CS:4350 Logic in Computer Science

Model Checking

Cesare Tinelli

Spring 2022

The University of Iowa

Credits

These slides are largely based on slides originally developed by **Andrei Voronkov** at the University of Manchester. Adapted by permission.

Outline

Putting it All Together

When we design a computational system, we would like to be sure that it will satisfy all requirements, including safety requirements

Putting it All Together

When we design a computational system, we would like to be sure that it will satisfy all requirements, including safety requirements

Now we can treat the safety problem as a logical problem

Putting it All Together

When we design a computational system, we would like to be sure that it will satisfy all requirements, including safety requirements

Now we can treat the safety problem as a logical problem

We can

- formally represent our system as a transition system
- express desired properties of the system as temporal formulas

Putting it All Together

When we design a computational system, we would like to be sure that it will satisfy all requirements, including safety requirements

Now we can treat the safety problem as a logical problem

We can

- formally represent our system as a transition system
- express desired properties of the system as temporal formulas

What is missing?

The Model Checking Problem

Given

1. a symbolic representation $\mathbb{S}$ of a transition system
2. an LTL formula F

check if every (some) computation of $\mathbb{S}$ satisfies F,
preferably fully automatically

Notation:

$\mathrm{Comp}(\mathbb{S})$:	set of all computation paths of $\mathbb{S}$
$\mathbb{S} \models F$:	holds if $\pi \models F$ for all $\pi \in \mathrm{Comp}(\mathbb{S})$

The Model Checking Problem

Given

1. a symbolic representation $\mathbb{S}$ of a transition system
2. an LTL formula F

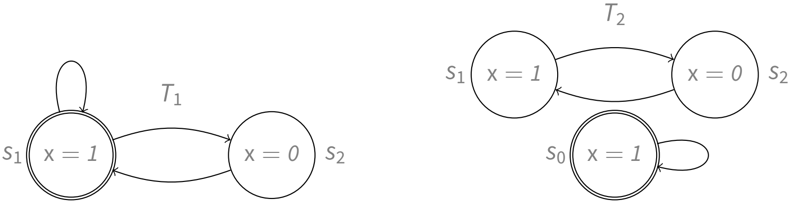
check if every (some) computation of $\mathbb{S}$ satisfies F,
preferably fully automatically

Notation:

$\mathrm{Comp}(\mathbb{S})$:	set of all computation paths of $\mathbb{S}$
$\mathbb{S} \models F$:	holds if $\pi \models F$ for all $\pi \in \mathrm{Comp}(\mathbb{S})$

Symbolic Representation and Transition Systems

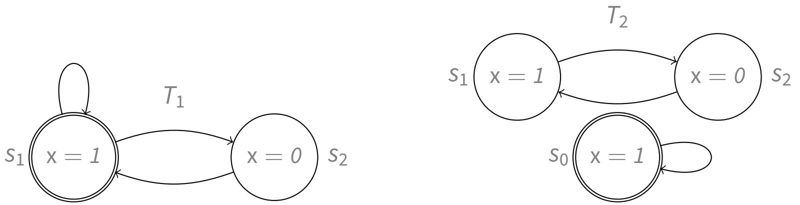
Consider the transition systems T_1 and T_2:



T_1 and T_2 have the same symbolic representation but satisfy different LTL formulas (e.g., $\Diamond\neg x$)

Symbolic Representation and Transition Systems

Consider the transition systems T_1 and T_2:

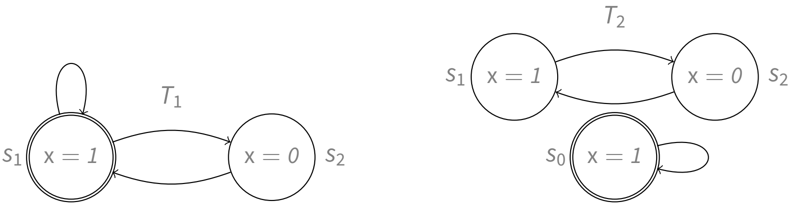


T_1 and T_2 have the same symbolic representation but satisfy different LTL formulas (e.g., $\Diamond\neg x$)

This happens only if one of the transition systems has two states with the same labelling function (e.g., s_0 and s_1 in T_2)

Symbolic Representation and Transition Systems

Consider the transition systems T_1 and T_2:



T_1 and T_2 have the same symbolic representation but satisfy different LTL formulas (e.g., $\Diamond \neg x$)

This happens only if one of the transition systems has two states with the same labelling function (e.g., s_0 and s_1 in T_2)

Such symbolic representations are *inadequate*: one cannot distinguish two different states by a state formula

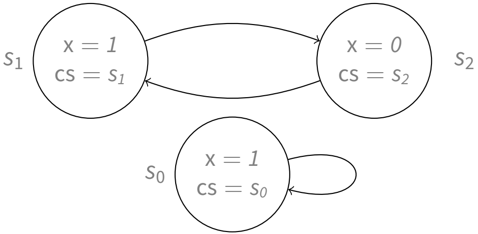
Making an Adequate Representation

If a transition system has different states labeled by the same interpretation, introduce a new state variable to distinguish such states

Making an Adequate Representation

If a transition system has different states labeled by the same interpretation, introduce a new state variable to distinguish such states

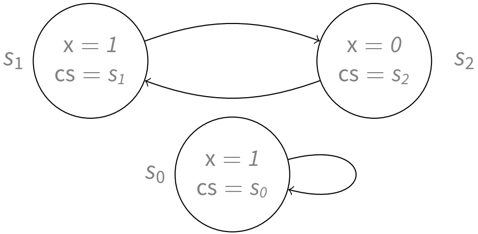
Example: One can add a *current state* variable cs with a unique value for each state



Making an Adequate Representation

If a transition system has different states labeled by the same interpretation, introduce a new state variable to distinguish such states

Example: One can add a *current state* variable cs with a unique value for each state



We will assume that different states always have different labelings

Reachability and Safety Properties

Reachability property: expressed by a formula for the form

$$\Diamond F$$

Reachability and Safety Properties

Reachability property: expressed by a formula for the form

$$\Diamond F$$

Safety/invariance property: expressed by a formula of the form

$$\Box F$$

In both cases, F is a PLFD formula

Reachability and Safety Properties

Reachability property: expressed by a formula for the form

$$\Diamond F$$

Safety/invariance property: expressed by a formula of the form

$$\Box F$$

In both cases, F is a PLFD formula

These are the most common problems arising in model checking

Reachability and Safety Properties

Reachability property: expressed by a formula for the form

$$\Diamond F$$

Safety/invariance property: expressed by a formula of the form

$$\Box F$$

In both cases, F is a PLFD formula

These are the most common problems arising in model checking

Terminology: With $\Diamond F$, usually F denotes a set of undesirable or *bad* states which a system should not reach

Reachability and Safety Properties

Reachability property: expressed by a formula for the form

$$\Diamond F$$

Safety/invariance property: expressed by a formula of the form

$$\Box F$$

In both cases, F is a PLFD formula

These are the most common problems arising in model checking

Note: $\mathbb{S} \not\models \Box F$ iff $\pi \models \Diamond\neg F$ for some $\pi \in \mathrm{Comp}(\mathbb{S})$

Reachability

Fix a transition system $\mathbb{S}$ with transition relation T over states S

We write $s_0 \to s_1$ if $(s_0, s_1) \in T$, (i.e., if there is a transition from state s_0 to state s_1)

Let $s \in S$

- s is *reachable in n steps from a state* $s_0 \in S$ if there exist states $s_1, \ldots, s_n \in S$ such that $s_n = s$ and $s_0 \to s_1 \to \cdots \to s_n$
- $s \in S$ is *reachable from a state* $s_0 \in S$ if s is reachable from s_0 in $n \geq 0$ steps
- $s \in S$ is *reachable in* $\mathbb{S}$ if s is reachable from some initial state of $\mathbb{S}$

Reachability

Fix a transition system $\mathbb{S}$ with transition relation T over states S

We write $s_0 \to s_1$ if $(s_0, s_1) \in T$, (i.e., if there is a transition from state s_0 to state s_1)

Let $s \in S$

- s is *reachable in n steps from a state $s_0 \in S$* if there exist states $s_1, \ldots, s_n \in S$ such that $s_n = s$ and $s_0 \to s_1 \to \cdots \to s_n$
- $s \in S$ is *reachable from a state $s_0 \in S$* if s is reachable from s_0 in $n \geq 0$ steps
- $s \in S$ is *reachable in $\mathbb{S}$* if s is reachable from some initial state of $\mathbb{S}$

Reachability

Fix a transition system $\mathbb{S}$ with transition relation T over states S

We write $s_0 \to s_1$ if $(s_0, s_1) \in T$, (i.e., if there is a transition from state s_0 to state s_1)

Let $s \in S$

- s is *reachable in n steps from a state $s_0 \in S$* if there exist states $s_1, \ldots, s_n \in S$ such that $s_n = s$ and $s_0 \to s_1 \to \cdots \to s_n$
- $s \in S$ is *reachable from a state $s_0 \in S$* if s is reachable from s_0 in $n \geq 0$ steps
- $s \in S$ is *reachable in $\mathbb{S}$* if s is reachable from some initial state of $\mathbb{S}$

Reachability

Fix a transition system $\mathbb{S}$ with transition relation T over states S

We write $s_0 \to s_1$ if $(s_0, s_1) \in T$, (i.e., if there is a transition from state s_0 to state s_1)

Let $s \in S$

- s is *reachable in n steps from a state* $s_0 \in S$ if there exist states $s_1, \ldots, s_n \in S$ such that $s_n = s$ and $s_0 \to s_1 \to \cdots \to s_n$
- $s \in S$ is *reachable from a state* $s_0 \in S$ if s is reachable from s_0 in $n \geq 0$ steps
- $s \in S$ is *reachable in* $\mathbb{S}$ if s is reachable from some initial state of $\mathbb{S}$

Reachability Properties and Graph Reachability

Theorem 1

A reachability property $\Diamond F$ holds on some computation path iff $s \models F$ for some reachable state s.

Reformulation of Reachability

$\mathbb{S}$ transition system with state variables $\boldsymbol{x} = x_1, \ldots, x_n$

Given

1. An *initial condition* $I(\boldsymbol{x})$, denoting the initial states of $\mathbb{S}$
2. A *transition formula* $T(\boldsymbol{x}, \boldsymbol{x}')$, denoting the transition relation of $\mathbb{S}$
3. A *final condition* $F(\boldsymbol{x})$, denoting a set of final states

is any final state reachable from an initial state?

Notation:

- $A(\boldsymbol{x})$ indicates that $\boldsymbol{x}$ are the free variables of A
- $A(\boldsymbol{x}, \boldsymbol{x}')$ indicates that $\boldsymbol{x}, \boldsymbol{x}'$ are the free variables of A with $\boldsymbol{x}' = x'_1, \ldots, x'_n$

Reformulation of Reachability

$\mathbb{S}$ transition system with state variables $\boldsymbol{x} = x_1, \ldots, x_n$

Given

1. An *initial condition* $I(\boldsymbol{x})$, denoting the initial states of $\mathbb{S}$
2. A *transition formula* $T(\boldsymbol{x}, \boldsymbol{x}')$, denoting the transition relation of $\mathbb{S}$
3. A *final condition* $F(\boldsymbol{x})$, denoting a set of final states

is any final state reachable from an initial state?

