Unbounded Model Checking

Stefan Ratschan

Katedra číslicového návrhu
Fakulta informačních technologií
České vysoké učení technické v Praze

Evropský sociální fond Praha & EU: Investujeme do vaší budoucnosti

Forever: Infinity

How to check that a certain model fulfills a specification?

Problem: transition systems have infinitely many paths of infinite length

Testing, bounded model checking: bounded time

Unbounded Model Checking

Today: method for proving LTL properties

So: no bound on time

$\mathbf{G}p$ (safety verification)

. . . and a little bit of $\mathbf{F}p$

. . . and even less of $p\mathbf{R}q$

Why?

Technical systems have limited lifetime

Why paths of infinite length??

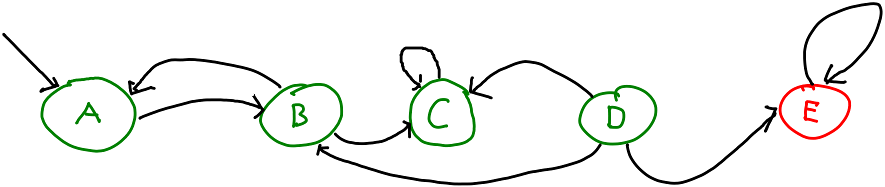
There is only a finite number of atoms on earth

Why natural numbers? Why real numbers?

In practice, lifetime can easily can be 10000000000000000000000 steps

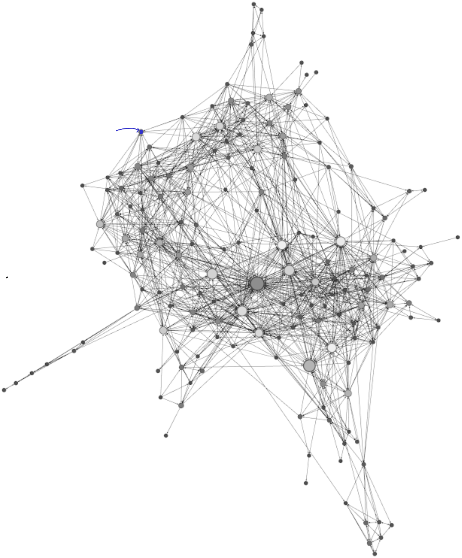
Safety Verification: **G** `ok`

Assume a transition system (S, S_0, R)



$\mathcal{I}(\texttt{ok}) = \{A, B, C, D\}$

How is it possible to convince somebody else that $\models \mathbf{G}\, \texttt{ok}$?



Safety Certificates

A set of states V is a *safety certificate* iff

- V contains every initial state: $S_0 \subseteq V$
- No transition leads out of V: $Post_R(V) \subseteq V$ where
  $Post_R(V) := \{x' \mid x \in V, (x, x') \in R\}$
- V contains only safe (i.e., no unsafe) states: $V \subseteq \mathcal{I}(\texttt{ok})$

Three conditions: *safety verification conditions*

First two conditions: *inductivity conditions*. What do they ensure?

Inductivity Conditions

- V contains every initial state: $S_0 \subseteq V$
- No transition leads out of V: $Post_R(V) \subseteq V$

Imply that V contains all reachable states:

$$\{\pi(k) \mid \pi \text{ path}, k \in \mathbb{N}_0\} \subseteq V$$

With third condition: all reachable states are safe.

The set of all reachable states $\{\pi(k) \mid \pi \text{ path}, k \in \mathbb{N}_0\}$ is also called *reach set*

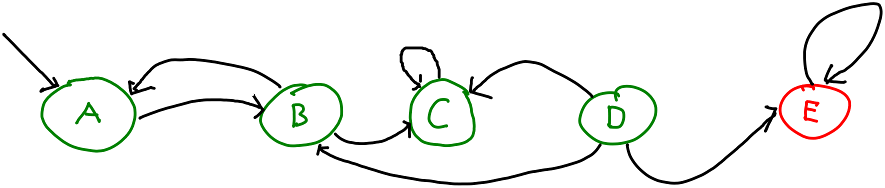
An arbitrary set that contains all reachable states, that is, any super-set of the reach set is called *invariant*.

