Modal transition systems: a foundation for the analysis of partial systems

Michael Huth
Computing and Information Sciences
Kansas State University

Joint work with Patrice Godefroid, Radha Jagadeesan, and David Schmidt.

Outline

1. Representing and analyzing partial systems

Outline

1. Representing and analyzing partial systems

2. Modal transition systems

Outline

1. Representing and analyzing partial systems
2. Modal transition systems
3. Property refutation and verification

Outline

1. Representing and analyzing partial systems

2. Modal transition systems

3. Property refutation and verification

4. Three instances of framework

Outline

1. Representing and analyzing partial systems

2. Modal transition systems

3. Property refutation and verification

4. Three instances of framework

5. Instrumented model checking

Outline

1. Representing and analyzing partial systems

2. Modal transition systems

3. Property refutation and verification

4. Three instances of framework

5. Instrumented model checking

6. Computing abstractions

Outline

1. Representing and analyzing partial systems

2. Modal transition systems

3. Property refutation and verification

4. Three instances of framework

5. Instrumented model checking

6. Computing abstractions

7. Assume-guarantee model checking

Outline

1. Representing and analyzing partial systems
2. Modal transition systems
3. Property refutation and verification
4. Three instances of framework
5. Instrumented model checking
6. Computing abstractions
7. Assume-guarantee model checking
8. Conclusions and future work

1 Representing and analyzing partial systems

Specification & analysis models are inherently *uncertain*. E.g.,

1 Representing and analyzing partial systems

Specification & analysis models are inherently *uncertain*. E.g.,

- non-determinism present in concurrent programs;

1 Representing and analyzing partial systems

Specification & analysis models are inherently *uncertain*. E.g.,

- non-determinism present in concurrent programs;
- "loose" system specifications; and

1 Representing and analyzing partial systems

Specification & analysis models are inherently *uncertain*. E.g.,

- non-determinism present in concurrent programs;
- "loose" system specifications; and
- imprecise control-flow graphs for inter-procedural flow analysis.

1 Representing and analyzing partial systems

Specification & analysis models are inherently *uncertain*. E.g.,

- non-determinism present in concurrent programs;
- "loose" system specifications; and
- imprecise control-flow graphs for inter-procedural flow analysis.

Specification formalisms (Z, UML, OCL, CFGs, etc) require "explicit" uncertainty to allow *partial* specification & analysis.

1 Representing and analyzing partial systems

Specification & analysis models are inherently *uncertain*. E.g.,

- non-determinism present in concurrent programs;
- "loose" system specifications; and
- imprecise control-flow graphs for inter-procedural flow analysis.

Specification formalisms (Z, UML, OCL, CFGs, etc) require "explicit" uncertainty to allow *partial* specification & analysis.

Modal transition systems express such aspects and generalize Dams's work (Dams 1996) to *partial* systems.

2 Modal transition systems

(Larsen & Thomsen 1988):

- state transition diagrams and bisimulation (Clarke et al. 1989) are too restrictive for refinement and implementation work;

2 Modal transition systems

(Larsen & Thomsen 1988):

- state transition diagrams and bisimulation (Clarke et al. 1989) are too restrictive for refinement and implementation work;
- propose systems whose state-transitions have *modalities*:

2 Modal transition systems

(Larsen & Thomsen 1988):

- state transition diagrams and bisimulation (Clarke et al. 1989) are too restrictive for refinement and implementation work;
- propose systems whose state-transitions have *modalities*:
  - an a-transition from state s to s' must occur (has to be realized in *all* refinements): $s \rightarrow^{a}_{\text{must}} s'$.

2 Modal transition systems

(Larsen & Thomsen 1988):

- state transition diagrams and bisimulation (Clarke et al. 1989) are too restrictive for refinement and implementation work;
- propose systems whose state-transitions have *modalities*:
  - an a-transition from state s to s' must occur (has to be realized in *all* refinements): $s \to^{a}_{\text{must}} s'$.
  - an a-transition from state s to s' may occur (has to be realized in *some* refinement; *uncertainty*): $s \to^{a}_{\text{may}} s'$;

2 Modal transition systems

(Larsen & Thomsen 1988):

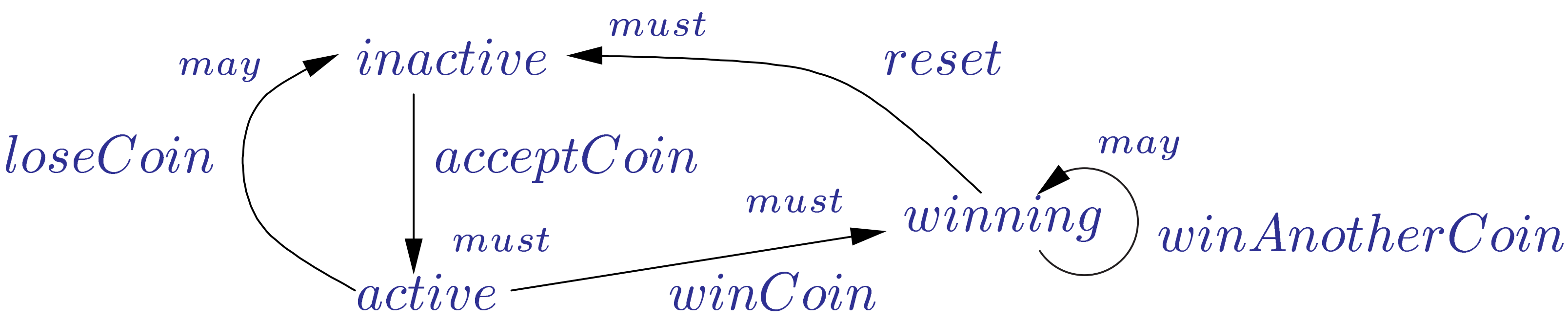
- state transition diagrams and bisimulation (Clarke et al. 1989) are too restrictive for refinement and implementation work;
- propose systems whose state-transitions have *modalities*:
  - an a-transition from state s to s' must occur (has to be realized in *all* refinements): $s \rightarrow^{a}_{\mathrm{must}} s'$.
  - an a-transition from state s to s' may occur (has to be realized in *some* refinement; *uncertainty*): $s \rightarrow^{a}_{\mathrm{may}} s'$;
  - an a-transition from state s to s' must not occur (has to be absent from all refinements): not $s \rightarrow^{a}_{\mathrm{may}} s'$;

2 Modal transition systems

(Larsen & Thomsen 1988):

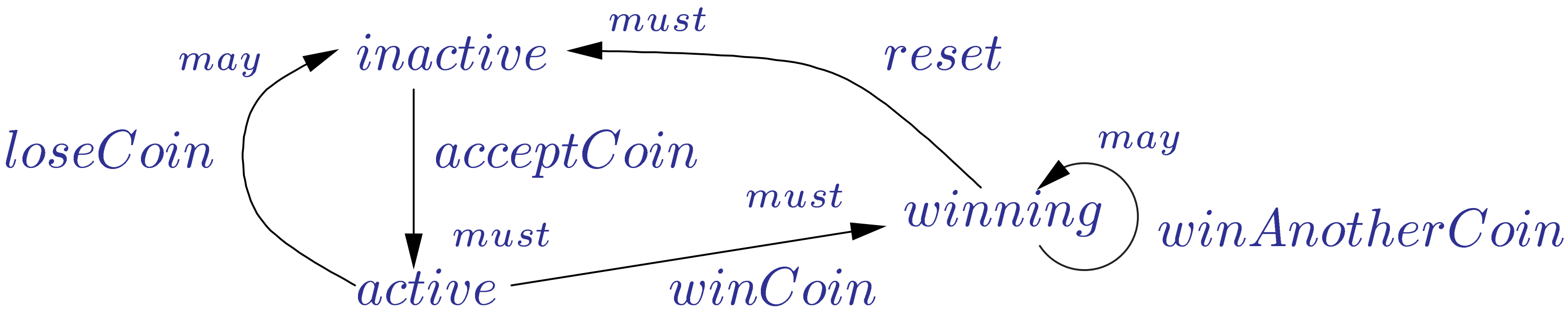
- state transition diagrams and bisimulation (Clarke et al. 1989) are too restrictive for refinement and implementation work;
- propose systems whose state-transitions have *modalities*:
  - an a-transition from state s to s' must occur (has to be realized in *all* refinements): $s \rightarrow^{a}_{\mathrm{must}} s'$.
  - an a-transition from state s to s' may occur (has to be realized in *some* refinement; *uncertainty*): $s \rightarrow^{a}_{\mathrm{may}} s'$;
  - an a-transition from state s to s' must not occur (has to be absent from all refinements): not $s \rightarrow^{a}_{\mathrm{may}} s'$;
- impose a consistency condition: $\rightarrow_{\mathrm{must}} \subseteq \rightarrow_{\mathrm{may}}$ (all required behavior is also possible).

A slot machine: specification

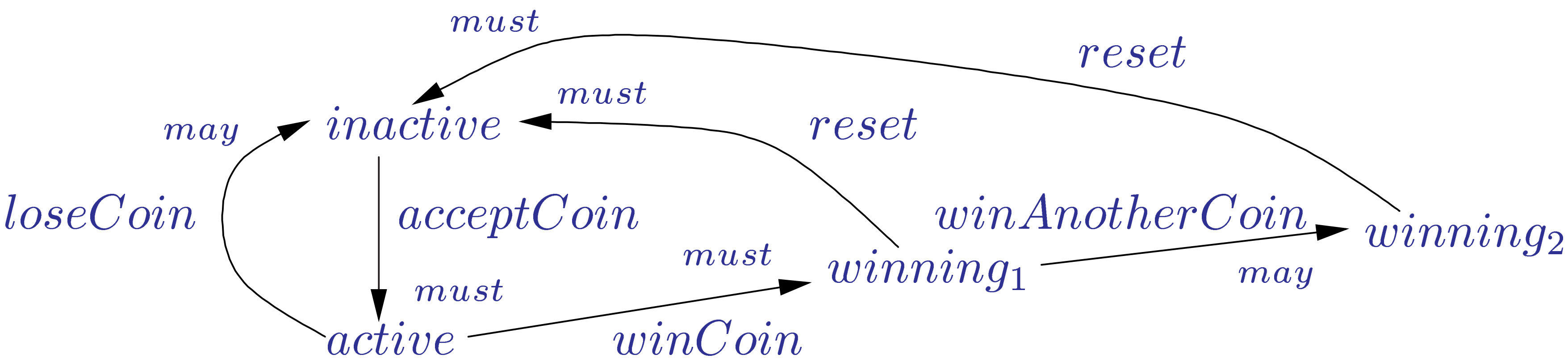


Consistency condition: *must*-transitions comprise $\rightarrow_{\mathrm{must}}$, *may*-and *must*-transitions comprise $\rightarrow_{\mathrm{may}}$.

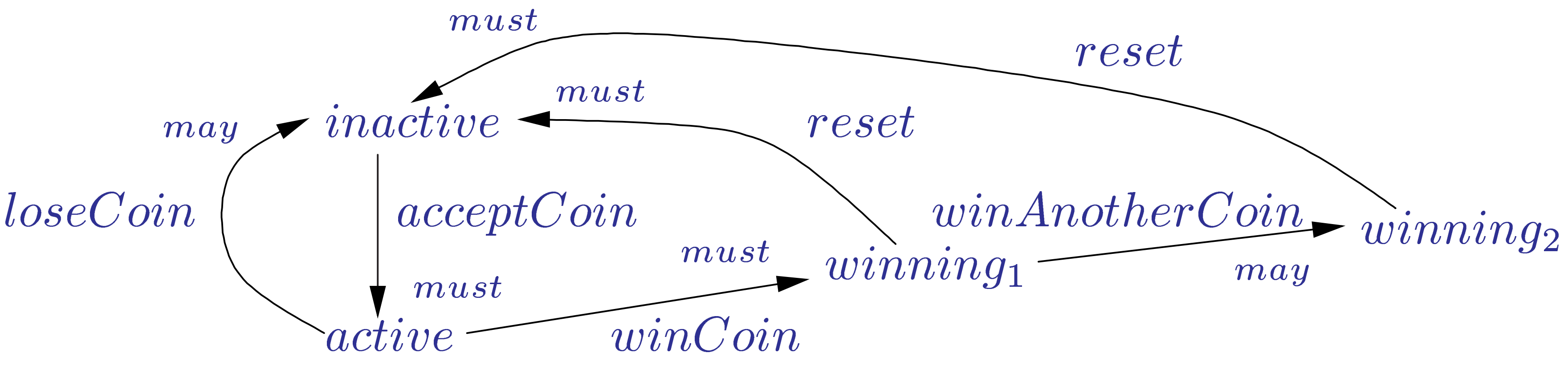
A slot machine: specification and refinement



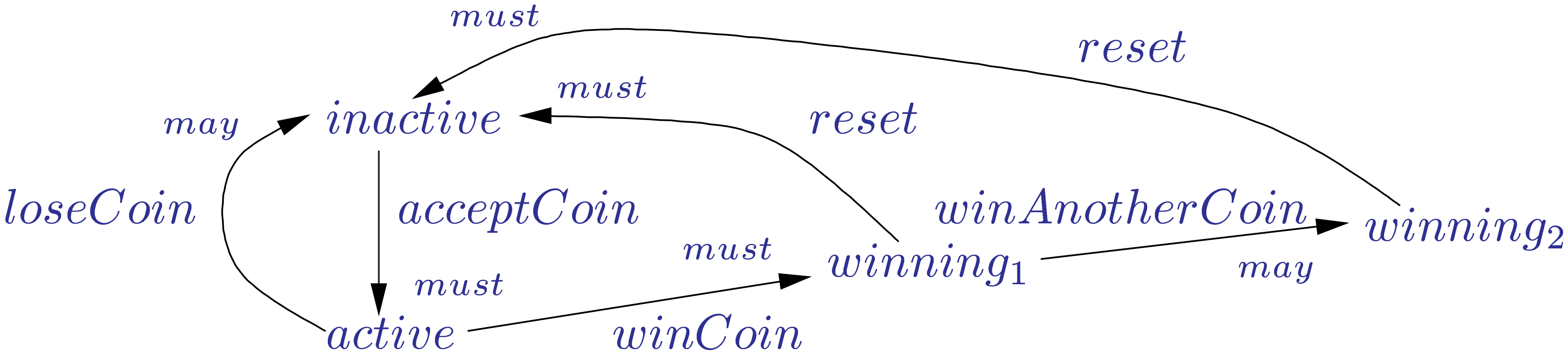
Consistency condition: *must*-transitions comprise $\rightarrow_{\text{must}}$, *may*-and *must*-transitions comprise $\rightarrow_{\text{may}}$.



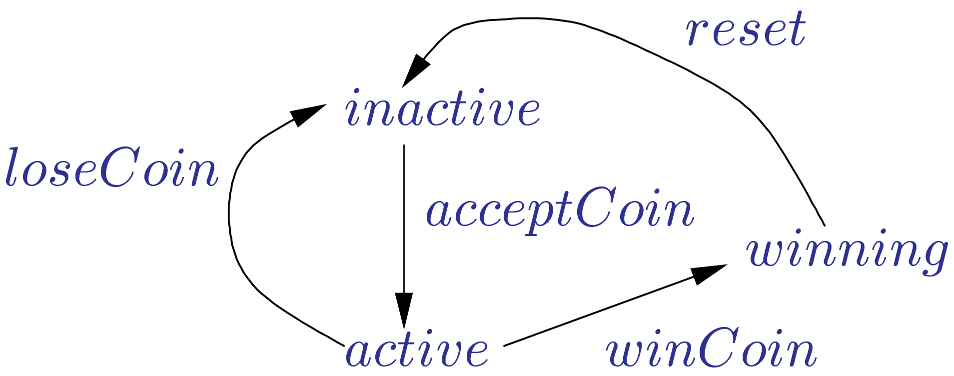
A slot machine: refinement



A slot machine: refinement and implementation



In implementations, all transitions are *must*-transitions:



Formal refinement and abstraction

State s *refines* state t (and t *abstracts* s) iff

Formal refinement and abstraction

State s *refines* state t (and t *abstracts* s) iff

1. (Larsen 1989) whenever $t \rightarrow^{a}_{\text{must}} t'$, there exists some s' that refines t' such that $s \rightarrow^{a}_{\text{must}} s'$;

Formal refinement and abstraction

State s *refines* state t (and t *abstracts* s) iff

1. (Larsen 1989) whenever $t \rightarrow^{a}_{\text{must}} t'$, there exists some s' that refines t' such that $s \rightarrow^{a}_{\text{must}} s'$;

2. (Larsen 1989) whenever $s \rightarrow^{a}_{\text{may}} s'$, there exists some t' that is refined by s' such that $t \rightarrow^{a}_{\text{may}} t'$;

Formal refinement and abstraction

State s *refines* state t (and t *abstracts* s) iff

1. (Larsen 1989) whenever $t \rightarrow^{a}_{\text{must}} t'$, there exists some s' that refines t' such that $s \rightarrow^{a}_{\text{must}} s'$;

2. (Larsen 1989) whenever $s \rightarrow^{a}_{\text{may}} s'$, there exists some t' that is refined by s' such that $t \rightarrow^{a}_{\text{may}} t'$;

3. $L^{must}(t) \subseteq L^{must}(s)$; and

4. $L^{may}(s) \subseteq L^{may}(t)$;

Formal refinement and abstraction

State s *refines* state t (and t *abstracts* s) iff

1. (Larsen 1989) whenever $t \rightarrow^{a}_{\mathrm{must}} t'$, there exists some s' that refines t' such that $s \rightarrow^{a}_{\mathrm{must}} s'$;
2. (Larsen 1989) whenever $s \rightarrow^{a}_{\mathrm{may}} s'$, there exists some t' that is refined by s' such that $t \rightarrow^{a}_{\mathrm{may}} t'$;
3. $L^{must}(t) \subseteq L^{must}(s)$; and
4. $L^{may}(s) \subseteq L^{may}(t)$;

where $L^{must}(t)$ ($L^{may}(t)$) are those state-propositions that must (may) be true at t.

Formal refinement and abstraction

State s *refines* state t (and t *abstracts* s) iff

1. (Larsen 1989) whenever $t \rightarrow^{a}_{\text{must}} t'$, there exists some s' that refines t' such that $s \rightarrow^{a}_{\text{must}} s'$;

2. (Larsen 1989) whenever $s \rightarrow^{a}_{\text{may}} s'$, there exists some t' that is refined by s' such that $t \rightarrow^{a}_{\text{may}} t'$;

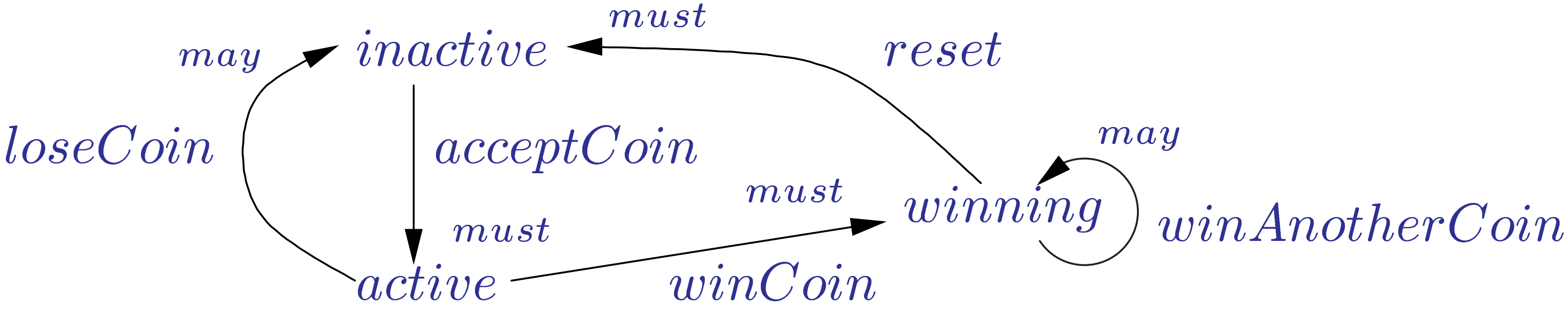
3. $L^{must}(t) \subseteq L^{must}(s)$; and

4. $L^{may}(s) \subseteq L^{may}(t)$;

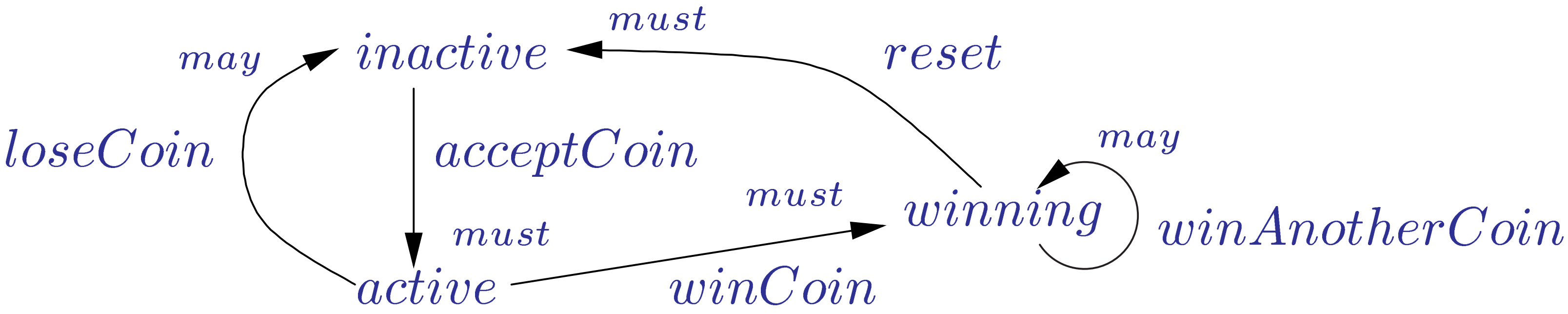
where $L^{must}(t)$ ($L^{may}(t)$) are those state-propositions that must (may) be true at t.

