CS-E3220 Declarative Progamming

Model-Checking and Validation

Jussi Rintanen

Department of Computer Science
Aalto University

November 6, 2019

This Lecture

- Model-checking in Temporal logics
- SAT-Based Bounded Model-Checking (BMC)
- Abstraction
- Counterexample-guided abstraction refinement (CEGAR)

Model-Checking

Test if a CTL formula is true in a transition system

$$A\mathcal{F}\,finalstate$$
$$A\mathcal{G}E\mathcal{F}\,progress$$

Test if an LTL formula is true in a transition system

$\mathcal{G}\mathcal{F}(green_1)$
$\mathcal{G}(\neg(green_1 \wedge red_1) \wedge \neg(green_1 \wedge yellow_1) \wedge \neg(yellow_1 \wedge red_1))$
$\mathcal{G}(green_1 \rightarrow F(red_1))$
$\mathcal{G}(red_1 \rightarrow (red_1 U yellow_1))$
$\mathcal{G}\neg(green_1 \wedge green_2)$

Model-Checking

- Model-checking: test $M \models \phi$ for given model M and formula ϕ
- Applications:
  - Computer-aided verification (LTL, CTL, CTL$*$)
  - Advanced databases: query evaluation (description logics)
- Easier than testing logical consequence, but can be hard

logic	logical consequence	model-checking
propositional logic	co-NP-complete	P-complete
PDL	EXP-complete	P-complete
CTL	EXP-complete	P-complete
CTL$*$	2-EXP-complete	PSPACE-complete
LTL	PSPACE-complete	PSPACE-complete

Why Model-Checking?

- Deductive verification: system models as formulas, theorem-proving
- Poor scalability of theorem-provers $\longrightarrow$ model-checking preferable
- Practical handling of transition system models with hundreds of thousands of states
- Since 1990ies, model-checking without explicitly represented system $\longrightarrow$ symbolic model-checking with OBDD & SAT:
  - OBDD-based model-checking for CTL
  - SAT-based model-checking for LTL

Model-Checking for Propositional Logic

```
fun MC(And(p,q),M) = boolAnd(MC (p,M),MC (q,M))
  | MC(Or(p,q),M) = boolOr(MC (p,M),MC (q,M))
  | MC(Neg p,M) = boolNot (p,M)
  | MC(Atom a,M) = M(a)
```

```
fun boolAnd(true,true) = true
  | boolAnd _ = false
fun boolOr(false,false) = false
  | boolOr _ = true
fun boolNot false = true
  | boolNot true = false
```

M is the model: mapping from propositional variables to $\{$ *true*, *false* $\}$

This algorithm runs in linear time if M is constant time (e.g. array indexed by propositional variables)

Model-Checking a Modal Logic with One Modality

1. Label each world w with $L(w) = \{x \in X | M \models_w x\}$
2. Consider all subformulas ϕ' of ϕ in the order of increasing length. For every world w, do $L(w) := L(w) \cup \{\phi'\}$ if
   - $\phi' = \psi_1 \wedge \psi_2$ and $\psi_1 \in L(w)$ and $\psi_2 \in L(w)$, or
   - $\phi' = \psi_1 \vee \psi_2$ and $\psi_1 \in L(w)$ or $\psi_2 \in L(w)$, or
   - $\phi' = \neg\psi_1$ and $\psi_1 \notin L(w)$, or
   - $\phi' = \Box\psi$ and $\psi \in L(w')$ for all w' such that wRw', or
   - $\phi' = \Diamond\psi$ and $\psi \in L(w')$ for some w' such that wRw'.

The runtime of this procedure is $O(|\phi| \times |W|)$

Model-Checking for Temporal Logics

- LTL, CTL, CTL$*$ talk about infinite paths, even when the transition system is finite
- P-time model-checking for PDL and CTL:
  - Label the computation graph with all subformulas ϕ_0 of ϕ
  - Truth in a node a function of truth of subformulas in the node and its successors
- PSPACE-hard model-checking for LTL and CTL$*$:
  - Need to consider exponentially many exponentially long paths