Note: this reformulation does not use temporal logic

Symbolic Reachability Checking

Main Idea: build a symbolic representation of the set of reachable states

Two main kinds of algorithm:

- forward reachability
- backward reachability

Symbolic Reachability Checking

Main Idea: build a symbolic representation of the set of reachable states

Two main kinds of algorithm:

- forward reachability
- backward reachability

An Efficient Encoding of PLFD in Propositional Logic

To reason about reachability it is convenient to use SAT solvers

This requires an encoding of PLFD to propositional logic

The encoding from the PLFD chapter does not scale well for large finite domains

An exponentially more compact encoding represents domains as sets of binary numbers

- Then, for a variable x with a domain of size 2^n for $n > 1$,
- n boolean variables are enough to represent x, instead of 2^n

An Efficient Encoding of PLFD in Propositional Logic

To reason about reachability it is convenient to use SAT solvers

This requires an encoding of PLFD to propositional logic

The encoding from the PLFD chapter does not scale well for large finite domains

An exponentially more compact encoding represents domains as sets of binary numbers

> Then, for a variable x with a domain of size 2^n for $n > 1$,
>
> n boolean variables are enough to represent x, instead of 2^n

An Efficient Encoding of PLFD in Propositional Logic

To reason about reachability it is convenient to use SAT solvers

This requires an encoding of PLFD to propositional logic

The encoding from the PLFD chapter does not scale well for large finite domains

An exponentially more compact encoding represents domains as sets of binary numbers

- Then, for a variable x with a domain of size 2^n for $n > 1$,
- n boolean variables are enough to represent x, instead of 2^n

An Efficient Encoding of PLFD in Propositional Logic

To reason about reachability it is convenient to use SAT solvers

This requires an encoding of PLFD to propositional logic

The encoding from the PLFD chapter does not scale well for large finite domains

An exponentially more compact encoding represents domains as sets of binary numbers

Then, for a variable x with a domain of size 2^n for $n > 1$,

n boolean variables are enough to represent x, instead of 2^n

An Efficient Encoding of PLFD in Propositional Logic

To reason about reachability it is convenient to use SAT solvers

This requires an encoding of PLFD to propositional logic

The encoding from the PLFD chapter does not scale well for large finite domains

An exponentially more compact encoding represents domains as sets of binary numbers

- Then, for a variable x with a domain of size 2^n for $n > 1$, n boolean variables are enough to represent x, instead of 2^n

An Efficient Encoding of PLFD in Propositional Logic

Suppose $|dom(x)| = 8$

Let $v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7$ be an arbitrary enumeration of $dom(x)$

Assign to each v_i the number i in binary:

$$\begin{aligned} b_x \quad = \quad \{ & v_0 \mapsto 000, v_1 \mapsto 001, v_2 \mapsto 010, v_3 \mapsto 011, \\ & v_4 \mapsto 100, v_5 \mapsto 101, v_6 \mapsto 110, v_7 \mapsto 111 \} \end{aligned}$$

If b is a binary number, let $b[k]$ denote its k-th least significant bit (e.g., $001[2] = 0$, $001[1] = 0$, $001[0] = 1$)

Encode atoms of the form $x = v_i$ as

$$x_2 = b_x(v_i)[2] \land x_1 = b_x(v_i)[1] \land x_0 = b_x(v_i)[0]$$

where x_2, x_1, x_0 are boolean variables for x

An Efficient Encoding of PLFD in Propositional Logic

Suppose $|dom(x)| = 8$

Let $v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7$ be an arbitrary enumeration of $dom(x)$

Assign to each v_i the number i in binary:

$$\begin{aligned} b_x \quad = \quad \{\, & v_0 \mapsto 000,\ v_1 \mapsto 001,\ v_2 \mapsto 010,\ v_3 \mapsto 011, \\ & v_4 \mapsto 100,\ v_5 \mapsto 101,\ v_6 \mapsto 110,\ v_7 \mapsto 111 \,\} \end{aligned}$$

If b is a binary number, let $b[k]$ denote its k-th least significant bit (e.g., $001[2] = 0$, $001[1] = 0$, $001[0] = 1$)

Encode atoms of the form $x = v_i$ as

$$x_2 = b_x(v_i)[2] \ \wedge\ x_1 = b_x(v_i)[1] \ \wedge\ x_0 = b_x(v_i)[0]$$

where x_2, x_1, x_0 are boolean variables for x

An Efficient Encoding of PLFD in Propositional Logic

Suppose $|dom(x)| = 8$

Let $v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7$ be an arbitrary enumeration of $dom(x)$

Assign to each v_i the number i in binary:

$$\begin{aligned} b_x \quad = \quad \{ & v_0 \mapsto 000,\ v_1 \mapsto 001,\ v_2 \mapsto 010,\ v_3 \mapsto 011, \\ & v_4 \mapsto 100,\ v_5 \mapsto 101,\ v_6 \mapsto 110,\ v_7 \mapsto 111 \} \end{aligned}$$

If b is a binary number, let $b[k]$ denote its k-th least significant bit
(e.g., $001[2] = 0$, $001[1] = 0$, $001[0] = 1$)

Encode atoms of the form $x = v_i$ as

$$x_2 = b_x(v_i)[2] \ \wedge \ x_1 = b_x(v_i)[1] \ \wedge \ x_0 = b_x(v_i)[0]$$

where x_2, x_1, x_0 are boolean variables for x

An Efficient Encoding of PLFD in Propositional Logic

Suppose $|dom(x)| = 8$

Let $v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7$ be an arbitrary enumeration of $dom(x)$

Assign to each v_i the number i in binary:

$$\begin{aligned} b_x \quad = \quad \{ & v_0 \mapsto 000, v_1 \mapsto 001, v_2 \mapsto 010, v_3 \mapsto 011, \\ & v_4 \mapsto 100, v_5 \mapsto 101, v_6 \mapsto 110, v_7 \mapsto 111 \} \end{aligned}$$

If b is a binary number, let $b[k]$ denote its k-th least significant bit
(e.g., $001[2] = 0$, $001[1] = 0$, $001[0] = 1$)

Encode atoms of the form $x = v_i$ as

$$x_2 = b_x(v_i)[2] \wedge x_1 = b_x(v_i)[1] \wedge x_0 = b_x(v_i)[0]$$

where x_2, x_1, x_0 are boolean variables for x

Binary Encoding Example

$$\begin{aligned} dom(\text{temp}) &= \{\mathit{0}, \mathit{150}, \mathit{160}, \mathit{170}, \mathit{180}, \mathit{190}, \mathit{200}, \mathit{210}\} \\ dom(\text{cont}) &= \{\mathit{none}, \mathit{burger}, \mathit{pizza}, \mathit{soup}\} \end{aligned}$$

$$\begin{aligned} b_{\text{temp}} &= \{\mathit{0} \mapsto 000, \mathit{150} \mapsto 001, \mathit{160} \mapsto 010, \mathit{170} \mapsto 011 \\ &\quad \mathit{180} \mapsto 100, \mathit{190} \mapsto 101, \mathit{200} \mapsto 110, \mathit{210} \mapsto 111\} \\ b_{\text{cont}} &= \{\mathit{none} \mapsto 00, \mathit{pizza} \mapsto 01, \mathit{burger} \mapsto 10, \mathit{soup} \mapsto 11\} \end{aligned}$$

Binary Encoding Example

$$
\begin{aligned}
b_{\text{temp}} &= \{\, \mathit{0} \mapsto 000,\ \mathit{150} \mapsto 001,\ \mathit{160} \mapsto 010,\ \mathit{170} \mapsto 011 \\
&\quad\ \mathit{180} \mapsto 100,\ \mathit{190} \mapsto 101,\ \mathit{200} \mapsto 110,\ \mathit{210} \mapsto 111 \,\} \\
b_{\text{cont}} &= \{\, \mathit{none} \mapsto 00,\ \mathit{pizza} \mapsto 01,\ \mathit{burger} \mapsto 10,\ \mathit{soup} \mapsto 11 \}
\end{aligned}
$$

The PLFD formula

$$\text{cont} = \mathit{pizza} \rightarrow \text{temp} \neq \mathit{200}$$

Binary Encoding Example

$$b_{\text{temp}} = \{\, \mathit{0} \mapsto 000,\ \mathit{150} \mapsto 001,\ \mathit{160} \mapsto 010,\ \mathit{170} \mapsto 011$$
$$\mathit{180} \mapsto 100,\ \mathit{190} \mapsto 101,\ \mathit{200} \mapsto 110,\ \mathit{210} \mapsto 111 \,\}$$
$$b_{\text{cont}} = \{\, \mathit{none} \mapsto 00,\ \mathit{pizza} \mapsto 01,\ \mathit{burger} \mapsto 10,\ \mathit{soup} \mapsto 11 \}$$

The PLFD formula

$$\text{cont} = \mathit{pizza} \rightarrow \text{temp} \neq \mathit{200}$$

is encoded as

$$(\text{cont}_1 = 0 \wedge \text{cont}_0 = 1) \rightarrow \neg(\text{temp}_2 = 1 \wedge \text{temp}_1 = 1 \wedge \text{temp}_0 = 0)$$

(with cont_1, cont_0, temp_2, temp_1, temp_0 boolean)

Binary Encoding Example

$$
\begin{aligned}
b_{\text{temp}} &= \{\, \mathit{0} \mapsto 000,\ \mathit{150} \mapsto 001,\ \mathit{160} \mapsto 010,\ \mathit{170} \mapsto 011 \\
&\quad\ \mathit{180} \mapsto 100,\ \mathit{190} \mapsto 101,\ \mathit{200} \mapsto 110,\ \mathit{210} \mapsto 111 \,\} \\
b_{\text{cont}} &= \{\, \mathit{none} \mapsto 00,\ \mathit{pizza} \mapsto 01,\ \mathit{burger} \mapsto 10,\ \mathit{soup} \mapsto 11 \}
\end{aligned}
$$

The PLFD formula

$$\text{cont} = \mathit{pizza} \to \text{temp} \neq \mathit{200}$$

is encoded as

$$(\text{cont}_1 = 0 \wedge \text{cont}_0 = 1) \to \neg(\text{temp}_2 = 1 \wedge \text{temp}_1 = 1 \wedge \text{temp}_0 = 0)$$

(with cont_1, cont_0, temp_2, temp_1, temp_0 boolean)

or, more compactly, as

$$(\neg\text{cont}_1 \wedge \text{cont}_0) \to \neg(\text{temp}_2 \wedge \text{temp}_1 \wedge \neg\text{temp}_0)$$

Binary Encoding

The translation is similar for every domain of cardinality 2^n for some $n > 1$

What if the cardinality of a domain $dom(x)$ is not a power of 2?

1. Let n be the smallest n such that $|dom(x)| < 2^n$
2. Encode as before but add constraint on x_i's to discard spurious values

Example

$\lvert dom(x) \rvert$	**Constraint**	**Discarded values**
7	$x_2 \wedge x_1 \to \neg x_0$	111
6	$x_2 \to \neg x_1$	110, 111
5	$x_2 \to \neg(x_1 \vee x_0)$	101, 110, 111
4	use only x_1, x_0 for x	none
3	$x_1 \to \neg x_0$	11
2	use only x_0 for x	none

Binary Encoding

The translation is similar for every domain of cardinality 2^n for some $n > 1$

What if the cardinality of a domain $dom(x)$ is not a power of 2?