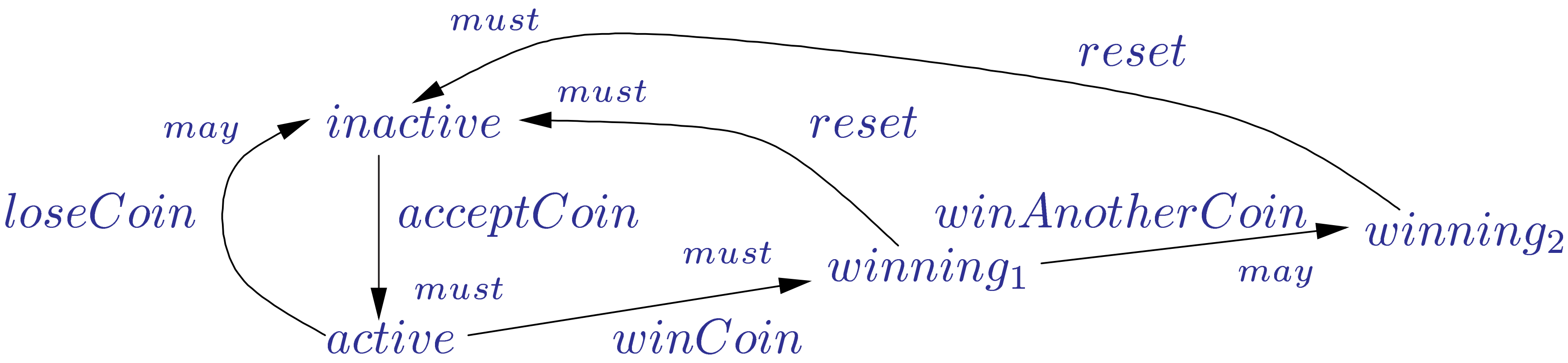
We write $\prec$ for the greatest such relation.

A slot machine: the witness for refinement

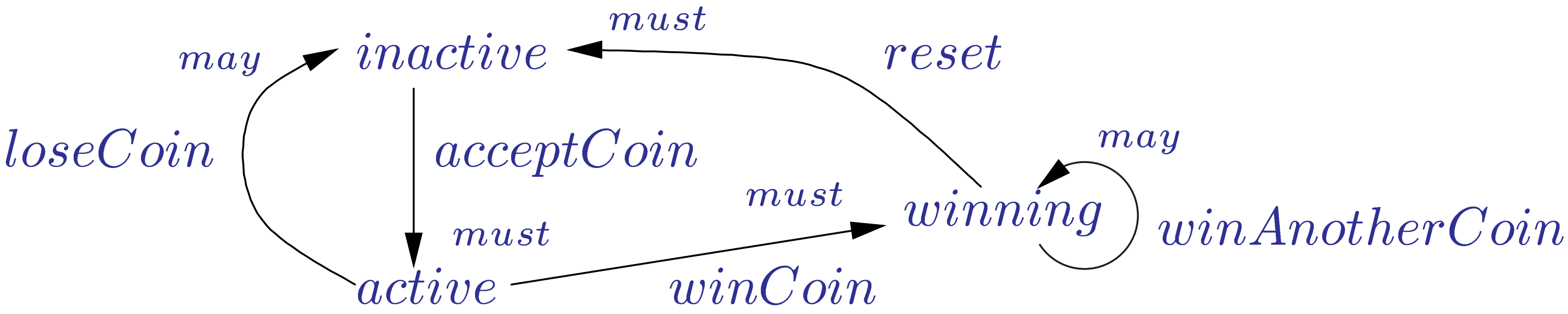


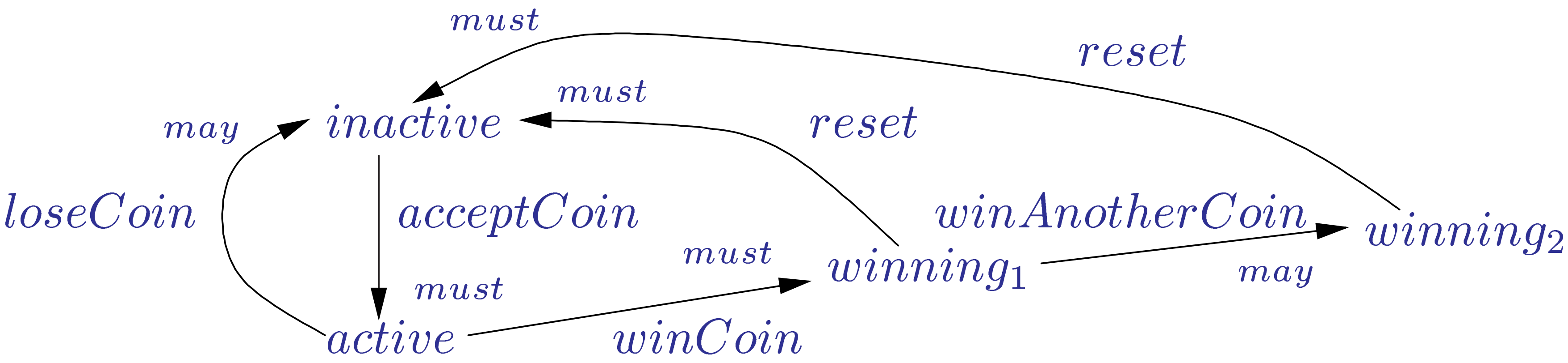
A slot machine: the witness for refinement





A slot machine: the witness for refinement





Refinement:

$\{(winning_1, winning), (winning_2, winning), (inactive, inactive), (active, active)\}$.

3 Property refutation and verification

Two satisfaction relations:

3 Property refutation and verification

Two satisfaction relations:

- $t \models^{\mathrm{nec}} \phi$: under-approximates "*all* refinements of t satisfy ϕ";

3 Property refutation and verification

Two satisfaction relations:

- $t \models^{\mathrm{nec}} \phi$: under-approximates "*all* refinements of t satisfy ϕ";
- $t \models^{\mathrm{pos}} \phi$: over-approximates "*some* refinement of t satisfies ϕ".

3 Property refutation and verification

Two satisfaction relations:

- $t \models^{\mathrm{nec}} \phi$: under-approximates "*all* refinements of t satisfy ϕ";
- $t \models^{\mathrm{pos}} \phi$: over-approximates "*some* refinement of t satisfies ϕ".

Sound framework for refutation and verification:

3 Property refutation and verification

Two satisfaction relations:

- $t \models^{\mathrm{nec}} \phi$: under-approximates "*all* refinements of t satisfy ϕ";
- $t \models^{\mathrm{pos}} \phi$: over-approximates "*some* refinement of t satisfies ϕ".

Sound framework for refutation and verification:

- To *verify* ϕ at state t, establish $t \models^{\mathrm{nec}} \phi$.

3 Property refutation and verification

Two satisfaction relations:

- $t \models^{\text{nec}} \phi$: under-approximates "*all* refinements of t satisfy ϕ";
- $t \models^{\text{pos}} \phi$: over-approximates "*some* refinement of t satisfies ϕ".

Sound framework for refutation and verification:

- To *verify* ϕ at state t, establish $t \models^{\text{nec}} \phi$.
- To *refute* ϕ at state t, establish $t \models^{\text{nec}} \neg\phi$.

3 Property refutation and verification

Two satisfaction relations:

- $t \models^{\mathrm{nec}} \phi$: under-approximates "*all* refinements of t satisfy ϕ";
- $t \models^{\mathrm{pos}} \phi$: over-approximates "*some* refinement of t satisfies ϕ".

Sound framework for refutation and verification:

- To *verify* ϕ at state t, establish $t \models^{\mathrm{nec}} \phi$.
- To *refute* ϕ at state t, establish $t \models^{\mathrm{nec}} \neg\phi$.

Related work: Cousot & Cousot (2000) define a *universal* and an *existential* abstraction; the latter one, used *in isolation* restricts sound reasoning to *safety* properties.

Temporal logic

Kozen's modal mu-calculus (Kozen 1983) (subsumes tool logics):

$$\phi ::= \bot \mid p \mid Z \mid \neg\phi \mid \phi \wedge \phi \mid \langle a \rangle\, \phi \mid \mu Z.\phi$$

Temporal logic

Kozen's modal mu-calculus (Kozen 1983) (subsumes tool logics):

$$\phi ::= \bot \mid p \mid Z \mid \neg\phi \mid \phi \wedge \phi \mid \langle a \rangle\, \phi \mid \mu Z.\phi$$

Semantic highlights — negation and $\models^{\mathrm{pos}}$ *not* considered in (Larsen 1989):

Temporal logic

Kozen's modal mu-calculus (Kozen 1983) (subsumes tool logics):

$$\phi ::= \bot \mid p \mid Z \mid \neg\phi \mid \phi \wedge \phi \mid \langle a \rangle\, \phi \mid \mu Z.\phi$$

Semantic highlights — negation and $\models^{\text{pos}}$ *not* considered in (Larsen 1989):

- $s \models^{\text{nec}} p$ iff $p \in L^{must}(s)$;

Temporal logic

Kozen's modal mu-calculus (Kozen 1983) (subsumes tool logics):

$$\phi ::= \bot \mid p \mid Z \mid \neg\phi \mid \phi \wedge \phi \mid \langle a \rangle\, \phi \mid \mu Z.\phi$$

Semantic highlights — negation and $\models^{\text{pos}}$ *not* considered in (Larsen 1989):

- $s \models^{\text{nec}} p$ iff $p \in L^{must}(s)$;
- $s \models^{\text{nec}} \neg\phi$ iff not $s \models^{\text{pos}} \phi$ (mutual induction: Kelb 1994);

Temporal logic

Kozen's modal mu-calculus (Kozen 1983) (subsumes tool logics):

$$\phi ::= \bot \mid p \mid Z \mid \neg\phi \mid \phi \wedge \phi \mid \langle a \rangle\, \phi \mid \mu Z.\phi$$

Semantic highlights — negation and $\models^{\text{pos}}$ *not* considered in (Larsen 1989):

- $s \models^{\text{nec}} p$ iff $p \in L^{must}(s)$;
- $s \models^{\text{nec}} \neg\phi$ iff not $s \models^{\text{pos}} \phi$ (mutual induction: Kelb 1994);
- $s \models^{\text{nec}} \langle a \rangle\, \phi$ iff for some s', $s \to^{a}_{\text{must}} s'$ and $s' \models^{\text{nec}} \phi$.

Temporal logic

Kozen's modal mu-calculus (Kozen 1983) (subsumes tool logics):

$$\phi ::= \bot \mid p \mid Z \mid \neg\phi \mid \phi \wedge \phi \mid \langle a \rangle\, \phi \mid \mu Z.\phi$$

Semantic highlights — negation and $\models^{\mathrm{pos}}$ *not* considered in (Larsen 1989):

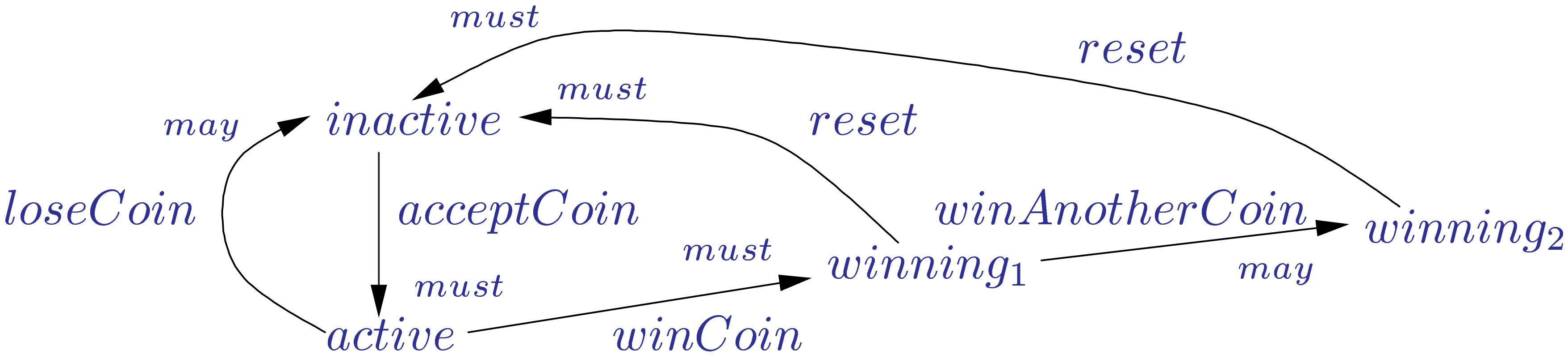
- $s \models^{\mathrm{nec}} p$ iff $p \in L^{must}(s)$;
- $s \models^{\mathrm{nec}} \neg\phi$ iff not $s \models^{\mathrm{pos}} \phi$ (mutual induction: Kelb 1994);
- $s \models^{\mathrm{nec}} \langle a \rangle\, \phi$ iff for some s', $s \rightarrow^{a}_{\mathrm{must}} s'$ and $s' \models^{\mathrm{nec}} \phi$.

The definition of $\models^{\mathrm{pos}}$ is as for $\models^{\mathrm{nec}}$, but we swap $\mathrm{nec} \leftrightarrow \mathrm{pos}$ and $\mathrm{must} \leftrightarrow \mathrm{may}$.

E.g., $s \models^{\mathrm{pos}} \langle a \rangle\, \phi$ iff for some s', $s \rightarrow^{a}_{\mathrm{may}} s'$ and $s' \models^{\mathrm{pos}} \phi$.

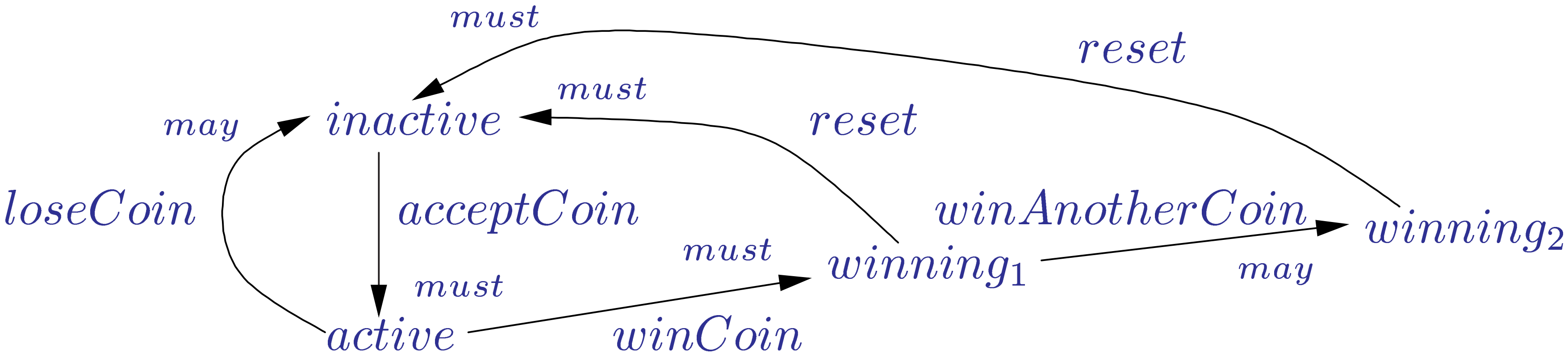
Example refutation

We don't have $\text{inactive} \models^{\text{nec}} \texttt{EF}\,(\text{winning} \wedge \texttt{EX}\,\text{winning})$:

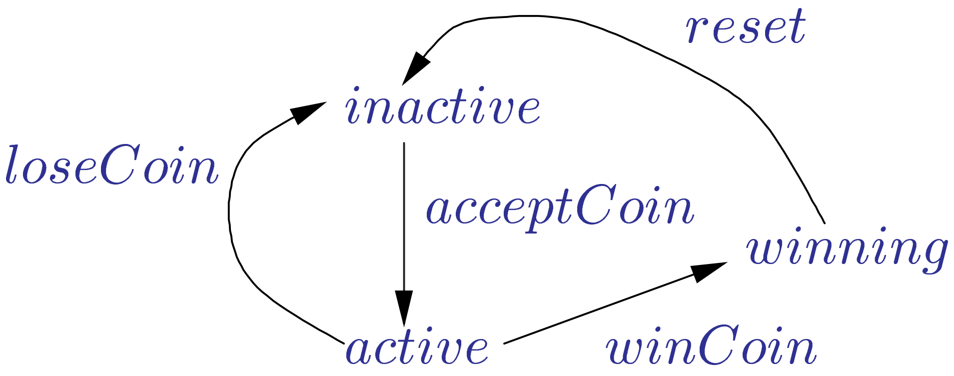


Example refutation

We don't have $\text{inactive} \models^{\text{nec}} \texttt{EF}\,(\text{winning} \wedge \texttt{EX}\,\text{winning})$:

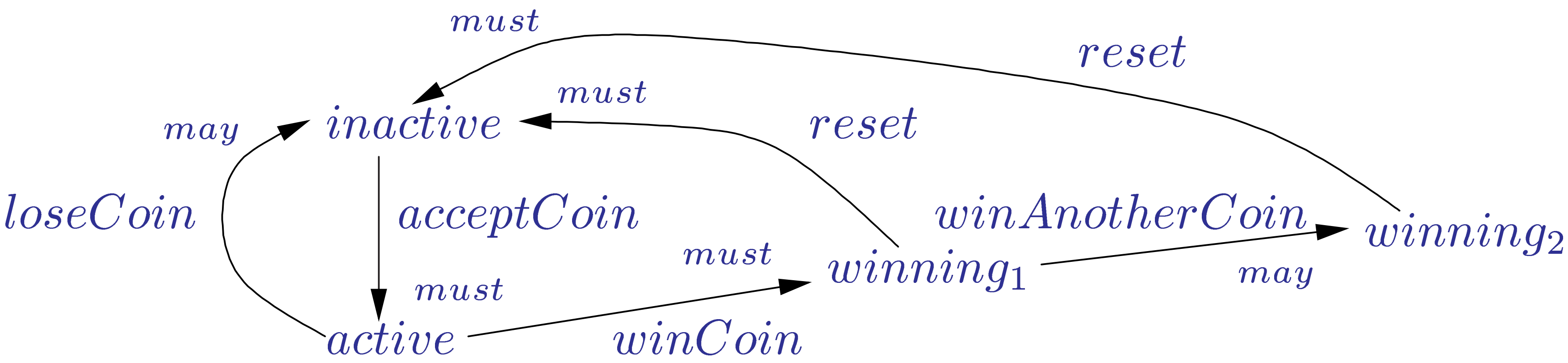


This makes sense as we have
$\text{inactive} \models^{\text{nec}} \neg\texttt{EF}\,(\text{winning} \wedge \texttt{EX}\,\text{winning})$:



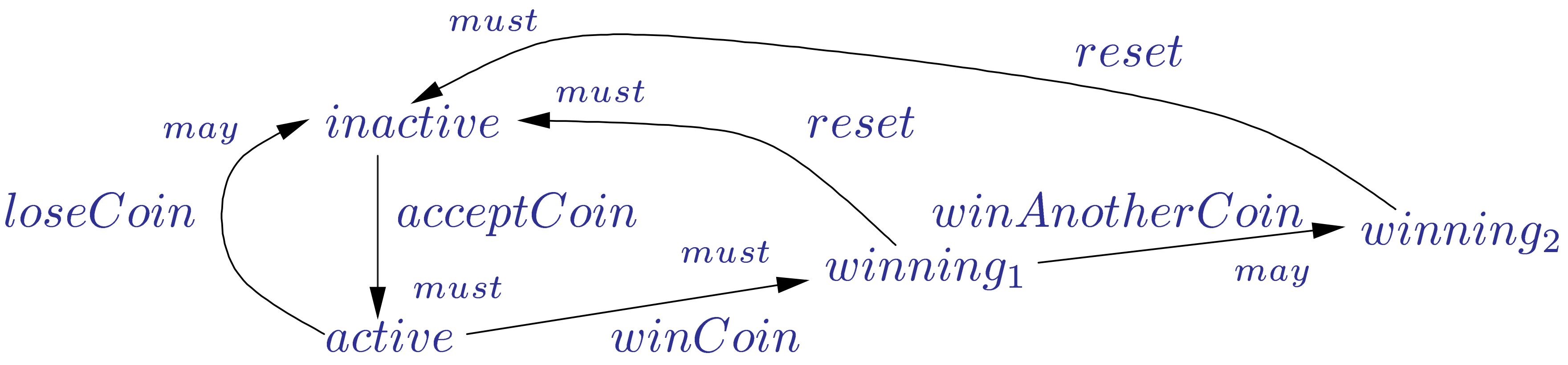
Example verification

We have inactive $\models^{\text{nec}}$ **EF** (winning):

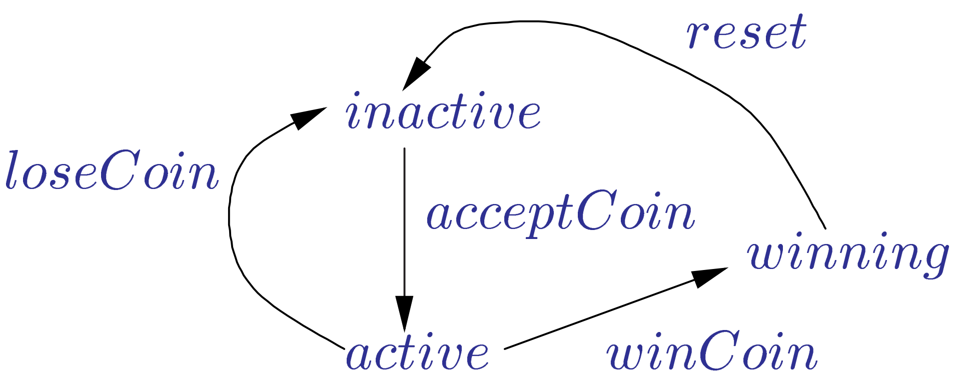


Example verification

We have inactive $\models^{\text{nec}}$ **EF** (winning):



So this property holds for all implementations, in particular for



Sound refutation and verification

For modal transition systems:

Sound refutation and verification

For modal transition systems:

- $t \models^{\text{nec}} \phi$ and $s \prec t$ imply $s \models^{\text{nec}} \phi$ (sound verification);

Sound refutation and verification

For modal transition systems:

- $t \models^{\text{nec}} \phi$ and $s \prec t$ imply $s \models^{\text{nec}} \phi$ (sound verification);
- not $t \models^{\text{nec}} \phi \wedge \neg\phi$ (consistency; sound refutation); and

Sound refutation and verification

For modal transition systems:

- $t \models^{\mathrm{nec}} \phi$ and $s \prec t$ imply $s \models^{\mathrm{nec}} \phi$ (sound verification);
- not $t \models^{\mathrm{nec}} \phi \wedge \neg\phi$ (consistency; sound refutation); and
- $t \models^{\mathrm{pos}} \phi \vee \neg\phi$ ("completeness").

Sound refutation and verification

For modal transition systems:

- $t \models^{\text{nec}} \phi$ and $s \prec t$ imply $s \models^{\text{nec}} \phi$ (sound verification);
- not $t \models^{\text{nec}} \phi \wedge \neg\phi$ (consistency; sound refutation); and
- $t \models^{\text{pos}} \phi \vee \neg\phi$ ("completeness").

Important application (revisited for assume-guarantee model checking): verifying

$$t \models^{\text{nec}} \phi \rightarrow \psi$$

Sound refutation and verification

For modal transition systems:

- $t \models^{\text{nec}} \phi$ and $s \prec t$ imply $s \models^{\text{nec}} \phi$ (sound verification);
- not $t \models^{\text{nec}} \phi \wedge \neg\phi$ (consistency; sound refutation); and
- $t \models^{\text{pos}} \phi \vee \neg\phi$ ("completeness").

Important application (revisited for assume-guarantee model checking): verifying

$$t \models^{\text{nec}} \phi \to \psi$$

means either refuting $t \models^{\text{pos}} \phi$ or verifying $t \models^{\text{nec}} \psi$.

4 Three instances of framework

4 Three instances of framework

(a) extended transition systems

4 Three instances of framework

(a) extended transition systems

(b) partial Kripke structures

4 Three instances of framework

(a) extended transition systems

(b) partial Kripke structures

(c) modal shape graphs

(a) Extended transition systems (Milner 1981)

- are labeled transition systems $\mathcal{E} = (\Sigma_E, \texttt{Act}, \rightarrow)$ together with a *divergence relation* $\uparrow\ \subseteq \Sigma_E \times \texttt{Act}$;

(a) Extended transition systems (Milner 1981)

- are labeled transition systems $\mathcal{E} = (\Sigma_E, \mathtt{Act}, \rightarrow)$ together with a *divergence relation* $\uparrow \subseteq \Sigma_E \times \mathtt{Act}$;
- we write $s \downarrow a$ (intuition: "all a-transitions out of s are present") when $s \uparrow a$ fails to hold;

(a) Extended transition systems (Milner 1981)

- are labeled transition systems $\mathcal{E} = (\Sigma_E, \mathtt{Act}, \rightarrow)$ together with a *divergence relation* $\uparrow\ \subseteq \Sigma_E \times \mathtt{Act}$;
- we write $s \downarrow a$ (intuition: "all a-transitions out of s are present") when $s \uparrow a$ fails to hold;
- example: $\Sigma_E \stackrel{\text{def}}{=} \{\mathtt{callp}, \mathtt{enterp}, \mathtt{runp}, \mathtt{exitp}\}$, with transitions $\mathtt{callp} \rightarrow^{\star} \mathtt{enterp}$, $\mathtt{enterp} \rightarrow^{\star} \mathtt{runp}$, $\mathtt{runp} \rightarrow^{\star} \mathtt{exitp}$, and $\mathtt{runp} \uparrow \star$.

(a) Extended transition systems (Milner 1981)

- are labeled transition systems $\mathcal{E} = (\Sigma_E, \mathtt{Act}, \rightarrow)$ together with a *divergence relation* $\uparrow \subseteq \Sigma_E \times \mathtt{Act}$;
- we write $s \downarrow a$ (intuition: "all a-transitions out of s are present") when $s \uparrow a$ fails to hold;
- example: $\Sigma_E \stackrel{\text{def}}{=} \{\mathtt{callp}, \mathtt{enterp}, \mathtt{runp}, \mathtt{exitp}\}$, with transitions $\mathtt{callp} \rightarrow^\star \mathtt{enterp}$, $\mathtt{enterp} \rightarrow^\star \mathtt{runp}$, $\mathtt{runp} \rightarrow^\star \mathtt{exitp}$, and $\mathtt{runp} \uparrow \star$.
- $\mathcal{E}$ as a modal transition system $\mathcal{M}[\mathcal{E}]$:

$$s \rightarrow_a^{must} s' \quad \text{iff} \quad s \rightarrow_a s'$$
$$s \rightarrow_a^{may} s' \quad \text{iff} \quad s \rightarrow_a s' \text{ or } s \uparrow a.$$

Example requires more than one action label. Then, $\mathtt{runp} \uparrow a$ for all a, possibly except "$\mathtt{exitp}$", "$\mathtt{enterp}$" and "$\mathtt{callp}$".

Partial bisimulation as abstraction

1. Greatest partial bisimulation (Milner 1981) on $\mathcal{E}$ is greatest abstraction on $\mathcal{M}[\mathcal{E}]$.

Partial bisimulation as abstraction

1. Greatest partial bisimulation (Milner 1981) on $\mathcal{E}$ is greatest abstraction on $\mathcal{M}[\mathcal{E}]$.

2. The "intuitionistic" semantics of Hennessy-Milner logic for $s \in \Sigma_E$ matches $\models^{\text{nec}}$ for $\mathcal{M}[\mathcal{E}]$.

Partial bisimulation as abstraction

1. Greatest partial bisimulation (Milner 1981) on $\mathcal{E}$ is greatest abstraction on $\mathcal{M}[\mathcal{E}]$.

2. The "intuitionistic" semantics of Hennessy-Milner logic for $s \in \Sigma_E$ matches $\models^{\mathrm{nec}}$ for $\mathcal{M}[\mathcal{E}]$.

Conclusion: The models, abstraction, and temporal logic semantics for partial bisimulation are a special instance of the corresponding notions for modal transition systems.

(b) Partial Kripke structures

- Bruns & Godefroid (1999) abstract propositional information at states ($L(s,p) \in \{\mathtt{F}, \mathtt{T}\}$) with a three-valued logic $\mathbf{3} \stackrel{\text{def}}{=} \{\bot < \mathtt{F}, \mathtt{T}\}$ (information ordering);

(b) Partial Kripke structures

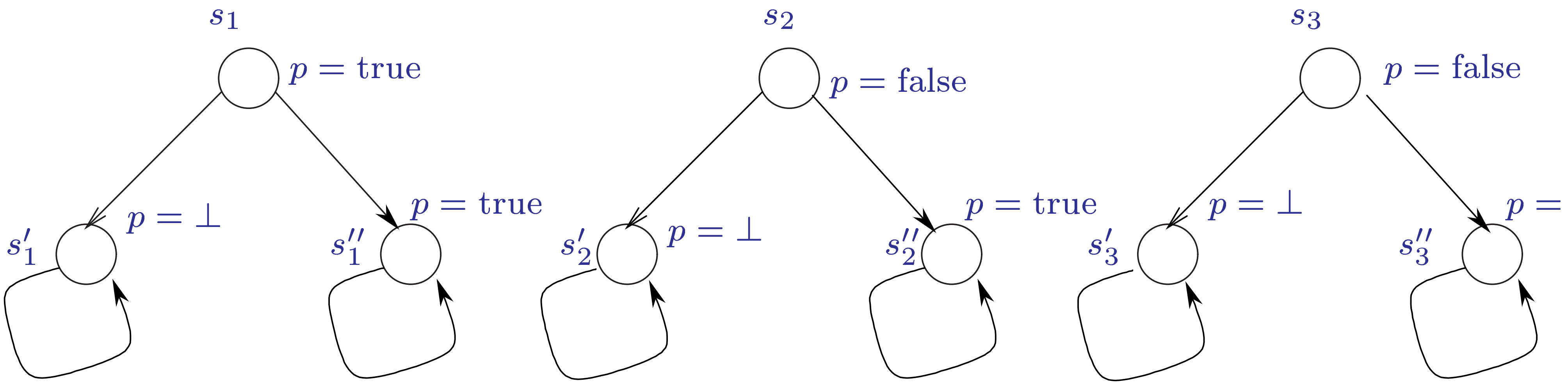
- Bruns & Godefroid (1999) abstract propositional information at states ($L(s,p) \in \{\mathtt{F}, \mathtt{T}\}$) with a three-valued logic $\mathbf{3} \stackrel{\text{def}}{=} \{\perp < \mathtt{F}, \mathtt{T}\}$ (information ordering);
- define an abstraction relation w.r.t. the order on $\mathbf{3}$; and

(b) Partial Kripke structures

- Bruns & Godefroid (1999) abstract propositional information at states ($L(s,p) \in \{\mathtt{F}, \mathtt{T}\}$) with a three-valued logic $\mathbf{3} \stackrel{\text{def}}{=} \{\bot < \mathtt{F}, \mathtt{T}\}$ (information ordering);
- define an abstraction relation w.r.t. the order on $\mathbf{3}$; and
- model check CTL formulas over such partial state spaces.

(b) Partial Kripke structures

- Bruns & Godefroid (1999) abstract propositional information at states ($L(s,p) \in \{\mathtt{F}, \mathtt{T}\}$) with a three-valued logic $\mathbf{3} \stackrel{\text{def}}{=} \{\perp < \mathtt{F}, \mathtt{T}\}$ (information ordering);
- define an abstraction relation w.r.t. the order on $\mathbf{3}$; and
- model check CTL formulas over such partial state spaces.



Conclusion

The order-isomorphism

$$
\begin{aligned}
\mathbf{M}[\Sigma] \quad &\cong \quad \Sigma \to \mathbf{3} \qquad\qquad (\Sigma \text{ discrete}) \\
(\{s \in \Sigma \mid f(s) = \text{true}\}, \{s \in \Sigma \mid f(s) = \perp \text{ or true}\}) \quad &\cong \quad f
\end{aligned}
$$

Conclusion

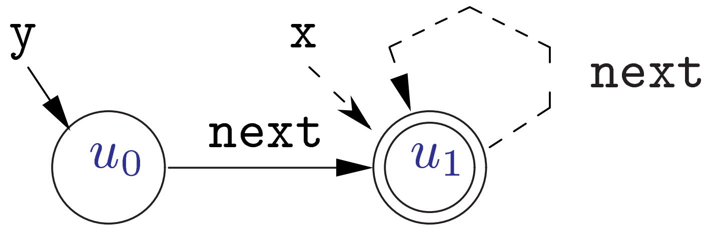
The order-isomorphism

$$\begin{aligned} \mathbf{M}[\Sigma] &\cong \Sigma \to \mathbf{3} \qquad (\Sigma \text{ discrete}) \\ (\{s \in \Sigma \mid f(s) = \mathrm{true}\}, \{s \in \Sigma \mid f(s) = \bot \text{ or } \mathrm{true}\}) &\cong f \end{aligned}$$

recaptures their models, abstraction relation ("completeness ordering" in Bruns (1999)), and CTL semantics as an important instance of the corresponding notions of modal transition systems.

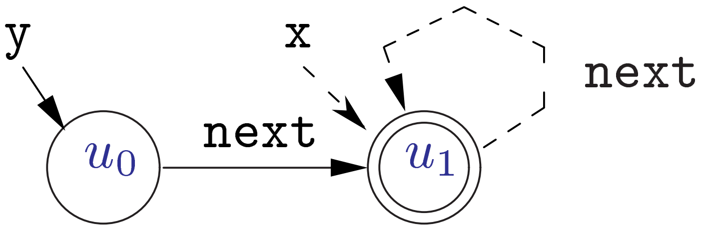
(c) Modal shape graphs

are approximations of concrete contents of heap storage. E.g. the modal shape graph



(c) Modal shape graphs

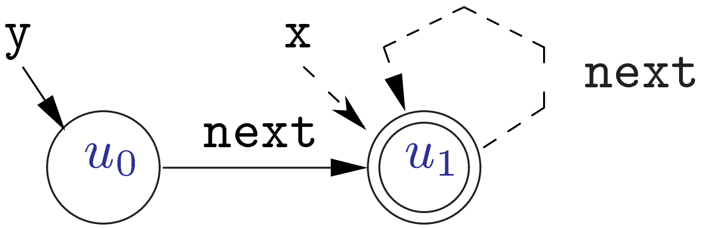
are approximations of concrete contents of heap storage. E.g. the modal shape graph



- approximates a singly linked list of length at least two.

(c) Modal shape graphs

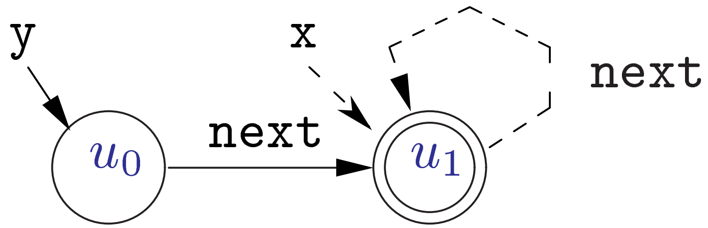
are approximations of concrete contents of heap storage. E.g. the modal shape graph



- approximates a singly linked list of length at least two.
- Local variables, `x` and `y`, point to objects.

(c) Modal shape graphs

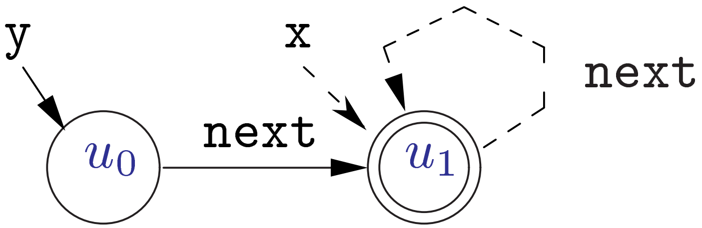
are approximations of concrete contents of heap storage. E.g. the modal shape graph



- approximates a singly linked list of length at least two.
- Local variables, `x` and `y`, point to objects.
- Objects have arcs to `next` fields; solid ones are must-, dotted ones are may-transitions.

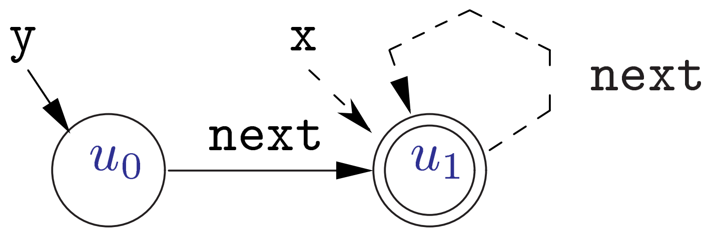
(c) Modal shape graphs

are approximations of concrete contents of heap storage. E.g. the modal shape graph

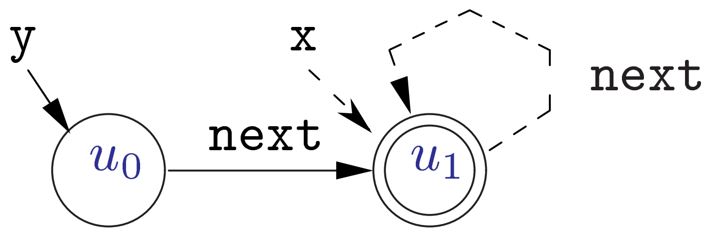


- approximates a singly linked list of length at least two.
- Local variables, `x` and `y`, point to objects.
- Objects have arcs to `next` fields; solid ones are must-, dotted ones are may-transitions.
- Double-circled objects are "summary nodes", possibly representing more than one object.

The shape graph



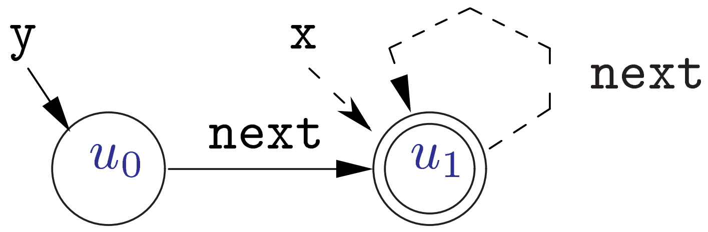
The shape graph and its coding via predicates



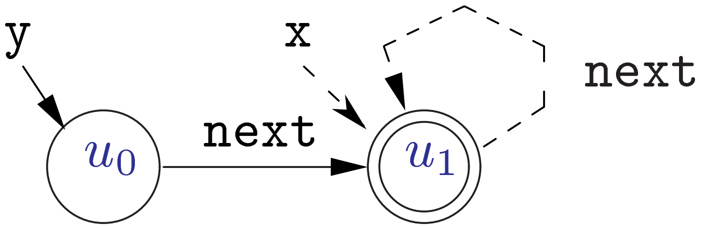
M. Sagiv, T. Reps, and R. Wilhelm (1999) use Kleene's ordering $0 < 1/2 < 1$: $\neg l = 1 - l$; $\wedge$ and $\vee$ are min and max (respectively).