CTL Model-Checking

- For every $w \in W$, $L(w) := \{x \in X | M \models_w x\}$
- Consider all subformulas ϕ' of ϕ in the order of increasing length. For every world w, do $L(w) := L(w) \cup \{\phi'\}$ if
  - $\phi' = \psi_1 \wedge \psi_2$ and $\psi_1 \in L(w)$ and $\psi_2 \in L(w)$, or
  - $\phi' = \psi_1 \vee \psi_2$ and $\psi_1 \in L(w)$ or $\psi_2 \in L(w)$, or
  - $\phi' = \neg\psi_1$ and $\psi_1 \notin L(w)$.
- Formulas $E(\phi \mathcal{U} \psi)$ and $E\mathcal{G}\phi$ handled in the following slides.
- All other operators can be reduced to the above:

$$
\begin{aligned}
E\mathcal{F}\phi &\equiv E(\top \mathcal{U} \phi) \\
A\mathcal{F}\varphi &\equiv \neg E\mathcal{G}\neg\varphi \\
A\mathcal{G}\varphi &\equiv \neg E(\top \mathcal{U} \neg\varphi) \\
A\mathcal{X}\varphi &\equiv \neg E\mathcal{X}\neg\varphi \\
A(\varphi \mathcal{U} \psi) &\equiv \neg E(\neg\psi \mathcal{U} (\neg\varphi \wedge \neg\psi)) \wedge \neg E\mathcal{G}\neg\psi
\end{aligned}
$$

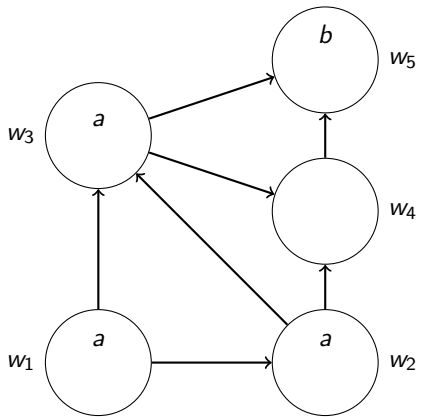
CTL Model-Checking: Labeling for $E(\phi \mathcal{U} \psi)$

For every world w, update $L(w) := L(w) \cup \{E(\phi \mathcal{U} \psi)\}$ if

- $\psi \in L(w)$ or
- $\phi \in L(w)$ and $E(\phi \mathcal{U} \psi) \in L(w')$ for some w' such that wRw'

until no more updates possible.

Example: $E(a \mathcal{U} b)$



Systematic algorithm given on the next slide...

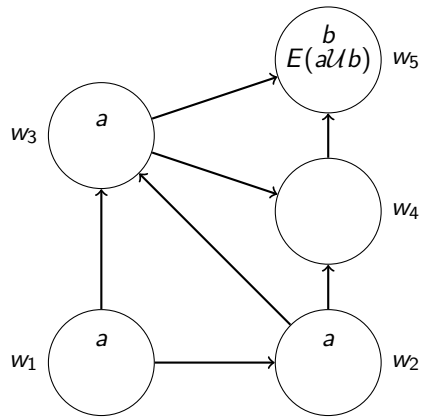
CTL Model-Checking: Labeling for $E(\phi \mathcal{U} \psi)$

For every world w, update $L(w) := L(w) \cup \{E(\phi \mathcal{U} \psi)\}$ if

- $\psi \in L(w)$ or
- $\phi \in L(w)$ and $E(\phi \mathcal{U} \psi) \in L(w')$ for some w' such that wRw'

until no more updates possible.

Example: $E(a \mathcal{U} b)$



Systematic algorithm given on the next slide...

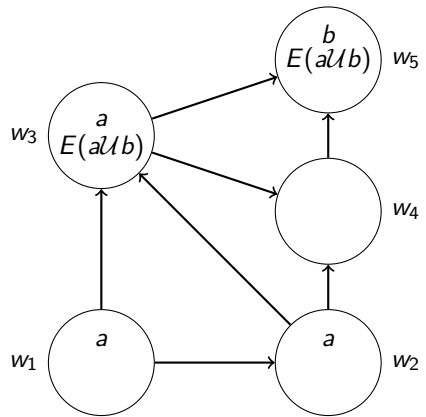
CTL Model-Checking: Labeling for $E(\phi \mathcal{U} \psi)$

For every world w, update $L(w) := L(w) \cup \{E(\phi \mathcal{U} \psi)\}$ if

- $\psi \in L(w)$ or
- $\phi \in L(w)$ and $E(\phi \mathcal{U} \psi) \in L(w')$ for some w' such that wRw'

until no more updates possible.