1. Let n be the smallest n such that $|dom(x)| < 2^n$
2. Encode as before but add constraint on x_i's to discard spurious values

Example

$\lvert dom(x)\rvert$	**Constraint**	**Discarded values**
7	$x_2 \wedge x_1 \rightarrow \neg x_0$	111
6	$x_2 \rightarrow \neg x_1$	110, 111
5	$x_2 \rightarrow \neg(x_1 \vee x_0)$	101, 110, 111
4	use only x_1, x_0 for x	none
3	$x_1 \rightarrow \neg x_0$	11
2	use only x_0 for x	none

Binary Encoding

The translation is similar for every domain of cardinality 2^n for some $n > 1$

What if the cardinality of a domain $dom(x)$ is not a power of 2?

1. Let n be the smallest n such that $|dom(x)| < 2^n$
2. Encode as before but add constraint on x_i's to discard spurious values

Example

$\lvert dom(x) \rvert$	**Constraint**	**Discarded values**
7	$x_2 \land x_1 \to \neg x_0$	111
6	$x_2 \to \neg x_1$	110, 111
5	$x_2 \to \neg(x_1 \lor x_0)$	101, 110, 111
4	use only x_1, x_0 for x	none
3	$x_1 \to \neg x_0$	11
2	use only x_0 for x	none

Binary Encoding

The translation is similar for every domain of cardinality 2^n for some $n > 1$

What if the cardinality of a domain $dom(x)$ is not a power of 2?

1. Let n be the smallest n such that $|dom(x)| < 2^n$
2. Encode as before but add constraint on x_i's to discard spurious values

Example

$\|dom(x)\|$	**Constraint**	**Discarded values**
7	$x_2 \wedge x_1 \rightarrow \neg x_0$	111
6	$x_2 \rightarrow \neg x_1$	110, 111
5	$x_2 \rightarrow \neg(x_1 \vee x_0)$	101, 110, 111
4	use only x_1, x_0 for x	none
3	$x_1 \rightarrow \neg x_0$	11
2	use only x_0 for x	none

Binary Encoding of Transition System States

Consider states described by state variables x, y, z

A state is then just a value from domain

$$S = dom(x) \times dom(y) \times dom(z)$$

1. If $|S| \leq 2^n$, encode D in binary as described before
2. Use boolean variables $x_0, \ldots, x_{n-1}$ to represent a state $s \in S$

Binary Encoding of Transition System States

Consider states described by state variables x, y, z

A state is then just a value from domain

$$S = dom(x) \times dom(y) \times dom(z)$$

1. If $|S| \leq 2^n$, encode D in binary as described before
2. Use boolean variables $x_0, \ldots, x_{n-1}$ to represent a state $s \in S$

Binary Encoding of Transition System States

Consider states described by state variables x, y, z

A state is then just a value from domain

$$S = dom(x) \times dom(y) \times dom(z)$$

1. If $|S| \leq 2^n$, encode D in binary as described before
2. Use boolean variables $x_0, \ldots, x_{n-1}$ to represent a state $s \in S$

We will consider only boolean state variables from now on

Reachability as a Decision Problem

$\boldsymbol{x} = x_1, \ldots, x_n$ with each x_i a boolean state variable

Given

1. a formula $I(\boldsymbol{x})$, the *initial condition*
2. a formula $T(\boldsymbol{x}, \boldsymbol{x}')$, the *transition formula*
3. a formula $F(\boldsymbol{x})$, the *final/reachability condition*

is there a sequence of states $s_0, \ldots, s_k$ such that

1. $s_0 \models I(\boldsymbol{x})$
2. $(s_{i-1}, s_i) \models T(\boldsymbol{x}, \boldsymbol{x}')$ for all $i = 0, \ldots, k-1$
3. $s_k \models F(\boldsymbol{x})$

Reachability as a Decision Problem

$\boldsymbol{x} = x_1, \ldots, x_n$ with each x_i a boolean state variable

Given

1. a formula $I(\boldsymbol{x})$, the *initial condition*
2. a formula $T(\boldsymbol{x}, \boldsymbol{x}')$, the *transition formula*
3. a formula $F(\boldsymbol{x})$, the *final/reachability condition*

is there a sequence of states $s_0, \ldots, s_k$ such that

1. $s_0 \models I(\boldsymbol{x})$
2. $(s_{i-1}, s_i) \models T(\boldsymbol{x}, \boldsymbol{x}')$ for all $i = 0, \ldots, k-1$
3. $s_k \models F(\boldsymbol{x})$

Reachability as a Decision Problem

$\boldsymbol{x} = x_1, \ldots, x_n$ with each x_i a boolean state variable

Given

1. a formula $I(\boldsymbol{x})$, the *initial condition*
2. a formula $T(\boldsymbol{x}, \boldsymbol{x}')$, the *transition formula*
3. a formula $F(\boldsymbol{x})$, the *final/reachability condition*

is there a sequence of states $s_0, \ldots, s_k$ such that

1. $s_0 \models I(\boldsymbol{x})$
2. $(s_{i-1}, s_i) \models T(\boldsymbol{x}, \boldsymbol{x}')$ for all $i = 0, \ldots, k-1$
3. $s_k \models F(\boldsymbol{x})$

Equivalently, is the following formula satisfiable for some $k \geq 0$?

$$I(\boldsymbol{x}_0) \wedge T(\boldsymbol{x}_0, \boldsymbol{x}_1) \wedge \cdots \wedge T(\boldsymbol{x}_{k-1}, \boldsymbol{x}_k) \wedge F(\boldsymbol{x}_k)$$

Reachability as a Decision Problem

$\boldsymbol{x} = x_1, \ldots, x_n$ with each x_i a boolean state variable

Given

1. a formula $I(\boldsymbol{x})$, the *initial condition*
2. a formula $T(\boldsymbol{x}, \boldsymbol{x}')$, the *transition formula*
3. a formula $F(\boldsymbol{x})$, the *final/reachability condition*

is there a sequence of states $s_0, \ldots, s_k$ such that

1. $s_0 \models I(\boldsymbol{x})$
2. $(s_{i-1}, s_i) \models T(\boldsymbol{x}, \boldsymbol{x}')$ for all $i = 0, \ldots, k-1$
3. $s_k \models F(\boldsymbol{x})$

Note: When that in this case, s_k is reachable from s_0 in k steps

Idea of Reachability-Checking Algorithms

Observation: If a final state is reachable from an initial state s_0, it is reachable from s_0 in some finite number k of steps

Approach:

- Starting with $k = 0$, construct a formula $R_k(\boldsymbol{x})$ denoting the set of states reachable in k steps
- If $R_k(\boldsymbol{x})$ is not satisfied by a final state, increase k and start again

Idea of Reachability-Checking Algorithms

Observation: If a final state is reachable from an initial state s_0, it is reachable from s_0 in some finite number k of steps

Approach:

- Starting with $k = 0$, construct a formula $R_k(\boldsymbol{x})$ denoting the set of states reachable in k steps
- If $R_k(\boldsymbol{x})$ is not satisfied by a final state, increase k and start again

Idea of Reachability-Checking Algorithms

Observation: If a final state is reachable from an initial state s_0, it is reachable from s_0 in some finite number k of steps

Approach:

- Starting with $k = 0$, construct a formula $R_k(\boldsymbol{x})$ denoting the set of states reachable in k steps
- If $R_k(\boldsymbol{x})$ is not satisfied by a final state, increase k and start again

Idea of Reachability-Checking Algorithms

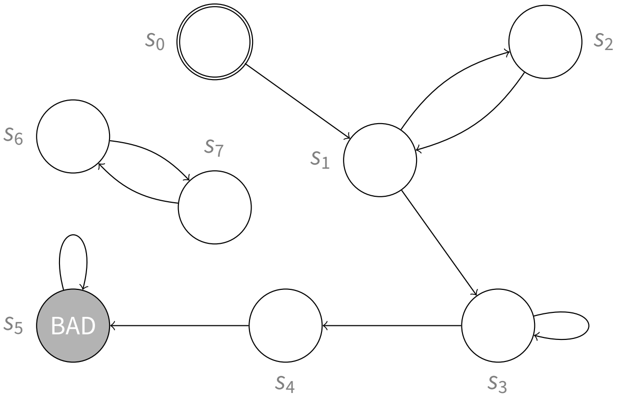
Observation: If a final state is reachable from an initial state s_0, it is reachable from s_0 in some finite number k of steps

Approach:

- Starting with $k = 0$, construct a formula $R_k(\boldsymbol{x})$ denoting the set of states reachable in k steps
- If $R_k(\boldsymbol{x})$ is not satisfied by a final state, increase k and start again

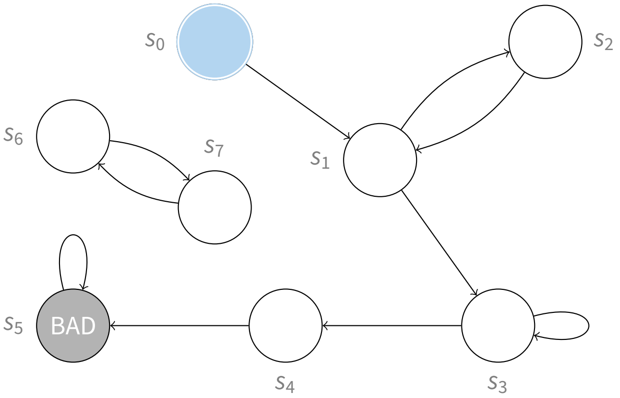
When does this process terminate?

Reachability in n steps



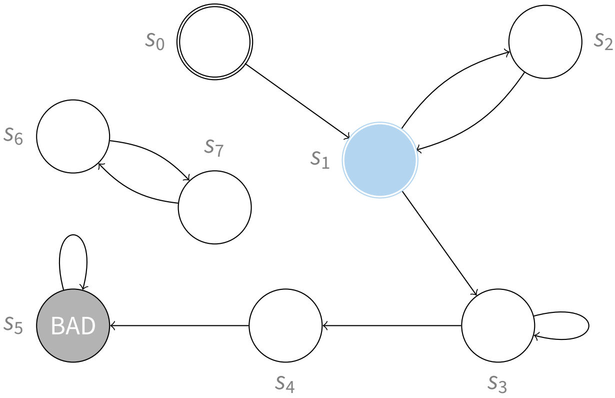
Reachability in n steps

States reachable in (exactly) 0 steps:



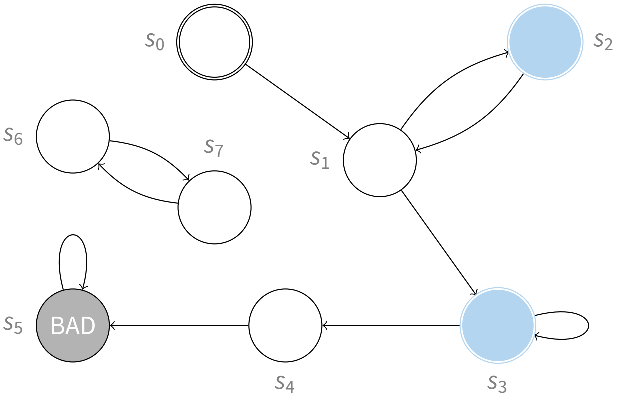
Reachability in n steps

States reachable in (exactly) **1** steps:



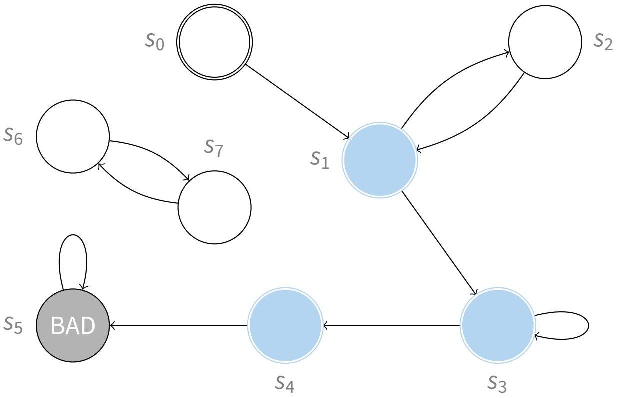
Reachability in n steps

States reachable in (exactly) 2 steps:



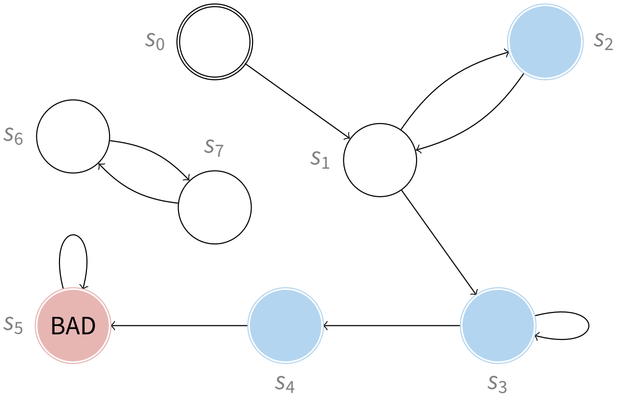
Reachability in n steps

States reachable in (exactly) 3 steps:



Reachability in n steps

States reachable in (exactly) 4 steps:



Simple Logical Analysis

Notation: If $\boldsymbol{z} = (z_1, \ldots, z_n)$ is a tuple of variables, $\exists \boldsymbol{z}\, F$ abbreviates $\exists z_1 \cdots \exists z_n\, F$

Lemma 2

Let $C(\boldsymbol{x})$ symbolically represent a set of states S_C. The formula

$$FR(\boldsymbol{x}) \quad \stackrel{\text{def}}{=} \quad \exists \boldsymbol{z}\, (C(\boldsymbol{z}) \wedge T(\boldsymbol{z}, \boldsymbol{x}))$$

denotes the set of states reachable from S_C in one step.

Simple Logical Analysis

Lemma 3

For all $n \geq 0$, the formula R_n, defined inductively by:

$$R_0(\boldsymbol{x}) \stackrel{\text{def}}{=} I(\boldsymbol{x}) \qquad\qquad R_{n+1}(\boldsymbol{x}) \stackrel{\text{def}}{=} \exists \boldsymbol{z}\,(R_n(\boldsymbol{z}) \wedge T(\boldsymbol{z}, \boldsymbol{x}))$$

denotes the set of states reachable in exactly n steps.

Simple Logical Analysis

Lemma 3

For all $n \geq 0$, the formula R_n, defined inductively by:

$$R_0(\boldsymbol{x}) \stackrel{\text{def}}{=} I(\boldsymbol{x}) \qquad\qquad R_{n+1}(\boldsymbol{x}) \stackrel{\text{def}}{=} \exists \boldsymbol{z}\,(R_n(\boldsymbol{z}) \wedge T(\boldsymbol{z}, \boldsymbol{x}))$$

denotes the set of states reachable in exactly n steps.

Note:

$$\begin{aligned}
R_n(\boldsymbol{x}_n) &= \exists \boldsymbol{z}\,(R_{n-1}(\boldsymbol{z}) \wedge T(\boldsymbol{z}, \boldsymbol{x}_n)) \\
&\equiv \exists \boldsymbol{x}_{n-1}\,(R_{n-1}(\boldsymbol{x}_{n-1}) \wedge T(\boldsymbol{x}_{n-1}, \boldsymbol{x}_n)) \\
&\equiv \exists \boldsymbol{x}_{n-1}\,(\exists \boldsymbol{x}_{n-2}\,(R_{n-2}(\boldsymbol{x}_{n-2}) \wedge T(\boldsymbol{x}_{n-2}, \boldsymbol{x}_{n-1})) \wedge T(\boldsymbol{x}_{n-1}, \boldsymbol{x}_n)) \\
&\equiv \exists \boldsymbol{x}_{n-1}\,(\exists \boldsymbol{x}_{n-2}\,(\cdots \exists \boldsymbol{x}_0\,(I(\boldsymbol{x}_0) \wedge T(\boldsymbol{x}_0, \boldsymbol{x}_1)) \cdots) \wedge T(\boldsymbol{x}_{n-1}, \boldsymbol{x}_n))
\end{aligned}$$

Simple Logical Analysis

Lemma 3

For all $n \geq 0$*, the formula* R_n*, defined inductively by:*

$$R_0(\boldsymbol{x}) \stackrel{\text{def}}{=} I(\boldsymbol{x}) \qquad R_{n+1}(\boldsymbol{x}) \stackrel{\text{def}}{=} \exists \boldsymbol{z}\,(R_n(\boldsymbol{z}) \wedge T(\boldsymbol{z}, \boldsymbol{x}))$$

denotes the set of states reachable in exactly n *steps.*

Note:

$$\begin{aligned}
R_n(\boldsymbol{x}_n) &= \exists \boldsymbol{z}\,(R_{n-1}(\boldsymbol{z}) \wedge T(\boldsymbol{z}, \boldsymbol{x}_n)) \\
&\equiv \exists \boldsymbol{x}_{n-1}\,(R_{n-1}(\boldsymbol{x}_{n-1}) \wedge T(\boldsymbol{x}_{n-1}, \boldsymbol{x}_n)) \\
&\equiv \exists \boldsymbol{x}_{n-1}\,(\exists \boldsymbol{x}_{n-2}\,(R_{n-2}(\boldsymbol{x}_{n-2}) \wedge T(\boldsymbol{x}_{n-2}, \boldsymbol{x}_{n-1})) \wedge T(\boldsymbol{x}_{n-1}, \boldsymbol{x}_n)) \\
&\equiv \exists \boldsymbol{x}_{n-1}\,(\exists \boldsymbol{x}_{n-2}\,(\cdots \exists \boldsymbol{x}_0\,(I(\boldsymbol{x}_0) \wedge T(\boldsymbol{x}_0, \boldsymbol{x}_1)) \cdots) \wedge T(\boldsymbol{x}_{n-1}, \boldsymbol{x}_n))
\end{aligned}$$

$R_n(\boldsymbol{x}_n)$ is equisatisfiable with $I(\boldsymbol{x}_0) \wedge T(\boldsymbol{x}_0, \boldsymbol{x}_1) \wedge \cdots \wedge T(\boldsymbol{x}_{n-1}, \boldsymbol{x}_n)$

Simple Forward Reachability Algorithm

Checks that it is possible to reach a state that satisfies F

procedure $SFReach(I, T, F)$
input: formulas $I(\boldsymbol{x}), T(\boldsymbol{x}, \boldsymbol{x}'), F(\boldsymbol{x})$
output: "yes" or "no" output
begin
  $i \;:=\; 0$
  $R \;:=\; I(\boldsymbol{x}_0)$
  loop
    if $R \wedge F(\boldsymbol{x}_i)$ is satisfiable
    then return "yes"
    $R \;:=\; R \wedge T(\boldsymbol{x}_i, \boldsymbol{x}_{i+1})$
    $i \;:=\; i + 1$
  end loop
end

Simple Forward Reachability Algorithm

Checks that it is possible to reach a state that satisfies F

procedure $SFReach(I, T, F)$
input: formulas $I(\boldsymbol{x}), T(\boldsymbol{x}, \boldsymbol{x}'), F(\boldsymbol{x})$
output: "yes" or "no" output
begin
$i \ := \ 0$
$R \ := \ I(\boldsymbol{x}_0)$
loop
if $R \wedge F(\boldsymbol{x}_i)$ is satisfiable
then return "yes"
$R \ := \ R \wedge T(\boldsymbol{x}_i, \boldsymbol{x}_{i+1})$
$i \ := \ i + 1$
end loop
end

How do we check the satisfiability of $R \wedge F(\boldsymbol{x}_i)$?

Simple Forward Reachability Algorithm

Checks that it is possible to reach a state that satisfies F

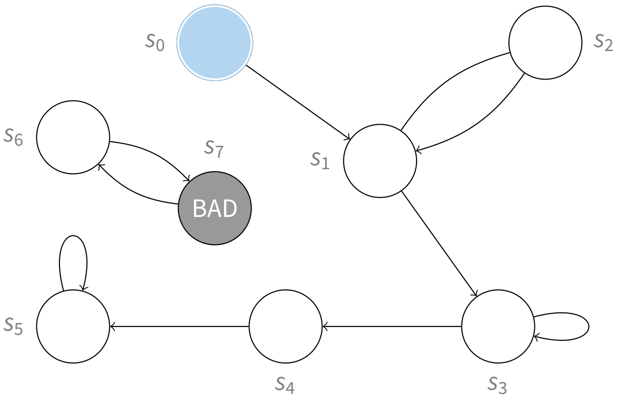
procedure $SFReach(I, T, F)$
input: formulas $I(\boldsymbol{x}), T(\boldsymbol{x}, \boldsymbol{x}'), F(\boldsymbol{x})$
output: "yes" or "no" output
begin
  $i := 0$
  $R := I(\boldsymbol{x}_0)$
  loop
    if $R \wedge F(\boldsymbol{x}_i)$ is satisfiable
    then return "yes"
    $R := R \wedge T(\boldsymbol{x}_i, \boldsymbol{x}_{i+1})$
    $i := i + 1$
  end loop
end

How do we check the satisfiability of $R \wedge F(\boldsymbol{x}_i)$?

Using SAT solvers!

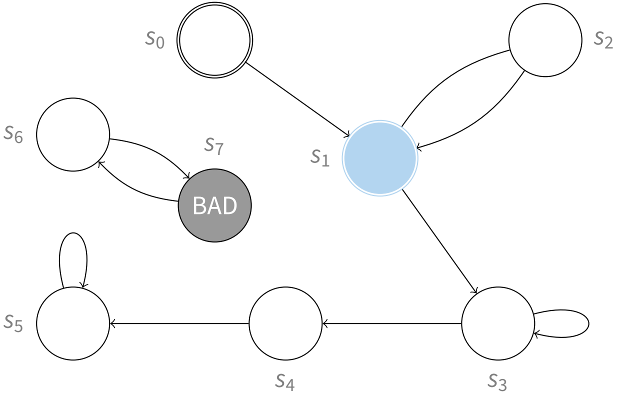
Termination

States reachable in (exactly) 0 steps:



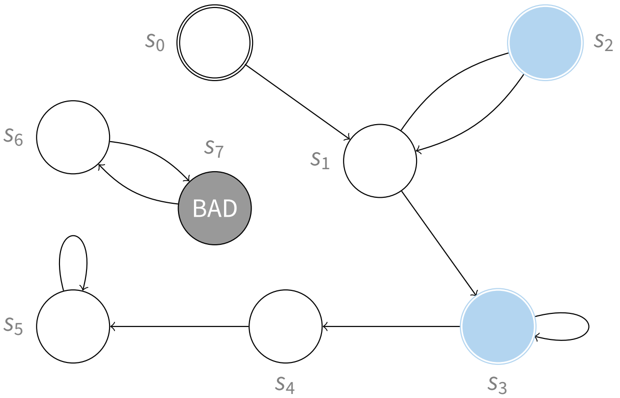
Termination

States reachable in (exactly) 1 steps:



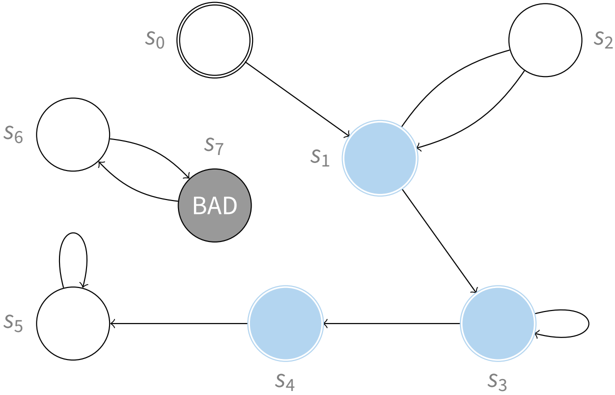
Termination

States reachable in (exactly) 2 steps:



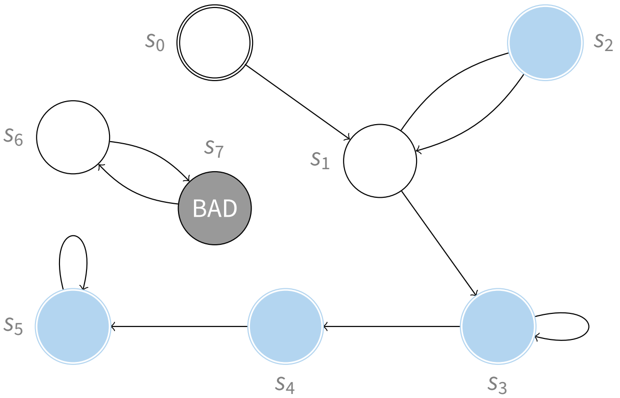
Termination

States reachable in (exactly) 3 steps:



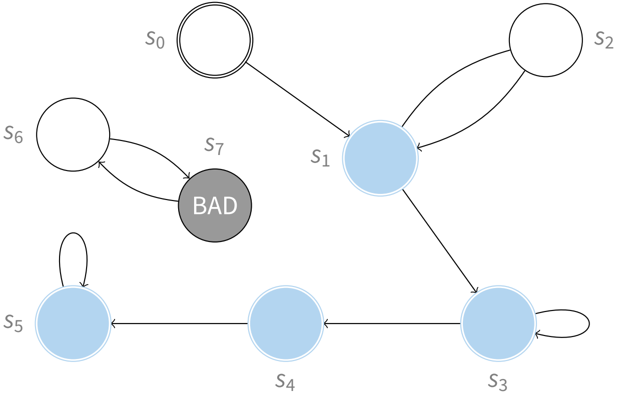
Termination

States reachable in (exactly) 4 steps:



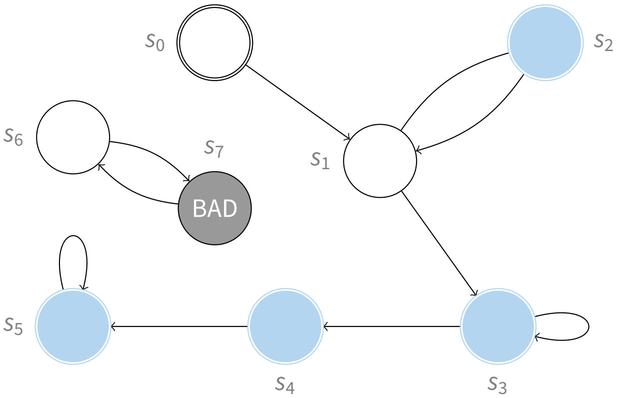
Termination

States reachable in (exactly) 5 steps:



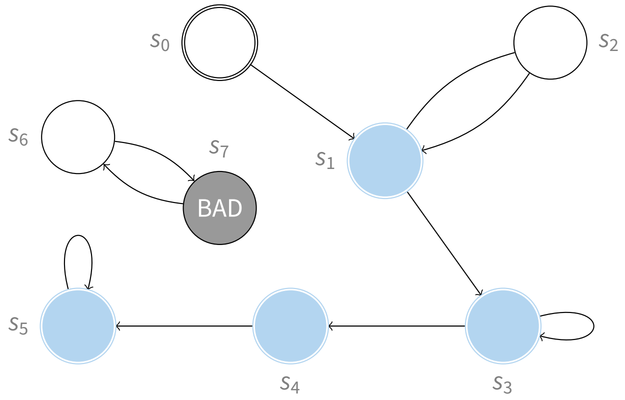
Termination

States reachable in (exactly) 6 steps:



Termination

States reachable in (exactly) 7 steps:



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

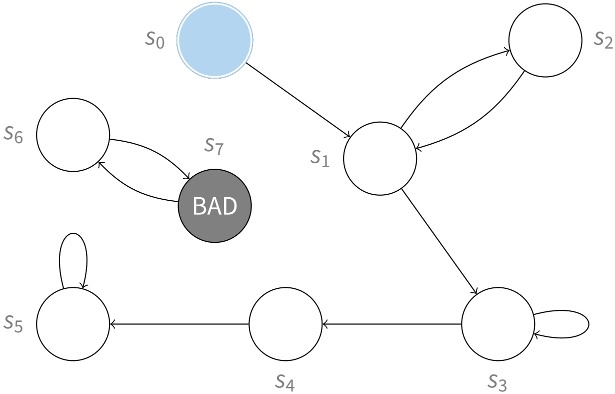
Reachability in $\leq n$ steps

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

$$\begin{aligned} R_{\leq 0}(\boldsymbol{x}) &\stackrel{\text{def}}{=} I(\boldsymbol{x}) \\ R_{\leq n+1}(\boldsymbol{x}) &\stackrel{\text{def}}{=} R_{\leq n}(\boldsymbol{x}) \lor \exists \boldsymbol{z}\,(R_{\leq n}(\boldsymbol{z}) \land T(\boldsymbol{z}, \boldsymbol{x})) \end{aligned}$$

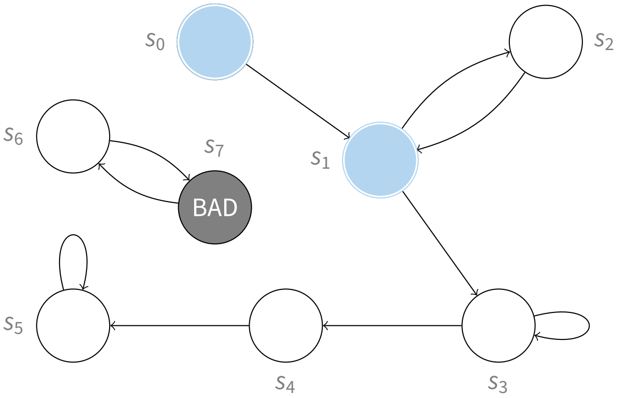
Reachability in $\leq n$ steps

States reachable in at most 0 steps:



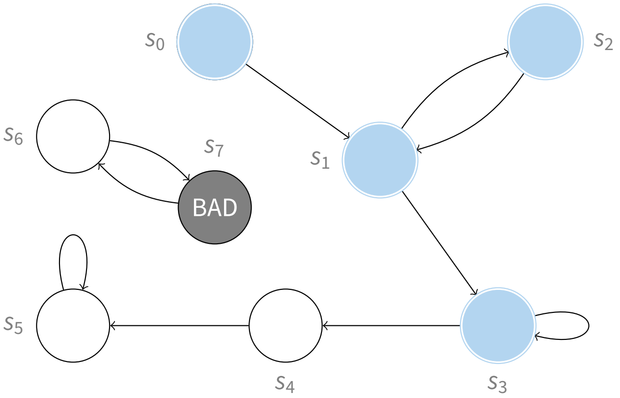
Reachability in $\leq n$ steps

States reachable in at most 1 steps:



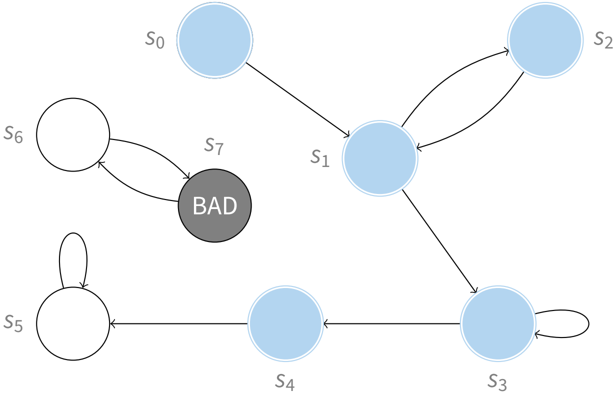
Reachability in $\leq n$ steps

States reachable in at most 2 steps:



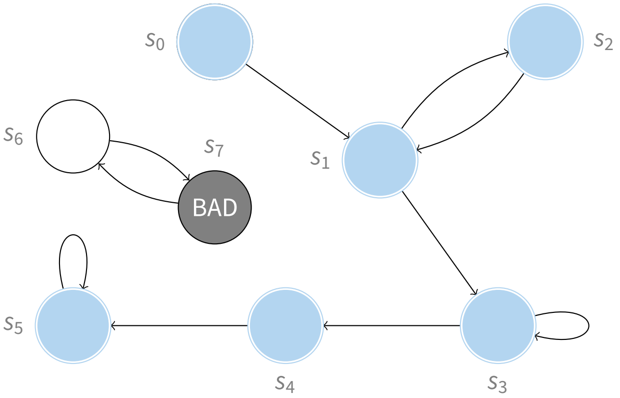
Reachability in $\leq n$ steps

States reachable in at most 3 steps:



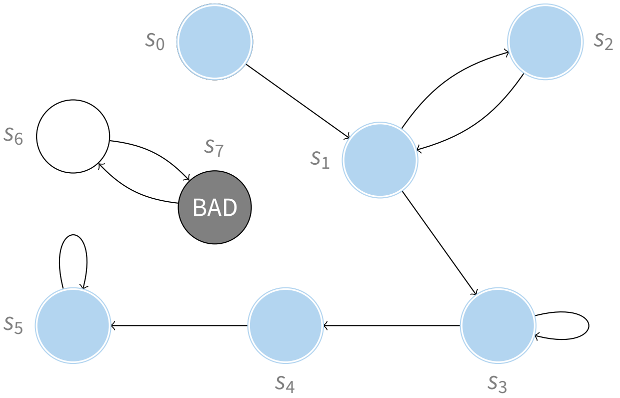
Reachability in $\leq n$ steps

States reachable in at most 4 steps:



Reachability in $\leq n$ steps

States reachable in at most 5 steps:



Full set of reachable states has been determined!