A set that fulfills the inductivity conditions is called *inductive invariant*.

Discussion

Is every invariant inductive?

Does every set that contains all reachable states
fulfill the inductivity conditions?



reach set: $\{A, B, C\}$

$\{A, B, C, D\}$? invariant, but not inductive

Does every transition system have an inductive invariant? reach set

Trivial inductive invariant:
contains all states of the given transition system

Questions

- How to check the safety verification conditions?
- How to compute a safety certificate?

How to Check Conditions?

Sometimes we have a guess for a safety certificate V:

- an expert can guess V, or
- can be part of documentation (see assert)

Safety verification conditions:

- $S_0 \subseteq V$
- $Post_R(V) \subseteq V$
- $V \subseteq \mathcal{I}(\mathtt{ok})$

Observation: set operations, especially subset relation

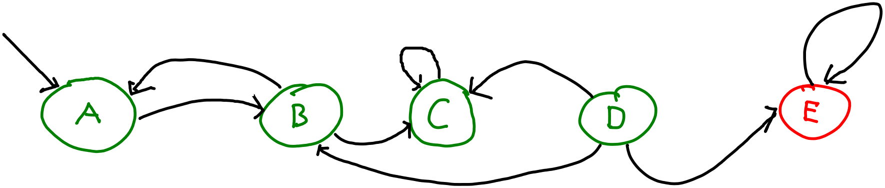
Checking: for small sets no problem

For large or infinite sets?

Symbolic Checking of Safety Verification Conditions

Assumption: Transition system, V, and ok given symbolically, using logical formulas:

$$(S, \{x \mid S_0(x)\}, \{(x, x') \mid R(x, x')\}), \{x \mid \mathtt{ok}(x)\}, \{x \mid V(x)\}$$



	x_1	x_2	x_3
A	$\top$	$\top$	$\top$
B	$\top$	$\top$	$\bot$
C	$\top$	$\bot$	$\top$
D	$\top$	$\bot$	$\bot$
E	$\bot$	$\bot$	$\bot$

$V = \{A, B, C\}$

S_0: $x_1 \wedge x_2 \wedge x_3$
R: $[x_1 \wedge x_2 \wedge x_3 \wedge x_1' \wedge x_2' \wedge \neg x_3'] \vee \dots$
ok: x_1
V: $x_1 \wedge [x_2 \vee x_3]$

Symbolic Checking of Safety Verification Conditions

Assumption: Transition system, V, and $\texttt{ok}$ given symbolically, using logical formulas:

$$(S, \{x \mid S_0(x)\}, \{(x, x') \mid R(x, x')\}), \{x \mid \texttt{ok}(x)\}, \{x \mid V(x)\}$$

After substituting:

- $S_0 \subseteq V$
- $\{x' \mid x \in V, (x, x') \in R\} \subseteq V$
- $V \subseteq \mathcal{I}(\texttt{ok})$

After simplification,

safety verification conditions in the form of predicate logical formulas:

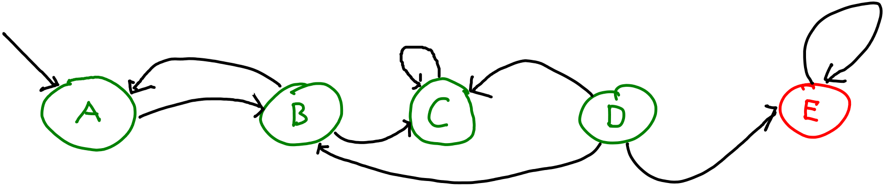
- $\forall x \,.\, S_0(x) \Rightarrow V(x)$
- $\forall x \forall x' \,.\, [V(x) \wedge R(x, x')] \Rightarrow V(x')$
- $\forall x \,.\, V(x) \Rightarrow \texttt{ok}(x)$

Example Continued

Safety verification conditions:

- $\forall x \,.\, S_0(x) \Rightarrow V(x)$
- $\forall x \forall x' \,.\, [V(x) \wedge R(x, x')] \Rightarrow V(x')$
- $\forall x \,.\, V(x) \Rightarrow \texttt{ok}(x)$

S_0: $x_1 \wedge x_2 \wedge x_3$
R: $\ldots$
$\texttt{ok}$: x_1
V: $x_1 \wedge [x_2 \vee x_3]$