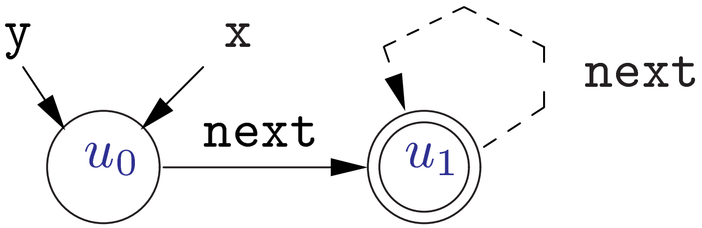
$x(u_0) = 0$	$next(u_0, u_0) = 0$
$x(u_1) = 1/2$	$next(u_0, u_1) = 1$
$y(u_0) = 1$	$next(u_1, u_0) = 0$
$y(u_1) = 0$	$next(u_1, u_1) = 1/2$
$sm(u_0) = 0$	proposition for summary nodes
$sm(u_1) = 1.$	

The same shape graph



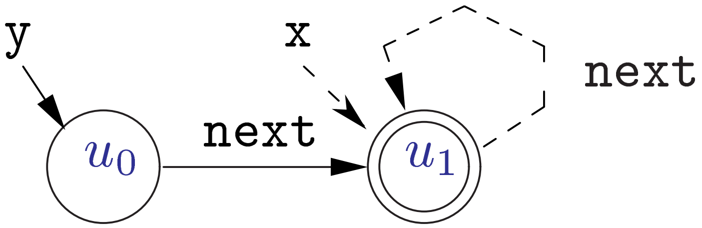
The same shape graph and the effect of `x = y`

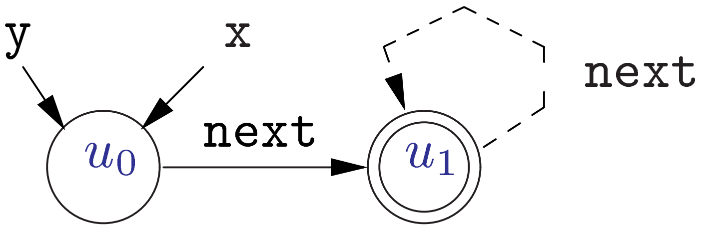




Note how the may-pointer `x` turns into a must-pointer.

The same shape graph and the effect of `x = y`

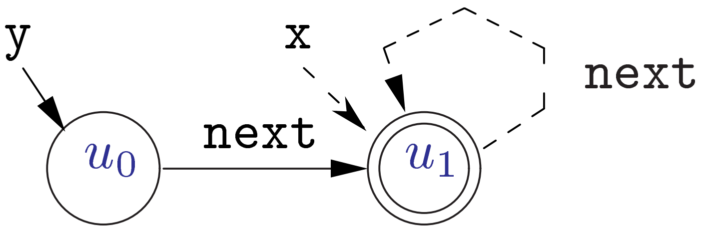




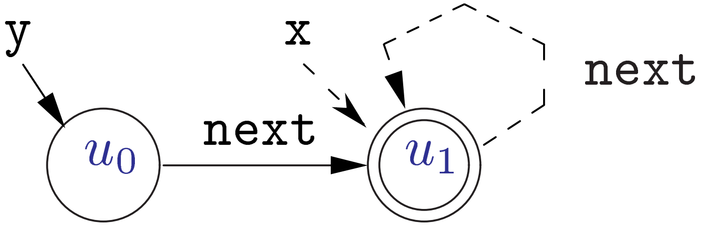
Note how the may-pointer `x` turns into a must-pointer. The general transfer function is:

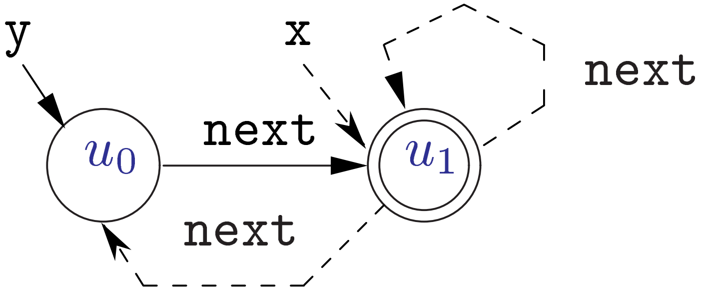
$$T[\texttt{x = y}] : x'(v) = y(v);\ \ \text{all other predicates } p' = p.$$

The shape graph



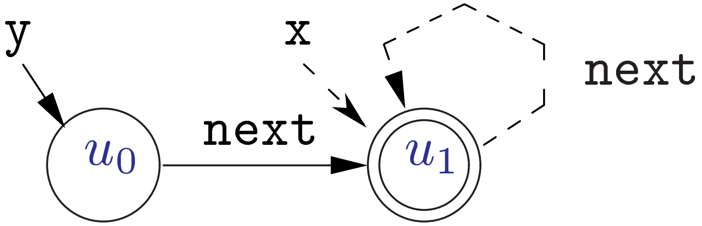
The shape graph and the effect of `x.next = y`

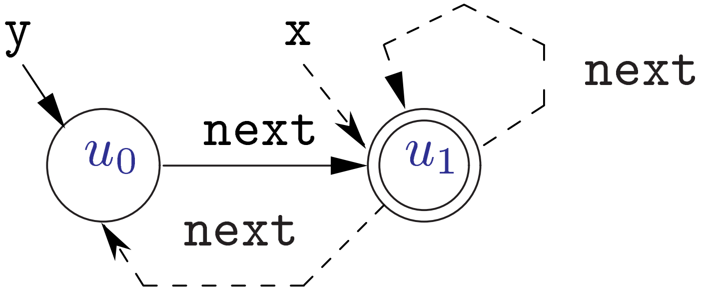




Note how the may-transition out of the summary node persists.

The shape graph and the effect of `x.next = y`





Note how the may-transition out of the summary node persists.

The general transfer function is:

$$T[\texttt{x.next = y}] : next'(v_1, v_2) = next(v_1, v_2) \wedge (sm(v_1) \vee \neg x(v_1)) \vee (x(v_1) \wedge y(v_2)); \text{ all other predicates } p' = p.$$

Shape (data-flow) analysis

1. A control-flow graph is extracted from the source program, where transfer functions (e.g. T[`x.next = y`]) annotate the arcs of the graph;

Shape (data-flow) analysis

1. A control-flow graph is extracted from the source program, where transfer functions (e.g. $T[$`x.next = y`$]$) annotate the arcs of the graph;

2. a finite-height sup-semilattice of data-flow values, based on shape graphs, is defined;

Shape (data-flow) analysis

1. A control-flow graph is extracted from the source program, where transfer functions (e.g. $T[$`x.next = y`$]$) annotate the arcs of the graph;

2. a finite-height sup-semilattice of data-flow values, based on shape graphs, is defined;

3. at the nodes (program points) of the control-flow graph, values of the sup-semilattice will be collected according to the meaning of transfer functions;

Shape (data-flow) analysis

1. A control-flow graph is extracted from the source program, where transfer functions (e.g. $T[$`x.next = y`$]$) annotate the arcs of the graph;

2. a finite-height sup-semilattice of data-flow values, based on shape graphs, is defined;

3. at the nodes (program points) of the control-flow graph, values of the sup-semilattice will be collected according to the meaning of transfer functions;

4. a least fixed-point calculation is undertaken on the flow equations induced by the control-flow graph.

Two collection semantics

In item 3,

- a *single* shape graph might be collected at each node ("independent attribute" analysis) or

Two collection semantics

In item 3,

- a *single* shape graph might be collected at each node ("independent attribute" analysis) or
- a set of shape graphs might be collected ("relational" analysis, as in Sagiv et al. 1999).

Two collection semantics

In item 3,

- a *single* shape graph might be collected at each node ("independent attribute" analysis) or
- a set of shape graphs might be collected ("relational" analysis, as in Sagiv et al. 1999).

In the former case, the join operation is the pointwise three-valued $\wedge$ which weakens solid arcs into dotted ones when dissimilar graphs are joined.

Two collection semantics

In item 3,

- a *single* shape graph might be collected at each node ("independent attribute" analysis) or
- a set of shape graphs might be collected ("relational" analysis, as in Sagiv et al. 1999).

In the former case, the join operation is the pointwise three-valued $\wedge$ which weakens solid arcs into dotted ones when dissimilar graphs are joined.

In the latter case, a bounded set union *widening* operation (Cousot & Cousot 1999) is employed so that no infinite set of graphs is constructed.

Independent attribute analysis example

```
y = Null;
while (x != Null) do
  t = y;
  y = x;
  x = x.next;
  y.next = t;
```

Independent attribute analysis example

```
y = Null;
while (x != Null) do
  t = y;
  y = x;
  x = x.next;
  y.next = t;
```

For the program above, we wish to verify/refute:

> "If the input to the program above is an acyclic list without any aliasing, pointed to by `x`, then the list pointed to by `y` upon program termination is also acyclic and without any aliasing."

(Forward) data-flow analysis of shape

We now perform (*** in class ***) a forward data-flow analysis for the program

```
y = Null;
while (x != Null) do
  t = y;
  y = x;
  x = x.next;
  y.next = t;
```

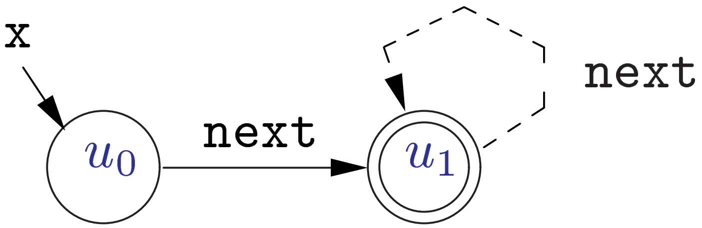
with the input shape of a linked list of length ≥ 2:

(Forward) data-flow analysis of shape

We now perform (*** in class ***) a forward data-flow analysis for the program

```
y = Null;
while (x != Null) do
  t = y;
  y = x;
  x = x.next;
  y.next = t;
```

with the input shape of a linked list of length ≥ 2:



Validating/refuting direction relationships

Sagiv et al. (1999) use three-valued predicate logic with a transitive closure operator to evaluate properties over the computed shape graph.

Validating/refuting direction relationships

Sagiv et al. (1999) use three-valued predicate logic with a transitive closure operator to evaluate properties over the computed shape graph. The *direction relationship* (Ghiya & Hendren 1996) states that an access path exists from the object named by `x` to an object named by `y`:

$$\mathtt{D}(\mathtt{x}, \mathtt{y}) \stackrel{\text{def}}{=} \mathtt{EF}_{\mathtt{next}}(\mathtt{x} \wedge \mathtt{EF}_{\mathtt{next}}\mathtt{y})$$

— an object, s, has atomic property `x` iff `x` points to s. Recall that $\mathtt{EF}_a\phi$ states, "there exists a path of a-labeled transitions such that, eventually, ϕ holds true."

Validating/refuting direction relationships

Sagiv et al. (1999) use three-valued predicate logic with a transitive closure operator to evaluate properties over the computed shape graph. The *direction relationship* (Ghiya & Hendren 1996) states that an access path exists from the object named by $\mathtt{x}$ to an object named by $\mathtt{y}$:

$$\mathtt{D}(\mathtt{x},\mathtt{y}) \stackrel{\text{def}}{=} \mathtt{EF}_{\mathtt{next}}(\mathtt{x} \wedge \mathtt{EF}_{\mathtt{next}}\,\mathtt{y})$$

— an object, s, has atomic property $\mathtt{x}$ iff $\mathtt{x}$ points to s. Recall that $\mathtt{EF}_a\phi$ states, "there exists a path of a-labeled transitions such that, eventually, ϕ holds true."

To validate that there is such a path from s, we check if $s \models^{\text{nec}} \mathtt{D}(\mathtt{x},\mathtt{y})$; to refute the existence of such a path, we check $s \models^{\text{nec}} \neg\mathtt{D}(\mathtt{x},\mathtt{y})$.

Validating/refuting interference relationships

The *interference relationship* (Ghiya & Hendren 1996) says that pointers `x` and `y` have access paths to a common heap node. It is encoded in the *inverse* transition system of $\rightarrow$:

$$\mathtt{I}(\mathtt{x}, \mathtt{y}) \stackrel{\text{def}}{=} (\mathtt{EF}_{\mathtt{next}^{-1}} \mathtt{x}) \wedge (\mathtt{EF}_{\mathtt{next}^{-1}} \mathtt{y}).$$

Validating/refuting interference relationships

The *interference relationship* (Ghiya & Hendren 1996) says that pointers `x` and `y` have access paths to a common heap node. It is encoded in the *inverse* transition system of $\rightarrow$:

$$\mathtt{I}(\mathtt{x},\mathtt{y}) \stackrel{\text{def}}{=} (\mathtt{EF}_{\mathtt{next}^{-1}}\mathtt{x}) \wedge (\mathtt{EF}_{\mathtt{next}^{-1}}\mathtt{y}).$$

We check $s \models^{\text{pos}} \mathtt{I}(\mathtt{x},\mathtt{y})$ to see if aliasing of object s by `x` and `y` is possible.

Validating/refuting interference relationships

The *interference relationship* (Ghiya & Hendren 1996) says that pointers `x` and `y` have access paths to a common heap node. It is encoded in the *inverse* transition system of $\rightarrow$:

$$\mathtt{I}(\mathtt{x},\mathtt{y}) \stackrel{\text{def}}{=} (\mathtt{EF}_{\mathtt{next}^{-1}}\mathtt{x}) \wedge (\mathtt{EF}_{\mathtt{next}^{-1}}\mathtt{y}).$$

We check $s \models^{\text{pos}} \mathtt{I}(\mathtt{x},\mathtt{y})$ to see if aliasing of object s by `x` and `y` is possible.

Similarly, we may want to try to validate that `x` and `y` definitely have access paths to a common heap node, s, by checking $s \models^{\text{nec}} \mathtt{I}(\mathtt{x},\mathtt{y})$.

Validating/refuting aliasing of pointers

Aliasing of pointers can be expressed: For

$$\mathtt{aliasing} \stackrel{\mathrm{def}}{=} \mathtt{EF}_{\mathtt{next}}(\bigvee_{\mathtt{x} \neq \mathtt{y}} \mathtt{x} \wedge \mathtt{y}),$$

the formulas

(a) $\mathtt{AG}_{\mathtt{next}} \neg \mathtt{aliasing}$,

(b) $\mathtt{AG}_{\mathtt{next}} \neg (\mathtt{x} \wedge \bigvee_{\mathtt{x} \neq \mathtt{y}} \mathtt{y})$, and

(c) $\mathtt{AG}_{\mathtt{next}} \neg (\mathtt{x} \wedge \mathtt{y})$

can then be used to check:

Validating/refuting aliasing of pointers

Aliasing of pointers can be expressed: For

$$\mathtt{aliasing} \stackrel{\text{def}}{=} \mathtt{EF}_{\mathtt{next}}(\bigvee_{\mathtt{x} \neq \mathtt{y}} \mathtt{x} \wedge \mathtt{y}),$$

the formulas

(a) $\mathtt{AG}_{\mathtt{next}} \neg \mathtt{aliasing}$,

(b) $\mathtt{AG}_{\mathtt{next}} \neg (\mathtt{x} \wedge \bigvee_{\mathtt{x} \neq \mathtt{y}} \mathtt{y})$, and

(c) $\mathtt{AG}_{\mathtt{next}} \neg (\mathtt{x} \wedge \mathtt{y})$

can then be used to check: (a) the absense of any kind of aliasing;

Validating/refuting aliasing of pointers

Aliasing of pointers can be expressed: For

$$\mathtt{aliasing} \stackrel{\text{def}}{=} \mathtt{EF}_{\mathtt{next}}(\bigvee_{\mathtt{x} \neq \mathtt{y}} \mathtt{x} \wedge \mathtt{y}),$$

the formulas

(a) $\mathtt{AG}_{\mathtt{next}} \neg \mathtt{aliasing}$,

(b) $\mathtt{AG}_{\mathtt{next}} \neg (\mathtt{x} \wedge \bigvee_{\mathtt{x} \neq \mathtt{y}} \mathtt{y})$, and

(c) $\mathtt{AG}_{\mathtt{next}} \neg (\mathtt{x} \wedge \mathtt{y})$

can then be used to check: (a) the absense of any kind of aliasing; (b) that `x` has no alias; and that

Validating/refuting aliasing of pointers

Aliasing of pointers can be expressed: For

$$\texttt{aliasing} \stackrel{\text{def}}{=} \texttt{EF}_{\texttt{next}}(\bigvee_{\texttt{x}\neq\texttt{y}} \texttt{x} \wedge \texttt{y}),$$

the formulas

(a) $\texttt{AG}_{\texttt{next}} \neg \texttt{aliasing}$,

(b) $\texttt{AG}_{\texttt{next}} \neg(\texttt{x} \wedge \bigvee_{\texttt{x}\neq\texttt{y}} \texttt{y})$, and

(c) $\texttt{AG}_{\texttt{next}} \neg(\texttt{x} \wedge \texttt{y})$

can then be used to check: (a) the absense of any kind of aliasing; (b) that `x` has no alias; and that (c) `x` and `y` never point to the same heap node. (Recall that $\texttt{AG}_a\phi$ states, "for all a-paths, it is globally true for all states along the path, that ϕ holds.")

Checking for cyclic data structures

We can check for possibly cyclic data structures. The predicate

$$\mathtt{cyclic} \stackrel{\mathrm{def}}{=} \bigvee_{\mathtt{x}\in\mathtt{AP}} \mathtt{x} \wedge \mathtt{EX}_{\mathtt{next}}\mathtt{EF}_{\mathtt{next}}\mathtt{x}$$

states that a heap node is pointed to by some `x` that has an access path to, presumably the same, heap node pointed to by `x`. (Recall that $\mathtt{EX}_a\phi$ says, "there exists an a-transition to a next state where ϕ holds.")

Recasting remaining parts of three-valued shape analysis

(M. Sagiv, T. Reps, and R. Wilhelm 1999): in the MTS framework

- their embedding theorem expresses *soundness* of the $\models^{\mathrm{nec}}$ semantics for refinement;

Recasting remaining parts of three-valued shape analysis

(M. Sagiv, T. Reps, and R. Wilhelm 1999): in the MTS framework

- their embedding theorem expresses *soundness* of the $\models^{\text{nec}}$ semantics for refinement;
- their `focus` operation, `instrumentation predicates`, and `coerce` operation have connections to assume-guarantee reasoning and generalized model checking (Bruns & Godefroid 2000).

5 Instrumented model checking

Overview:

- How to instrument model checks of partial Kripke structures.

5 Instrumented model checking

Overview:

- How to instrument model checks of partial Kripke structures.
- How to instrument model checks of modal transition systems.

5 Instrumented model checking

Overview:

- How to instrument model checks of partial Kripke structures.
- How to instrument model checks of modal transition systems.
- How to improve precision: generalized model checking.

Model checking partial Kripke structures

- A partial Kripke structure $\mathcal{M} = (\Sigma_M, \rightarrow, L^{must}, L^{may})$ determines two ordinary Kripke structures with the same state set and transition relation;
  - a "pessimistic" one, $\mathcal{M}_{\mathrm{p}} \stackrel{\text{def}}{=} (\Sigma_M, \rightarrow, L^{must})$, and
  - an "optimistic" one, $\mathcal{M}_{\mathrm{o}} \stackrel{\text{def}}{=} (\Sigma_M, \rightarrow, L^{may})$.