Termination

Let S_n the set of states reachable in $\leq n$ steps

Key properties for termination:

1. $S_n \subseteq S_{n+1}$ for all $n \leq 0$
2. the state space is finite

Consequences:

- there is k such that $S_k = S_{k+1}$
- for such k we have $R_{\leq k}(\boldsymbol{x}) = R_{\leq k+1}(\boldsymbol{x})$

Termination

Let S_n the set of states reachable in $\leq n$ steps

Key properties for termination:

1. $S_n \subseteq S_{n+1}$ for all $n \leq 0$
2. the state space is finite

Consequences:

- there is k such that $S_k = S_{k+1}$
- for such k we have $R_{\leq k}(\boldsymbol{x}) \equiv R_{\leq k+1}(\boldsymbol{x})$

Forward Reachability Algorithm

procedure $FReach(I, T, F)$
input: formulas $I(\boldsymbol{x}), T(\boldsymbol{x}, \boldsymbol{x}'), F(\boldsymbol{x})$
output: "yes" or "no"
begin
 $R(\boldsymbol{x}) \;:=\; I(\boldsymbol{x})$
 loop
  if $R(\boldsymbol{x}) \wedge F(\boldsymbol{x})$ is satisfiable
  then return "yes"
  $R'(\boldsymbol{x}) \;:=\; R(\boldsymbol{x}) \vee \exists \boldsymbol{z}\,(R(\boldsymbol{z}) \wedge T(\boldsymbol{z}, \boldsymbol{x}))$
  if $R(\boldsymbol{x}) \equiv R'(\boldsymbol{x})$ **then return** "no"
  $R(\boldsymbol{x}) \;:=\; R'(\boldsymbol{x})$
 end loop
end

Forward Reachability Algorithm

procedure $FReach(I, T, F)$
input: formulas $I(\boldsymbol{x}), T(\boldsymbol{x}, \boldsymbol{x}'), F(\boldsymbol{x})$
output: "yes" or "no"
begin
$R(\boldsymbol{x}) \ := \ I(\boldsymbol{x})$
loop
if $R(\boldsymbol{x}) \land F(\boldsymbol{x})$ is satisfiable
then return "yes"
$R'(\boldsymbol{x}) \ := \ R(\boldsymbol{x}) \lor \exists \boldsymbol{z}\, (R(\boldsymbol{z}) \land T(\boldsymbol{z}, \boldsymbol{x}))$
if $R(\boldsymbol{x}) \equiv R'(\boldsymbol{x})$ **then return** "no"
$R(\boldsymbol{x}) \ := \ R'(\boldsymbol{x})$
end loop
end

Implementation?

Forward Reachability Algorithm

procedure $FReach(I, T, F)$
input: formulas $I(\boldsymbol{x}), T(\boldsymbol{x}, \boldsymbol{x}'), F(\boldsymbol{x})$
output: "yes" or "no"
begin
  $R(\boldsymbol{x}) \ := \ I(\boldsymbol{x})$
  loop
    if $R(\boldsymbol{x}) \wedge F(\boldsymbol{x})$ is satisfiable
    then return "yes"
    $R'(\boldsymbol{x}) \ := \ R(\boldsymbol{x}) \vee \exists \boldsymbol{z}\,(R(\boldsymbol{z}) \wedge T(\boldsymbol{z}, \boldsymbol{x}))$
    if $R(\boldsymbol{x}) \equiv R'(\boldsymbol{x})$ **then return** "no"
    $R(\boldsymbol{x}) \ := \ R'(\boldsymbol{x})$
  end loop
end

Implementation?

Conjunction and disjunction

Forward Reachability Algorithm

procedure $FReach(I, T, F)$
input: formulas $I(\boldsymbol{x}), T(\boldsymbol{x}, \boldsymbol{x}'), F(\boldsymbol{x})$
output: "yes" or "no"
begin
$R(\boldsymbol{x}) \ := \ I(\boldsymbol{x})$
loop
if $R(\boldsymbol{x}) \wedge F(\boldsymbol{x})$ is satisfiable
then return "yes"
$R'(\boldsymbol{x}) \ := \ R(\boldsymbol{x}) \vee \exists \boldsymbol{z}\, (R(\boldsymbol{z}) \wedge T(\boldsymbol{z}, \boldsymbol{x}))$
if $R(\boldsymbol{x}) \equiv R'(\boldsymbol{x})$ **then return** "no"
$R(\boldsymbol{x}) \ := \ R'(\boldsymbol{x})$
end loop
end

Implementation?

Conjunction and disjunction
Quantification

Forward Reachability Algorithm

procedure $FReach(I, T, F)$
input: formulas $I(\boldsymbol{x}), T(\boldsymbol{x}, \boldsymbol{x}'), F(\boldsymbol{x})$
output: "yes" or "no"
begin
 $R(\boldsymbol{x}) \ := \ I(\boldsymbol{x})$
 loop
  if $R(\boldsymbol{x}) \wedge F(\boldsymbol{x})$ is satisfiable
  then return "yes"
  $R'(\boldsymbol{x}) \ := \ R(\boldsymbol{x}) \vee \exists \boldsymbol{z}\, (R(\boldsymbol{z}) \wedge T(\boldsymbol{z}, \boldsymbol{x}))$
  if $R(\boldsymbol{x}) \equiv R'(\boldsymbol{x})$ **then return** "no"
  $R(\boldsymbol{x}) \ := \ R'(\boldsymbol{x})$
 end loop
end

Implementation?

Conjunction and disjunction
Quantification
Satisfiability checking

Forward Reachability Algorithm

procedure $FReach(I, T, F)$
input: formulas $I(\boldsymbol{x}), T(\boldsymbol{x}, \boldsymbol{x}'), F(\boldsymbol{x})$
output: "yes" or "no"
begin
$R(\boldsymbol{x}) \ := \ I(\boldsymbol{x})$
loop
if $R(\boldsymbol{x}) \wedge F(\boldsymbol{x})$ is satisfiable
then return "yes"
$R'(\boldsymbol{x}) \ := \ R(\boldsymbol{x}) \vee \exists \boldsymbol{z}\, (R(\boldsymbol{z}) \wedge T(\boldsymbol{z}, \boldsymbol{x}))$
if $R(\boldsymbol{x}) \equiv R'(\boldsymbol{x})$ **then return** "no"
$R(\boldsymbol{x}) \ := \ R'(\boldsymbol{x})$
end loop
end

Implementation?

Conjunction and disjunction
Quantification
Satisfiability checking
Equivalence checking

Forward Reachability Algorithm

procedure $FReach(I, T, F)$
input: formulas $I(\boldsymbol{x}), T(\boldsymbol{x}, \boldsymbol{x}'), F(\boldsymbol{x})$
output: “yes” or “no”
begin
$R(\boldsymbol{x}) \ := \ I(\boldsymbol{x})$
loop
if $R(\boldsymbol{x}) \wedge F(\boldsymbol{x})$ is satisfiable
then return “yes”
$R'(\boldsymbol{x}) \ := \ R(\boldsymbol{x}) \vee \exists \boldsymbol{z}\, (R(\boldsymbol{z}) \wedge T(\boldsymbol{z}, \boldsymbol{x}))$
if $R(\boldsymbol{x}) \equiv R'(\boldsymbol{x})$ **then return** “no”
$R(\boldsymbol{x}) \ := \ R'(\boldsymbol{x})$
end loop
end

Implementation?

Use QBF techiques or
OBDDs and OBDD algorithms

Conjunction and disjunction
Quantification
Satisfiability checking
Equivalence checking

Main Issues with Forward Reachability Algorithms

Forward reachability behaves in the same way,
independently of the set of final states

In other words, it is not goal oriented

Backward Reachability

Idea:

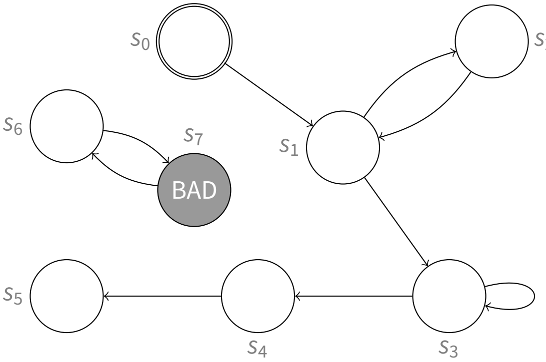
- instead of going forward in the state transition graph, go backward
- swap initial and final states and invert the transition relation

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

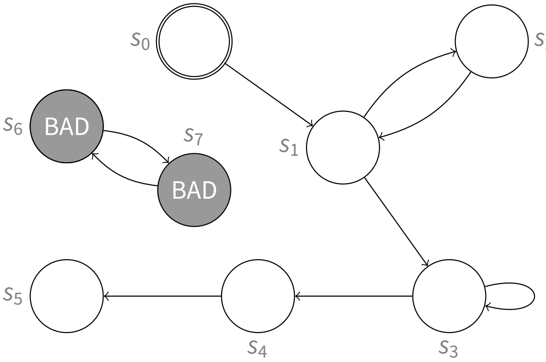


Backward Reachability in $\leq n$ steps

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

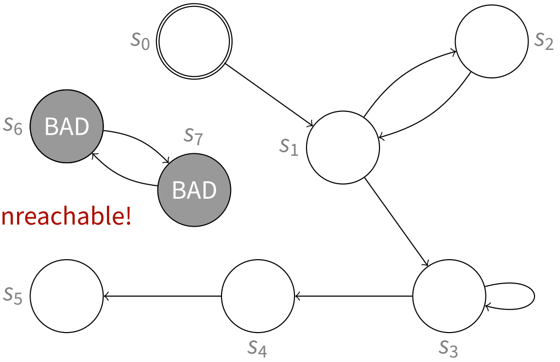


Backward Reachability in $\leq n$ steps

Idea:

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

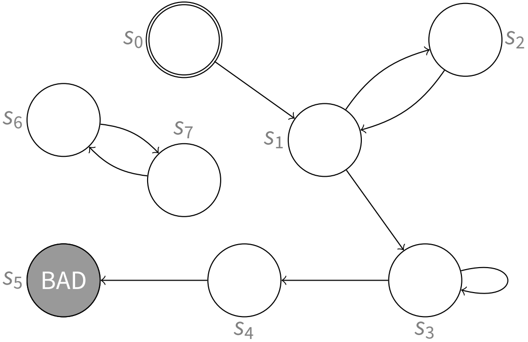
Number of backward steps: 1



Bad states are unreachable!

