Formalization: Variables and Domains

variable	domain	explanation
st_coffee	$\{0, 1\}$	drink storage contains coffee
st_beer	$\{0, 1\}$	drink storage contains beer
disp	$\{\mathit{none}, \mathit{beer}, \mathit{coffee}\}$	content of drink dispenser
coins	$\{\mathit{0}, \mathit{1}, \mathit{2}, \mathit{3}\}$	number of coins in the slot
customer	$\{\mathit{none}, \mathit{student}, \mathit{prof}\}$	customer

Temporal properties of states

Using LTL we can express temporal properties of states:

- Students only drink coffee:
  $\Box(\text{customer} = \textit{student} \rightarrow \text{disp} = \textit{coffe})$
- Fairness to customers:
  $\Box((\text{customer} = \textit{student} \rightarrow \Diamond\text{customer} = \textit{prof}) \wedge (\text{customer} = \textit{prof} \rightarrow \Diamond\text{customer} = \textit{student}))$
- drinks are dispensed infinitely many times:
  $\Box\Diamond(\neg\text{disp} = \textit{none})$

Can we also express properties of transitions ?

Temporal properties of states

Using LTL we can express temporal properties of states:

- Students only drink coffee:
  $\Box(\text{customer} = \textit{student} \rightarrow \text{disp} = \textit{coffe})$
- Fairness to customers:
  $\Box((\text{customer} = \textit{student} \rightarrow \Diamond\text{customer} = \textit{prof}) \wedge (\text{customer} = \textit{prof} \rightarrow \Diamond\text{customer} = \textit{student}))$
- drinks are dispensed infinitely many times:
  $\Box\Diamond(\neg\text{disp} = \textit{none})$

Can we also express properties of transitions ?

Temporal properties of states

Using LTL we can express temporal properties of states:

- Students only drink coffee:
  $\Box(\text{customer} = \textit{student} \rightarrow \text{disp} = \textit{coffe})$
- Fairness to customers:
  $\Box((\text{customer} = \textit{student} \rightarrow \Diamond\text{customer} = \textit{prof}) \wedge (\text{customer} = \textit{prof} \rightarrow \Diamond\text{customer} = \textit{student}))$
- drinks are dispensed infinitely many times:
  $\Box\Diamond(\neg\text{disp} = \textit{none})$

Can we also express properties of transitions ?

Temporal properties of states

Using LTL we can express temporal properties of states:

- Students only drink coffee:
  $\Box(\text{customer} = \textit{student} \rightarrow \text{disp} = \textit{coffe})$
- Fairness to customers:
  $\Box((\text{customer} = \textit{student} \rightarrow \Diamond\text{customer} = \textit{prof}) \wedge (\text{customer} = \textit{prof} \rightarrow \Diamond\text{customer} = \textit{student}))$
- drinks are dispensed infinitely many times:
  $\Box\Diamond(\neg\text{disp} = \textit{none})$

Can we also express properties of transitions ?

Temporal properties of states

Using LTL we can express temporal properties of states:

- Students only drink coffee:
  $\Box(\text{customer} = \mathit{student} \rightarrow \text{disp} = \mathit{coffe})$
- Fairness to customers:
  $\Box((\text{customer} = \mathit{student} \rightarrow \Diamond\text{customer} = \mathit{prof}) \wedge (\text{customer} = \mathit{prof} \rightarrow \Diamond\text{customer} = \mathit{student}))$
- drinks are dispensed infinitely many times:
  $\Box\Diamond(\neg\text{disp} = \mathit{none})$

Can we also express properties of transitions ?

Transitions

1. *Recharge* which results in the drink storage having both beer and coffee.
2. *Customer_arrives*, after which a customer appears at the machine.
3. *Customer_leaves*, after which the customer leaves.
4. *Coin_insert*, when the customer inserts a coin in the machine.
5. *Dispense_beer*, when the customer presses the button to get a can of beer.
6. *Dispense_coffee*, when the customer presses the button to get a cup of coffee.
7. *Take_drink*, when the customer removes a drink from the dispenser.

Reasoning About Transitions

Consider the following properties:

1. "one cannot have two beers in a row without inserting a coin".
2. "If we never have two recharge transitions in a row, then the next transition after a recharge must be a customer arrival".

Note that they are about transitions, not about states.

Reasoning About Transitions

Consider the following properties:

1. "one cannot have two beers in a row without inserting a coin".
2. "If we never have two recharge transitions in a row, then the next transition after a recharge must be a customer arrival".

Note that they are about transitions, not about states.

How can one represent these properties?

Reasoning About Transitions

Consider the following properties:

1. "one cannot have two beers in a row without inserting a coin".
2. "If we never have two recharge transitions in a row, then the next transition after a recharge must be a customer arrival".

Note that they are about transitions, not about states.

How can one represent these properties?

Introduce a state variable denoting the next transition.

Example

$$\mathit{Recharge} \stackrel{\text{def}}{=} \mathrm{tr} = \mathit{Recharge} \land \mathrm{customer} = \mathit{none} \land \mathrm{st\_coffee}' \land \mathrm{st\_beer}' \land \mathit{only}(\mathrm{st\_coffee}, \mathrm{st\_beer}, \mathrm{tr}).$$

$$\mathit{Dispense\_beer} \stackrel{\text{def}}{=} \mathrm{tr} = \mathit{Dispense\_beer} \land \mathrm{customer} = \mathit{student} \land \mathrm{st\_beer} \land \mathrm{disp} = \mathit{none} \land (\mathrm{coins} = \mathit{2} \lor \mathrm{coins} = \mathit{3}) \land \mathrm{disp}' = \mathit{beer} \land (\mathrm{coins} = \mathit{2} \to \mathrm{coins}' = \mathit{0}) \land (\mathrm{coins} = \mathit{3} \to \mathrm{coins}' = \mathit{1}) \land \mathit{only}(\mathrm{st\_beer}, \mathrm{disp}, \mathrm{coins}).$$

$$\mathit{Customer\_arrives} \stackrel{\text{def}}{=} \mathrm{tr} = \mathit{Customer\_arrives} \land \mathrm{customer} = \mathit{none} \land \mathrm{customer}' \neq \mathit{none} \land \mathit{only}(\mathrm{customer}, \mathrm{tr})$$

$$\mathit{Coin\_insert} \stackrel{\text{def}}{=} \mathrm{tr} = \mathit{Coin\_insert} \land \mathrm{customer} \neq \mathit{none} \land \mathrm{coins} \neq \mathit{3} \land (\mathrm{coins} = \mathit{0} \to \mathrm{coins}' = \mathit{1}) \land (\mathrm{coins} = \mathit{1} \to \mathrm{coins}' = \mathit{2}) \land (\mathrm{coins} = \mathit{2} \to \mathrm{coins}' = \mathit{3}) \land \mathit{only}(\mathrm{coins}, \mathrm{tr}).$$

Representing Temporal Properties of Transitions

1. One cannot have two beers without inserting a coin in between getting them.

$$\Box(\mathrm{tr} = \mathit{Dispense\_beer} \to \bigcirc(\Box \mathrm{tr} \neq \mathit{Dispence\_beer} \;\vee\; \mathrm{tr} \neq \mathit{Dispence\_beer}\, \mathcal{U}\, \mathrm{tr} = \mathit{Insert\_coin}))$$

2. If we never have two recharge transitions in a row, then the next transition after a recharge must be a customer arrival.

$$\Box(\mathrm{tr} = \mathit{Recharge} \to \bigcirc \mathrm{tr} \neq \mathit{Recharge}) \to \Box(\mathrm{tr} = \mathit{Recharge} \to \bigcirc \mathrm{tr} = \mathit{Customer\_arrives})$$

3. The value of customer can only be changed as a result of either *Customer_arrives* or *Customer_leaves*.

$$\Box(\textstyle\bigwedge_{v \in dom(\mathrm{customer})}(\mathrm{customer} = v \wedge \bigcirc \mathrm{customer} \neq v) \to \mathrm{tr} = \mathit{Customer\_arrives} \vee \mathrm{tr} = \mathit{Customer\_leaves})$$

Representing Temporal Properties of Transitions

1. One cannot have two beers without inserting a coin in between getting them.

$$\Box(\mathrm{tr} = \mathit{Dispense\_beer} \rightarrow \bigcirc(\Box\mathrm{tr} \neq \mathit{Dispence\_beer} \vee \mathrm{tr} \neq \mathit{Dispence\_beer}\, \mathcal{U}\, \mathrm{tr} = \mathit{Insert\_coin}))$$

2. If we never have two recharge transitions in a row, then the next transition after a recharge must be a customer arrival.

$$\Box(\mathrm{tr} = \mathit{Recharge} \rightarrow \bigcirc\mathrm{tr} \neq \mathit{Recharge}) \rightarrow \Box(\mathrm{tr} = \mathit{Recharge} \rightarrow \bigcirc\mathrm{tr} = \mathit{Customer\_arrives})$$