Model checking partial Kripke structures

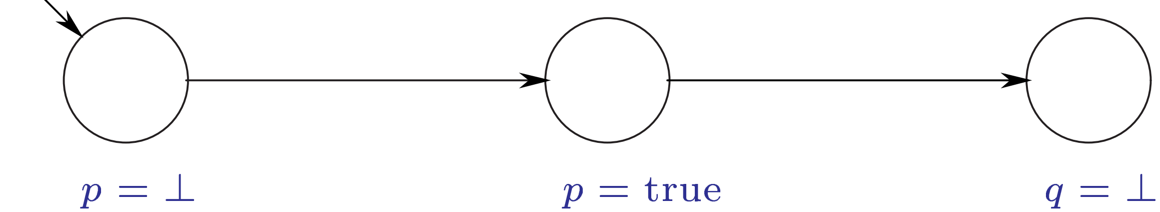
- A partial Kripke structure $\mathcal{M} = (\Sigma_M, \rightarrow, L^{must}, L^{may})$ determines two ordinary Kripke structures with the same state set and transition relation;
  - a "pessimistic" one, $\mathcal{M}_{\mathrm{p}} \stackrel{\text{def}}{=} (\Sigma_M, \rightarrow, L^{must})$, and
  - an "optimistic" one, $\mathcal{M}_{\mathrm{o}} \stackrel{\text{def}}{=} (\Sigma_M, \rightarrow, L^{may})$.
- If ϕ does not contain any negations (or implications), we have
  - $(\mathcal{M}, s) \models^{\text{nec}} \phi$ if $(\mathcal{M}_{\mathrm{p}}, s) \models \phi$;
  - $(\mathcal{M}, s) \models^{\text{pos}} \phi$ if $(\mathcal{M}_{\mathrm{o}}, s) \models \phi$.

Model checking partial Kripke structures

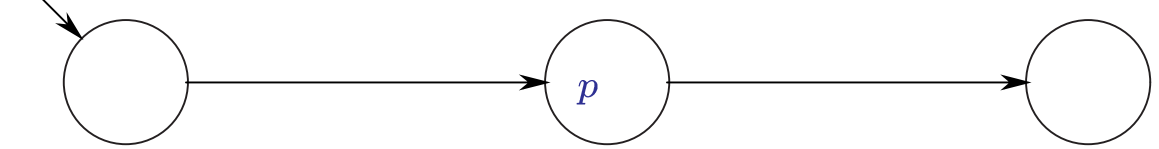
- A partial Kripke structure $\mathcal{M} = (\Sigma_M, \rightarrow, L^{must}, L^{may})$ determines two ordinary Kripke structures with the same state set and transition relation;
  - a "pessimistic" one, $\mathcal{M}_\mathrm{p} \stackrel{\mathrm{def}}{=} (\Sigma_M, \rightarrow, L^{must})$, and
  - an "optimistic" one, $\mathcal{M}_\mathrm{o} \stackrel{\mathrm{def}}{=} (\Sigma_M, \rightarrow, L^{may})$.
- If ϕ does not contain any negations (or implications), we have
  - $(\mathcal{M}, s) \models^{\mathrm{nec}} \phi$ if $(\mathcal{M}_\mathrm{p}, s) \models \phi$;
  - $(\mathcal{M}, s) \models^{\mathrm{pos}} \phi$ if $(\mathcal{M}_\mathrm{o}, s) \models \phi$.
- Finally, (i) translate *any* ϕ into an equivalent negation normal form $T(\phi)$, and (ii) apply previous item to $T(\phi)$.

Example

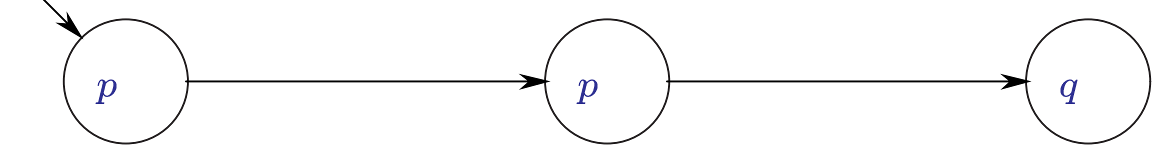
partial Kripke structure



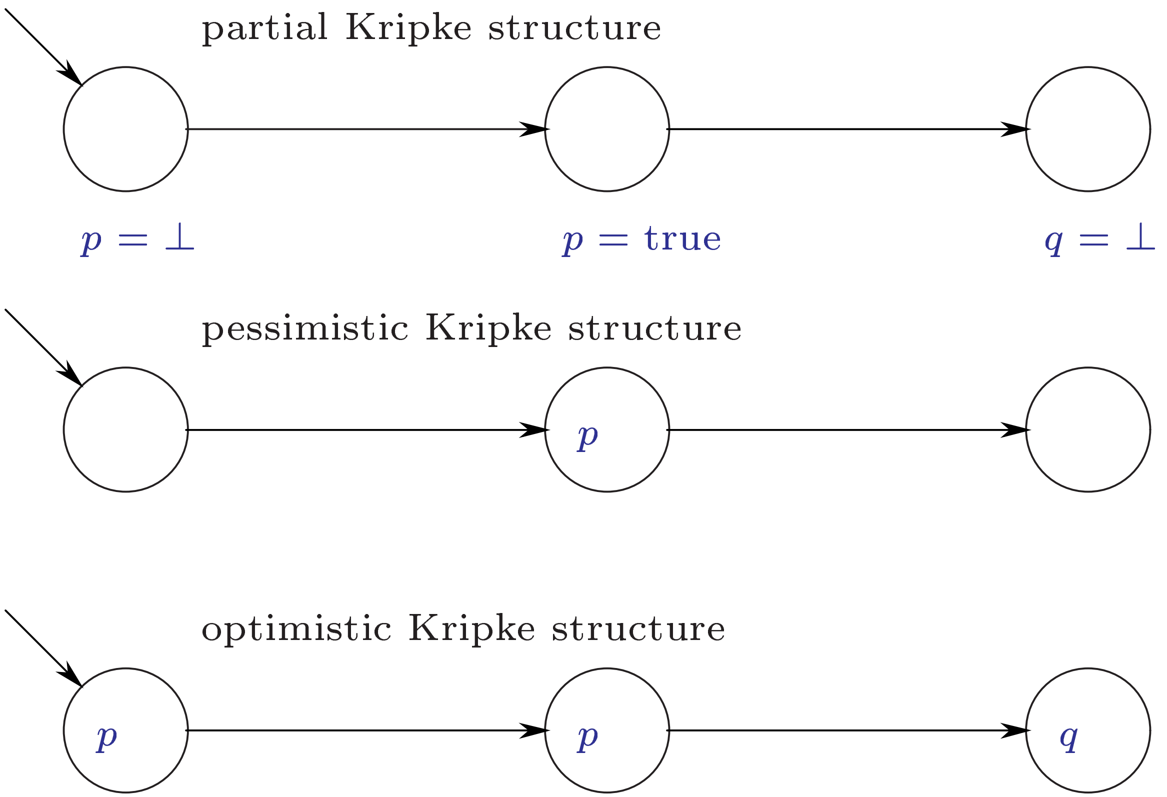
pessimistic Kripke structure



optimistic Kripke structure

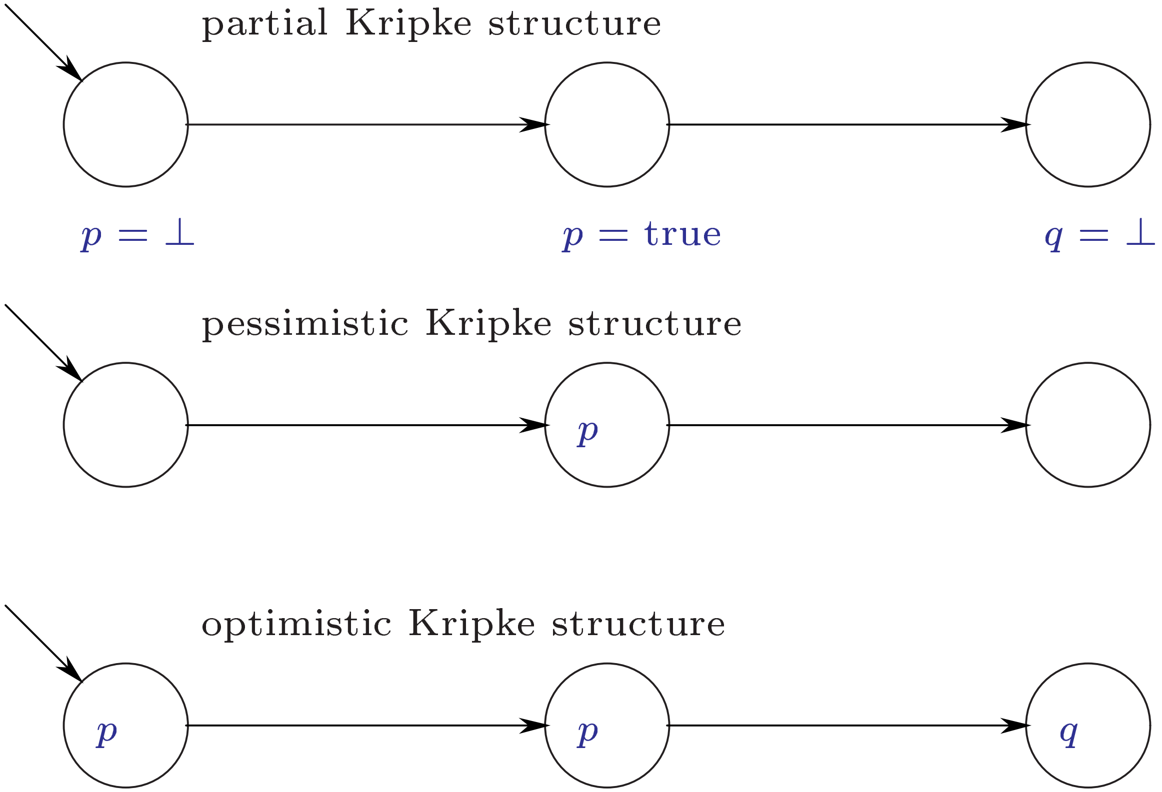


Example



We don't have $\mathcal{M} \models^{\text{nec}} \mathtt{A}[p\,\mathtt{U}\,q]$ as we don't have $\mathcal{M}_{\text{p}} \models \mathtt{A}[p\,\mathtt{U}\,q]$.

Example



We don't have $\mathcal{M} \models^{\mathrm{nec}} \mathtt{A}[p \,\mathtt{U}\, q]$ as we don't have $\mathcal{M}_{\mathrm{p}} \models \mathtt{A}[p \,\mathtt{U}\, q]$.

We do have $\mathcal{M} \models^{\mathrm{pos}} \mathtt{A}[p \,\mathtt{U}\, q]$ as we do have $\mathcal{M}_{\mathrm{o}} \models \mathtt{A}[p \,\mathtt{U}\, q]$.

Model checking modal transition systems

1. Put all may-information into transitions; pack propositional information into states. Thus, we may drop the p clause of the logic.

Model checking modal transition systems

1. Put all may-information into transitions; pack propositional information into states. Thus, we may drop the p clause of the logic.

2. Translate ϕ into a negation normal form $T(\phi)$ over a complement-closed set of actions: $\beta \in \mathtt{Act} \cup \{\bar{\alpha} \mid \alpha \in \mathtt{Act}\}$.

Model checking modal transition systems

1. Put all may-information into transitions; pack propositional information into states. Thus, we may drop the p clause of the logic.

2. Translate ϕ into a negation normal form $T(\phi)$ over a complement-closed set of actions: $\beta \in \texttt{Act} \cup \{\bar{\alpha} \mid \alpha \in \texttt{Act}\}$.

3. Extend the may-component and must-component of the given modal transition system to make them complement-closed.

Model checking modal transition systems

1. Put all may-information into transitions; pack propositional information into states. Thus, we may drop the p clause of the logic.

2. Translate ϕ into a negation normal form $T(\phi)$ over a complement-closed set of actions: $\beta \in \mathtt{Act} \cup \{\bar{\alpha} \mid \alpha \in \mathtt{Act}\}$.

3. Extend the may-component and must-component of the given modal transition system to make them complement-closed.

4. Model check $T(\phi)$ on the extended must-component for $\models^{\mathrm{nec}}$ and on the extended may-component for $\models^{\mathrm{pos}}$.

Step 1

Put all may-information into transitions; pack propositional information into states. (As I understand, this is what happens in the Bandera tool suite.) Thus, may drop the p clause ($\alpha \in \texttt{Act}$), resulting in the logic

$$\phi ::= \top \mid Z \mid \neg\phi \mid \phi \wedge \phi \mid \langle\alpha\rangle\,\phi \mid \mu Z.\phi$$

which is expressive enough to model CTL, CTL*, and LTL in an *action-based* encoding.

Step 1

Put all may-information into transitions; pack propositional information into states. (As I understand, this is what happens in the Bandera tool suite.) Thus, may drop the p clause ($\alpha \in \texttt{Act}$), resulting in the logic

$$\phi ::= \top \mid Z \mid \neg\phi \mid \phi \wedge \phi \mid \langle\alpha\rangle\,\phi \mid \mu Z.\phi$$

which is expressive enough to model CTL, CTL*, and LTL in an *action-based* encoding.

The translation $\phi \mapsto T(\phi)$ should preserve important fragments of that logic, such as CTL and LTL.

Step 2: part one

Given ϕ generated by

$$\phi ::= \top \mid Z \mid \neg\phi \mid \phi \wedge \phi \mid \langle\alpha\rangle\, \phi \mid \mu Z.\phi$$

the negation normal form $T(\phi)$ is defined over

$$\phi ::= \bot \mid \top \mid Z \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \langle\beta\rangle\, \phi \mid [\beta]\phi \mid \mu Z.\phi \mid \nu Z.\phi$$

where β ranges over $\mathtt{Act} \cup \{\bar{\alpha} \mid \alpha \in \mathtt{Act}\}$.

Step 2: part one

Given ϕ generated by

$$\phi ::= \top \mid Z \mid \neg\phi \mid \phi \wedge \phi \mid \langle\alpha\rangle\,\phi \mid \mu Z.\phi$$

the negation normal form $T(\phi)$ is defined over

$$\phi ::= \bot \mid \top \mid Z \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \langle\beta\rangle\,\phi \mid [\beta]\phi \mid \mu Z.\phi \mid \nu Z.\phi$$

where β ranges over $\mathtt{Act} \cup \{\bar{\alpha} \mid \alpha \in \mathtt{Act}\}$.

The translation T leaves $\top$, Z, and $\neg Z$ unchanged. Further,
$T(\neg\top) \stackrel{\text{def}}{=} \bot$, $T(\neg\neg\phi) \stackrel{\text{def}}{=} T(\phi)$, $T(\neg(\phi_1 \wedge \phi_2)) \stackrel{\text{def}}{=} T(\neg\phi_1) \vee T(\neg\phi_2)$,
$T(\neg\langle\alpha\rangle\,\phi) \stackrel{\text{def}}{=} [\alpha]T(\neg\phi)$, $T(\neg\mu Z.\phi) \stackrel{\text{def}}{=} \nu Z.T(\neg\phi)$,
$T(\phi_1 \wedge \phi_2) \stackrel{\text{def}}{=} T(\phi_1) \wedge T(\phi_2)$, $T(\langle\alpha\rangle\,\phi) \stackrel{\text{def}}{=} \langle\alpha\rangle\,T(\phi)$, and
$T(\mu Z.\phi) \stackrel{\text{def}}{=} \mu Z.T(\phi)$.

Step 2: part two

We defined $\phi \mapsto T(\phi)$ by structural induction on ϕ. On the resulting formula, we transform all matching subformulas according to

$$T([\alpha]\top) \stackrel{\text{def}}{=} \top \text{ and } T([\alpha]\bot) \stackrel{\text{def}}{=} \langle\overline{\alpha}\rangle\,\top$$

and denote the result by $T(\phi)$ again.

- Note that $[\alpha]\top$ (all α-next states satisfy "true") and $\top$ ("true") are semantically equivalent.
- Note that $s \models^{\text{nec}} [\alpha]\bot$ (all possible α-next states satisfy "false") holds iff there are no α-may-transitions with s as source.
- Consequence: all minimal existential subformulas of $T(\phi)$ are thus of the form $\langle\alpha\rangle\,\phi$ or $\langle\overline{\alpha}\rangle\,\phi$, where ϕ is $\top$ or some Z.

Step 3

A modal transition system $\mathcal{A}$ consists of

- a must-component $\mathcal{A}^{\mathrm{must}} = (\Sigma_A, \mathtt{Act}, \rightarrow_{\mathrm{must}})$ and
- a may-component $\mathcal{A}^{\mathrm{may}} = (\Sigma_A, \mathtt{Act}, \rightarrow_{\mathrm{may}})$.

Step 3

A modal transition system $\mathcal{A}$ consists of

- a must-component $\mathcal{A}^{\mathrm{must}} = (\Sigma_A, \mathtt{Act}, \to_{\mathrm{must}})$ and
- a may-component $\mathcal{A}^{\mathrm{may}} = (\Sigma_A, \mathtt{Act}, \to_{\mathrm{may}})$.

We define $\overline{\mathcal{A}} = (\overline{\mathcal{A}}^{\mathrm{must}}, \overline{\mathcal{A}}^{\mathrm{may}})$ with set of states

$$\Sigma_{\overline{A}} \stackrel{\mathrm{def}}{=} \Sigma_A \cup \{s_\perp\} \qquad (s_\perp \notin \Sigma_A)$$

and set of action labels $\mathtt{Act} \cup \{\overline{\alpha} \mid \alpha \in \mathtt{Act}\}$. This MTS retains all $\mathtt{may}$-transitions and $\mathtt{must}$-transitions of $\mathcal{A}$, and *adds* new transitions:

Step 3

A modal transition system $\mathcal{A}$ consists of

- a must-component $\mathcal{A}^{\mathrm{must}} = (\Sigma_A, \mathtt{Act}, \rightarrow_{\mathrm{must}})$ and
- a may-component $\mathcal{A}^{\mathrm{may}} = (\Sigma_A, \mathtt{Act}, \rightarrow_{\mathrm{may}})$.

We define $\overline{\mathcal{A}} = (\overline{\mathcal{A}}^{\mathrm{must}}, \overline{\mathcal{A}}^{\mathrm{may}})$ with set of states

$$\Sigma_{\overline{A}} \stackrel{\mathrm{def}}{=} \Sigma_A \cup \{s_\perp\} \qquad (s_\perp \notin \Sigma_A)$$

and set of action labels $\mathtt{Act} \cup \{\overline{\alpha} \mid \alpha \in \mathtt{Act}\}$. This MTS retains all `may`-transitions and `must`-transitions of $\mathcal{A}$, and *adds* new transitions:

$$\begin{array}{rcll} (s, \overline{\alpha}, s_\perp) \in \ \rightarrow_{\mathrm{must}} & \text{if} & \text{for all } s' \in \Sigma_A,\ (s, \alpha, s') \notin \ \rightarrow_{\mathrm{may}} & (1) \\ (s, \overline{\alpha}, s') \in \ \rightarrow_{\mathrm{may}} & \text{if} & (s, \alpha, s') \in \ \rightarrow_{\mathrm{may}} \setminus \rightarrow_{\mathrm{must}} . & (2) \end{array}$$

Computing the extended must-component

Computing the component $\overline{\mathcal{A}}^{\text{must}}$, we have to

- prove/compute the must-transitions of $\mathcal{A}^{must}$, and
- prove the *absence* of may-transitions in $\mathcal{A}^{may}$ to establish the must-transitions into $s_\perp$:

$$(s, \overline{\alpha}, s_\perp) \in\ \rightarrow_{\text{must}} \text{ if for all } s' \in \Sigma_A,\ (s, \alpha, s') \notin\ \rightarrow_{\text{may}}$$

Computing the extended must-component

Computing the component $\overline{\mathcal{A}}^{\text{must}}$, we have to

- prove/compute the must-transitions of $\mathcal{A}^{must}$, and
- prove the *absence* of may-transitions in $\mathcal{A}^{may}$ to establish the must-transitions into $s_\perp$:

$$(s, \overline{\alpha}, s_\perp) \in \;\rightarrow_{\text{must}} \;\text{ if for all } s' \in \Sigma_A,\; (s, \alpha, s') \notin \;\rightarrow_{\text{may}}$$

We can decide $(\mathcal{A}, s) \models^{\text{nec}} \phi$ by model checking $T(\phi)$ on the pointed *labeled transition system* $(\overline{\mathcal{A}}^{\text{must}}, s)$.