- $[x_1 \wedge x_2 \wedge x_3] \Rightarrow [x_1 \wedge [x_2 \vee x_3]]$
- $[x_1 \wedge [x_2 \vee x_3] \wedge R(x_1, x_2, x_3, x_1', x_2', x_3')] \Rightarrow [x_1' \wedge [x_2' \vee x_3']]$
- $[x_1 \wedge [x_2 \vee x_3]] \Rightarrow x_1$

What about the quantifiers? $\models$

Automatic Check Using Solver

We have to prove:

- $S_0(x) \Rightarrow V(x)$
- $[V(x) \wedge R(x, x')] \Rightarrow V(x')$
- $V(x) \Rightarrow \mathtt{ok}(x)$

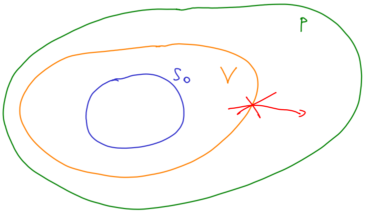
But SAT solvers do not prove, they check satisfiability!

It suffices to check unsatisfiability of

- $\neg[S_0(x) \Rightarrow V(x)]$
- $\neg[[V(x) \wedge R(x, x')] \Rightarrow V(x')]$
- $\neg[V(x) \Rightarrow \mathtt{ok}(x)]$

which is

- $S_0(x) \wedge \neg V(x)$
- $V(x) \wedge R(x, x') \wedge \neg V(x')$
- $V(x) \wedge \neg\mathtt{ok}(x)$



Computation of Safety Certificate

If safety certificate not yet known?

In finite case, we can view transition relation as a directed graph.

So: use graph algorithms, which ones?

$\models$ **G** ok? How to discover this?

Checking **G** ok by Depth-First Search

Notation: for finite path t, $|t|$ denotes its length, $last(t) = t(|t| - 1)$

search(t, V) // *modifies V, denote input value by V^{in}*
In: finite path t, $\{t(i) \mid 0 \leq i < |t| - 1\} \subseteq V \subseteq \mathcal{I}(\texttt{ok})$
Out: if $\perp$ then $\not\models$ $\mathbf{G}\texttt{ok}$ else

- $V^{in} \subseteq V \subseteq \mathcal{I}(\texttt{ok})$
- every state that can be reached from $last(t)$ without reaching a state in V^{in} is in V

if $last(t) \in V$ **then**
    return $\top$
else if $last(t) \notin \mathcal{I}(ok)$ **then** // *counter-example found*
    return $\perp$
else
    $V \leftarrow V \cup \{last(t)\}$
    return $\forall s' \,.\, R(last(t), s') \Rightarrow$ search(ts', V)

Checking **G** ok by Depth-First Search

Initial call:

$V \leftarrow \emptyset$
if for all $s \in S_0$, search(s, V) **then**
  return V // *reach set, hence a safety certificate*
else
  return $\perp$ // *ideally, return counter-example here*

Explicit State Model Checking

Checking the correctness of transition systems by
going through the individual states

Basis: classical graph algorithms

Depth-first search:

- resulting counter-example tend to be long
- may get lost in irrelevant parts of graph
- does not work for systems with infinitely many states

Breadth-first, best-first:
high memory requirements (queue)

Best-first search (*directed model checking*) mainly useful in cases where

- there are good heuristics
- the goal is mainly to find counter-examples
  (see also BMC for planning)

Set Based Algorithm

- $S_0 \subseteq V$
- $Post_R(V) \subseteq V$
- $V \subseteq \mathcal{I}(\mathtt{ok})$