3. The value of customer can only be changed as a result of either *Customer_arrives* or *Customer_leaves*.

$$\Box(\bigwedge_{v \in dom(\mathrm{customer})}(\mathrm{customer} = v \wedge \bigcirc\mathrm{customer} \neq v) \rightarrow \mathrm{tr} = \mathit{Customer\_arrives} \vee \mathrm{tr} = \mathit{Customer\_leaves})$$

Representing Temporal Properties of Transitions

1. One cannot have two beers without inserting a coin in between getting them.

$$\Box(\mathrm{tr} = \mathit{Dispense\_beer} \rightarrow \bigcirc(\Box \mathrm{tr} \neq \mathit{Dispence\_beer} \ \vee \ \mathrm{tr} \neq \mathit{Dispence\_beer} \, \mathcal{U} \, \mathrm{tr} = \mathit{Insert\_coin}))$$

2. If we never have two recharge transitions in a row, then the next transition after a recharge must be a customer arrival.

$$\Box(\mathrm{tr} = \mathit{Recharge} \rightarrow \bigcirc \mathrm{tr} \neq \mathit{Recharge}) \rightarrow \Box(\mathrm{tr} = \mathit{Recharge} \rightarrow \bigcirc \mathrm{tr} = \mathit{Customer\_arrives})$$

3. The value of customer can only be changed as a result of either *Customer_arrives* or *Customer_leaves*.

$$\Box(\bigwedge_{v \in dom(\mathrm{customer})}(\mathrm{customer} = v \wedge \bigcirc \mathrm{customer} \neq v) \rightarrow \mathrm{tr} = \mathit{Customer\_arrives} \vee \mathrm{tr} = \mathit{Customer\_leaves})$$

Representing Temporal Properties of Transitions

1. One cannot have two beers without inserting a coin in between getting them.

$$\square(\mathrm{tr} = \mathit{Dispense\_beer} \rightarrow \bigcirc(\square \mathrm{tr} \neq \mathit{Dispence\_beer} \ \vee \\ \mathrm{tr} \neq \mathit{Dispence\_beer}\ \mathcal{U}\ \mathrm{tr} = \mathit{Insert\_coin}))$$

2. If we never have two recharge transitions in a row, then the next transition after a recharge must be a customer arrival.

$$\square(\mathrm{tr} = \mathit{Recharge} \rightarrow \bigcirc \mathrm{tr} \neq \mathit{Recharge}) \rightarrow \\ \square(\mathrm{tr} = \mathit{Recharge} \rightarrow \bigcirc \mathrm{tr} = \mathit{Customer\_arrives})$$

3. The value of customer can only be changed as a result of either *Customer_arrives* or *Customer_leaves*.

$$\square(\bigwedge_{v \in dom(\mathrm{customer})}(\mathrm{customer} = v \wedge \bigcirc \mathrm{customer} \neq v) \rightarrow \\ \mathrm{tr} = \mathit{Customer\_arrives} \vee \mathrm{tr} = \mathit{Customer\_leaves})$$

Representing Temporal Properties of Transitions

1. One cannot have two beers without inserting a coin in between getting them.

$$\Box(\mathrm{tr} = \mathit{Dispense\_beer} \rightarrow \bigcirc(\Box \mathrm{tr} \neq \mathit{Dispence\_beer} \ \lor \\ \mathrm{tr} \neq \mathit{Dispence\_beer}\ \mathcal{U}\ \mathrm{tr} = \mathit{Insert\_coin}))$$

2. If we never have two recharge transitions in a row, then the next transition after a recharge must be a customer arrival.

$$\Box(\mathrm{tr} = \mathit{Recharge} \rightarrow \bigcirc \mathrm{tr} \neq \mathit{Recharge}) \rightarrow \\ \Box(\mathrm{tr} = \mathit{Recharge} \rightarrow \bigcirc \mathrm{tr} = \mathit{Customer\_arrives})$$

3. The value of customer can only be changed as a result of either *Customer_arrives* or *Customer_leaves*.

$$\Box(\bigwedge_{v \in dom(\mathrm{customer})}(\mathrm{customer} = v \land \bigcirc \mathrm{customer} \neq v) \rightarrow \\ \mathrm{tr} = \mathit{Customer\_arrives} \lor \mathrm{tr} = \mathit{Customer\_leaves})$$

Representing Temporal Properties of Transitions

1. One cannot have two beers without inserting a coin in between getting them.

$$\Box(\mathrm{tr} = \mathit{Dispense\_beer} \rightarrow \bigcirc(\Box \mathrm{tr} \neq \mathit{Dispence\_beer} \ \vee \\ \mathrm{tr} \neq \mathit{Dispence\_beer}\ \mathcal{U}\ \mathrm{tr} = \mathit{Insert\_coin}))$$

2. If we never have two recharge transitions in a row, then the next transition after a recharge must be a customer arrival.

$$\Box(\mathrm{tr} = \mathit{Recharge} \rightarrow \bigcirc \mathrm{tr} \neq \mathit{Recharge}) \rightarrow \\ \Box(\mathrm{tr} = \mathit{Recharge} \rightarrow \bigcirc \mathrm{tr} = \mathit{Customer\_arrives})$$

3. The value of customer can only be changed as a result of either *Customer_arrives* or *Customer_leaves*.

$$\Box(\bigwedge_{v \in dom(\mathrm{customer})}(\mathrm{customer} = v \wedge \bigcirc \mathrm{customer} \neq v) \rightarrow \\ \mathrm{tr} = \mathit{Customer\_arrives} \vee \mathrm{tr} = \mathit{Customer\_leaves})$$

Putting it All Together

When we design a system, we would like to be sure that it will satisfy all requirements, such as safety.

Now we can treat the safety problem as a mathematical problem. We can

- formally represent our system as a transition system (the symbolic representation);
- express the desired properties of the system in temporal logic.

What is missing?

Putting it All Together

When we design a system, we would like to be sure that it will satisfy all requirements, such as safety.

Now we can treat the safety problem as a mathematical problem. We can

- formally represent our system as a transition system (the symbolic representation);
- express the desired properties of the system in temporal logic.

What is missing?

Putting it All Together

When we design a system, we would like to be sure that it will satisfy all requirements, such as safety.

Now we can treat the safety problem as a mathematical problem. We can

- formally represent our system as a transition system (the symbolic representation);
- express the desired properties of the system in temporal logic.

What is missing?

The Model Checking Problem

Model Checking problem:

Given

1. a symbolic representation of a transition system;
2. a temporal formula F,

check if every (some) computation of the system satisfies this formula, preferably in a fully automatic way.

Reachability and Safety Properties

A reachability property is expressed by a formula

$$\Diamond F,$$

where F is a propositional formula.

A safety property is expressed by a formula

$$\Box F,$$

where F is a propositional formula.

Reachability and safety properties are the most common problems arising in model checking. They are dual to each other: if we can check one of them, we can check the other one too:

- $\Box F \equiv \neg \Diamond \neg F$;
- $\Diamond F \equiv \neg \Box \neg F$.

We cannot reach an unsafe state if and only if all states we can visit are safe.

Reachability and Safety Properties

A reachability property is expressed by a formula

$$\lozenge F,$$

where F is a propositional formula.

A safety property is expressed by a formula

$$\square F,$$

where F is a propositional formula.

Reachability and safety properties are the most common problems arising in model checking. They are dual to each other: if we can check one of them, we can check the other one too:

- $\square F \equiv \neg\lozenge\neg F$;
- $\lozenge F \equiv \neg\square\neg F$.

We cannot reach an unsafe state if and only if all states we can visit are safe.

Reachability and Safety Properties

A reachability property is expressed by a formula

$$\Diamond F,$$

where F is a propositional formula.

A safety property is expressed by a formula

$$\Box F,$$

where F is a propositional formula.

Reachability and safety properties are the most common problems arising in model checking. They are dual to each other: if we can check one of them, we can check the other one too:

- $\Box F \equiv \neg\Diamond\neg F$;
- $\Diamond F \equiv \neg\Box\neg F$.

We cannot reach an unsafe state if and only if all states we can visit are safe.

Reachability and Safety Properties

A reachability property is expressed by a formula

$$\Diamond F,$$

where F is a propositional formula.

A safety property is expressed by a formula

$$\Box F,$$

where F is a propositional formula.

Reachability and safety properties are the most common problems arising in model checking. They are dual to each other: if we can check one of them, we can check the other one too:

- $\Box F \equiv \neg \Diamond \neg F$;
- $\Diamond F \equiv \neg \Box \neg F$.

We cannot reach an unsafe state if and only if all states we can visit are safe.

Reachability

Fix a transition system $\mathbb{S}$ with the transition relation T. We write $s_0 \to s_1$ for $(s_0, s_1) \in T$ (that is, if there is a transition from s_0 to s_1).