Step 4

Theorem 4.2.1 (Correctness of model checking algorithm) *Given a MTS $\mathcal{A}$ and a (closed) modal mu-calculus formula ϕ, let $T(\phi)$ and $\overline{\mathcal{A}}$ be the formula and MTS (respectively) as defined above. For any state $s \in \Sigma_A$, we then have*

1. $(\mathcal{A}, s) \models^{\mathrm{nec}} \phi$ *iff* $(\overline{\mathcal{A}}^{\mathrm{must}}, s) \models T(\phi)$;
2. $(\mathcal{A}, s) \models^{\mathrm{pos}} \phi$ *iff* $(\overline{\mathcal{A}}^{\mathrm{may}}, s) \models T(\phi)$; *and*
3. *not* $(\mathcal{A}, s) \models^{\mathrm{pos}} \phi$ *iff* $(\overline{\mathcal{A}}^{\mathrm{may}}, s) \not\models T(\phi)$.

Note that the righthand sides can be decided by conventional model checks of pointed labeled transition systems.

Losing precision

(Bruns & Godefroid 2000) Ideally, we want $s \models^{\mathrm{nec}} \phi$ iff for all refinements s' of s in *concrete/complete* systems, the conventional $s' \models \phi$ holds.

Losing precision

(Bruns & Godefroid 2000) Ideally, we want $s \models^{\text{nec}} \phi$ iff for all refinements s' of s in *concrete/complete* systems, the conventional $s' \models \phi$ holds.

For example, let $q \in L^{must}(s)$, $p \in L^{may}(s) \setminus L^{must}(s)$, and $\phi \stackrel{\text{def}}{=} q \wedge (p \to p)$. (Note that ϕ is satisfiable, but not valid.) Then:

Losing precision

(Bruns & Godefroid 2000) Ideally, we want $s \models^{\text{nec}} \phi$ iff for all refinements s' of s in *concrete/complete* systems, the conventional $s' \models \phi$ holds.

For example, let $q \in L^{must}(s)$, $p \in L^{may}(s) \setminus L^{must}(s)$, and $\phi \stackrel{\text{def}}{=} q \wedge (p \rightarrow p)$. (Note that ϕ is satisfiable, but not valid.) Then:

- We don't have $s \models^{\text{nec}} \phi$, because we don't have $s \models^{\text{nec}} p \rightarrow p$: we cannot refute p (i.e. not $s \models^{\text{nec}} \neg p$) and we cannot verify p (i.e. not $s \models^{\text{nec}} p$).

Losing precision

(Bruns & Godefroid 2000) Ideally, we want $s \models^{\text{nec}} \phi$ iff for all refinements s' of s in *concrete/complete* systems, the conventional $s' \models \phi$ holds.

For example, let $q \in L^{must}(s)$, $p \in L^{may}(s) \setminus L^{must}(s)$, and $\phi \stackrel{\text{def}}{=} q \wedge (p \to p)$. (Note that ϕ is satisfiable, but not valid.) Then:

- We don't have $s \models^{\text{nec}} \phi$, because we don't have $s \models^{\text{nec}} p \to p$: we cannot refute p (i.e. not $s \models^{\text{nec}} \neg p$) and we cannot verify p (i.e. not $s \models^{\text{nec}} p$).
- But ϕ holds in *all* complete refinements of s, because $s' \models p \to p$ holds in all concrete models.

Losing precision

(Bruns & Godefroid 2000) Ideally, we want $s \models^{\text{nec}} \phi$ iff for all refinements s' of s in *concrete/complete* systems, the conventional $s' \models \phi$ holds.

For example, let $q \in L^{must}(s)$, $p \in L^{may}(s) \setminus L^{must}(s)$, and $\phi \stackrel{\text{def}}{=} q \wedge (p \rightarrow p)$. (Note that ϕ is satisfiable, but not valid.) Then:

- We don't have $s \models^{\text{nec}} \phi$, because we don't have $s \models^{\text{nec}} p \rightarrow p$: we cannot refute p (i.e. not $s \models^{\text{nec}} \neg p$) and we cannot verify p (i.e. not $s \models^{\text{nec}} p$).

- But ϕ holds in *all* complete refinements of s, because $s' \models p \rightarrow p$ holds in all concrete models.

Our semantics for $\models^{\text{nec}}$ only loses precision for disjunctions ($\vee$).
Our semantics for $\models^{\text{pos}}$ only loses precision for conjunctions ($\wedge$).

Generalized model checking

Definition 4.2.2 [Generalized model checking problem (Bruns & Godefroid 2000)] Given a state s of a partial Kripke structure $\mathcal{M}$ and some ϕ, is there a state s' of a *Kripke* structure $\mathcal{M}'$ such that (i) $(\mathcal{M}', s')$ refines $(\mathcal{M}, s)$ and (ii) $(\mathcal{M}', s') \models \phi$?

Generalized model checking

Definition 4.2.2 [Generalized model checking problem (Bruns & Godefroid 2000)] Given a state s of a partial Kripke structure $\mathcal{M}$ and some ϕ, is there a state s' of a *Kripke* structure $\mathcal{M}'$ such that (i) $(\mathcal{M}', s')$ refines $(\mathcal{M}, s)$ and (ii) $(\mathcal{M}', s') \models \phi$?

We write

$$\mathrm{GMC}(\mathcal{M}, s, \phi)$$

for the decision problem above. E.g. $\mathrm{GMC}(\mathcal{M}, s, \phi) = \mathrm{true}$ means that there is a pointed Kripke structure $(\mathcal{M}', s')$ which refines $(\mathcal{M}, s)$ such that $(\mathcal{M}', s') \models \phi$.

Generalized model checking

Definition 4.2.2 [Generalized model checking problem (Bruns & Godefroid 2000)] Given a state s of a partial Kripke structure $\mathcal{M}$ and some ϕ, is there a state s' of a *Kripke* structure $\mathcal{M}'$ such that (i) $(\mathcal{M}', s')$ refines $(\mathcal{M}, s)$ and (ii) $(\mathcal{M}', s') \models \phi$?

We write

$$\mathrm{GMC}(\mathcal{M}, s, \phi)$$

for the decision problem above. E.g. $\mathrm{GMC}(\mathcal{M}, s, \phi) = \mathsf{true}$ means that there is a pointed Kripke structure $(\mathcal{M}', s')$ which refines $(\mathcal{M}, s)$ such that $(\mathcal{M}', s') \models \phi$.

We have soundness: $(\mathcal{M}, s) \models^{\mathrm{nec}} \phi$ implies not $\mathrm{GMC}(\mathcal{M}, s, \neg\phi)$.

Generalized model checking

Definition 4.2.2 [Generalized model checking problem (Bruns & Godefroid 2000)] Given a state s of a partial Kripke structure $\mathcal{M}$ and some ϕ, is there a state s' of a *Kripke* structure $\mathcal{M}'$ such that (i) $(\mathcal{M}', s')$ refines $(\mathcal{M}, s)$ and (ii) $(\mathcal{M}', s') \models \phi$?

We write

$$\mathrm{GMC}(\mathcal{M}, s, \phi)$$

for the decision problem above. E.g. $\mathrm{GMC}(\mathcal{M}, s, \phi) = \mathrm{true}$ means that there is a pointed Kripke structure $(\mathcal{M}', s')$ which refines $(\mathcal{M}, s)$ such that $(\mathcal{M}', s') \models \phi$.

We have soundness: $(\mathcal{M}, s) \models^{\mathrm{nec}} \phi$ implies not $\mathrm{GMC}(\mathcal{M}, s, \neg\phi)$.

Completeness means: not $\mathrm{GMC}(\mathcal{M}, s, \neg\phi)$ implies $(\mathcal{M}, s) \models^{\mathrm{nec}} \phi$.

Instrumented use of $\mathrm{GMC}(\mathcal{M}, s, \phi)$

We can compute $\models^{\mathrm{nec}}$ *without loss of precision* by computing one instance of the generalized model checking problem:

$$s \models^{\mathrm{nec}} \phi \qquad \stackrel{\mathrm{def}}{=} \qquad \text{not } \mathrm{GMC}(\mathcal{M}, s, \neg\phi).$$

Instrumented use of $\mathrm{GMC}(\mathcal{M}, s, \phi)$

We can compute $\models^{\mathrm{nec}}$ *without loss of precision* by computing one instance of the generalized model checking problem:

$$s \models^{\mathrm{nec}} \phi \qquad \stackrel{\mathrm{def}}{=} \qquad \text{not } \mathrm{GMC}(\mathcal{M}, s, \neg\phi).$$

Dually, we can compute $\models^{\mathrm{pos}}$ without loss of precision by setting:

$$s \models^{\mathrm{pos}} \phi \qquad \stackrel{\mathrm{def}}{=} \qquad \mathrm{GMC}(\mathcal{M}, s, \phi).$$

Instrumented use of $\mathrm{GMC}(\mathcal{M}, s, \phi)$

We can compute $\models^{\mathrm{nec}}$ *without loss of precision* by computing one instance of the generalized model checking problem:

$$s \models^{\mathrm{nec}} \phi \qquad \stackrel{\mathrm{def}}{=} \qquad \text{not } \mathrm{GMC}(\mathcal{M}, s, \neg\phi).$$

Dually, we can compute $\models^{\mathrm{pos}}$ without loss of precision by setting:

$$s \models^{\mathrm{pos}} \phi \qquad \stackrel{\mathrm{def}}{=} \qquad \mathrm{GMC}(\mathcal{M}, s, \phi).$$

Generalized model checking may be defined similarly over other classes of partial models, e.g. modal transition systems.

Alternatively, there are linear translations of MTSs into partial Kripke structures that preserve refinement.

Satisfiability and GMC

- Recall: ϕ is *satisfiable* iff there is some Kripke structure $(\mathcal{M}, s)$ such that $(\mathcal{M}, s) \models \phi$ holds.

Satisfiability and GMC

- Recall: ϕ is *satisfiable* iff there is some Kripke structure $(\mathcal{M}, s)$ such that $(\mathcal{M}, s) \models \phi$ holds.

- Consider the *universal* partial Kripke structure $\mathcal{U}$: one state s_U, one transition $s_U \to s_U$, and possible labeling for all propositions: $L^{may}(s_U) = \mathtt{AP}$, $L^{must}(s_U) = \{\}$; "nothing guaranteed, everything possible".

Satisfiability and GMC

- Recall: ϕ is *satisfiable* iff there is some Kripke structure $(\mathcal{M}, s)$ such that $(\mathcal{M}, s) \models \phi$ holds.

- Consider the *universal* partial Kripke structure $\mathcal{U}$: one state s_U, one transition $s_U \rightarrow s_U$, and possible labeling for all propositions: $L^{may}(s_U) = \texttt{AP}$, $L^{must}(s_U) = \{\}$; "nothing guaranteed, everything possible".

- *Every* pointed Kripke structure $(\mathcal{M}', s')$ is a refinement of $(\mathcal{U}, s_U)$.

Satisfiability and GMC

- Recall: ϕ is *satisfiable* iff there is some Kripke structure $(\mathcal{M}, s)$ such that $(\mathcal{M}, s) \models \phi$ holds.
- Consider the *universal* partial Kripke structure $\mathcal{U}$: one state s_U, one transition $s_U \to s_U$, and possible labeling for all propositions: $L^{may}(s_U) = \mathtt{AP}$, $L^{must}(s_U) = \{\}$; “nothing guaranteed, everything possible”.
- *Every* pointed Kripke structure $(\mathcal{M}', s')$ is a refinement of $(\mathcal{U}, s_U)$.
- Thus, ϕ is satisfiable iff $\mathrm{GMC}(\mathcal{U}, s_U, \phi)$ holds.

Satisfiability and GMC

- Recall: ϕ is *satisfiable* iff there is some Kripke structure $(\mathcal{M}, s)$ such that $(\mathcal{M}, s) \models \phi$ holds.
- Consider the *universal* partial Kripke structure $\mathcal{U}$: one state s_U, one transition $s_U \to s_U$, and possible labeling for all propositions: $L^{may}(s_U) = \mathtt{AP}$, $L^{must}(s_U) = \{\}$; "nothing guaranteed, everything possible".
- *Every* pointed Kripke structure $(\mathcal{M}', s')$ is a refinement of $(\mathcal{U}, s_U)$.
- Thus, ϕ is satisfiable iff $\mathrm{GMC}(\mathcal{U}, s_U, \phi)$ holds.
- Consequence: worst-case complexity of GMC is never less than the one for satisfiability.

Model checking and GMC

Consider pointed Kripke structures $(\mathcal{M}_1, s_1)$ and $(\mathcal{M}_2, s_2)$.

- Then these models are essentially *complete/concrete*, pointed partial Kripke structure.

- For complete structures:

  – refinement equals "bisimulation";

Model checking and GMC

Consider pointed Kripke structures $(\mathcal{M}_1, s_1)$ and $(\mathcal{M}_2, s_2)$.

- Then these models are essentially *complete/concrete*, pointed partial Kripke structure.
- For complete structures:
  - refinement equals “bisimulation”;
  - $\models^{\mathrm{nec}}$ equals $\models^{\mathrm{pos}}$; and

Model checking and GMC

Consider pointed Kripke structures $(\mathcal{M}_1, s_1)$ and $(\mathcal{M}_2, s_2)$.

- Then these models are essentially *complete/concrete*, pointed partial Kripke structure.
- For complete structures:
  - refinement equals "bisimulation";
  - $\models^{\mathrm{nec}}$ equals $\models^{\mathrm{pos}}$; and
  - $(\mathcal{M}_1, s_1) \models \phi$ iff $\mathrm{GMC}(\mathcal{M}_1, s_1, \phi)$.

Model checking and GMC

Consider pointed Kripke structures $(\mathcal{M}_1, s_1)$ and $(\mathcal{M}_2, s_2)$.

- Then these models are essentially *complete/concrete*, pointed partial Kripke structure.
- For complete structures:
  - refinement equals "bisimulation";
  - $\models^{\mathrm{nec}}$ equals $\models^{\mathrm{pos}}$; and
  - $(\mathcal{M}_1, s_1) \models \phi$ iff $\mathrm{GMC}(\mathcal{M}_1, s_1, \phi)$.
- Consequence: Generalized model checking and model checking coincide (as problems) on complete structures; in that case, you are well advised to run the algorithms for model checking. :-))

Complexity and algorithms

- Lesson: GMC has two extremal points: satisfiability (universal model $\mathcal{U}$) and model checking (Kripke structures).

Complexity and algorithms

- Lesson: GMC has two extremal points: satisfiability (universal model $\mathcal{U}$) and model checking (Kripke structures).
- Algorithms: For any partial Kripke structure $(\mathcal{M}, s_0)$ and CTL or LTL formula ϕ, the generalized model checking problem reduces to checking the emptiness of an alternating Büchi word automata (quadratic in state space, exponential in length of ϕ).

Complexity and algorithms

- Lesson: GMC has two extremal points: satisfiability (universal model $\mathcal{U}$) and model checking (Kripke structures).
- Algorithms: For any partial Kripke structure $(\mathcal{M}, s_0)$ and CTL or LTL formula ϕ, the generalized model checking problem reduces to checking the emptiness of an alternating Büchi word automata (quadratic in state space, exponential in length of ϕ).
- Complexity: For LTL and CTL, the generalized model checking problem is EXPTIME-complete.

Complexity and algorithms

- Lesson: GMC has two extremal points: satisfiability (universal model $\mathcal{U}$) and model checking (Kripke structures).
- Algorithms: For any partial Kripke structure $(\mathcal{M}, s_0)$ and CTL or LTL formula ϕ, the generalized model checking problem reduces to checking the emptiness of an alternating Büchi word automata (quadratic in state space, exponential in length of ϕ).

- Complexity: For LTL and CTL, the generalized model checking problem is EXPTIME-complete.
- Consequence: For CTL, satisfiability checking and generalized model checking are equally hard. For LTL, however, satisfiability checking is much simpler (PSPACE-complete) than generalized model checking!

6 Computing abstract modal transition systems

Overview:

- Relational specification of abstract modal transition systems.

6 Computing abstract modal transition systems

Overview:

- Relational specification of abstract modal transition systems.
- Computing
  - boolean abstractions of modal transition systems
  - boolean/cartesian abstractions of modal transition systems.

6 Computing abstract modal transition systems

Overview:

- Relational specification of abstract modal transition systems.
- Computing
  - boolean abstractions of modal transition systems
  - boolean/cartesian abstractions of modal transition systems.
- Abstraction refinement for modal transition systems.

Relational specification of abstract MTSs

Definition 4.2.3 [Relational abstraction (Dams 1996)] Let $\mathcal{A}_1 = (\texttt{Act}, \mathcal{A}_1^{\mathrm{must}}, \mathcal{A}_1^{\mathrm{may}})$ be a (abstract or concrete) MTS. Given a set of (designated) abstract states, Σ_{A_2}, and a total relation $\rho\colon \Sigma_{A_1} \times \Sigma_{A_2}$, we define $\mathcal{A}_2 = (\Sigma_{A_2}, \texttt{Act}, \rightarrow_{\mathrm{must}}, \rightarrow_{\mathrm{may}})$:

Relational specification of abstract MTSs

Definition 4.2.3 [Relational abstraction (Dams 1996)] Let $\mathcal{A}_1 = (\mathtt{Act}, \mathcal{A}_1^{\mathrm{must}}, \mathcal{A}_1^{\mathrm{may}})$ be a (abstract or concrete) MTS. Given a set of (designated) abstract states, Σ_{A_2}, and a total relation $\rho\colon \Sigma_{A_1} \times \Sigma_{A_2}$, we define $\mathcal{A}_2 = (\Sigma_{A_2}, \mathtt{Act}, \rightarrow_{\mathrm{must}}, \rightarrow_{\mathrm{may}})$:

- $a_2 \rightarrow^{\alpha}_{\mathrm{must}} a_2'$ iff for all $a_1 \in \Sigma_{A_1}$ with $a_1 \,\rho\, a_2$ there exists $a_1' \in \Sigma_{A_1}$ such that $a_1' \,\rho\, a_2'$ and $a_1 \rightarrow^{\alpha}_{\mathrm{must}} a_1'$;

Relational specification of abstract MTSs

Definition 4.2.3 [Relational abstraction (Dams 1996)] Let $\mathcal{A}_1 = (\mathtt{Act}, \mathcal{A}_1^{\mathrm{must}}, \mathcal{A}_1^{\mathrm{may}})$ be a (abstract or concrete) MTS. Given a set of (designated) abstract states, Σ_{A_2}, and a total relation $\rho\colon \Sigma_{A_1} \times \Sigma_{A_2}$, we define $\mathcal{A}_2 = (\Sigma_{A_2}, \mathtt{Act}, \rightarrow_{\mathrm{must}}, \rightarrow_{\mathrm{may}})$:

- $a_2 \rightarrow^{\alpha}_{\mathrm{must}} a_2'$ iff for all $a_1 \in \Sigma_{A_1}$ with $a_1 \,\rho\, a_2$ there exists $a_1' \in \Sigma_{A_1}$ such that $a_1' \,\rho\, a_2'$ and $a_1 \rightarrow^{\alpha}_{\mathrm{must}} a_1'$;
- $a_2 \rightarrow^{\alpha}_{\mathrm{may}} a_2'$ iff there exist $a_1 \in \Sigma_{A_1}$ and $a_1' \in \Sigma_{A_1}$ such that $a_1 \,\rho\, a_2$, $a_1' \,\rho\, a_2'$, and $a_1 \rightarrow^{\alpha}_{\mathrm{may}} a_1'$.

Relational specification of abstract MTSs

Definition 4.2.3 [Relational abstraction (Dams 1996)] Let $\mathcal{A}_1 = (\mathtt{Act}, \mathcal{A}_1^{\mathrm{must}}, \mathcal{A}_1^{\mathrm{may}})$ be a (abstract or concrete) MTS. Given a set of (designated) abstract states, Σ_{A_2}, and a total relation $\rho \colon \Sigma_{A_1} \times \Sigma_{A_2}$, we define $\mathcal{A}_2 = (\Sigma_{A_2}, \mathtt{Act}, \rightarrow_{\mathrm{must}}, \rightarrow_{\mathrm{may}})$:

- $a_2 \rightarrow^{\alpha}_{\mathrm{must}} a_2'$ iff for all $a_1 \in \Sigma_{A_1}$ with $a_1 \, \rho \, a_2$ there exists $a_1' \in \Sigma_{A_1}$ such that $a_1' \, \rho \, a_2'$ and $a_1 \rightarrow^{\alpha}_{\mathrm{must}} a_1'$;

- $a_2 \rightarrow^{\alpha}_{\mathrm{may}} a_2'$ iff there exist $a_1 \in \Sigma_{A_1}$ and $a_1' \in \Sigma_{A_1}$ such that $a_1 \, \rho \, a_2$, $a_1' \, \rho \, a_2'$, and $a_1 \rightarrow^{\alpha}_{\mathrm{may}} a_1'$.