Example: $E(a \mathcal{U} b)$



Systematic algorithm given on the next slide...

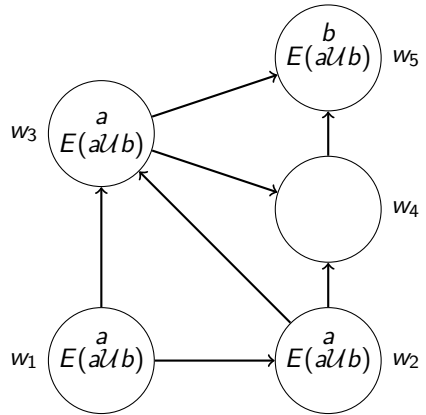
CTL Model-Checking: Labeling for $E(\phi \mathcal{U} \psi)$

For every world w, update $L(w) := L(w) \cup \{E(\phi \mathcal{U} \psi)\}$ if

- $\psi \in L(w)$ or
- $\phi \in L(w)$ and $E(\phi \mathcal{U} \psi) \in L(w')$ for some w' such that wRw'

until no more updates possible.

Example: $E(a \mathcal{U} b)$



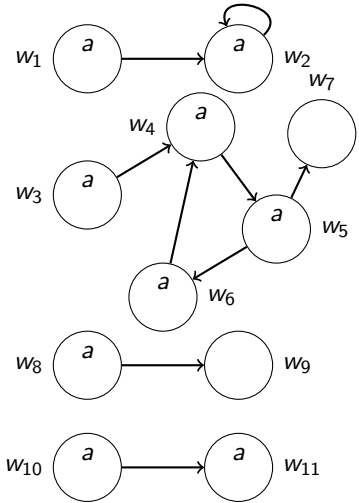
Systematic algorithm given on the next slide...

CTL Model-Checking: Labeling for $E(\phi \mathcal{U} \psi)$

PROCEDURE EU(α,β)
$T := \{w \in W | \beta \in L(w)\}$;
FOR EACH $w \in T$ DO $L(w) := L(w) \cup \{E(\alpha \mathcal{U} \beta)\}$;
WHILE $T \neq \emptyset$ DO
  take any $w \in T$;
  $T := T \setminus \{w\}$;
  FOR EACH t such that tRw DO
    IF $\alpha \in L(t)$ and $E(\alpha \mathcal{U} \beta) \notin L(t)$ THEN
      $L(t) := L(t) \cup \{E(\alpha \mathcal{U} \beta)\}$;
      $T := T \cup \{t\}$;
    END IF
  END FOR
END WHILE

CTL Model-Checking: Labeling for $E\mathcal{G}\phi$

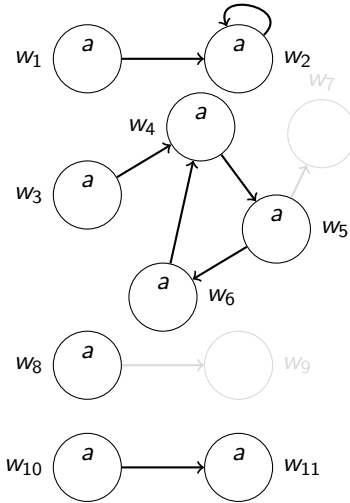
- Let $W_\phi = \{w \in W | \phi \in L(w)\}$
- Let $G = \langle W_\phi, R \cap (W_\phi \times W_\phi) \rangle$
- Find strongly connected components of G
- For each SCC C such that $|C| > 1$ or wRw for some $w \in C$, do $L(w) := L(w) \cup \{E\mathcal{G}\phi\}$ for all $w \in C$
- For each $w \in W_\phi$, if $E\mathcal{G}\phi \in L(w')$ for some w' such that wRw' then $L(w) := L(w) \cup \{E\mathcal{G}\phi\}$



Systematic algorithm given on the next slide...