- A state s is reachable in n steps from a state s_0 if there exists a sequence of states $s_1, \ldots, s_n$ such that $s_n = s$ and

$$s_0 \to s_1 \to \ldots \to s_n.$$

- A state s is reachable from a state s_0 if s is reachable from s_0 in $n \geq 0$ steps.

Reachability

Fix a transition system $\mathbb{S}$ with the transition relation T. We write $s_0 \to s_1$ for $(s_0, s_1) \in T$ (that is, if there is a transition from s_0 to s_1).

- A state s is reachable in n steps from a state s_0 if there exists a sequence of states $s_1, \ldots, s_n$ such that $s_n = s$ and

$$s_0 \to s_1 \to \ldots \to s_n.$$

- A state s is reachable from a state s_0 if s is reachable from s_0 in $n \geq 0$ steps.

Reachability

Fix a transition system $\mathbb{S}$ with the transition relation T. We write $s_0 \to s_1$ for $(s_0, s_1) \in T$ (that is, if there is a transition from s_0 to s_1).

- A state s is reachable in n steps from a state s_0 if there exists a sequence of states $s_1, \ldots, s_n$ such that $s_n = s$ and

$$s_0 \to s_1 \to \ldots \to s_n.$$

- A state s is reachable from a state s_0 if s is reachable from s_0 in $n \geq 0$ steps.

Reachability Properties and Graph Reachability

Theorem. Let F be a propositional formula. The formula $\Diamond F$ holds on some computation path if and only if there exists an initial state s_0 and a state s such that $s \models F$ and s is reachable from s_0.

Reformulation of Reachability

Given

1. Initial condition I representing a set of initial states;
2. Final condition F representing a set of final states;
3. formula Tr representing the transition relation of a transition system $\mathbb{S}$,

is any final state reachable from an initial state in $\mathbb{S}$?

Symbolic Reachability Checking

- Idea: build a symbolic representation of the set of reachable states.
- Two main kinds of algorithm:
  - forward reachability;
  - backward reachability.

Symbolic Reachability Checking

- Idea: build a symbolic representation of the set of reachable states.
- Two main kinds of algorithm:
  - forward reachability;
  - backward reachability.

Reformulation as a Decision Problem

Given

1. a formula $I(\bar{x})$, called the initial condition;
2. a formula $F(\bar{x})$, called the final condition;
3. formula $T(\bar{x}, \bar{x}')$, called the transition formula

does there exist a sequence of states $s_0, \ldots, s_n$ such that

1. $s_0 \models I(\bar{x})$;
2. $s_n \models F(\bar{x})$;
3. For all $i = 0, \ldots, n-1$ we have $(s_{i-1}, s_i) \models T(\bar{x}, \bar{x}')$.

Note that in this case s_n is reachable from s_0 in n steps.

Idea of Reachability-Checking Algorithms

If a final state is reachable from an initial state, then it is reachable from an initial state in some number n of steps.

For a given number n, find a symbolic representation of the set of states reachable from from an initial state in n steps. If this formula is not satisfied in a final state, increase n and start again.

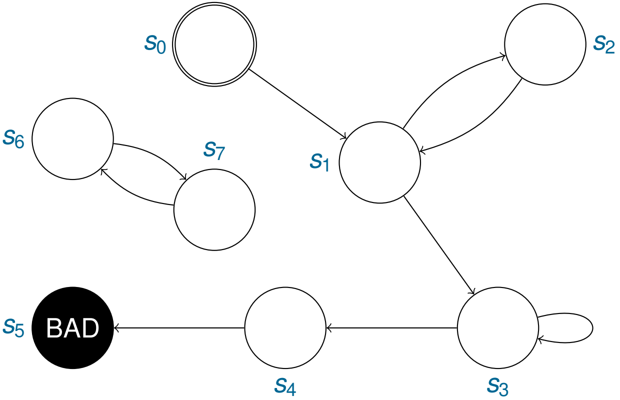
Idea of Reachability-Checking Algorithms

If a final state is reachable from an initial state, then it is reachable from an initial state in some number n of steps.

For a given number n, find a symbolic representation of the set of states reachable from from an initial state in n steps. If this formula is not satisfied in a final state, increase n and start again.

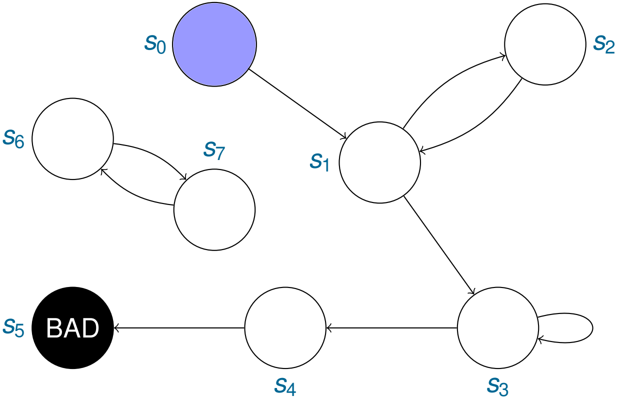
Reachability in n steps

Number of steps:



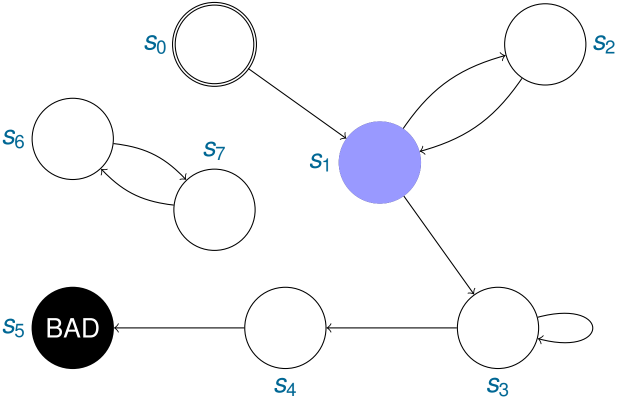
Reachability in n steps

Number of steps: 0



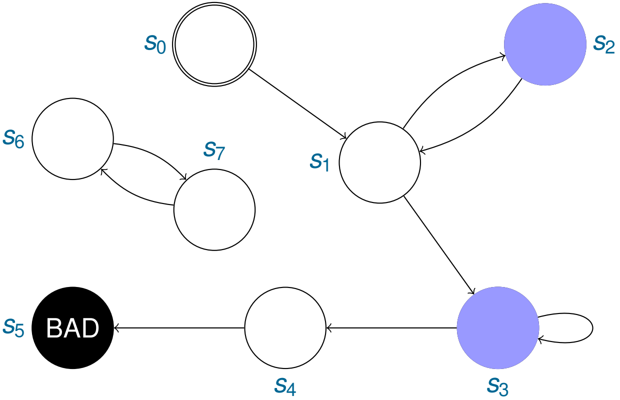
Reachability in n steps

Number of steps: 1



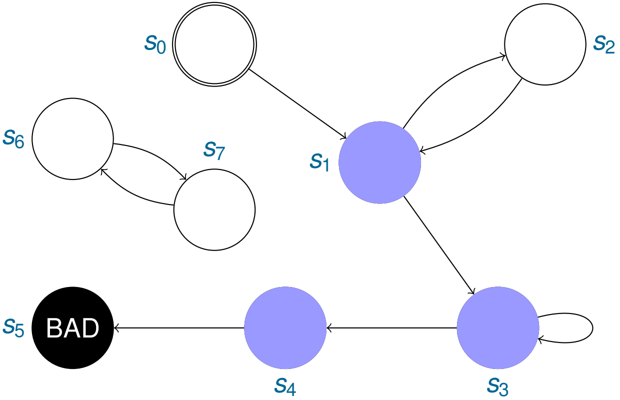
Reachability in n steps

Number of steps: 2



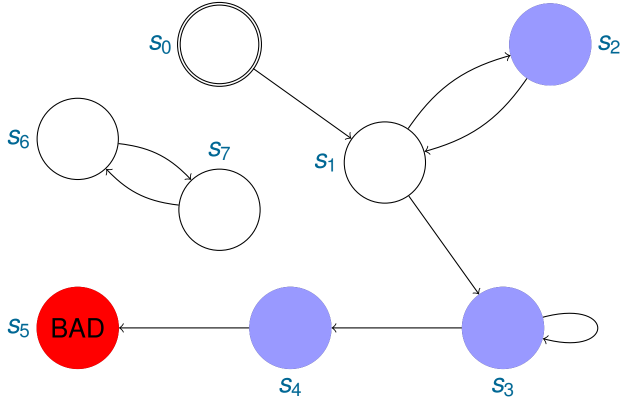
Reachability in n steps

Number of steps: 3



Reachability in n steps

Number of steps: 4



Simple Logical Analysis

Lemma

Let $C(\bar{x})$ symbolically represent a set of states S. Define

$$FR(\bar{x}) \stackrel{\text{def}}{=} \exists \bar{x}_1 (C(\bar{x}_1) \wedge T(\bar{x}_1, \bar{x})).$$

Then $FR(\bar{x})$ represents the set of states reachable from S in one step.