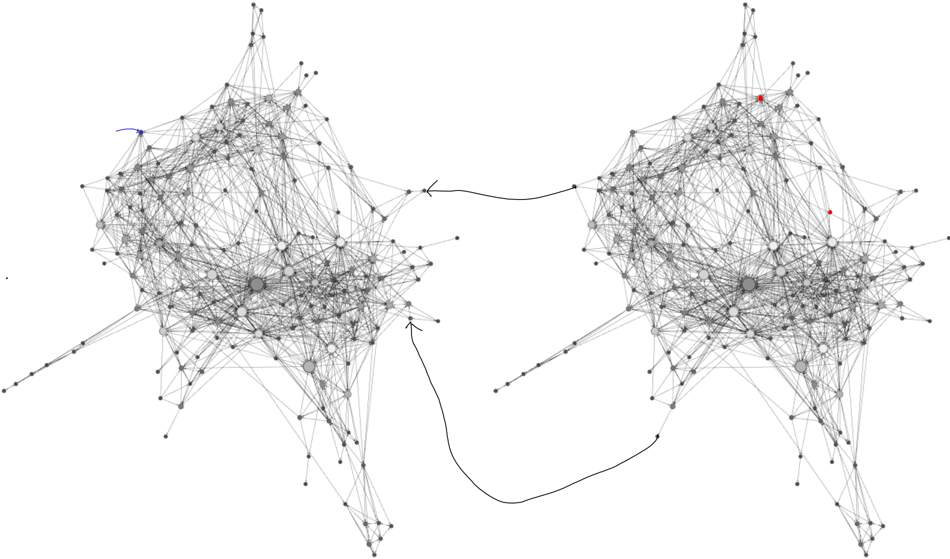
Set fulfilling first condition? S_0

... second condition? Add additional states

$V \leftarrow S_0$
while there is a transition (x, x') such that $x \in V$, $x' \notin V$ **do**
    $V \leftarrow V \cup \{x' \mid (x, x') \in R, x \in V\}$ // $V \cup Post_R(V)$
return V

If resulting $V \subseteq \mathcal{I}(\mathtt{ok})$ then V fulfills verification conditions.

Computation of Safety Certificates



Analyis of Algorithm

$V \leftarrow S_0$
while there is a transition (x, x') such that $x \in V$, $x' \notin V$ **do**
  $V \leftarrow V \cup \{x' \mid (x, x') \in R, x \in V\}$
return V

The resulting set V is an inductive invariant
  (and so contains all reachable states)

Will it also fulfill the third verification condition $V \subseteq \mathcal{I}(\texttt{ok})$?

In k-th cycle, V is the set of states reachable in k steps.

During algorithm execution V never contains unreachable states

The algorithm computes the set of reachable states

Conclusion: If the property $\mathbf{G}p$ holds, then
  the result is a safety certificate

Algorithm Termination

Algorithm for computing inductive invariants:

$V \leftarrow S_0$
while there is a transition (x, x') such that $x \in V$, $x' \notin V$ **do**
    $V \leftarrow V \cup \{x' \mid (x, x') \in R, x \in V\}$
return V

- If V does not yet contain all reachable states,
  the algorithm continues further.
- The algorithm always terminates (for finite transition systems)

Efficient Computation of Inductive Invariants

Problems:

- Slow convergence, non-termination of infinite systems
- Complex sets

In practice:

let V be a superset of S_0
while there is a transition (x, x') such that $x \in V$, $x' \notin V$ **do**
  let V be a superset of $V \cup \{x' \mid (x, x') \in R, x \in V\}$
return V

Still we have:

- If the algorithm terminates, then V is an inductive invariant
- In k-th cycle, V is a superset of the set of states reachable in k steps.

But: In general, result is a superset of reachable states.

Efficient Computation of Inductive Invariants

How much to overapproximate?

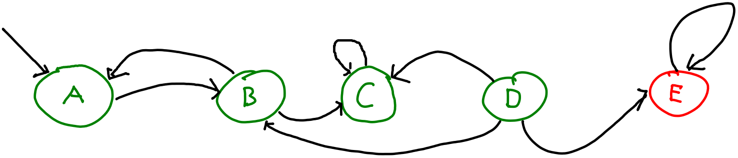
- Too much overapproximation might result in violation of $V \subseteq \mathcal{I}(\texttt{ok})$
- Too small overapproximation might result in termination problems (in infinite state case) or set representation blows up

Special data structures for compact set representation:
  BDD (Binary Decision Diagram)

symbolic model checking

Backward Computation

Instead of a set such that fulfills the safety verification conditions
we can also compute its complement.

