: Replace abstract by **some** lower maximal actions

Refinement notion (2)

How to handle nondeterminism at the abstract level?

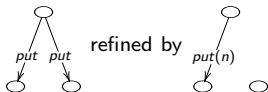
Resolvable



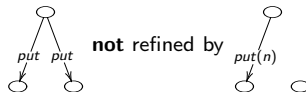
Refinement notion (2)

How to handle nondeterminism at the abstract level?

Resolvable



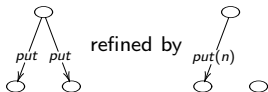
Persistent



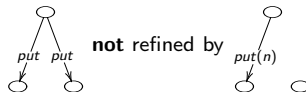
Refinement notion (2)

How to handle nondeterminism at the abstract level?

Resolvable

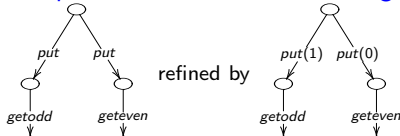


Persistent



Persistent: allows to abstractly enforce different concrete behavior

Example: exist **puts** where afterwards either **getodd** or **geteven** possible



Refinement: extended bisimulation

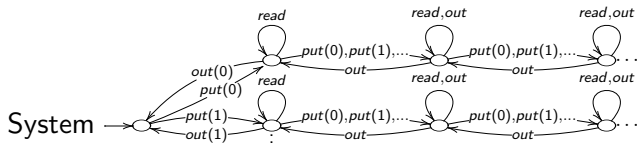
\sqsubseteq -refinement is bisimulation sensitive to order on actions
[Thomsen 87]:

An **abstract** transition has to be matched with a **concrete** one where the **action** is the **same** or a **more** specialized one.

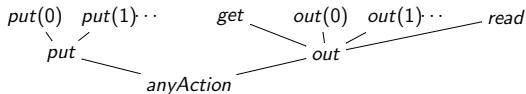
A **concrete** transition has to be matched with an **abstract** one where the **action** is the **same** or a **less** specialized one.

Satisfaction unsound for signature extension

Satisfaction is not preserved when actions are added:



satisfies $\langle put \rangle [out(0)] ff$ for order



How to remedy?

The problem:

- New upper bounds break naive semantics.
- New upper bounds desired for incremental modeling.
- So banning the introduction of new upper bounds is too restrictive.

Partial order with consistency

Remedy to the problem:

Usage of **consistency predicate**, denoted \curvearrowright , saying which pairs of actions can have common specialized versions.

- Consistency is symmetric,
- pairs having upper bound must be consistent, and
- consistency is closed under abstraction of either action.

Consistency predicate is determined by the modeler.

Partial order with consistency

Remedy to the problem:

Usage of **consistency predicate**, denoted \curvearrowright , saying which pairs of actions can have common specialized versions.

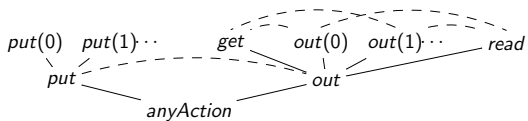
- Consistency is symmetric,
- pairs having upper bound must be consistent, and
- consistency is closed under abstraction of either action.

Consistency predicate is determined by the modeler.

Signature extensions: add actions and/or remove consistencies (without violating above constraints).

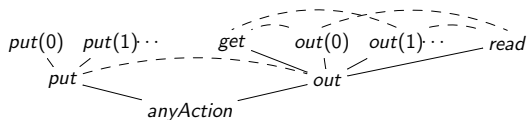
Signature extension

Order with consistency (non-implicit consistency: dashed bows)

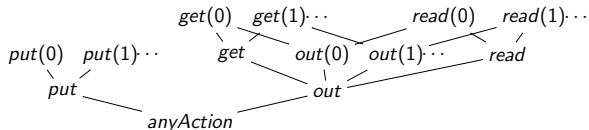


Signature extension

Order with consistency (non-implicit consistency: dashed bows)

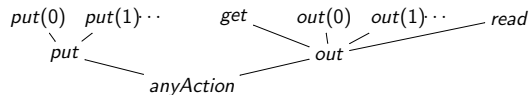


is extended to

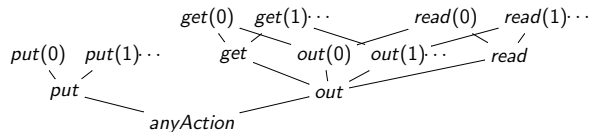


Non-signature-extension

Order with consistency



is **not** extended by



Satisfaction

Intuition for abstract systems and order with consistency:

$$\langle a \rangle \phi \equiv \bigvee_{a' \sqsupseteq a} \langle a' \rangle \phi \text{ and}$$

$$[a] \phi \equiv \bigwedge_{a'} \text{has common specialized action with } a [a'] \phi$$