CTL Model-Checking: Labeling for $E\mathcal{G}\phi$

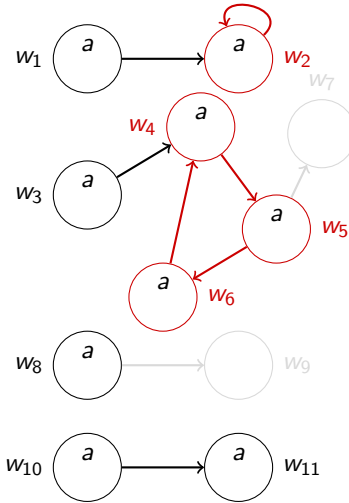
- Let $W_\phi = \{w \in W | \phi \in L(w)\}$
- Let $G = \langle W_\phi, R \cap (W_\phi \times W_\phi) \rangle$
- Find strongly connected components of G
- For each SCC C such that $|C| > 1$ or wRw for some $w \in C$, do $L(w) := L(w) \cup \{E\mathcal{G}\phi\}$ for all $w \in C$
- For each $w \in W_\phi$, if $E\mathcal{G}\phi \in L(w')$ for some w' such that wRw' then $L(w) := L(w) \cup \{E\mathcal{G}\phi\}$



Systematic algorithm given on the next slide...

CTL Model-Checking: Labeling for $E\mathcal{G}\phi$

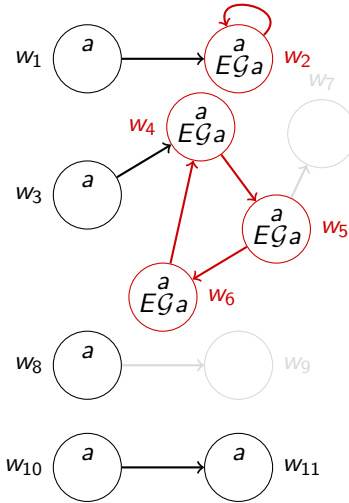
- Let $W_\phi = \{w \in W | \phi \in L(w)\}$
- Let $G = \langle W_\phi, R \cap (W_\phi \times W_\phi) \rangle$
- Find strongly connected components of G
- For each SCC C such that $|C| > 1$ or wRw for some $w \in C$, do $L(w) := L(w) \cup \{E\mathcal{G}\phi\}$ for all $w \in C$
- For each $w \in W_\phi$, if $E\mathcal{G}\phi \in L(w')$ for some w' such that wRw' then $L(w) := L(w) \cup \{E\mathcal{G}\phi\}$



Systematic algorithm given on the next slide...

CTL Model-Checking: Labeling for $E\mathcal{G}\phi$

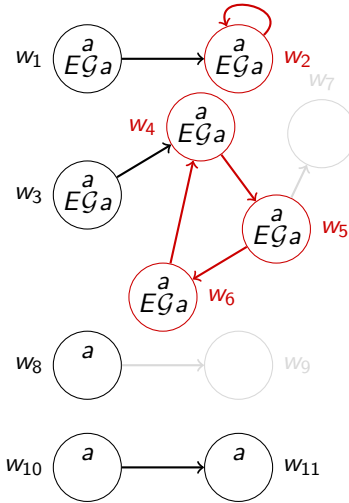
- Let $W_\phi = \{w \in W | \phi \in L(w)\}$
- Let $G = \langle W_\phi, R \cap (W_\phi \times W_\phi) \rangle$
- Find strongly connected components of G
- For each SCC C such that $|C| > 1$ or wRw for some $w \in C$, do $L(w) := L(w) \cup \{E\mathcal{G}\phi\}$ for all $w \in C$
- For each $w \in W_\phi$, if $E\mathcal{G}\phi \in L(w')$ for some w' such that wRw' then $L(w) := L(w) \cup \{E\mathcal{G}\phi\}$



Systematic algorithm given on the next slide...

CTL Model-Checking: Labeling for $E\mathcal{G}\phi$

- Let $W_\phi = \{w \in W | \phi \in L(w)\}$
- Let $G = \langle W_\phi, R \cap (W_\phi \times W_\phi) \rangle$
- Find strongly connected components of G
- For each SCC C such that $|C| > 1$ or wRw for some $w \in C$, do $L(w) := L(w) \cup \{E\mathcal{G}\phi\}$ for all $w \in C$
- For each $w \in W_\phi$, if $E\mathcal{G}\phi \in L(w')$ for some w' such that wRw' then $L(w) := L(w) \cup \{E\mathcal{G}\phi\}$



Systematic algorithm given on the next slide...