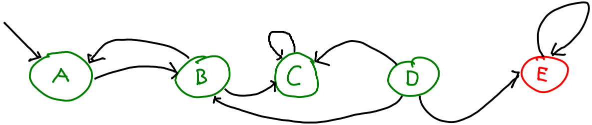


$\{D, E\}$

Especially: set of states that leads to an unsafe state:
backward reach set

If the result does not contain any initial state, safety proven.

Backward Computation



For all the above algorithms there are versions that work in opposite direction: backwards from set of unsafe states to set of initial states.

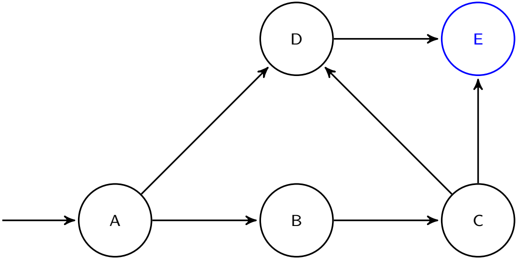
No need to re-implement: Use transition system (S, S_0^-, R^-) where

- $S_0^- = S \setminus \mathcal{I}(\texttt{ok}) = \{s \mid s \models \neg\texttt{ok}\}$
- $R^- = \{(x', x) \mid (x, x') \in R\}$

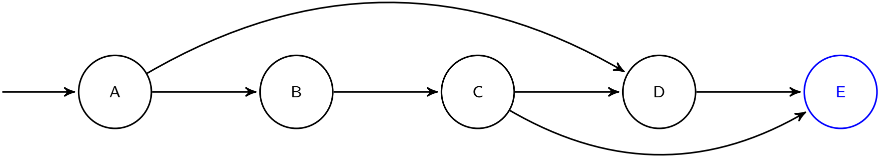
Check $\models \mathbf{G}\neg\texttt{ok}^-$, where $\mathcal{I}(\texttt{ok}^-) = S_0$.

Certificate for **F** goal

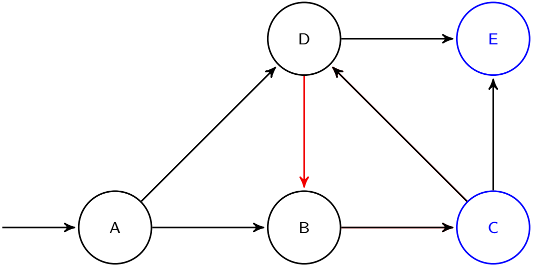
How is it possible to convince somebody else that $(S, S_0, R) \models \mathbf{F}$ goal?



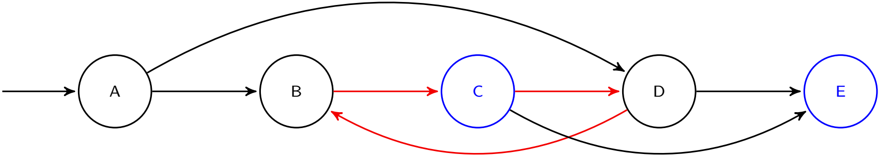
$\models \mathbf{F}$ goal where $\mathcal{I}(goal) = \{E\}$?



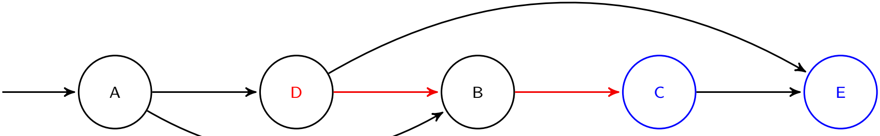
topological sort, strict linear (total) order on all states



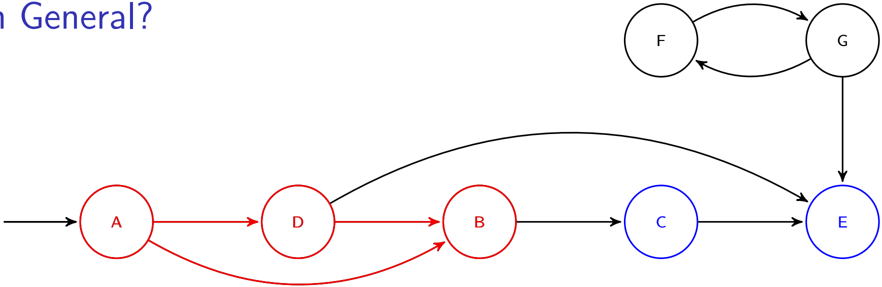
$\models$ **F** goal where $\mathcal{I}(goal) = \{C, E\}$? topological sort?