Lemma 4.2.4 *Given $\mathcal{A}_1$ and $\mathcal{A}_2$ as above, $\mathcal{A}_2$ is a MTS and ρ is a refinement.*

Proof: Instructive Exercise. ;-))

Example

Let $\mathcal{A}_1$ be a complete MTS with set of stores for three integer variables `x`, `y`, and `z` as (infinite) state space. The assignment statement `x = z` induces all state transitions, e.g. one from $c = \{\mathtt{x} \mapsto i, \mathtt{y} \mapsto j, \mathtt{z} \mapsto k\}$ to $c' = \{\mathtt{x} \mapsto k, \mathtt{y} \mapsto j, \mathtt{z} \mapsto k\}$.

Example

Let $\mathcal{A}_1$ be a complete MTS with set of stores for three integer variables `x`, `y`, and `z` as (infinite) state space. The assignment statement `x = z` induces all state transitions, e.g. one from $c = \{\mathtt{x} \mapsto i, \mathtt{y} \mapsto j, \mathtt{z} \mapsto k\}$ to $c' = \{\mathtt{x} \mapsto k, \mathtt{y} \mapsto j, \mathtt{z} \mapsto k\}$.

The predicates $\phi_1 \stackrel{\text{def}}{=} \mathtt{odd}(\mathtt{x})$, $\phi_2 \stackrel{\text{def}}{=} (\mathtt{y} > \mathtt{0})$, and $\phi_3 \stackrel{\text{def}}{=} (\mathtt{z} < \mathtt{0})$ induce an equivalence relation on Σ_{A_1}. Let Σ_{A_2} be the set of all *equivalence-closed subsets* of the states of $\mathcal{A}_1$.

Example

Let $\mathcal{A}_1$ be a complete MTS with set of stores for three integer variables `x`, `y`, and `z` as (infinite) state space. The assignment statement `x = z` induces all state transitions, e.g. one from $c = \{\mathtt{x} \mapsto i, \mathtt{y} \mapsto j, \mathtt{z} \mapsto k\}$ to $c' = \{\mathtt{x} \mapsto k, \mathtt{y} \mapsto j, \mathtt{z} \mapsto k\}$.

The predicates $\phi_1 \stackrel{\text{def}}{=} \mathtt{odd}(\mathtt{x})$, $\phi_2 \stackrel{\text{def}}{=} (\mathtt{y} > \mathtt{0})$, and $\phi_3 \stackrel{\text{def}}{=} (\mathtt{z} < \mathtt{0})$ induce an equivalence relation on Σ_{A_1}. Let Σ_{A_2} be the set of all *equivalence-closed subsets* of the states of $\mathcal{A}_1$.

- There is a `may`-transition from the state $\phi_1 \wedge \phi_2 \wedge \phi_3$ to the states $\phi_1 \wedge \phi_2 \wedge \phi_3$ and $\neg\phi_1 \wedge \phi_2 \wedge \phi_3$;

Example

Let $\mathcal{A}_1$ be a complete MTS with set of stores for three integer variables `x`, `y`, and `z` as (infinite) state space. The assignment statement `x = z` induces all state transitions, e.g. one from $c = \{\mathtt{x} \mapsto i, \mathtt{y} \mapsto j, \mathtt{z} \mapsto k\}$ to $c' = \{\mathtt{x} \mapsto k, \mathtt{y} \mapsto j, \mathtt{z} \mapsto k\}$.

The predicates $\phi_1 \stackrel{\text{def}}{=} \mathtt{odd}(\mathtt{x})$, $\phi_2 \stackrel{\text{def}}{=} (\mathtt{y} > \mathtt{0})$, and $\phi_3 \stackrel{\text{def}}{=} (\mathtt{z} < \mathtt{0})$ induce an equivalence relation on Σ_{A_1}. Let Σ_{A_2} be the set of all *equivalence-closed subsets* of the states of $\mathcal{A}_1$.

- There is a `may`-transition from the state $\phi_1 \wedge \phi_2 \wedge \phi_3$ to the states $\phi_1 \wedge \phi_2 \wedge \phi_3$ and $\neg\phi_1 \wedge \phi_2 \wedge \phi_3$;
- there is a `must`-transition from $\phi_1 \wedge \phi_2 \wedge \phi_3$ to the disjunction of monomials $(\phi_1 \wedge \phi_2 \wedge \phi_3) \vee (\neg\phi_1 \wedge \phi_2 \wedge \phi_3)$; but

Example

Let $\mathcal{A}_1$ be a complete MTS with set of stores for three integer variables `x`, `y`, and `z` as (infinite) state space. The assignment statement `x = z` induces all state transitions, e.g. one from $c = \{\mathtt{x} \mapsto i, \mathtt{y} \mapsto j, \mathtt{z} \mapsto k\}$ to $c' = \{\mathtt{x} \mapsto k, \mathtt{y} \mapsto j, \mathtt{z} \mapsto k\}$.

The predicates $\phi_1 \stackrel{\text{def}}{=} \mathtt{odd}(\mathtt{x})$, $\phi_2 \stackrel{\text{def}}{=} (\mathtt{y} > \mathtt{0})$, and $\phi_3 \stackrel{\text{def}}{=} (\mathtt{z} < \mathtt{0})$ induce an equivalence relation on Σ_{A_1}. Let Σ_{A_2} be the set of all *equivalence-closed subsets* of the states of $\mathcal{A}_1$.

- There is a `may`-transition from the state $\phi_1 \wedge \phi_2 \wedge \phi_3$ to the states $\phi_1 \wedge \phi_2 \wedge \phi_3$ and $\neg\phi_1 \wedge \phi_2 \wedge \phi_3$;
- there is a `must`-transition from $\phi_1 \wedge \phi_2 \wedge \phi_3$ to the disjunction of monomials $(\phi_1 \wedge \phi_2 \wedge \phi_3) \vee (\neg\phi_1 \wedge \phi_2 \wedge \phi_3)$; but
- there is *no* `must`-transition from $\phi_1 \wedge \phi_2 \wedge \phi_3$ to any monomial, e.g. $\phi_1 \wedge \phi_2 \wedge \phi_3$ or $\neg\phi_1 \wedge \phi_2 \wedge \phi_3$.

Pre and post

Notation. For any predicate η on a set of states Σ_S, the post operator and weakest precondition are defined as

$$\begin{aligned}
\texttt{post}(\eta) &\stackrel{\text{def}}{=} \{s' \in \Sigma_S \mid \exists s \in \Sigma_S : s \models \eta, s \to s'\} \\
\tilde{\texttt{pre}}(\eta) &\stackrel{\text{def}}{=} \{s \in \Sigma_S \mid \forall s' \in \Sigma_S : s \to s' \text{ implies } s' \models \eta\}.
\end{aligned}$$

Pre and post

Notation. For any predicate η on a set of states Σ_S, the post operator and weakest precondition are defined as

$$\begin{aligned} \texttt{post}(\eta) &\stackrel{\text{def}}{=} \{s' \in \Sigma_S \mid \exists s \in \Sigma_S : s \models \eta, s \to s'\} \\ \tilde{\texttt{pre}}(\eta) &\stackrel{\text{def}}{=} \{s \in \Sigma_S \mid \forall s' \in \Sigma_S : s \to s' \text{ implies } s' \models \eta\}. \end{aligned}$$

Needed property. For any predicates η, ψ on states, satisfiability witnesses s' for $\texttt{post}(\psi) \wedge \eta$ correspond to satisfiability witnesses s for $\psi \wedge \neg\tilde{\texttt{pre}}(\neg\eta)$ via $s \to s'$ (even in the presence of deadlock!).

Proof: Instructive Exercise. ;-))

Methodological assumptions

- In our motivating applications, an abstract program is built by converting each program statement from a transformer operating on concrete states to a transformer operating on abstract states, in the style of the example above.

Methodological assumptions

- In our motivating applications, an abstract program is built by converting each program statement from a transformer operating on concrete states to a transformer operating on abstract states, in the style of the example above.

- Thus, we may focus on the abstraction of a *single* program statement, *allowing us to consider MTSs without explicit action labels.*

Methodological assumptions

- In our motivating applications, an abstract program is built by converting each program statement from a transformer operating on concrete states to a transformer operating on abstract states, in the style of the example above.
- Thus, we may focus on the abstraction of a *single* program statement, *allowing us to consider MTSs without explicit action labels.*
- For such program statements and a quantifier-free formula η, we assume that $\tilde{\texttt{pre}}(\eta)$ is *quantifier-free* as well.

Methodological assumptions

- In our motivating applications, an abstract program is built by converting each program statement from a transformer operating on concrete states to a transformer operating on abstract states, in the style of the example above.
- Thus, we may focus on the abstraction of a *single* program statement, *allowing us to consider MTSs without explicit action labels.*
- For such program statements and a quantifier-free formula η, we assume that $\tilde{\texttt{pre}}(\eta)$ is *quantifier-free* as well.
- The latter is the case for usual programming language constructs and enables the use of decision procedures as realized in tools such as SVC or PVS.

Boolean abstraction: specification

Let $\mathcal{S}$ be a LTS. Given a finite set of quantifier-free formulas of first-order logic,

$$\Phi = \{\phi_1, \phi_2, \ldots, \phi_n\},$$

we derive a finite-state abstract MTS $\mathcal{B}_\Phi$ as in Definition 4.2.3 ($\mathcal{A}_1$ is $\mathcal{S}$ and $\mathcal{A}_2$ is $\mathcal{B}_\Phi$):

Boolean abstraction: specification

Let $\mathcal{S}$ be a LTS. Given a finite set of quantifier-free formulas of first-order logic,

$$\Phi = \{\phi_1, \phi_2, \ldots, \phi_n\},$$

we derive a finite-state abstract MTS $\mathcal{B}_\Phi$ as in Definition 4.2.3 ($\mathcal{A}_1$ is $\mathcal{S}$ and $\mathcal{A}_2$ is $\mathcal{B}_\Phi$): (i) $\Sigma_{B_\Phi} \subseteq \{0, 1\}^n$;

Boolean abstraction: specification

Let $\mathcal{S}$ be a LTS. Given a finite set of quantifier-free formulas of first-order logic,

$$\Phi = \{\phi_1, \phi_2, \ldots, \phi_n\},$$

we derive a finite-state abstract MTS $\mathcal{B}_\Phi$ as in Definition 4.2.3 ($\mathcal{A}_1$ is $\mathcal{S}$ and $\mathcal{A}_2$ is $\mathcal{B}_\Phi$): (i) $\Sigma_{B_\Phi} \subseteq \{0,1\}^n$; (ii) for $b \in \{0,1\}^n$, we define a monomial $\langle b, \Phi \rangle$ whose ith conjunct is ϕ_i if $b_i = 1$, and $\neg\phi_i$ otherwise;

Boolean abstraction: specification

Let $\mathcal{S}$ be a LTS. Given a finite set of quantifier-free formulas of first-order logic,

$$\Phi = \{\phi_1, \phi_2, \ldots, \phi_n\},$$

we derive a finite-state abstract MTS $\mathcal{B}_\Phi$ as in Definition 4.2.3 ($\mathcal{A}_1$ is $\mathcal{S}$ and $\mathcal{A}_2$ is $\mathcal{B}_\Phi$): (i) $\Sigma_{B_\Phi} \subseteq \{0,1\}^n$; (ii) for $b \in \{0,1\}^n$, we define a monomial $\langle b, \Phi\rangle$ whose ith conjunct is ϕ_i if $b_i = 1$, and $\neg\phi_i$ otherwise; (iii) $\rho \stackrel{\text{def}}{=} \rho_{\mathrm{b}}^{\Phi} \subseteq \Sigma_S \times \Sigma_{B_\Phi}$, where

$$s\, \rho_{\mathrm{b}}^{\Phi}\, b \qquad \text{iff} \qquad s \models \langle b, \Phi\rangle.$$

We make ρ_{b}^{Φ} total by setting $\Sigma_{B_\Phi} \stackrel{\text{def}}{=} \{b \in \{0,1\}^n \mid \exists s \in \Sigma_S \colon s\, \rho_{\mathrm{b}}^{\Phi}\, b\}$.

Boolean abstraction: implementation of may

The set of `may`-successors of a state set ψ is

$$\mathtt{next}(\psi)^{\mathrm{may}}_{\mathrm{b}} \stackrel{\mathrm{def}}{=} \{b' \in \Sigma_B \mid \psi \wedge \neg\tilde{\mathtt{pre}}(\neg\langle b', \Phi\rangle) \text{ is satisfiable } \}.$$

Recall

$\tilde{\mathtt{pre}}(\neg\langle b', \Phi\rangle) \stackrel{\mathrm{def}}{=} \{s \in \Sigma_S \mid \forall s' \in \Sigma_S \colon s \to s' \text{ implies } s' \models \neg\langle b', \Phi\rangle\}.$

Boolean abstraction: implementation of may

The set of `may`-successors of a state set ψ is

$$\mathtt{next}(\psi)^{\mathrm{may}}_{\mathrm{b}} \stackrel{\mathrm{def}}{=} \{b' \in \Sigma_B \mid \psi \wedge \neg \tilde{\mathtt{pre}}(\neg\langle b', \Phi\rangle) \text{ is satisfiable } \}.$$

Recall

$\tilde{\mathtt{pre}}(\neg\langle b', \Phi\rangle) \stackrel{\mathrm{def}}{=} \{s \in \Sigma_S \mid \forall s' \in \Sigma_S : s \to s' \text{ implies } s' \models \neg\langle b', \Phi\rangle\}$.

If ψ is a boolean combination of $\{\phi_i \mid m \leq i \leq n\}$, we compute $\mathtt{next}(\psi)^{\mathrm{may}}_{\mathrm{b}}$ as a BDD $H^{may}(\psi, \mathrm{true}, m)$:

$$H^{may}(\psi, \eta, i) \stackrel{\mathrm{def}}{=} \begin{cases} (x_i \wedge H^{may}(\psi, \eta \wedge \phi_i, i+1)) & \\ \vee(\neg x_i \wedge H^{may}(\psi, \eta \wedge \neg\phi_i, i+1) & \text{if } 0 < i \leq n, \\ 1 \qquad \text{if } i = n+1 \text{ and } \psi \wedge \neg\tilde{\mathtt{pre}}(\neg\eta) \text{ is satisfiable,} & \\ 0 \qquad \text{if } i = n+1 \text{ and } \psi \wedge \neg\tilde{\mathtt{pre}}(\neg\eta) \text{ is unsatisfiable.} & \end{cases}$$

Boolean abstraction: implementation of must

The set of $\texttt{must}$-successors of a state set ψ is

$$\texttt{next}(\psi)^{\mathrm{must}}_{\mathrm{b}} \stackrel{\mathrm{def}}{=} \{b' \in \Sigma_B \mid \psi \wedge \tilde{\texttt{pre}}(\neg\langle b', \Phi\rangle) \text{ is unsatisfiable } \}.$$

Recall

$\tilde{\texttt{pre}}(\neg\langle b', \Phi\rangle) \stackrel{\mathrm{def}}{=} \{s \in \Sigma_S \mid \forall s' \in \Sigma_S : s \rightarrow s' \text{ implies } s' \models \neg\langle b', \Phi\rangle\}.$

Boolean abstraction: implementation of must

The set of **must**-successors of a state set ψ is

$$\mathtt{next}(\psi)_{\mathrm{b}}^{\mathrm{must}} \stackrel{\mathrm{def}}{=} \{b' \in \Sigma_B \mid \psi \wedge \tilde{\mathtt{pre}}(\neg\langle b', \Phi\rangle) \text{ is unsatisfiable } \}.$$

Recall

$\tilde{\mathtt{pre}}(\neg\langle b', \Phi\rangle) \stackrel{\mathrm{def}}{=} \{s \in \Sigma_S \mid \forall s' \in \Sigma_S \colon s \rightarrow s' \text{ implies } s' \models \neg\langle b', \Phi\rangle\}$.

If ψ is a boolean combination of $\{\phi_i \mid m \leq i \leq n\}$, we compute $\mathtt{next}(\psi)_{\mathrm{b}}^{\mathrm{must}}$ as a BDD $H^{must}(\psi, \text{true}, m)$:

$$H^{must}(\psi, \eta, i) \stackrel{\mathrm{def}}{=} \begin{cases} (x_i \wedge H^{must}(\psi, \eta \wedge \phi_i, i+1)) & \\ \vee(\neg x_i \wedge H^{must}(\psi, \eta \wedge \neg\phi_i, i+1) & \text{if } 0 < i \leq n, \\ 1 & \text{if } i = n+1 \text{ and } \psi \wedge \tilde{\mathtt{pre}}(\neg\eta) \text{ is unsatisfiable}, \\ 0 & \text{if } i = n+1 \text{ and } \psi \wedge \tilde{\mathtt{pre}}(\neg\eta) \text{ is satisfiable}. \end{cases}$$

Soundness, completeness, and cost

- For H^{must} and H^{may}, interpret "unknown" results from call to theorem prover as "satisfiable".

Soundness, completeness, and cost

- For H^{must} and H^{may}, interpret "unknown" results from call to theorem prover as "satisfiable".

- If $H^{may}(\langle b, \Phi \rangle, \mathrm{true}, m)$ applied to b' renders 1, then $b \rightarrow_{\mathrm{may}} b'$ in $\mathcal{B}_\Phi$ (converse true if theorem prover has a good day :-)).

Soundness, completeness, and cost

- For H^{must} and H^{may}, interpret "unknown" results from call to theorem prover as "satisfiable".

- If $H^{may}(\langle b, \Phi\rangle, \mathrm{true}, m)$ applied to b' renders 1, then $b \rightarrow_{\mathrm{may}} b'$ in $\mathcal{B}_\Phi$ (converse true if theorem prover has a good day :-)).

- If $H^{must}(\langle b, \Phi\rangle, \mathrm{true}, m)$ applied to b' renders 1, then $b \rightarrow_{\mathrm{must}} b'$ in $\mathcal{B}_\Phi$ (converse true if theorem prover has a good day :-)).

Soundness, completeness, and cost

- For H^{must} and H^{may}, interpret "unknown" results from call to theorem prover as "satisfiable".
- If $H^{may}(\langle b, \Phi \rangle, \mathrm{true}, m)$ applied to b' renders 1, then $b \rightarrow_{\mathrm{may}} b'$ in $\mathcal{B}_\Phi$ (converse true if theorem prover has a good day :-)).
- If $H^{must}(\langle b, \Phi \rangle, \mathrm{true}, m)$ applied to b' renders 1, then $b \rightarrow_{\mathrm{must}} b'$ in $\mathcal{B}_\Phi$ (converse true if theorem prover has a good day :-)).
- The overall computation makes $O(2^n)$ calls to the theorem prover: the *same cost* as computing the may-component only!

Boolean/cartesian abstraction: motivation

Revisiting our example with $\phi_1 \stackrel{\text{def}}{=} \texttt{odd}(\texttt{x})$, $\phi_2 \stackrel{\text{def}}{=} (\texttt{y} > \texttt{0})$, $\phi_3 \stackrel{\text{def}}{=} (\texttt{z} < \texttt{0})$ and the assignment `x = z`;

Boolean/cartesian abstraction: motivation

Revisiting our example with $\phi_1 \stackrel{\text{def}}{=} \texttt{odd}(\texttt{x})$, $\phi_2 \stackrel{\text{def}}{=} (\texttt{y} > 0)$, $\phi_3 \stackrel{\text{def}}{=} (\texttt{z} < 0)$ and the assignment `x = z`; if $\Phi = \{\phi_2, \phi_3\}$, then $\mathcal{B}_\Phi$ has four states, each one with a `must`-transition to itself.