Define a sequence of formulas R_n for reachability in n states:

$$\begin{aligned} R_0(\bar{x}) &\stackrel{\text{def}}{=} I(\bar{x}) \\ &\cdots \\ R_n(\bar{x}) &\stackrel{\text{def}}{=} \exists \bar{x}_{n-1}(R_{n-1}(\bar{x}_{n-1}) \wedge T(\bar{x}_{n-1}, \bar{x})) \end{aligned}$$

Simple Logical Analysis

Lemma

Let $C(\bar{x})$ symbolically represent a set of states S. Define

$$FR(\bar{x}) \stackrel{\text{def}}{=} \exists \bar{x}_1 (C(\bar{x}_1) \land T(\bar{x}_1, \bar{x})).$$

Then $FR(\bar{x})$ represents the set of states reachable from S in one step.

Define a sequence of formulas R_n for reachability in n states:

$$\begin{array}{rcl} R_0(\bar{x}) & \stackrel{\text{def}}{=} & I(\bar{x}) \\ & \ldots & \\ R_n(\bar{x}) & \stackrel{\text{def}}{=} & \exists \bar{x}_{n-1} (R_{n-1}(\bar{x}_{n-1}) \land T(\bar{x}_{n-1}, \bar{x})) \end{array}$$

One-sided Forward Reachability Algorithm

procedure $FReach(I, T, F)$
input: formulas I, T, F
output: "yes" or no output
begin
  $i := 0$;
  $R := I(\bar{x}_0)$;
  loop
    if $R \wedge F(\bar{x}_i)$ is satisfiable **then** **return** "yes" ;
    $R := R \wedge T(\bar{x}_i, \bar{x}_{i+1})$;
    $i := i + 1$;
  end **loop**
end

Implementation? Use SAT solvers.
Bounded Model Checking (BMC) when we fix an upper bound on i.

One-sided Forward Reachability Algorithm

procedure $FReach(I, T, F)$
input: formulas I, T, F
output: "yes" or no output
begin
  $i := 0$;
  $R := I(\bar{x}_0)$;
  loop
    if $R \wedge F(\bar{x}_i)$ is satisfiable **then** **return** "yes" ;
    $R := R \wedge T(\bar{x}_i, \bar{x}_{i+1})$;
    $i := i + 1$;
  end **loop**
end

Lemma:

$$I(\bar{x}_0) \wedge T(\bar{x}_0, \bar{x}_1) \wedge T(\bar{x}_1, \bar{x}_2) \wedge \ldots \wedge T(\bar{x}_{i-1}, \bar{x}_i) \wedge F(\bar{x}_i)$$

is satisfiable if and only if there is a state s reachable in i steps, such that $s \models F$.

Implementation? Use SAT solvers.
Bounded Model Checking (BMC) when we fix an upper bound on i.

One-sided Forward Reachability Algorithm

procedure $FReach(I, T, F)$
input: formulas I, T, F
output: "yes" or no output
begin
  $i := 0$;
  $R := I(\bar{x}_0)$;
  loop
    if $R \wedge F(\bar{x}_i)$ is satisfiable **then** **return** "yes" ;
    $R := R \wedge T(\bar{x}_i, \bar{x}_{i+1})$;
    $i := i + 1$;
  end **loop**
end

Lemma:

$$I(\bar{x}_0) \wedge T(\bar{x}_0, \bar{x}_1) \wedge T(\bar{x}_1, \bar{x}_2) \wedge \ldots \wedge T(\bar{x}_{i-1}, \bar{x}_i) \wedge F(\bar{x}_i)$$

is satisfiable if and only if there is a state s reachable in i steps, such that $s \models F$.

Implementation? Use SAT solvers.
Bounded Model Checking (BMC) when we fix an upper bound on i.

One-sided Forward Reachability Algorithm

procedure $FReach(I, T, F)$
input: formulas I, T, F
output: "yes" or no output
begin
  $i := 0$;
  $R := I(\bar{x}_0)$;
  loop
    if $R \wedge F(\bar{x}_i)$ is satisfiable **then** **return** "yes" ;
    $R := R \wedge T(\bar{x}_i, \bar{x}_{i+1})$;
    $i := i + 1$;
  end **loop**
end

Lemma:

$$I(\bar{x}_0) \wedge T(\bar{x}_0, \bar{x}_1) \wedge T(\bar{x}_1, \bar{x}_2) \wedge \ldots \wedge T(\bar{x}_{i-1}, \bar{x}_i) \wedge F(\bar{x}_i)$$

is satisfiable if and only if there is a state s reachable in i steps, such that $s \models F$.

Implementation? Use SAT solvers.

Bounded Model Checking (BMC) when we fix an upper bound on i.

One-sided Forward Reachability Algorithm

```
procedure FReach(I, T, F)
input: formulas I, T, F
output: "yes" or no output
begin
  i   :=  0 ;
  R   :=  I(x̄_0) ;
  loop
    if R ∧ F(x̄_i) is satisfiable then return "yes" ;
    R   :=  R ∧ T(x̄_i, x̄_{i+1}) ;
    i   :=  i + 1 ;
  end loop
end
```

Lemma:

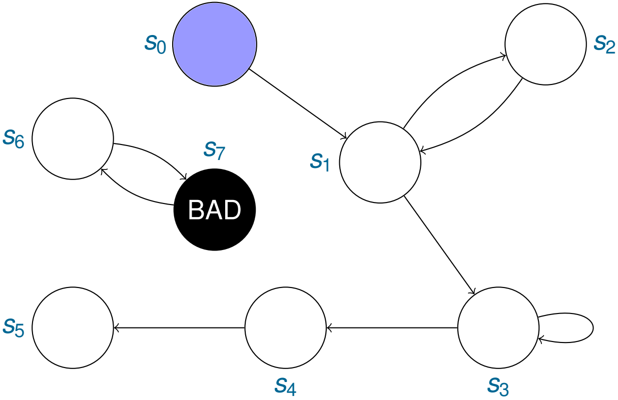
$$I(\bar{x}_0) \wedge T(\bar{x}_0, \bar{x}_1) \wedge T(\bar{x}_1, \bar{x}_2) \wedge \ldots \wedge T(\bar{x}_{i-1}, \bar{x}_i) \wedge F(\bar{x}_i)$$

is satisfiable if and only if there is a state s reachable in i steps, such that $s \models F$.

Implementation? Use SAT solvers.
Bounded Model Checking (BMC) when we fix an upper bound on i.

Termination?

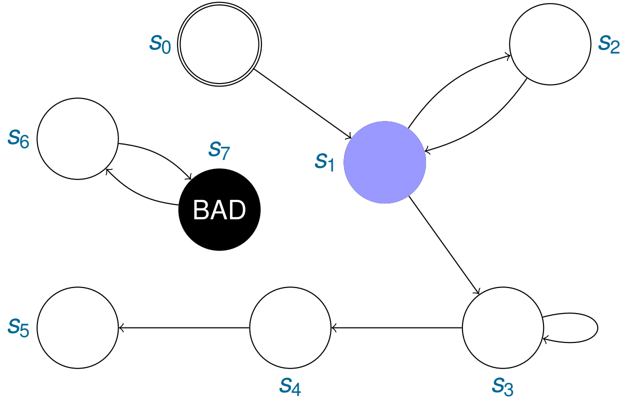
Number of steps: 0



When no final state is reachable, the algorithm does not terminate.

Termination?

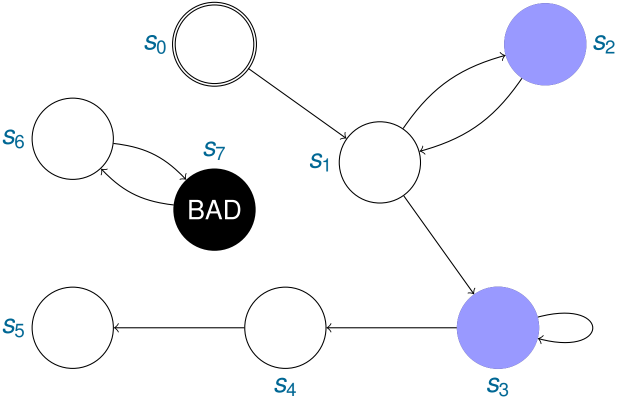
Number of steps: 1



When no final state is reachable, the algorithm does not terminate.

Termination?