But in state C we already reached the goal!



In General?



Strict linear order on $\{A, D, B\}$, leaving set reaches goal.

Certificate for **F**goal:

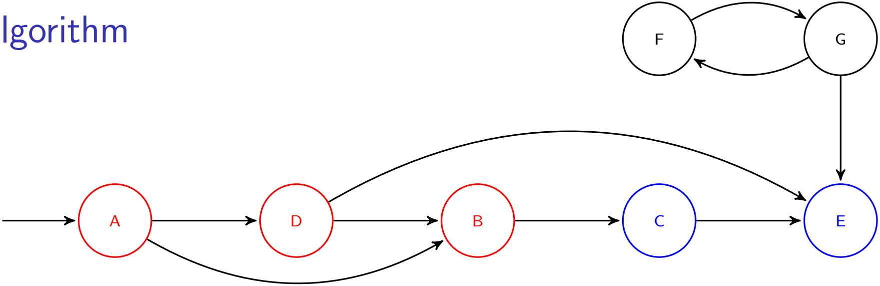
pair $(V, \succ)$ where $\succ$ is a strict linear order on V s.t.

- $S_0 \subseteq V \subseteq S$
- for all $(s, s') \in R$, $s \in V$, $s' \notin V$ implies $s' \in \mathcal{I}(goal)$
- for all $(s, s') \in (R \cap V \times V)$, $s \succ s'$

Similar objects:

- ranking function for proving program termination
- Lyapunov functions for proving stability of continuous systems

Algorithm



pair $(V, \succ)$ where $\succ$ is a strict linear order on V s.t.

- $S_0 \subseteq V \subseteq S$
- for all $(s, s') \in R$, $s \in V$, $s' \notin V$ implies $s' \in \mathcal{I}(goal)$
- for all $(s, s') \in (R \cap V \times V)$, $s \succ s'$

Algorithm principles:

- depth-first traversal,
- stop in goal states,
- for cycles return $\perp$.

Checking **F** goal by Depth-First Search

search(t,V) // *modifies V, denote input value by V^{in}*

In: finite path t, list V

Out: if $\bot$ then $\not\models$ **F** goal else

- V^{in} is a suffix of V
- for all $(s, s') \in R$, s.t. $s \in V$, $s' \in V$, $s' \notin \mathcal{I}(goal)$, s occurs in the list V before s'.
- every state that can be reached from $last(t)$ without reaching a state in $\mathcal{I}(goal)$ or V^{in} is in V

if $last(t) \notin \mathcal{I}(goal) \wedge \exists i \in \{0, \dots, |t| - 2\} . t(i) = last(t)$ **then**
    return $\bot$
else if $last(t) \in \mathcal{I}(goal) \vee last(t) \in V$ **then**
    return $\top$
else
    $r \leftarrow \forall s' . R(last(t), s') \Rightarrow \text{search}(ts', V)$
    $V \leftarrow last(t) :: V$
    return r

Checking **F** goal by Depth-First Search

Initial call:

$V \leftarrow <>$
if for all $s \in S_0$, search(s, V) **then**
    return V
else
    return $\perp$ // *ideally, return counter-example here*

Checking **F** goal by Depth-First Search

search(t,V)

if $\mathit{last}(t) \notin \mathcal{I}(\mathit{goal}) \wedge \exists i \in \{0, \dots, |t|-2\} \;.\; t(i) = \mathit{last}(t)$ **then**

  return $\bot$

else if $\mathit{last}(t) \in \mathcal{I}(\mathit{goal}) \vee \mathit{last}(t) \in V$ **then**

  return $\top$

else

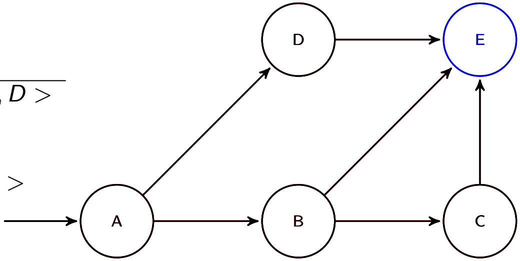
  $r \leftarrow \forall s' \;.\; R(\mathit{last}(t), s') \Rightarrow \text{search}(ts', V)$