Boolean/cartesian abstraction: motivation

Revisiting our example with $\phi_1 \stackrel{\text{def}}{=} \texttt{odd}(\texttt{x})$, $\phi_2 \stackrel{\text{def}}{=} (\texttt{y} > \texttt{0})$, $\phi_3 \stackrel{\text{def}}{=} (\texttt{z} < \texttt{0})$ and the assignment `x = z`; if $\Phi = \{\phi_2, \phi_3\}$, then $\mathcal{B}_\Phi$ has four states, each one with a `must`-transition to itself. However, adding the predicate ϕ_1 to Φ, there are no `must`-transitions with the abstract state $\phi_1 \wedge \phi_2 \wedge \phi_3$ (111) as a source in $\mathcal{B}_{\{\phi_1\} \cup \Phi}$. This is quite annoying, since the `y` information is lost even though it plays no role in the assignment `x = z`.

Boolean/cartesian abstraction: motivation

Revisiting our example with $\phi_1 \stackrel{\mathrm{def}}{=} \mathtt{odd}(\mathtt{x})$, $\phi_2 \stackrel{\mathrm{def}}{=} (\mathtt{y} > \mathtt{0})$, $\phi_3 \stackrel{\mathrm{def}}{=} (\mathtt{z} < \mathtt{0})$ and the assignment `x = z`; if $\Phi = \{\phi_2, \phi_3\}$, then $\mathcal{B}_\Phi$ has four states, each one with a `must`-transition to itself. However, adding the predicate ϕ_1 to Φ, there are no `must`-transitions with the abstract state $\phi_1 \wedge \phi_2 \wedge \phi_3$ (111) as a source in $\mathcal{B}_{\{\phi_1\}\cup\Phi}$. This is quite annoying, since the `y` information is lost even though it plays no role in the assignment `x = z`. But before we saw that there *is* a `must`-transition from $\phi_1 \wedge \phi_2 \wedge \phi_3$ to the *disjunction* $(\phi_1 \wedge \phi_2 \wedge \phi_3) \vee (\neg\phi_1 \wedge \phi_2 \wedge \phi_3)$ that correctly captures the "absence of effect" on `y`.

Boolean/cartesian abstraction: motivation

Revisiting our example with $\phi_1 \stackrel{\text{def}}{=} \texttt{odd(x)}$, $\phi_2 \stackrel{\text{def}}{=} (\texttt{y} > \texttt{0})$, $\phi_3 \stackrel{\text{def}}{=} (\texttt{z} < \texttt{0})$ and the assignment `x = z`; if $\Phi = \{\phi_2, \phi_3\}$, then $\mathcal{B}_\Phi$ has four states, each one with a `must`-transition to itself. However, adding the predicate ϕ_1 to Φ, there are no `must`-transitions with the abstract state $\phi_1 \wedge \phi_2 \wedge \phi_3$ (111) as a source in $\mathcal{B}_{\{\phi_1\}\cup\Phi}$. This is quite annoying, since the `y` information is lost even though it plays no role in the assignment `x = z`. But before we saw that there *is* a `must`-transition from $\phi_1 \wedge \phi_2 \wedge \phi_3$ to the *disjunction* $(\phi_1 \wedge \phi_2 \wedge \phi_3) \vee (\neg\phi_1 \wedge \phi_2 \wedge \phi_3)$ that correctly captures the "absence of effect" on `y`.

Calculating `must`-transitions whose targets can be abstract states of the above kind is generally too expensive: given n predicates, there are a possible 2^{2^n} such states. Therefore, we use cartesian abstraction (bka "independent attribute analysis").

Boolean/cartesian abstraction: specification

Given $\mathcal{S}$ and Φ as before, we specify a finite-state abstract MTS $\mathcal{C}_\Phi$ ($\mathcal{A}_1$ is $\mathcal{S}$ and $\mathcal{A}_2$ is $\mathcal{C}_\Phi$):

Boolean/cartesian abstraction: specification

Given $\mathcal{S}$ and Φ as before, we specify a finite-state abstract MTS $\mathcal{C}_\Phi$ ($\mathcal{A}_1$ is $\mathcal{S}$ and $\mathcal{A}_2$ is $\mathcal{C}_\Phi$): (i) $\Sigma_{C_\Phi} \subseteq \{0, 1, \star\}^n$;

Boolean/cartesian abstraction: specification

Given $\mathcal{S}$ and Φ as before, we specify a finite-state abstract MTS $\mathcal{C}_\Phi$ ($\mathcal{A}_1$ is $\mathcal{S}$ and $\mathcal{A}_2$ is $\mathcal{C}_\Phi$): (i) $\Sigma_{C_\Phi} \subseteq \{0, 1, \star\}^n$; (ii) for $c \in \{0, 1, \star\}^n$, we define a monomial, $\langle c, \Phi \rangle$, whose ith conjunct is ϕ_i if $c_i = 1$, $\neg\phi_i$ if $c_i = 0$, and $\texttt{true}$ otherwise;

Boolean/cartesian abstraction: specification

Given $\mathcal{S}$ and Φ as before, we specify a finite-state abstract MTS $\mathcal{C}_\Phi$ ($\mathcal{A}_1$ is $\mathcal{S}$ and $\mathcal{A}_2$ is $\mathcal{C}_\Phi$): (i) $\Sigma_{C_\Phi} \subseteq \{0,1,\star\}^n$; (ii) for $c \in \{0,1,\star\}^n$, we define a monomial, $\langle c, \Phi \rangle$, whose ith conjunct is ϕ_i if $c_i = 1$, $\neg\phi_i$ if $c_i = 0$, and $\texttt{true}$ otherwise; (iii) $\rho \stackrel{\text{def}}{=} \rho_{\text{c}}^{\Phi} \circ \rho_{\text{b}}^{\Phi}$, where

$$b \, \rho_{\text{c}}^{\Phi} \, c \qquad \text{iff} \qquad \text{for } 1 \leq i \leq n,\ c_i \neq \star \text{ implies } b_i = c_i.$$

We make ρ_{c}^{Φ} total by setting
$\Sigma_{C_\Phi} \stackrel{\text{def}}{=} \{c \in \{0,1,\star\}^n \mid \exists b \in \Sigma_{B_\Phi} : b \, \rho_{\text{c}}^{\Phi} \, c\}$.

Boolean/cartesian abstraction: specification

Given $\mathcal{S}$ and Φ as before, we specify a finite-state abstract MTS $\mathcal{C}_\Phi$ ($\mathcal{A}_1$ is $\mathcal{S}$ and $\mathcal{A}_2$ is $\mathcal{C}_\Phi$): (i) $\Sigma_{C_\Phi} \subseteq \{0, 1, \star\}^n$; (ii) for $c \in \{0, 1, \star\}^n$, we define a monomial, $\langle c, \Phi \rangle$, whose ith conjunct is ϕ_i if $c_i = 1$, $\neg\phi_i$ if $c_i = 0$, and **`true`** otherwise; (iii) $\rho \stackrel{\text{def}}{=} \rho_{\mathrm{c}}^{\Phi} \circ \rho_{\mathrm{b}}^{\Phi}$, where

$$b \, \rho_{\mathrm{c}}^{\Phi} \, c \qquad \text{iff} \qquad \text{for } 1 \leq i \leq n,\ c_i \neq \star \text{ implies } b_i = c_i.$$

We make ρ_{c}^{Φ} total by setting $\Sigma_{C_\Phi} \stackrel{\text{def}}{=} \{c \in \{0, 1, \star\}^n \mid \exists b \in \Sigma_{B_\Phi} : b \, \rho_{\mathrm{c}}^{\Phi} \, c\}$.

Note: (a) the meaning of $\star$ (“don’t care”);

Boolean/cartesian abstraction: specification

Given $\mathcal{S}$ and Φ as before, we specify a finite-state abstract MTS $\mathcal{C}_\Phi$ ($\mathcal{A}_1$ is $\mathcal{S}$ and $\mathcal{A}_2$ is $\mathcal{C}_\Phi$): (i) $\Sigma_{C_\Phi} \subseteq \{0, 1, \star\}^n$; (ii) for $c \in \{0, 1, \star\}^n$, we define a monomial, $\langle c, \Phi \rangle$, whose ith conjunct is ϕ_i if $c_i = 1$, $\neg\phi_i$ if $c_i = 0$, and `true` otherwise; (iii) $\rho \stackrel{\text{def}}{=} \rho_{\mathrm{c}}^{\Phi} \circ \rho_{\mathrm{b}}^{\Phi}$, where

$$b \, \rho_{\mathrm{c}}^{\Phi} \, c \qquad \text{iff} \qquad \text{for } 1 \leq i \leq n,\ c_i \neq \star \text{ implies } b_i = c_i.$$

We make ρ_{c}^{Φ} total by setting
$\Sigma_{C_\Phi} \stackrel{\text{def}}{=} \{c \in \{0, 1, \star\}^n \mid \exists b \in \Sigma_{B_\Phi} : b \, \rho_{\mathrm{c}}^{\Phi} \, c\}$.

Note: (a) the meaning of $\star$ (“don’t care”); (b) the abstract MTS, $\mathcal{C}'_\Phi$, obtained by abstracting $\mathcal{B}_\Phi$ with ρ_{c}^{Φ}, is less precise than $\mathcal{C}_\Phi$;

Boolean/cartesian abstraction: specification

Given $\mathcal{S}$ and Φ as before, we specify a finite-state abstract MTS $\mathcal{C}_\Phi$ ($\mathcal{A}_1$ is $\mathcal{S}$ and $\mathcal{A}_2$ is $\mathcal{C}_\Phi$): (i) $\Sigma_{C_\Phi} \subseteq \{0, 1, \star\}^n$; (ii) for $c \in \{0, 1, \star\}^n$, we define a monomial, $\langle c, \Phi \rangle$, whose ith conjunct is ϕ_i if $c_i = 1$, $\neg\phi_i$ if $c_i = 0$, and `true` otherwise; (iii) $\rho \stackrel{\text{def}}{=} \rho_{\mathrm{c}}^{\Phi} \circ \rho_{\mathrm{b}}^{\Phi}$, where

$$b \, \rho_{\mathrm{c}}^{\Phi} \, c \qquad \text{iff} \qquad \text{for } 1 \leq i \leq n,\ c_i \neq \star \text{ implies } b_i = c_i.$$

We make ρ_{c}^{Φ} total by setting
$\Sigma_{C_\Phi} \stackrel{\text{def}}{=} \{c \in \{0, 1, \star\}^n \mid \exists b \in \Sigma_{B_\Phi} : b \, \rho_{\mathrm{c}}^{\Phi} \, c\}$.

Note: (a) the meaning of $\star$ ("don't care"); (b) the abstract MTS, $\mathcal{C}'_\Phi$, obtained by abstracting $\mathcal{B}_\Phi$ with ρ_{c}^{Φ}, is less precise than $\mathcal{C}_\Phi$; (c) we can model the abstraction relation in our logic: $s \, (\rho_{\mathrm{c}}^{\Phi} \circ \rho_{\mathrm{b}}^{\Phi}) \, c$ iff $s \models \langle c, \Phi \rangle$.

Boolean/cartesian abstraction: implementation of may

Now, we use a *ternary decision diagram* (TDD):

$$G^{may}(\psi,\eta,i) \stackrel{\text{def}}{=} \begin{cases} ([1/x_i] \wedge G^{may}(\psi,\eta\wedge\phi_i,i+1)) \vee ([0/x_i] \wedge G^{may}(\psi,\eta\wedge\neg\phi_i,i+1)) \\ \vee([\star/x_i] \wedge G^{may}(\psi,\eta,i+1)) \qquad \text{if } 0 < i \leq n, \\ 1 \qquad \text{if } i = n+1 \text{ and } \psi \wedge \neg\tilde{\mathtt{pre}}(\neg\eta) \text{ is satisfiable,} \\ 0 \qquad \text{if } i = n+1 \text{ and } \psi \wedge \neg\tilde{\mathtt{pre}}(\neg\eta) \text{ is unsatisfiable.} \end{cases}$$

Boolean/cartesian abstraction: implementation of may

Now, we use a *ternary decision diagram* (TDD):

$$G^{may}(\psi,\eta,i) \stackrel{\text{def}}{=} \begin{cases} ([1/x_i] \wedge G^{may}(\psi,\eta \wedge \phi_i, i+1)) \vee ([0/x_i] \wedge G^{may}(\psi,\eta \wedge \neg\phi_i, i+1)) \\ \vee([\star/x_i] \wedge G^{may}(\psi,\eta,i+1)) \qquad \text{if } 0 < i \leq n, \\ 1 \qquad \text{if } i = n+1 \text{ and } \psi \wedge \neg\tilde{\texttt{pre}}(\neg\eta) \text{ is satisfiable,} \\ 0 \qquad \text{if } i = n+1 \text{ and } \psi \wedge \neg\tilde{\texttt{pre}}(\neg\eta) \text{ is unsatisfiable.} \end{cases}$$

The function G^{may} essentially computes the abstract post-operator of SLAM.

Boolean/cartesian abstraction: implementation of may

Now, we use a *ternary decision diagram* (TDD):

$$G^{may}(\psi, \eta, i) \stackrel{\text{def}}{=} \begin{cases} ([1/x_i] \wedge G^{may}(\psi, \eta \wedge \phi_i, i+1)) \vee ([0/x_i] \wedge G^{may}(\psi, \eta \wedge \neg\phi_i, i+1)) \\ \vee([\star/x_i] \wedge G^{may}(\psi, \eta, i+1)) \qquad \text{if } 0 < i \leq n, \\ 1 \qquad \text{if } i = n+1 \text{ and } \psi \wedge \neg \tilde{\mathtt{pre}}(\neg\eta) \text{ is satisfiable}, \\ 0 \qquad \text{if } i = n+1 \text{ and } \psi \wedge \neg \tilde{\mathtt{pre}}(\neg\eta) \text{ is unsatisfiable}. \end{cases}$$

The function G^{may} essentially computes the abstract post-operator of SLAM. The TDD $G^{may}(\psi, \mathrm{true}, m)$ represents the set of `may`-successors of ψ:

$$\mathtt{next}(\psi)_{\mathrm{c}}^{\mathrm{may}} \stackrel{\text{def}}{=} \{c' \in \Sigma_C \mid \psi \wedge \neg \tilde{\mathtt{pre}}(\neg\langle c', \Phi\rangle) \text{ is satisfiable } \}.$$

Boolean/cartesian abstraction: implementation of must

The TDD $G^{must}(\psi, \text{true}, m)$ represents the set of `must`-successors of ψ:

$$G^{must}(\psi,\eta,i) \stackrel{\text{def}}{=} \begin{cases} ([1/x_i] \wedge G^{must}(\psi, \eta \wedge \phi_i, i+1)) \vee ([0/x_i] \wedge G^{must}(\psi, \eta \wedge \neg\phi_i, i+1)) \\ \vee([\star/x_i] \wedge G^{must}(\psi, \eta, i+1)) \qquad \text{if } 0 < i \leq n, \\ 1 \qquad \text{if } i = n+1 \text{ and } \psi \wedge \tilde{\texttt{pre}}(\neg\eta) \text{ is unsatisfiable,} \\ 0 \qquad \text{if } i = n+1 \text{ and } \psi \wedge \tilde{\texttt{pre}}(\neg\eta) \text{ is satisfiable.} \end{cases}$$

Concretely,

$$\texttt{next}(\psi)_{\mathrm{c}}^{\mathrm{must}} \stackrel{\text{def}}{=} \{c' \in \Sigma_C \mid \psi \wedge \tilde{\texttt{pre}}(\neg\langle c', \Phi\rangle) \text{ is unsatisfiable } \}.$$

Soundness, completeness, and cost

- For G^{must} and G^{may}, interpret “unknown” results from call to theorem prover as “satisfiable”.

Soundness, completeness, and cost

- For G^{must} and G^{may}, interpret "unknown" results from call to theorem prover as "satisfiable".
- If $G^{may}(\langle c, \Phi\rangle, \mathrm{true}, m)$ applied to c' renders 1, then $c \rightarrow_{\mathrm{may}} c'$ in $\mathcal{C}_\Phi$ (converse true if theorem prover has a good day :-)).

Soundness, completeness, and cost

- For G^{must} and G^{may}, interpret "unknown" results from call to theorem prover as "satisfiable".

- If $G^{may}(\langle c, \Phi\rangle, \mathrm{true}, m)$ applied to c' renders 1, then $c \rightarrow_{\mathrm{may}} c'$ in $\mathcal{C}_\Phi$ (converse true if theorem prover has a good day :-)).

- If $G^{must}(\langle c, \Phi\rangle, \mathrm{true}, m)$ applied to c' renders 1, then $c \rightarrow_{\mathrm{must}} c'$ in $\mathcal{C}_\Phi$ (converse true if theorem prover has a good day :-)).

Soundness, completeness, and cost

- For G^{must} and G^{may}, interpret “unknown” results from call to theorem prover as “satisfiable”.
- If $G^{may}(\langle c, \Phi\rangle, \mathrm{true}, m)$ applied to c' renders 1, then $c \rightarrow_{\mathrm{may}} c'$ in $\mathcal{C}_\Phi$ (converse true if theorem prover has a good day :-)).
- If $G^{must}(\langle c, \Phi\rangle, \mathrm{true}, m)$ applied to c' renders 1, then $c \rightarrow_{\mathrm{must}} c'$ in $\mathcal{C}_\Phi$ (converse true if theorem prover has a good day :-)).
- The overall computation makes $O(3^n)$ calls to the theorem prover: the *same cost* as computing the may-component only!

Abstraction refinement

- In the boolean case, abstraction refinement works for the may-component only! Recall our example of adding ϕ_1 to $\{\phi_2, \phi_3\}$.
- In the boolean/cartesian case, abstraction refinement works for may- and must-components.

Abstraction refinement

- In the boolean case, abstraction refinement works for the may-component only! Recall our example of adding ϕ_1 to $\{\phi_2, \phi_3\}$.
- In the boolean/cartesian case, abstraction refinement works for may- and must-components.

Theorem 4.2.5 (Incremental refinement) *For $\Phi = \{\phi_1, \phi_2, \ldots, \phi_n\}$ and $\Psi = \Phi \cup \{\phi_{n+1}, \phi_{n+2}, \ldots, \phi_{n+m}\}$, the MTS $\mathcal{C}_\Psi$ is a refinement of the MTS $\mathcal{C}_\Phi$.*

Abstraction refinement

- In the boolean case, abstraction refinement works for the may-component only! Recall our example of adding ϕ_1 to $\{\phi_2, \phi_3\}$.
- In the boolean/cartesian case, abstraction refinement works for may- and must-components.

Theorem 4.2.5 (Incremental refinement) *For* $\Phi = \{\phi_1, \phi_2, \ldots, \phi_n\}$ *and* $\Psi = \Phi \cup \{\phi_{n+1}, \phi_{n+2}, \ldots, \phi_{n+m}\}$, *the MTS* $\mathcal{C}_\Psi$ *is a refinement of the MTS* $\mathcal{C}_\Phi$*.*

Proof. The refinement $\rho \subseteq \Sigma_{C_\Psi} \times \Sigma_{C_\Phi}$ is given by $\{(c', c) \mid c$ is a prefix of $c'\}$. $\square$

7 Assume-guarantee model checking

*** to be completed ***

8 Conclusions and future work

Modal transition systems

1. their abstraction notion, and temporal logic semantics are a robust and executable framework for a range of applications in the design and analysis of software.

8 Conclusions and future work

Modal transition systems

1. their abstraction notion, and temporal logic semantics are a robust and executable framework for a range of applications in the design and analysis of software.

2. allow for sound reasoning about partial or incomplete designs and programs, thereby recovering bisimulation techniques on total or complete designs.

8 Conclusions and future work

Modal transition systems

1. their abstraction notion, and temporal logic semantics are a robust and executable framework for a range of applications in the design and analysis of software.

2. allow for sound reasoning about partial or incomplete designs and programs, thereby recovering bisimulation techniques on total or complete designs.

3. exploit all four possible interactions of path quantifiers and state-transition modalities.

8 Conclusions and future work

Modal transition systems

1. their abstraction notion, and temporal logic semantics are a robust and executable framework for a range of applications in the design and analysis of software.

2. allow for sound reasoning about partial or incomplete designs and programs, thereby recovering bisimulation techniques on total or complete designs.

3. exploit all four possible interactions of path quantifiers and state-transition modalities.

4. await novel applications, e.g., compositional pointer analysis (Whaley & Rinard 1999).