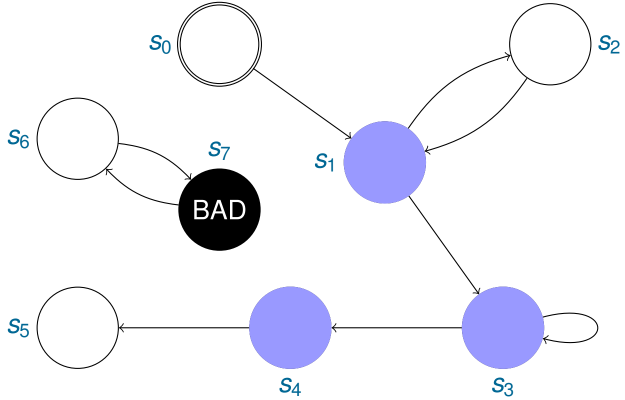
Number of steps: 2



When no final state is reachable, the algorithm does not terminate.

Termination?

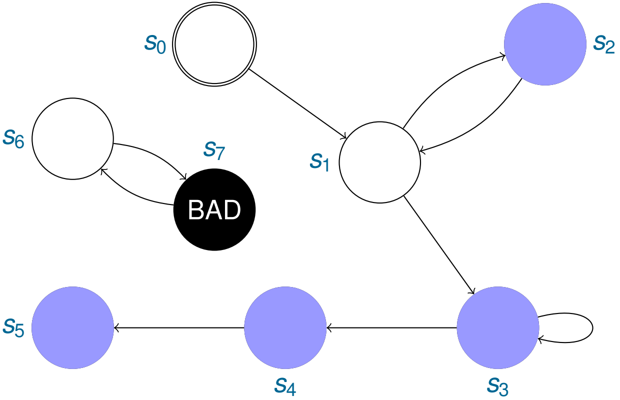
Number of steps: 3



When no final state is reachable, the algorithm does not terminate.

Termination?

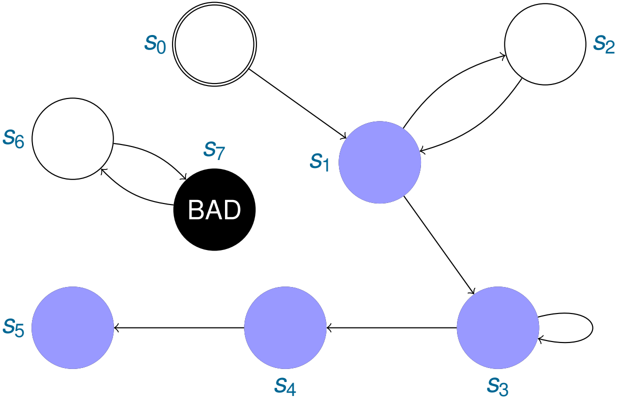
Number of steps: 4



When no final state is reachable, the algorithm does not terminate.

Termination?

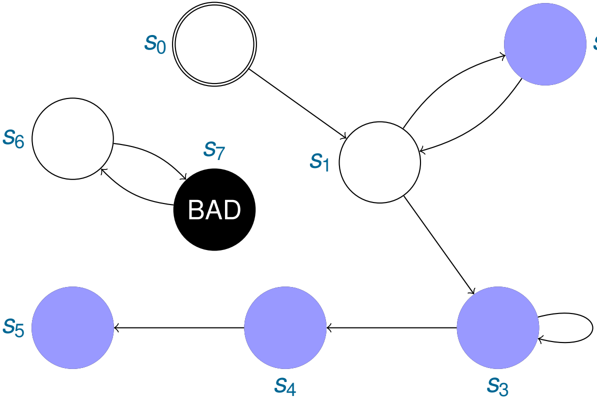
Number of steps: 5



When no final state is reachable, the algorithm does not terminate.

Termination?

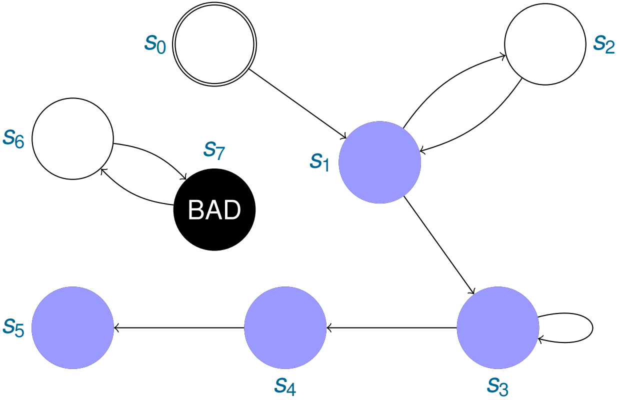
Number of steps: 6



When no final state is reachable, the algorithm does not terminate.

Termination?

Number of steps: 7



When no final state is reachable, the algorithm does not terminate.

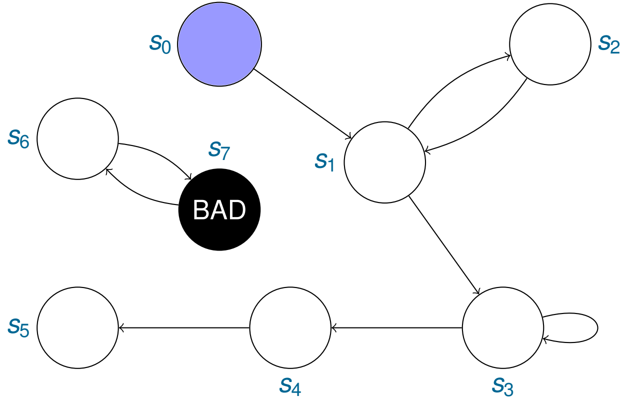
Reachability in $\leq n$ steps

Define a sequence of formulas $R_{\leq n}$ for reachability in $\leq n$ states:

$$
\begin{array}{rcl}
R_{\leq 0}(\bar{x}) & \stackrel{\text{def}}{=} & I(\bar{x}) \\
 & \cdots & \\
R_{\leq n}(\bar{x}) & \stackrel{\text{def}}{=} & R_{\leq n-1}(\bar{x}) \vee \exists \bar{x}_{n-1}(R_{\leq n-1}(\bar{x}_{n-1}) \wedge T(\bar{x}_{n-1}, \bar{x}))
\end{array}
$$

Reachability in $\leq n$ steps

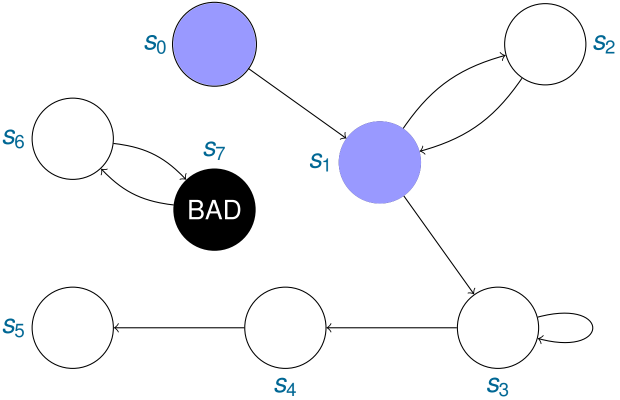
Number of steps: 0



The set of states will change no more.

Reachability in $\leq n$ steps

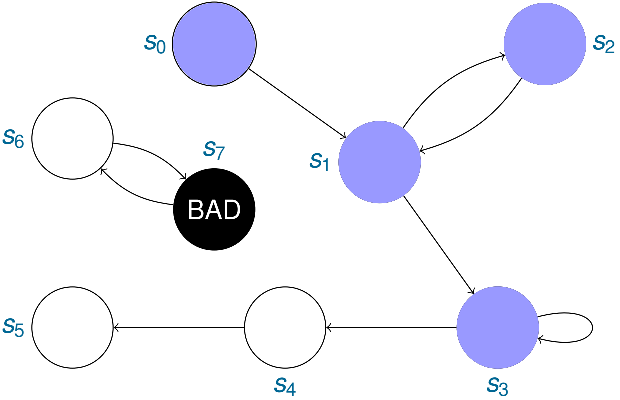
Number of steps: 1



The set of states will change no more.

Reachability in $\leq n$ steps

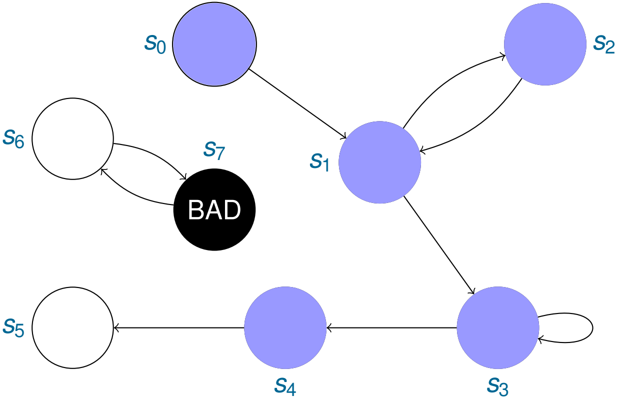
Number of steps: 2



The set of states will change no more.