Satisfaction

Intuition for abstract systems and order with consistency:

$$\langle a \rangle \phi \equiv \bigvee_{a' \sqsupseteq a} \langle a' \rangle \phi \text{ and}$$

$$[a] \phi \equiv \bigwedge_{a' \text{ is consistent with } a} [a'] \phi$$

Satisfaction

Intuition for abstract systems and order with consistency:

$$\langle a \rangle \phi \equiv \bigvee_{a' \sqsupseteq a} \langle a' \rangle \phi \text{ and}$$

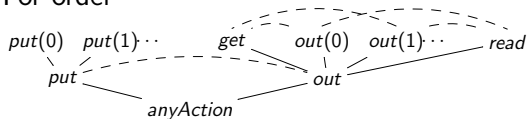
$$[a] \phi \equiv \bigwedge_{a' \text{ is consistent with } a} [a'] \phi$$

$\langle a \rangle \phi$ holds at s iff there is a transition (s, a', s') with $a' \sqsupseteq a$ such that s' satisfies ϕ .

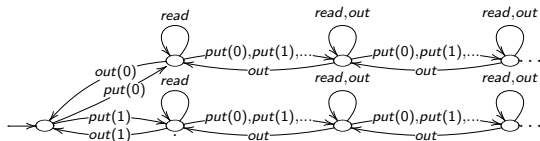
$[a] \phi$ holds at s iff for any transition (s, a', s') with a' consistent with a state s' satisfies ϕ .

Satisfaction example

For order



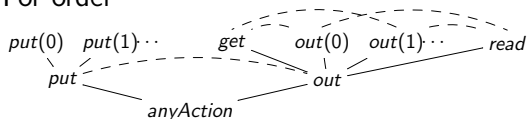
system



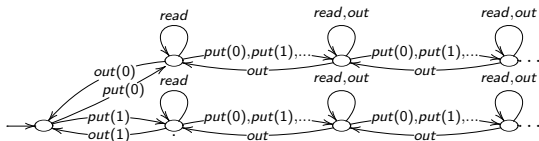
satisfies $\nu X.([anyAction]X \wedge \mu Y.(\langle out \rangle Y \vee [out]ff))$,

Satisfaction example

For order



system

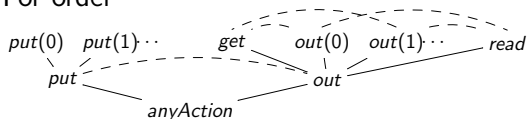


satisfies $\nu X.([anyAction]X \wedge \mu Y.(\langle out \rangle Y \vee [out]ff))$,
 but not $\langle put \rangle[out(0)]ff$ nor its negation $[put]\langle out(0) \rangle tt$.

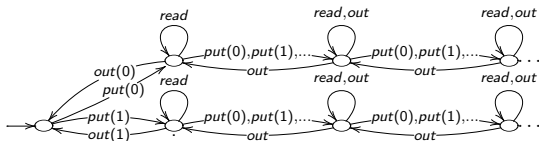
Satisfaction is three-valued

Satisfaction example

For order



system



satisfies $\nu X.([anyAction]X \wedge \mu Y.(\langle out \rangle Y \vee [out]ff))$,
 but not $\langle put \rangle [out(0)]ff$ nor its negation $[put] \langle out(0) \rangle tt$.

Satisfaction is three-valued, not four-valued.

Soundness

Theorem

- $((\text{Act}, \sqsubseteq), \curvearrowright)$ extension of $((\text{Act}', \sqsubseteq'), \curvearrowright')$.
- T_1 transition system over Act , T_2 transition system over Act' such that $T_1 \sqsubseteq$ -refines T_2 .

If $T_2 (\sqsubseteq', \curvearrowright')$ -satisfies ϕ , then $T_1 (\sqsubseteq, \curvearrowright)$ -satisfies ϕ .

Reduction to μ -calculus satisfaction

Reduce our ordered-action satisfaction to classical (unordered) μ -calculus satisfaction.

Advantage of such reduction is the reuse of

- existing theory,
- algorithms, and
- tool support.