  $V \leftarrow \mathit{last}(t) :: V$

  return r

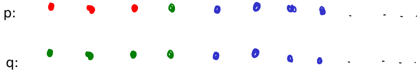
$\mathcal{I}(\mathit{goal}) = \{E\}$

t	V^{in}	V^{out}
A	$<>$	$< A, B, C, D >$
AD	$<>$	$< D >$
ADE	$<>$	$<>$
AB	$< D >$	$< B, C, D >$
ABE	$< D >$	$< D >$
ABC	$< D >$	$< C, D >$



Certificate for $p\mathbf{R}q$

$\pi \models p\mathbf{R}q$ iff for all j, [if for all $i < j$, $\pi^i \not\models p$, then $\pi^j \models q$]



$\mathbf{G}\phi$ is equivalent to $\bot\mathbf{R}\phi$

Certificate for $\mathbf{G}\texttt{ok}$:

- $\forall x \,.\, S_0(x) \Rightarrow V(x)$
- $\forall x \forall x' \,.\, [V(x) \wedge R(x, x')] \Rightarrow V(x')$
- $\forall x \,.\, V(x) \Rightarrow \texttt{ok}(x)$

If $p \neq \bot$, weaker conditions will suffice:

- $\forall x \,.\, S_0(x) \Rightarrow V(x)$
- $\forall x \forall x' \,.\, [V(x) \wedge R(x, x') \wedge \neg p(x)] \Rightarrow V(x')$
- $\forall x \,.\, V(x) \Rightarrow q(x)$

Certificate for $p\mathbf{R}q$: Example

$\pi \models p\mathbf{R}q$ iff for all j, [if for all $i < j$, $\pi^i \not\models p$, then $\pi^j \models q$]

$\mathcal{I}(p) = \{C, D, E\}$
$\mathcal{I}(q) = \{A, B, C, D\}$

- $\forall x \,.\, S_0(x) \Rightarrow V(x)$
- $\forall x \forall x' \,.\, [V(x) \wedge R(x, x') \wedge \neg p(x)] \Rightarrow V(x')$
- $\forall x \,.\, V(x) \Rightarrow q(x)$

Certificate: $\{A, B, C\}$

Arbitrary LTL formulas

In the case of a finite number of states there are methods for checking arbitrary LTL (CTL) formulas.

Tools:

- SPIN: `http://spinroot.com`
- SAL: `http://sal.csl.sri.com`
- NuSMV: `http://nusmv.fbk.eu`

Turing Award 2007:
Clarke/Emerson/Sifakis

Infinite Number of States

Principle also works for infinite state space

Example:
Flat computer programs over integers:
state space $\{1, \ldots, loc\} \times \mathbb{N}^k$,
where k is number of program variables, loc is the number of program lines

Here, and in several other cases:
safety verification undecidable.

But: Often works in practice, see area of "software model checking"
(see MI-FME)

Better than modeling the problem using variables with finite domain $\{0, \ldots, 2^{64} - 1\}$

Further examples till end of semester

Model Checking for Planning

Here: Non-determinism corresponds to decisions we can take.
Example: Should we switch on coal-fired power station? Yes/no

Given:

- Transition system $(S, \{s_0\}, R)$, where
  s_0 is the current state of the system, a
- LTL formula ϕ,

Find: path π, s.t. $\pi \models \phi$
(plan what to do from current state s_0)

Idea: Such a path is a counter-example to $\models \neg\phi$

Find plan by applying a method for checking $\models \neg\phi$

Unbounded Model Checking for Planning: Examples

We want a plan fulfilling **G**ok

- Method: Check ¬**G**ok, that is **F**¬ok
- Result: counter-example to **F**¬ok, that is,
  path, that will cycle outside of $\{s \mid s \models \neg\text{ok}\}$, in $\mathcal{I}(\text{ok})$

We want a plan fulfilling **F**goal

- Method: Check ¬**F**goal, that is **G**¬goal
- Result: counter-example to **G**¬goal, that is,
  path, that will reach $\mathcal{I}(\text{goal})$

When using BMC:

If $\text{BMC}(\neg\phi, n)$ holds, then there is no plan of length n,
that ensures that the requirement holds forever.