Reachability in $\leq n$ steps

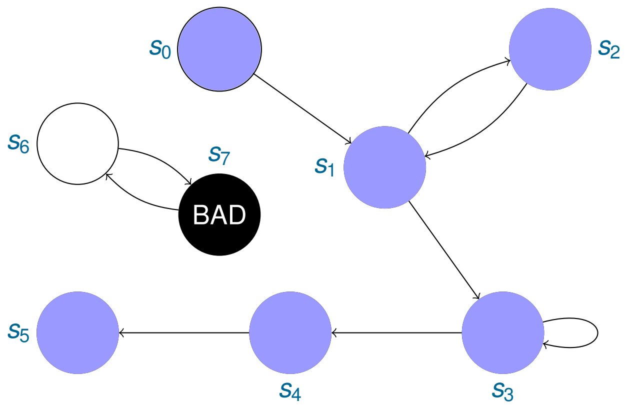
Number of steps: 3



The set of states will change no more.

Reachability in $\leq n$ steps

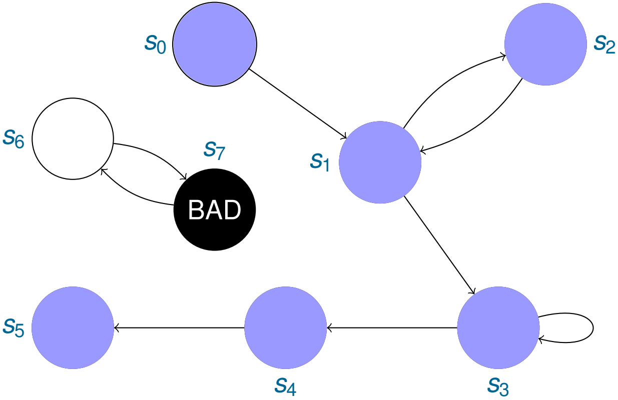
Number of steps: 4



The set of states will change no more.

Reachability in $\leq n$ steps

Number of steps: 5



The set of states will change no more.

Termination

Denote by S_n the set of states reachable from an initial state in $\leq n$ steps.
Key properties for termination.

- $S_i \subseteq S_{i+1}$ for all i;
- the system has a finite number of states;
- therefore, there exists a number k such that $S_k = S_{k+1}$;
- for such k we have $R_{\leq k}(\bar{x}) \equiv R_{\leq k+1}(\bar{x})$.

Complete Forward Reachability Algorithm

procedure $FReach(I, T, F)$
input: formulas I, T, F
output: "yes" or "no"
begin
  $i \; := \; 0$;
  $R_0(\bar{x}) \; := \; I(\bar{x})$;
  loop
    if $R_i(\bar{x}) \wedge F(\bar{x})$ is satisfiable **then** **return** "yes" ;

  end loop
end

Implementation?

Conjunction and disjunction
Quantification
Satisfiability checking
Equivalence checking

Complete Forward Reachability Algorithm

procedure *FReach*(I, T, F)
input: formulas I, T, F
output: “yes” or “no”
begin
  $i := 0$;
  $R_0(\bar{x}) := I(\bar{x})$;
  loop
    if $R_i(\bar{x}) \wedge F(\bar{x})$ is satisfiable **then** **return** “yes” ;
    $R_{i+1}(\bar{x}) := R_i(\bar{x}) \vee \exists \bar{x}_i (R_i(\bar{x}_i) \wedge T(\bar{x}_i, \bar{x}))$;

  end **loop**
end

Implementation?

Conjunction and disjunction
Quantification
Satisfiability checking
Equivalence checking

Complete Forward Reachability Algorithm

procedure $FReach(I, T, F)$
input: formulas I, T, F
output: “yes” or “no”
begin
  $i \;:=\; 0$;
  $R_0(\bar{x}) \;:=\; I(\bar{x})$;
  loop
    if $R_i(\bar{x}) \wedge F(\bar{x})$ is satisfiable **then** **return** “yes” ;
    $R_{i+1}(\bar{x}) \;:=\; R_i(\bar{x}) \vee \exists \bar{x}_i (R_i(\bar{x}_i) \wedge T(\bar{x}_i, \bar{x}))$;
    if $R_i(\bar{x}) \equiv R_{i+1}(\bar{x})$ **then** **return** “no” ;
  end **loop**
end

Implementation?

Conjunction and disjunction
Quantification
Satisfiability checking
Equivalence checking

Complete Forward Reachability Algorithm

procedure $FReach(I, T, F)$
input: formulas I, T, F
output: “yes” or “no”
begin
  $i \; := \; 0$;
  $R_0(\bar{x}) \; := \; I(\bar{x})$;
  loop
    if $R_i(\bar{x}) \wedge F(\bar{x})$ is satisfiable **then** **return** “yes” ;
    $R_{i+1}(\bar{x}) \; := \; R_i(\bar{x}) \vee \exists \bar{x}_i (R_i(\bar{x}_i) \wedge T(\bar{x}_i, \bar{x}))$;
    if $R_i(\bar{x}) \equiv R_{i+1}(\bar{x})$ **then** **return** “no” ;
    $i \; := \; i + 1$;
  end **loop**
end

Implementation?

Conjunction and disjunction
Quantification
Satisfiability checking
Equivalence checking

Complete Forward Reachability Algorithm

procedure $FReach(I, T, F)$
input: formulas I, T, F
output: "yes" or "no"
begin
  $i := 0$;
  $R_0(\bar{x}) := I(\bar{x})$;
  loop
    if $R_i(\bar{x}) \wedge F(\bar{x})$ is satisfiable **then** **return** "yes" ;
    $R_{i+1}(\bar{x}) := R_i(\bar{x}) \vee \exists \bar{x}_i (R_i(\bar{x}_i) \wedge T(\bar{x}_i, \bar{x}))$;
    if $R_i(\bar{x}) \equiv R_{i+1}(\bar{x})$ **then** **return** "no" ;
    $i := i + 1$;
  end **loop**
end

Implementation?

Conjunction and disjunction
Quantification
Satisfiability checking
Equivalence checking

Complete Forward Reachability Algorithm

```
procedure FReach(I, T, F)
input: formulas I, T, F
output: "yes" or "no"
begin
  i := 0 ;
  R_0(x̄) := I(x̄) ;
  loop
    if R_i(x̄) ∧ F(x̄) is satisfiable then return "yes" ;
     R_{i+1}(x̄) := R_i(x̄) ∨ ∃x̄_i(R_i(x̄_i) ∧ T(x̄_i, x̄)) ;
      if R_i(x̄) ≡ R_{i+1}(x̄) then return "no" ;
      i := i + 1 ;
   end loop
end
```

Implementation?

Conjunction and disjunction
Quantification
Satisfiability checking
Equivalence checking

Complete Forward Reachability Algorithm

procedure $FReach(I, T, F)$
input: formulas I, T, F
output: “yes” or “no”
begin
  $i := 0$;
  $R_0(\bar{x}) := I(\bar{x})$;
  loop
    if $R_i(\bar{x}) \wedge F(\bar{x})$ is satisfiable **then** **return** “yes” ;
    $R_{i+1}(\bar{x}) := R_i(\bar{x}) \vee \exists \bar{x}_i(R_i(\bar{x}_i) \wedge T(\bar{x}_i, \bar{x}))$;
    if $R_i(\bar{x}) \equiv R_{i+1}(\bar{x})$ **then** **return** “no” ;
    $i := i + 1$;
  end **loop**
end

Implementation?

Conjunction and disjunction
Quantification
Satisfiability checking
Equivalence checking

Complete Forward Reachability Algorithm

procedure $FReach(I, T, F)$
input: formulas I, T, F
output: "yes" or "no"
begin
  $i \ := \ 0$;
  $R_0(\bar{x}) \ := \ I(\bar{x})$;
  loop
    if $R_i(\bar{x}) \wedge F(\bar{x})$ is satisfiable **then** **return** "yes" ;
    $R_{i+1}(\bar{x}) \ := \ R_i(\bar{x}) \vee \exists \bar{x}_i(R_i(\bar{x}_i) \wedge T(\bar{x}_i, \bar{x}))$;
    if $R_i(\bar{x}) \equiv R_{i+1}(\bar{x})$ **then** **return** "no" ;
    $i \ := \ i + 1$;
  end **loop**
end

Implementation?

Conjunction and disjunction
Quantification
Satisfiability checking
Equivalence checking

Complete Forward Reachability Algorithm

procedure $FReach(I, T, F)$
input: formulas I, T, F
output: "yes" or "no"
begin
  $i := 0$;
  $R_0(\bar{x}) := I(\bar{x})$;
  loop
    if $R_i(\bar{x}) \wedge F(\bar{x})$ is satisfiable **then** **return** "yes" ;
    $R_{i+1}(\bar{x}) := R_i(\bar{x}) \vee \exists \bar{x}_i(R_i(\bar{x}_i) \wedge T(\bar{x}_i, \bar{x}))$;
    if $R_i(\bar{x}) \equiv R_{i+1}(\bar{x})$ **then** **return** "no" ;
    $i := i + 1$;
  end **loop**
end

Implementation?