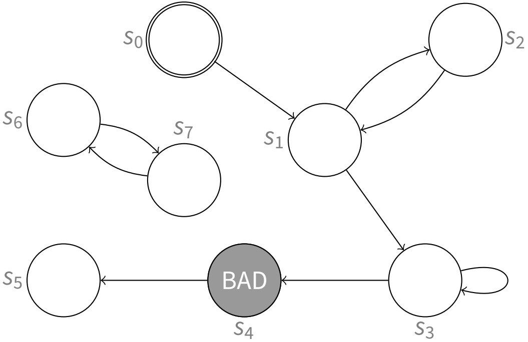
Backward Reachability in *n* steps

Number of backward steps: 0



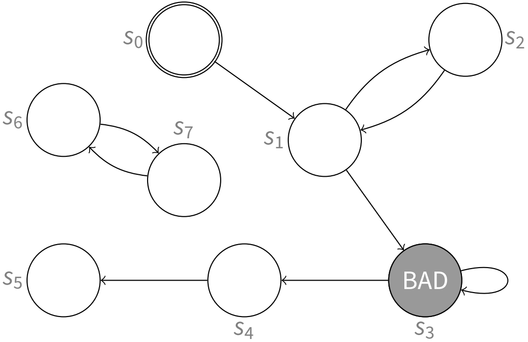
Backward Reachability in n steps

Number of backward steps: 1



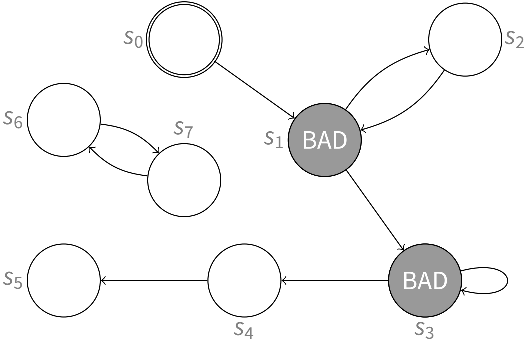
Backward Reachability in n steps

Number of backward steps: 2



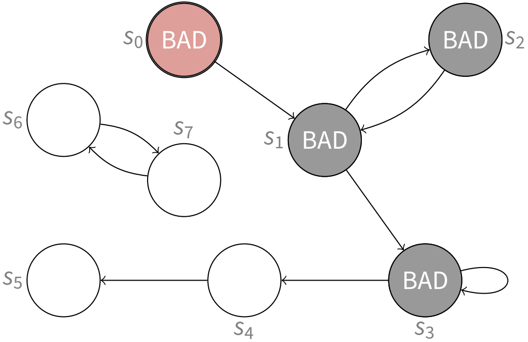
Backward Reachability in *n* steps

Number of backward steps: 3



Backward Reachability in *n* steps

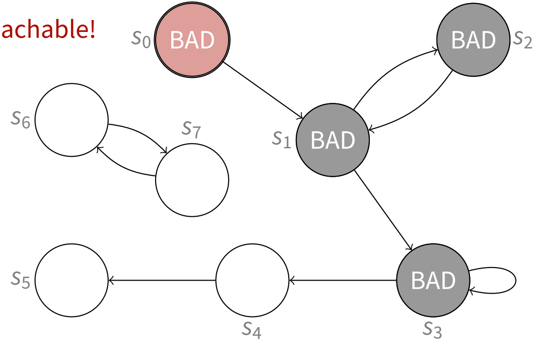
Number of backward steps: 4



Backward Reachability in n steps

Number of backward steps: 4

Bad states are reachable!



Backward Reachability

S_0 is *backward reachable from F in n steps*
if F is reachable from S_0 in n steps

Backward Reachability

S_0 is *backward reachable from F in n steps*
if F is reachable from S_0 in n steps

Lemma 4
Let $C(\boldsymbol{x})$ symbolically represent a set of states S_C. The formula

$$BR(\boldsymbol{x}) \quad \stackrel{\text{def}}{=} \quad \exists \boldsymbol{z}\,(T(\boldsymbol{x}, \boldsymbol{z}) \wedge C(\boldsymbol{z}))$$

denotes the set of states backward reachable from S_C in one step.

Backward Reachability Algorithm

Same as the forward reachability algorithms, but

- swap I with F
- invert the transition relation T

Backward Reachability Algorithm

Same as the forward reachability algorithms, but

- swap I with F
- invert the transition relation T

procedure $BReach(I, T, F)$
input: formulas I, T, F
output: “yes” or “no”
begin
$R(\boldsymbol{x}) \ := \ F(\boldsymbol{x})$
loop
if $R(\boldsymbol{x}) \wedge I(\boldsymbol{x})$ is satisfiable **then**
return “yes”
$R'(\boldsymbol{x}) \ := \ R(\boldsymbol{x}) \vee \exists \boldsymbol{z}\,(T(\boldsymbol{x}, \boldsymbol{z}) \wedge R(\boldsymbol{z}))$
if $R(\boldsymbol{x}) \equiv R'(\boldsymbol{x})$ **then return** “no”
$R(\boldsymbol{x}) \ := \ R'(\boldsymbol{x})$
end loop
end

Backward Reachability Algorithm

Same as the forward reachability algorithms, but

- swap I with F
- invert the transition relation T

procedure $BReach(I, T, F)$
input: formulas I, T, F
output: "yes" or "no"
begin
  $R(\boldsymbol{x}) \ := \ F(\boldsymbol{x})$
  loop
    if $R(\boldsymbol{x}) \wedge I(\boldsymbol{x})$ is satisfiable **then**
      return "yes"
    $R'(\boldsymbol{x}) \ := \ R(\boldsymbol{x}) \vee \exists \boldsymbol{z}\, (T(\boldsymbol{x}, \boldsymbol{z}) \wedge R(\boldsymbol{z}))$
    if $R(\boldsymbol{x}) \equiv R'(\boldsymbol{x})$ **then return** "no"
    $R(\boldsymbol{x}) \ := \ R'(\boldsymbol{x})$
  end loop
end

procedure $FReach(I, T, F)$
input: formulas I, T, F
output: "yes" or "no"
begin
  $R(\boldsymbol{x}) \ := \ I(\boldsymbol{x})$
  loop
    if $R(\boldsymbol{x}) \wedge F(\boldsymbol{x})$ is satisfiable **then**
      return "yes"
    $R'(\boldsymbol{x}) \ := \ R(\boldsymbol{x}) \vee \exists \boldsymbol{z}\, (R(\boldsymbol{z}) \wedge T(\boldsymbol{z}, \boldsymbol{x}))$
    if $R(\boldsymbol{x}) \equiv R'(\boldsymbol{x})$ **then return** "no"
    $R(\boldsymbol{x}) \ := \ R'(\boldsymbol{x})$
  end loop
end

Backward Reachability Algorithm

Same as the forward reachability algorithms, but

- swap I with F
- invert the transition relation T

procedure $BReach(I, T, F)$
input: formulas I, T, F
output: "yes" or "no"
begin
  $R(\boldsymbol{x}) \;:=\; F(\boldsymbol{x})$
  loop
    if $R(\boldsymbol{x}) \wedge I(\boldsymbol{x})$ is satisfiable **then**
      return "yes"
    $R'(\boldsymbol{x}) \;:=\; R(\boldsymbol{x}) \vee \exists \boldsymbol{z}\,(T(\boldsymbol{x}, \boldsymbol{z}) \wedge R(\boldsymbol{z}))$
    if $R(\boldsymbol{x}) \equiv R'(\boldsymbol{x})$ **then return** "no"
    $R(\boldsymbol{x}) \;:=\; R'(\boldsymbol{x})$
  end loop
end

procedure $FReach(I, T, F)$
input: formulas I, T, F
output: "yes" or "no"
begin
  $R(\boldsymbol{x}) \;:=\; I(\boldsymbol{x})$
  loop
    if $R(\boldsymbol{x}) \wedge F(\boldsymbol{x})$ is satisfiable **then**
      return "yes"
    $R'(\boldsymbol{x}) \;:=\; R(\boldsymbol{x}) \vee \exists \boldsymbol{z}\,(R(\boldsymbol{z}) \wedge T(\boldsymbol{z}, \boldsymbol{x}))$
    if $R(\boldsymbol{x}) \equiv R'(\boldsymbol{x})$ **then return** "no"
    $R(\boldsymbol{x}) \;:=\; R'(\boldsymbol{x})$
  end loop
end

Checking Invariant Properties

Reachability checking can be used to prove invariant properties too

To check whether a state property P is invariant for a system $\mathbb{S}$:

$$\mathbb{S} \models \Box P$$

we can check the reachability in $\mathbb{S}$ of $\neg P$

Reason: P is invariant iff $\neg P$ is unreachable

However, there are more direct and often more efficient ways to check for invariance

Checking Invariant Properties

Reachability checking can be used to prove invariant properties too

To check whether a state property P is invariant for a system $\mathbb{S}$:

$$\mathbb{S} \models \Box P$$

we can check the reachability in $\mathbb{S}$ of $\neg P$

Reason: P is invariant iff $\neg P$ is unreachable

However, there are more direct and often more efficient ways to check for invariance

Checking Invariant Properties

Reachability checking can be used to prove invariant properties too

To check whether a state property P is invariant for a system $\mathbb{S}$:

$$\mathbb{S} \models \Box P$$

we can check the reachability in $\mathbb{S}$ of $\neg P$

Reason: F is invariant iff $\neg P$ is unreachable

However, there are more direct and often more efficient ways to check for invariance

Checking Invariant Properties

Reachability checking can be used to prove invariant properties too

To check whether a state property P is invariant for a system $\mathbb{S}$:

$$\mathbb{S} \models \Box P$$

we can check the reachability in $\mathbb{S}$ of $\neg P$

Reason: F is invariant iff $\neg P$ is unreachable

However, there are more direct and often more efficient ways to check for invariance

Invariant Checking by Temporal Induction

Consider system $\mathbb{S}$ with initial condition $I(\boldsymbol{x})$ and transition formula $T(\boldsymbol{x}, \boldsymbol{x}')$

Theorem 5
$P(\boldsymbol{x})$ is invariant for $\mathbb{S}$ if the following entailments hold in PLFD:

(base case) $I(\boldsymbol{x}) \models P(\boldsymbol{x})$
(inductive step) $P(\boldsymbol{x}) \wedge T(\boldsymbol{x}, \boldsymbol{x}') \models P(\boldsymbol{x}')$

Invariant Checking by Temporal Induction

Consider system $\mathbb{S}$ with initial condition $I(\boldsymbol{x})$ and transition formula $T(\boldsymbol{x}, \boldsymbol{x}')$

Theorem 5

$P(\boldsymbol{x})$ is invariant for $\mathbb{S}$ if the following entailments hold in PLFD:

(base case)	$I(\boldsymbol{x}) \models P(\boldsymbol{x})$
(inductive step)	$P(\boldsymbol{x}) \wedge T(\boldsymbol{x}, \boldsymbol{x}') \models P(\boldsymbol{x}')$

Invariant Checking by Temporal Induction

Consider system $\mathbb{S}$ with initial condition $I(\boldsymbol{x})$ and transition formula $T(\boldsymbol{x}, \boldsymbol{x}')$

Theorem 5

$P(\boldsymbol{x})$ is invariant for $\mathbb{S}$ if the following entailments hold in PLFD:

$$\begin{array}{rl} \textit{(base case)} & I(\boldsymbol{x}) \models P(\boldsymbol{x}) \\ \textit{(inductive step)} & P(\boldsymbol{x}) \wedge T(\boldsymbol{x}, \boldsymbol{x}') \models P(\boldsymbol{x}') \end{array}$$

iff

- $I(\boldsymbol{x}) \wedge \neg P(\boldsymbol{x})$ *is unsatisfiable and*
- $P(\boldsymbol{x}) \wedge T(\boldsymbol{x}, \boldsymbol{x}') \wedge \neg P(\boldsymbol{x}')$ *is unsatisfiable*

Invariant Checking by Temporal Induction

Consider system $\mathbb{S}$ with initial condition $I(\boldsymbol{x})$ and transition formula $T(\boldsymbol{x}, \boldsymbol{x}')$

Theorem 5
$P(\boldsymbol{x})$ is invariant for $\mathbb{S}$ if the following entailments hold in PLFD:

(base case)	$I(\boldsymbol{x}) \models P(\boldsymbol{x})$
(inductive step)	$P(\boldsymbol{x}) \wedge T(\boldsymbol{x}, \boldsymbol{x}') \models P(\boldsymbol{x}')$

iff

- $I(\boldsymbol{x}) \wedge \neg P(\boldsymbol{x})$ *is unsatisfiable and*
- $P(\boldsymbol{x}) \wedge T(\boldsymbol{x}, \boldsymbol{x}') \wedge \neg P(\boldsymbol{x}')$ *is unsatisfiable*

In that case, P is *(temporally) inductive* for $\mathbb{S}$

Invariant Checking by Temporal Induction

Consider system $\mathbb{S}$ with initial condition $I(\boldsymbol{x})$ and transition formula $T(\boldsymbol{x}, \boldsymbol{x}')$