CTL Model-Checking: Labeling for $E\mathcal{G}\phi$

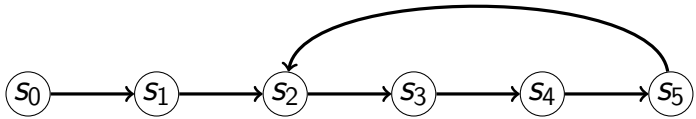
PROCEDURE EG(α)
$S' := \{w \in W | \alpha \in L(w)\}$;
$SCC := \{C | C$ is a SCC of $\langle S', R \cap (S' \times S') \rangle, |C| \geq 1$ or there is $w \in C$ with $wRw\}$;
$T := \{w \in C | C \in SCC\}$;
FOR EACH $w \in T$ DO $L(w) := L(w) \cup \{E\mathcal{G}\alpha\}$;
WHILE $T \neq \emptyset$ DO
  take any $w \in T$;
  $T := T \backslash \{w\}$;
  FOR EACH $t \in S'$ such that tRw DO
    IF $E\mathcal{G}\alpha \notin L(t)$ THEN
      $L(t) := L(t) \cup \{E\mathcal{G}\alpha\}$;
      $T := T \cup \{t\}$;
    END IF
  END FOR
END WHILE

Bounded Model-Checking for LTL

- SAT-based LTL model-checking solves two exponential problems at the same time:
  - Find a path in an exponential-size transition graph
  - Test that the LTL formula is satisfied on that path

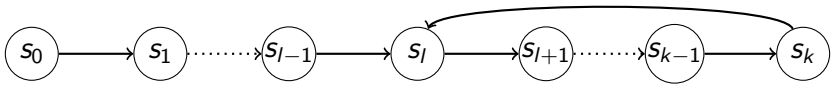
  Large size of transition systems masks the PSPACE-hardness of LTL model-checking
- LTL model-checking popular since 1999, due to very good scalability of SAT solvers
- Use of CTL model-checking decreasing, due to transition systems being very large

Bounded Model-Checking for LTL



- Model-checking limited by size of computation graphs (1M, 1B)
- Symbolic methods (OBDD, SAT) can relax this limitation
- Biere et al. (1999) reduced LTL model-checking to SAT
- Consider paths of the form $s_0, s_1, \ldots, \overbrace{s_l, \ldots, s_k}, \overbrace{s_l, \ldots, s_k}, s_l, \ldots$
- The segment $s_l, \ldots, s_k$ repeats an infinite number of times
- Encode this path with a loop as a propositional formula

Bounded Model-Checking for LTL



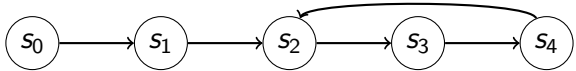
$$[\![x]\!]_i^{l,k} = x@i$$
$$[\![\neg x]\!]_i^{l,k} = \neg x@i$$
$$[\![\phi_1 \vee \phi_2]\!]_i^{l,k} = [\![\phi_1]\!]_i^{l,k} \vee [\![\phi_2]\!]_i^{l,k}$$
$$[\![\phi_1 \wedge \phi_2]\!]_i^{l,k} = [\![\phi_1]\!]_i^{l,k} \wedge [\![\phi_2]\!]_i^{l,k}$$

$$[\![\mathcal{X}\phi]\!]_i^{l,k} = [\![\phi]\!]_{succ(i)}^{l,k}$$
$$[\![\mathcal{G}\phi]\!]_i^{l,k} = \bigwedge_{j=\min(l,i)}^{k} [\![\phi]\!]_j^{l,k}$$
$$[\![\mathcal{F}\phi]\!]_i^{l,k} = \bigvee_{j=\min(l,i)}^{k} [\![\phi]\!]_j^{l,k}$$
$$[\![\phi\mathcal{U}\psi]\!]_i^{l,k} = \bigvee \begin{array}{l} \bigvee_{j=i}^{k} ([\![\psi]\!]_j^{l,k} \wedge \bigwedge_{z=i}^{j-1} [\![\phi]\!]_z^{l,k}) \\ \bigvee_{j=l}^{i-1} ([\![\psi]\!]_j^{l,k} \wedge \bigwedge_{z=l}^{j-1} [\![\phi]\!]_z^{l,k} \wedge \bigwedge_{z=i}^{k} [\![\phi]\!]_z^{l,k}) \end{array}$$
$$[\![\phi R\psi]\!]_i^{l,k} = \text{ similar}$$