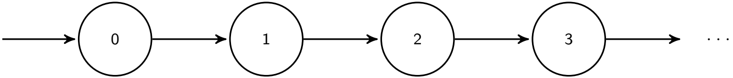
There might be a longer plan

Unbounded model checking finds plans of arbitrary length

Dynamic programming, reinforcement learning ...

Relationship to Mathematical Induction

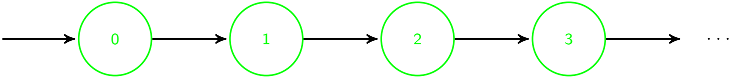
Transition system $(\cdot, \{0\}, \{(x, x') \mid x' = x + 1\})$

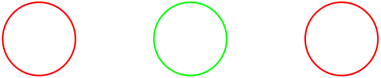


$\mathbf{G}p$, where p describes the property we want to prove.

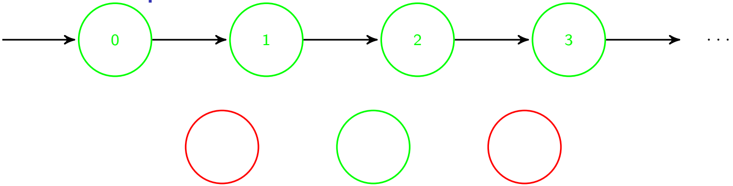
Example: $\mathcal{I}(p) = \{n \mid P\}$, where $P \equiv 0 + 1 + \cdots + n = \frac{n(n+1)}{2}$

$\forall x \in \mathbb{N}_0 \,.\, P(x) \;\Leftrightarrow \models \mathbf{G}p \;\Leftrightarrow\;$ all reachable sets are green:





Relationship to Mathematical Induction



Verification conditions (formulation based on logical formulas):

- $\forall x \,.\, S_0(x) \Rightarrow V(x)$
- $\forall x \forall x' \,.\, [V(x) \wedge R(x, x')] \Rightarrow V(x')$
- $\forall x \,.\, V(x) \Rightarrow P(x)$

For $V = P$:

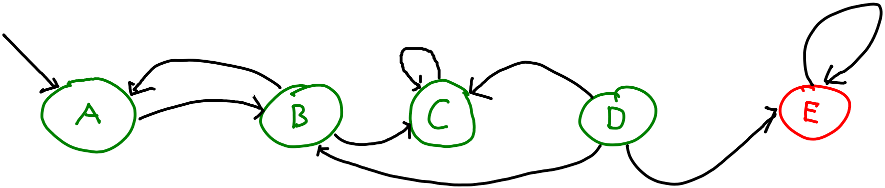
- $\forall x \,.\, S_0(x) \Rightarrow P(x)$
- $\forall x \forall x' \,.\, [P(x) \wedge R(x, x')] \Rightarrow P(x')$
- $\forall x \,.\, P(x) \Rightarrow P(x)$

Final condition holds trivally, the rest is classical mathematical induction!

Relationship to Mathematical Induction

- $S_0 \subseteq V$
- $\{x' \mid x \in V, (x, x') \in R\} \subseteq V$
- $V \subseteq \mathcal{I}(p)$

Why not, in general, choose V as $\mathcal{I}(p)$?



Cannot happen for standard mathematical induction on natural numbers.

In different mathematical structures a different choice is used, i.e., for proving a certain property a different property is proven that implies the given property.

Elevator example

https://youtu.be/pRSZTuEPacU

https:
//www.thyssenkrupp-elevator.com/uk/products/elevators/twin

Conclusion

Certificates

It is possible to check an infinite number of paths of infinite length!

Principles: induction, linear orders, etc.

Literature: Principles are used all over computer science/mathematics, but are hardly every explicitely explained.

Now:

- We know finite state models, we are able to specify properties, and to check them (automatically).
- Rest of semester: more expressive models
  (infinite number of states, probability etc.)