Conjunction and disjunction
Quantification
Satisfiability checking
Equivalence checking

Complete Forward Reachability Algorithm

procedure $FReach(I, T, F)$
input: formulas I, T, F
output: "yes" or "no"
begin
  $i \ := \ 0$;
  $R_0(\bar{x}) \ := \ I(\bar{x})$;
  loop
    if $R_i(\bar{x}) \land F(\bar{x})$ is satisfiable **then** **return** "yes" ;
    $R_{i+1}(\bar{x}) \ := \ R_i(\bar{x}) \lor \exists \bar{x}_i (R_i(\bar{x}_i) \land T(\bar{x}_i, \bar{x}))$;
    if $R_i(\bar{x}) \equiv R_{i+1}(\bar{x})$ **then** **return** "no" ;
    $i \ := \ i + 1$;
  end **loop**
end

Implementation?
Use OBDDs and OBDD algorithms

Conjunction and disjunction
Quantification
Satisfiability checking
Equivalence checking

Main Problems with the Forward Reachability Algorithms

Forward reachability behave in the same way independently of the set of final states.

In other words, they are not goal oriented.

Backward Reachability in $\leq n$ steps

Idea:

- instead of going forward in the state transition graph, go backward;
- swap initial and final states and invert the transition relation.

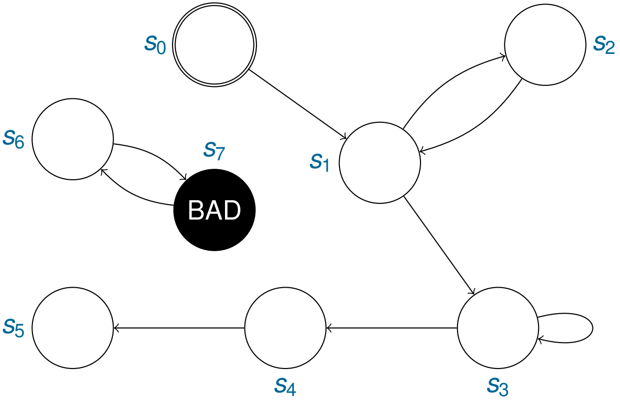
Number of backward steps:

Backward Reachability in $\leq n$ steps

Idea:

- instead of going forward in the state transition graph, go backward;
- swap initial and final states and invert the transition relation.

Number of backward steps: 0

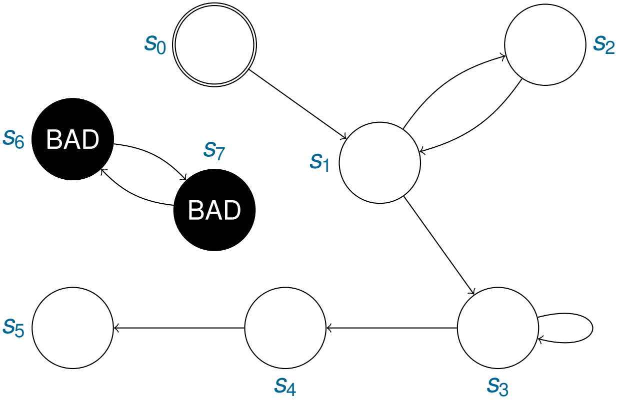


Backward Reachability in $\leq n$ steps

Idea:

- instead of going forward in the state transition graph, go backward;
- swap initial and final states and invert the transition relation.

Number of backward steps: 1

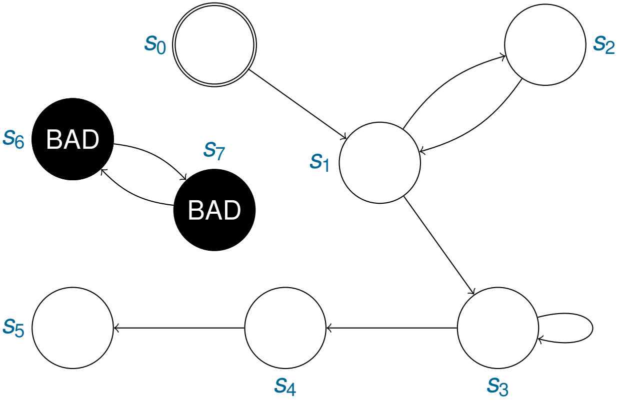


Backward Reachability in $\leq n$ steps

Idea:

- instead of going forward in the state transition graph, go backward;
- swap initial and final states and invert the transition relation.

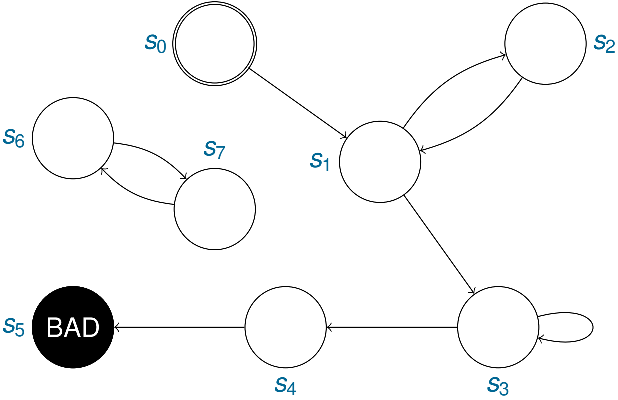
Number of backward steps: 1



Unreachable!

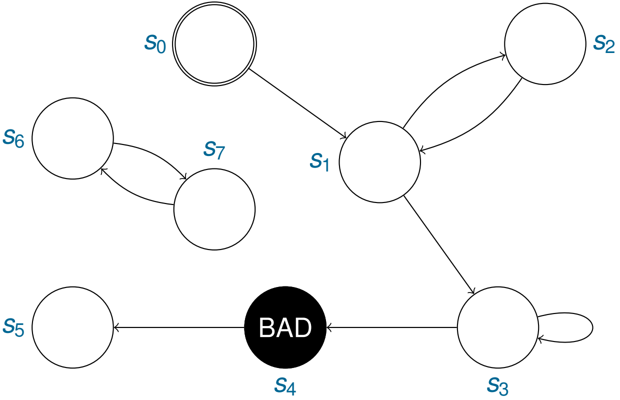
Backward Reachability in *n* steps

Number of backward steps: 0



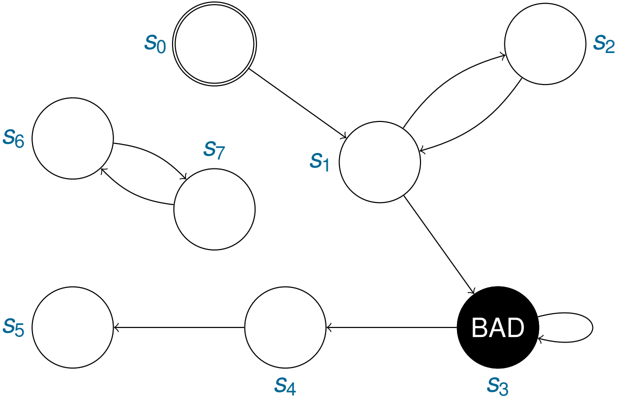
Backward Reachability in n steps

Number of backward steps: 1



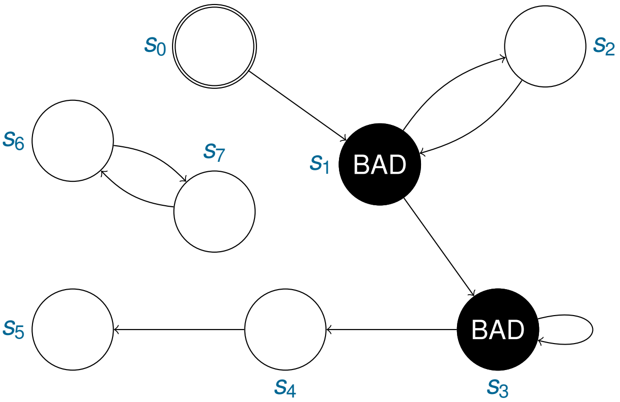
Backward Reachability in n steps

Number of backward steps: 2



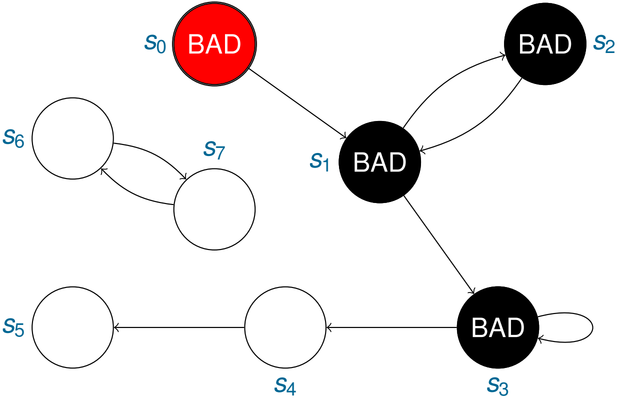
Backward Reachability in n steps

Number of backward steps: 3



Backward Reachability in n steps

Number of backward steps: 4



Reachable!