Here: $succ(i) = i + 1$ if $i < k$, $succ(k) = l$

ϕ is true on some path (representable as a (k, l)-loop) iff $[\![\phi]\!]_0^{l,k} \wedge I \wedge T[X@1/X'] \wedge \cdots \wedge T[X@(k-1)/X, X@k/X'] \wedge T[X@k/X, X@l/X']$ is satisfiable

Bounded Model-Checking for LTL



$$
\begin{aligned}
[\![\mathcal{X}\mathcal{G}\,a]\!]_0^{2,4} &= a@1 \wedge a@2 \wedge a@3 \wedge a@4 \\
[\![a\,\mathcal{U}\,b]\!]_1^{2,4} &= b@1 \\
&\quad \vee (b@2 \wedge a@1) \\
&\quad \vee (b@3 \wedge a@1 \wedge a@2) \\
&\quad \vee (b@4 \wedge a@1 \wedge a@2 \wedge a@3)
\end{aligned}
$$

Bounded Model-Checking for LTL

- 2-D search: Test formulas for (k, l)-loops with increasing k, l
- 1-D search: Encode choice of l in formulas for fixed k, increase k
- Strengths:
  - Very effective in finding faulty behaviors (for "low" values of k)
  - Applicable to far bigger systems than OBDDs
- Weaknesses:
  - No general method for deciding when to stop increasing k
  - Hence: Often not a practical method for proving correctness

Abstraction in Verification

- Correctness of a system can sometimes be determined from an abstraction of a system
  - Ignore some aspects of the system model
  - Abstracted system has more possible behaviors than the original system
  - If all executions of abstracted systems have given property, then so do those of the original system
- Reasoning about the abstract system can be far easier
  - Fewer state variables $\longrightarrow$ number of abstract states smaller
- Applications:
  - Software verification (conventional program code)
  - sequential circuits (CPUs)
  - others

How to Abstract a System Model

Abstracting a state = distinctions between states eliminated
Abstracting state variables = distinctions between values eliminated

Examples of abstracting state variables

- integers x, y by $f_{x,y} : \mathbb{Z} \times \mathbb{Z} \to \{0, 1\}$ such that
  $$f_{x,y}(x, y) = \begin{cases} 1 & \text{if } x > y \\ 0 & \text{otherwise} \end{cases}$$
- Boolean b by $f_b(v) = 0$ (constant function: variable eliminated)
- weekday d by $f_d(Sat) = 0$, $f_d(Sun) = 0$, $f_d(d) = 1$ otherwise

How to Abstract a System Model

Abstracting states: s mapped to $s' = f(s)$ by mapping values of each variable or subset of variables to its abstraction:

- state variables X partitioned to $X_1, \ldots, X_n$
- abstraction functions are $f_1, \ldots, f_n$
- abstract state variables are $y_1, \ldots, y_n$
- value $s'(y_i)$ in the abstracted state is obtained by f_i:

$$s'(y_i) = f_i(x_1^i, \ldots, x_{j_i}^i)$$

where $X_i = \{x_1^i, \ldots, x_{j_i}^i\}$

Equivalence relation $\equiv_f$ induced by f: $s_1 \equiv_f s_2$ iff $f(s_1) = f(s_2)$

A Simple Special Class of Abstractions

We next only consider abstracting Boolean variables one by one

- Boolean state variables x are abstracted either
  - by $f_x(v) = 0$ (eliminating the variable), or
  - $f_x(v) = v$ (retaining the variable)
- A subset of state variables is eliminated, others remain intact

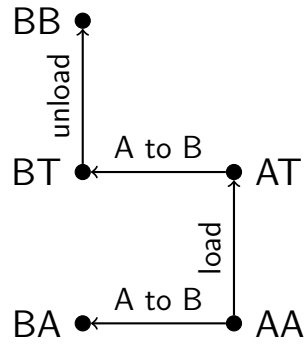
How to Abstract a System Model

- Simple abstraction: eliminate some state variable(s)
- Eliminate x from formula ϕ: generate $\exists x.\phi \leftrightarrow \phi[\top/x] \vee \phi[\bot/x]$
- Eliminate x from effect e:
  - Replace all assignments $x := B$ by ϵ
  - Replace $\mathsf{eITE}(\phi, e_1, e_2)$ by $\mathsf{eITE}(\exists x.\phi, e_1', e_2')$ where e_i' is e_i with x eliminated
- This allows eliminating x from any transition rule (ϕ, e)