First reduction step:

Use intermediate order (for getting finiteness wrt action):

Identify actions whenever they behave equally wrt order and consistency predicate on the actions occurring in the property.

$$\Delta_1 \sqsubseteq' \Delta_2 \text{ iff } \forall a_2 \in \Delta_2: \exists a_1 \in \Delta_1: a_1 \sqsubseteq a_2$$

$$\Delta_1 \frown' \Delta_2 \text{ iff } \exists a_1 \in \Delta_1: \exists a_2 \in \Delta_2: a_1 \frown a_2$$

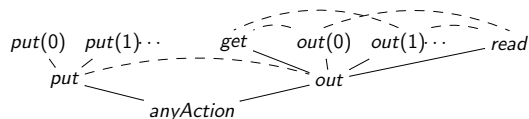
First reduction step:

Use intermediate order (for getting finiteness wrt action):

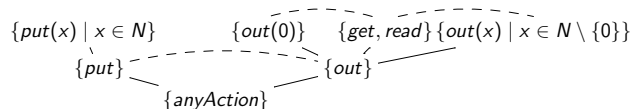
Identify actions whenever they behave equally wrt order and consistency predicate on the actions occurring in the property.

$$\Delta_1 \sqsubseteq' \Delta_2 \text{ iff } \forall a_2 \in \Delta_2: \exists a_1 \in \Delta_1: a_1 \sqsubseteq a_2$$

$$\Delta_1 \frown' \Delta_2 \text{ iff } \exists a_1 \in \Delta_1: \exists a_2 \in \Delta_2: a_1 \frown a_2$$



becomes, wrt property-actions *anyAction*, *out*, *put*, *out(0)*:



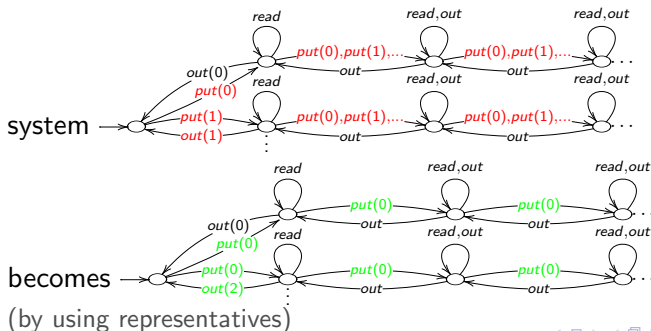
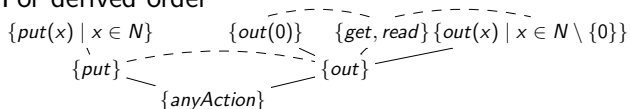
Second reduction step:

Replace labels in transition system by their equivalent class:

Second reduction step:

Replace labels in transition system by their equivalent class:

For derived order



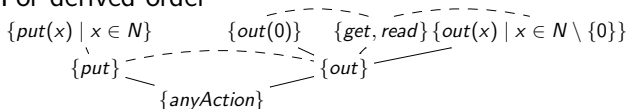
Third reduction step:

Replace $\langle a \rangle \phi$ by $\bigvee_{a' \sqsupseteq a} \langle a' \rangle \phi$,
and $[a] \phi$ by $\bigwedge_{a' \sim a} [a'] \phi$,
adapted to the derived order and actions.

Third reduction step:

Replace $\langle a \rangle \phi$ by $\bigvee_{a' \supseteq a} \langle a' \rangle \phi$,
 and $[a] \phi$ by $\bigwedge_{a' \sim_a} [a'] \phi$,
 adapted to the derived order and actions.

For derived order



property $\langle put \rangle [out(0)] ff$ becomes

$$(\langle put(0) \rangle \underbrace{([out(0)]ff \wedge [read]ff \wedge [out]ff \wedge [anyAction]ff)}_{\hat{\phi}}) \vee (\langle put \rangle \hat{\phi})$$

Correctness

Reduction is sound and complete:

Theorem

$T \models_{\underline{\underline{E}}} \phi$ iff transformed T satisfies transformed ϕ via μ -calculus satisfaction.

Related work

Lattice automata (map transitions to lattice-elements):

We do **not** use lattice operations. We do **not** want lattices: `put` and `get` should not have common specialized actions.

Action refinement (replace action by a unique process):

We replace an action by a number of actions, differing from instance to instance.

Parameterized systems:

We have local parameters: irreducible to parameterized systems in case of recursion.

Conclusion

- Illustrated: usefulness of abstract actions in properties.
- Illustrated: naive satisfaction unsound for signature extension.
- Defined: ordered action with consistency relation; sound satisfaction for signature extension.
- Developed: reduction of our satisfaction to classical μ -calculus satisfaction.