Backward Reachability

If S_n is reachable from S_0 in n steps, we say that S_0 is backward reachable from S_0 in n steps.

Backward Reachability

If S_n is reachable from S_0 in n steps, we say that S_0 is backward reachable from S_0 in n steps.

Lemma

Let $C(\bar{x})$ symbolically represent a set of states S. Define

$$BR(\bar{x}) \stackrel{\text{def}}{=} \exists \bar{x}_1 (C(\bar{x}_1) \land T(\bar{x}, \bar{x}_1)).$$

Then $BR(\bar{x})$ represents the set of states backward reachable from S in one step.

Complete Backward Reachability Algorithm

Same as the forward reachability algorithms, but

- Swap I with F;
- Use the inverse of the transition relation T.

procedure $BReach(I, T, F)$
input: formulas I, T, F
output: "yes" or "no"
begin
  $i := 0$;
  $R_0(\bar{x}) := F(\bar{x})$;
  loop
    if $R_i(\bar{x}) \wedge I(\bar{x})$ is satisfiable **then return** "yes" ;
    $R_{i+1}(\bar{x}) := R_i(\bar{x}) \vee \exists \bar{x}_i (R_i(\bar{x}_i) \wedge T(\bar{x}, \bar{x}_i))$;
    if $R_i(\bar{x}) \equiv R_{i+1}(\bar{x})$ **then return** "no" ;
    $i := i + 1$;
  end loop
end

Complete Backward Reachability Algorithm

Same as the forward reachability algorithms, but

- Swap I with F;
- Use the inverse of the transition relation T.

procedure $BReach(I, T, F)$
input: formulas I, T, F
output: "yes" or "no"
begin
  $i := 0$;
  $R_0(\bar{x}) := F(\bar{x})$;
  loop
    if $R_i(\bar{x}) \wedge I(\bar{x})$ is satisfiable **then return** "yes" ;
    $R_{i+1}(\bar{x}) := R_i(\bar{x}) \vee \exists \bar{x}_i (R_i(\bar{x}_i) \wedge T(\bar{x}, \bar{x}_i))$;
    if $R_i(\bar{x}) \equiv R_{i+1}(\bar{x})$ **then return** "no" ;
    $i := i + 1$;
  end loop
end

Other Properties

- There are general model-checking algorithm for arbitrary LTL properties.

Summary

- model checking
- safety properties as reachability
- symbolic reachability checking
- one-sided forward reachability (satisfiability algorithms)
- full forward/backward reachability (QBF/OBDD)

SAT competition results

Two winners: Congratulations!!!

Sivert Aasnaess (1st place)

Tomer Galor (2nd place)

SAT competition results

Random problems generated near the crossover point.

Simple problems vars 3-6 (100 problems):

	sat	sat avg. time	unsat	unsat avg. time	unknown	inconsist
sivert	34	0.10s	66	0.12s	0	0
tomer	30	0s	66	0s	0	0

SAT competition results

Random problems generated near the crossover point.

Simple problems vars 3-6 (100 problems):

	sat	sat avg. time	unsat	unsat avg. time	unknown	inconsist
sivert	34	0.10s	66	0.12s	0	0
tomer	30	0s	66	0s	0	0

Medium problems vars 10-30 (200 problems):

	sat	sat avg. time	unsat	unsat avg. time	unknown	inconsist
sivert	129	0.3s	81	0.4s	0	0
tomer	129	0.03s	81	0.0148	0	0

SAT competition results

Random problems generated near the crossover point.

Simple problems vars 3-6 (100 problems):

	sat	sat avg. time	unsat	unsat avg. time	unknown	inconsist
sivert	34	0.10s	66	0.12s	0	0
tomer	30	0s	66	0s	0	0

Medium problems vars 10-30 (200 problems):

	sat	sat avg. time	unsat	unsat avg. time	unknown	inconsist
sivert	129	0.3s	81	0.4s	0	0
tomer	129	0.03s	81	0.0148	0	0

Hard problems vars 50-180 (131 problems):

	sat	sat avg. time	unsat	unsat avg. time	unknown	inconsist
sivert	40	3.1s	34	3.4s	57	0
tomer	19	22s	21	17	91	0

SAT competition results

Random problems generated near the crossover point.

Simple problems vars 3-6 (100 problems):

	sat	sat avg. time	unsat	unsat avg. time	unknown	inconsist
sivert	34	0.10s	66	0.12s	0	0
tomer	30	0s	66	0s	0	0
minisat	34	0s	66	0s	0	0

Medium problems vars 10-30 (200 problems):

	sat	sat avg. time	unsat	unsat avg. time	unknown	inconsist
sivert	129	0.3s	81	0.4s	0	0
tomer	129	0.03s	81	0.0148	0	0

Hard problems vars 50-180 (131 problems):

	sat	sat avg. time	unsat	unsat avg. time	unknown	inconsist
sivert	40	3.1s	34	3.4s	57	0
tomer	19	22s	21	17	91	0

SAT competition results

Random problems generated near the crossover point.

Simple problems vars 3-6 (100 problems):

	sat	sat avg. time	unsat	unsat avg. time	unknown	inconsist
sivert	34	0.10s	66	0.12s	0	0
tomer	30	0s	66	0s	0	0
minisat	34	0s	66	0s	0	0

Medium problems vars 10-30 (200 problems):

	sat	sat avg. time	unsat	unsat avg. time	unknown	inconsist
sivert	129	0.3s	81	0.4s	0	0
tomer	129	0.03s	81	0.0148	0	0
minisat	129	0s	81	0s	0	0

Hard problems vars 50-180 (131 problems):

	sat	sat avg. time	unsat	unsat avg. time	unknown	inconsist
sivert	40	3.1s	34	3.4s	57	0
tomer	19	22s	21	17	91	0

SAT competition results

Random problems generated near the crossover point.

Simple problems vars 3-6 (100 problems):

	sat	sat avg. time	unsat	unsat avg. time	unknown	inconsist
sivert	34	0.10s	66	0.12s	0	0
tomer	30	0s	66	0s	0	0
minisat	34	0s	66	0s	0	0

Medium problems vars 10-30 (200 problems):

	sat	sat avg. time	unsat	unsat avg. time	unknown	inconsist
sivert	129	0.3s	81	0.4s	0	0
tomer	129	0.03s	81	0.0148	0	0
minisat	129	0s	81	0s	0	0

Hard problems vars 50-180 (131 problems):

	sat	sat avg. time	unsat	unsat avg. time	unknown	inconsist
sivert	40	3.1s	34	3.4s	57	0
tomer	19	22s	21	17	91	0
minisat	61	0s	70	0s	0	0

Short summary of the course (I)

Propositional Logic:

- satisfiability, validity, equivalence
- formalising problems
- splitting algorithm, polarity, pure atom
- CNF, CNF transformation
- clausal form, definitional clausal form transformation
- satisfiability of sets of clauses: DPLL, splitting+unit propagation, pure literal, tautology removal, Horn clauses.
- satisfiability of general formulas: semantic tableaux

Probabilistic analysis of satisfiability:

- random clause generation, transition function
- sharp transitions, easy-hard problems
- randomized algorithms for satisfiability:
  GSAT, WSAT, GSAT with Random Walks

Short summary of the course (II)

OBDDs: compact representation of Boolean functions

- BDT, OBDDs, building OBDDs, if-then-else normal form
- satisfiability, validity, equivalence checking for OBDDs
- alg. on OBDDs: disjunction, conjunc., quantification

QBF: Quantified Boolean Formulas

- syntax, semantics
- bound and free occurrences of variables
- rectification, prenex form, CNF transformation
- sat., validity can be reduced to evaluation of closed formulas
- evaluating QBF formulas:
  splitting, DPLL, pure literal, universal literal
- OBDD representation of QBF

Short summary of the course (III)

Propositional Logic of Finite Domains (PLFD):

- syntax, semantics
- translation of propositional logic into PLFD and back
- satisfiability checking: semantic tableaux (new rules)

Transition Systems:

- states, transitions
- symbolic representation of sets of states, transitions
- preconditions, postconditions, frame problem

Short summary of the course (IV)

Linear Temporal Logic LTL:

reasoning about temporal properties of transition systems

- syntax, semantics, temporal operators $\bigcirc$, $\Diamond$, $\Box$, $\mathcal{U}$, $\mathcal{R}$
- properties that can be expressed by LTL
- checking whether properties true/false on all/some paths of a transition system
- equivalence of LTL formulas, how to show non-equivalence

Model Checking:

- checking reachability and safety
- forward symbolic model checking of the reachability property
- one-sided forward reachability (using satisfiability algorithms)
- full forward/backward reachability (QBF/OBDD)