Example

- Load object in truck at A:
  (TatA$\wedge$OatA, (OatA := 0; OinT := 1))
- Unload object from truck at B:
  (TatB$\wedge$OinT, (OinT := 0; OatB := 1))
- Move truck from A to B:
  (TatA, (TatA := 0;TatB := 1))

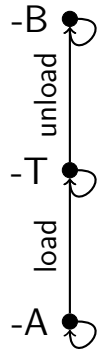
Possible states AA, BA, AT, BT, BB



Example, Abstracted

Abstract away both TatA and TatB:

- Load object in truck at A:
  (OatA, (OatA := 0; OinT := 1))
- Unload object from truck at B:
  (OinT, (OinT := 0; OatB := 1))
- Move truck from A to B: ($\top$, ϵ)



Possible states:

abstract	concrete
-B	BB
-T	AT, BT
-A	AA, BA

Arrows in the abstraction are a superset of the arrows in the original system

Properties of Abstractions

Let T be a transition system and T' its abstraction (w.r.t vars Z).
Let ϕ be a formula and $\phi' = \exists Z.\phi$.

Theorems:

- If the path in T has length n, then the path in T' has length $\leq n$
- If there is path to ϕ in T, then there is a path to ϕ' in T'

Corollary:

- If there is no path to ϕ' in T', then there is no path to ϕ in T

Properties of Abstractions: Spurious Paths

- If there is path p to ϕ' in T', then there might be no path to ϕ in T. Here p is called a spurious path.
- Core question in model-checking with abstractions: What to do with spurious paths?
  - We want to test if LTL formula ϕ can be true on a path
  - There is such a path in the abstracted system
  - Does this path have a counterpart in the concrete system?

Reachability Checking with Abstraction

Test if any state satisfying ϕ is reachable:

1. Build initial abstraction of transition system (not abstracting ϕ)
2. Test if ϕ is reachable in the abstracted system.
3. If ϕ not reachable in abstraction, also not reachable in unabstracted system. Stop.
4. Let $t_1, \ldots, t_n$ be the transition sequence reaching ϕ
5. If $t_1, \ldots, t_n$ is a transition sequence also in the original system, stop.
6. Generate a new (less abstract) abstraction.
7. Go to step 2.

Counterexample-Guided Abstraction Refinement

A spurious path has been found

How is the abstraction refined?

1. Given $t_1, \ldots, t_n$, identify the maximal prefix $t_1, \ldots, t_i$ that is executable in the original transition system
2. t_{i+1} is not possible in the original system, only in the abstraction
3. Consider the corresponding state sequences $s'_0, \ldots, s'_i$ and $s_0, \ldots, s_i$
4. Transition t_{i+1} is possible in s'_i, but not in s_i
5. This difference suggests how the abstraction must be changed!

Counterexample-Guided Abstraction Refinement

The (spurious) plan to transport object from A to B:

1. load object in vehicle in location A
2. unload object from vehicle in location B

The state sequences:

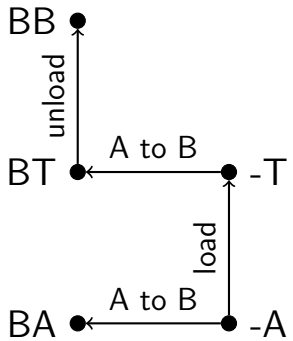
action	state	original system					abstract system				
		TatA	TatB	OatA	OinT	OatB	TatA	TatB	OatA	OinT	OatB
load	s_0	1	0	1	0	0	-	-	1	0	0
unload	s_1	1	0	0	1	0	-	-	0	1	0
	s_2	NA	NA	NA	NA	NA	-	-	0	0	1

In the original system, unload's precondition TatB$\wedge$ OinT is false!
Refine the abstraction: include the state variable TatB.

Example, Abstraction Refined

- Load object in truck at A:
  (OatA, (OatA := 0; OinT := 1))
- Unload object from truck at B:
  (TatB∧OinT, (OinT := 0; OatB := 1))
- Move truck from A to B:
  (⊤, (TatB := 1))

Possible states -A, BA, -T, BT, BB



Counterexample-Guided Abstraction Refinement

- CEGAR applicable to all leading methods
  - Explicit state reachability analysis (e.g. breadth-first search)
  - Explicit state model-checking
  - OBDD-based reachability & model-checking
  - SAT-based reachability & model-checking
- Critical in SW model-checking, where abstraction is unavoidable (due to the typically infinite state space)