Theorem 5
$P(\boldsymbol{x})$ is invariant for $\mathbb{S}$ if the following entailments hold in PLFD:

(base case)	$I(\boldsymbol{x}) \models P(\boldsymbol{x})$
(inductive step)	$P(\boldsymbol{x}) \wedge T(\boldsymbol{x}, \boldsymbol{x}') \models P(\boldsymbol{x}')$

Problem: Not all invariants are inductive

Example 1: Inductive vs. Invariant

$$dom(x_1) = dom(x_2) = \{0, 1, 2, 3, 4, 5, 6, 7\}$$

$$\begin{aligned} I(x_1, x_2) &\stackrel{\text{def}}{=} x_1 = 0 \land x_2 = 1 \\ T(x_1, x_2, x_1', x_2') &\stackrel{\text{def}}{=} (x_2 \neq 3 \rightarrow x_2' = x_2 + 1) \\ &\land (x_2 = 3 \rightarrow x_2' = 0) \\ &\land x_1' = x_2 \end{aligned}$$

Note: This system can be encoded faithfully in PLFD (and so in PL)

Example 1: Inductive vs. Invariant

$dom(x_1) = dom(x_2) = \{0, 1, 2, 3, 4, 5, 6, 7\}$

$$
\begin{aligned}
I(x_1, x_2) &\stackrel{\text{def}}{=} x_1 = 0 \land x_2 = 1 \\
T(x_1, x_2, x_1', x_2') &\stackrel{\text{def}}{=} (x_2 \neq 3 \rightarrow x_2' = x_2 + 1) \\
&\land\ (x_2 = 3 \rightarrow x_2' = 0) \\
&\land\ x_1' = x_2
\end{aligned}
$$

$$P(x_1, x_2) \stackrel{\text{def}}{=} 0 \leq x_2 \land x_2 \leq 3$$

Inductive? Invariant?

Example 1: Inductive vs. Invariant

$dom(x_1) = dom(x_2) = \{0, 1, 2, 3, 4, 5, 6, 7\}$

$$
\begin{aligned}
I(x_1, x_2) &\stackrel{\text{def}}{=} x_1 = 0 \land x_2 = 1 \\
T(x_1, x_2, x_1', x_2') &\stackrel{\text{def}}{=} (x_2 \neq 3 \rightarrow x_2' = x_2 + 1) \\
&\land\ (x_2 = 3 \rightarrow x_2' = 0) \\
&\land\ x_1' = x_2
\end{aligned}
$$

$$P(x_1, x_2) \stackrel{\text{def}}{=} 0 \leq x_2 \land x_2 \leq 3$$ Inductive? Invariant?

base) $I(x_1, x_2) \models P(x_1, x_2)$?

step) $P(x_1, x_2) \land T(x_1, x_2, x_1', x_2') \models P(x_1', x_2')$?

Example 1: Inductive vs. Invariant

$$dom(x_1) = dom(x_2) = \{ 0, 1, 2, 3, 4, 5, 6, 7 \}$$

$$\begin{aligned}
I(x_1, x_2) &\stackrel{\text{def}}{=} x_1 = 0 \wedge x_2 = 1 \\
T(x_1, x_2, x_1', x_2') &\stackrel{\text{def}}{=} (x_2 \neq 3 \rightarrow x_2' = x_2 + 1) \\
&\wedge\ (x_2 = 3 \rightarrow x_2' = 0) \\
&\wedge\ x_1' = x_2
\end{aligned}$$

✔ ✔

$$P(x_1, x_2) \stackrel{\text{def}}{=} 0 \leq x_2 \wedge x_2 \leq 3$$

Inductive? Invariant?

base) $I(x_1, x_2) \models P(x_1, x_2)$? ✔

step) $P(x_1, x_2) \wedge T(x_1, x_2, x_1', x_2') \models P(x_1', x_2')$? ✔

Example 1: Inductive vs. Invariant

$$dom(x_1) = dom(x_2) = \{\, 0, 1, 2, 3, 4, 5, 6, 7 \,\}$$

$$\begin{aligned}
I(x_1, x_2) &\stackrel{\text{def}}{=} x_1 = 0 \land x_2 = 1 \\
T(x_1, x_2, x_1', x_2') &\stackrel{\text{def}}{=} (x_2 \neq 3 \rightarrow x_2' = x_2 + 1) \\
&\land \quad (x_2 = 3 \rightarrow x_2' = 0) \\
&\land \quad x_1' = x_2
\end{aligned}$$

$$P(x_1, x_2) \stackrel{\text{def}}{=} x_2 \leq 4$$

Inductive? Invariant?

base) $I(x_1, x_2) \models P(x_1, x_2)$?

step) $P(x_1, x_2) \land T(x_1, x_2, x_1', x_2') \models P(x_1', x_2')$?

Example 1: Inductive vs. Invariant

$$dom(x_1) = dom(x_2) = \{0, 1, 2, 3, 4, 5, 6, 7\}$$

$$
\begin{aligned}
I(x_1, x_2) &\stackrel{\text{def}}{=} x_1 = 0 \land x_2 = 1 \\
T(x_1, x_2, x_1', x_2') &\stackrel{\text{def}}{=} (x_2 \neq 3 \rightarrow x_2' = x_2 + 1) \\
&\land \; (x_2 = 3 \rightarrow x_2' = 0) \\
&\land \; x_1' = x_2
\end{aligned}
$$

$$P(x_1, x_2) \stackrel{\text{def}}{=} x_2 \leq 4$$

✗

Inductive? Invariant?

base) $I(x_1, x_2) \models P(x_1, x_2)$? ✔

step) $P(x_1, x_2) \land T(x_1, x_2, x_1', x_2') \models P(x_1', x_2')$? ✗

$\{x_1 \mapsto 1, x_2 \mapsto 4\}, \{x_1' \mapsto 4, x_2' \mapsto 5\}$

Example 1: Inductive vs. Invariant

$dom(x_1) = dom(x_2) = \{0, 1, 2, 3, 4, 5, 6, 7\}$

$$
\begin{aligned}
I(x_1, x_2) &\stackrel{\text{def}}{=} x_1 = 0 \land x_2 = 1 \\
T(x_1, x_2, x_1', x_2') &\stackrel{\text{def}}{=} (x_2 \neq 3 \rightarrow x_2' = x_2 + 1) \\
&\land\ (x_2 = 3 \rightarrow x_2' = 0) \\
&\land\ x_1' = x_2
\end{aligned}
$$

$$P(x_1, x_2) \stackrel{\text{def}}{=} x_2 \leq 4$$

Inductive? ✗ Invariant? ✔

base) $I(x_1, x_2) \models P(x_1, x_2)$? ✔

step) $P(x_1, x_2) \land T(x_1, x_2, x_1', x_2') \models P(x_1', x_2')$? ✗

$\{x_1 \mapsto 1, x_2 \mapsto 4\}, \{x_1' \mapsto 4, x_2' \mapsto 5\}$

state $\{x_1 \mapsto 0, x_2 \mapsto 4\}$ is unreachable!

Example 1: Inductive vs. Invariant

$$dom(x_1) = dom(x_2) = \{0, 1, 2, 3, 4, 5, 6, 7\}$$

$$\begin{aligned}
I(x_1, x_2) &\stackrel{\text{def}}{=} x_1 = 0 \land x_2 = 1 \\
T(x_1, x_2, x_1', x_2') &\stackrel{\text{def}}{=} (x_2 \neq 3 \rightarrow x_2' = x_2 + 1) \\
&\land \quad (x_2 = 3 \rightarrow x_2' = 0) \\
&\land \quad x_1' = x_2
\end{aligned}$$

$$P(x_1, x_2) \stackrel{\text{def}}{=} x_1 < x_2$$

Inductive? Invariant?

base) $I(x_1, x_2) \models P(x_1, x_2)$?

step) $P(x_1, x_2) \land T(x_1, x_2, x_1', x_2') \models P(x_1', x_2')$?

Example 1: Inductive vs. Invariant

$$dom(x_1) = dom(x_2) = \{\, 0, 1, 2, 3, 4, 5, 6, 7 \,\}$$

$$\begin{aligned} I(x_1, x_2) &\stackrel{\text{def}}{=} x_1 = 0 \land x_2 = 1 \\ T(x_1, x_2, x_1', x_2') &\stackrel{\text{def}}{=} (x_2 \neq 3 \rightarrow x_2' = x_2 + 1) \\ &\land \ (x_2 = 3 \rightarrow x_2' = 0) \\ &\land \ x_1' = x_2 \end{aligned}$$

$$P(x_1, x_2) \stackrel{\text{def}}{=} x_1 < x_2$$

✗

Inductive? Invariant?

base) $I(x_1, x_2) \models P(x_1, x_2)$? ✓

step) $P(x_1, x_2) \land T(x_1, x_2, x_1', x_2') \models P(x_1', x_2')$? ✗

$\{\, x_1 \mapsto 2, x_2 \mapsto 3 \,\}, \{\, x_1' \mapsto 3, x_2' \mapsto 0 \,\}$

Example 1: Inductive vs. Invariant

$$dom(x_1) = dom(x_2) = \{0,1,2,3,4,5,6,7\}$$

$$\begin{aligned}
I(x_1,x_2) &\stackrel{\text{def}}{=} x_1 = 0 \land x_2 = 1 \\
T(x_1,x_2,x_1',x_2') &\stackrel{\text{def}}{=} (x_2 \neq 3 \rightarrow x_2' = x_2 + 1) \\
&\land\ (x_2 = 3 \rightarrow x_2' = 0) \\
&\land\ x_1' = x_2
\end{aligned}$$

$$P(x_1,x_2) \stackrel{\text{def}}{=} x_1 < x_2$$

✗ Inductive? ✗ Invariant?

base) $I(x_1,x_2) \models P(x_1,x_2)$? ✔

step) $P(x_1,x_2) \land T(x_1,x_2,x_1',x_2') \models P(x_1',x_2')$? ✗

$\{x_1 \mapsto 2, x_2 \mapsto 3\}, \{x_1' \mapsto 3, x_2' \mapsto 0\}$

state $\{x_1 \mapsto 2, x_2 \mapsto 3\}$ is reachable!

Example 1: Inductive vs. Invariant

$$dom(x_1) = dom(x_2) = \{\, 0, 1, 2, 3, 4, 5, 6, 7 \,\}$$

$$\begin{aligned}
I(x_1, x_2) &\stackrel{\text{def}}{=} x_1 = 0 \land x_2 = 1 \\
T(x_1, x_2, x_1', x_2') &\stackrel{\text{def}}{=} (x_2 \neq 3 \rightarrow x_2' = x_2 + 1) \\
&\quad \land \; (x_2 = 3 \rightarrow x_2' = 0) \\
&\quad \land \; x_1' = x_2
\end{aligned}$$

$$P(x_1, x_2) \stackrel{\text{def}}{=} 0 < x_1$$

Inductive? Invariant?

base) $I(x_1, x_2) \models P(x_1, x_2)$?

step) $P(x_1, x_2) \land T(x_1, x_2, x_1', x_2') \models P(x_1', x_2')$?

Example 1: Inductive vs. Invariant

$dom(x_1) = dom(x_2) = \{0, 1, 2, 3, 4, 5, 6, 7\}$

$$
\begin{aligned}
I(x_1, x_2) &\stackrel{\text{def}}{=} x_1 = 0 \land x_2 = 1 \\
T(x_1, x_2, x_1', x_2') &\stackrel{\text{def}}{=} (x_2 \neq 3 \rightarrow x_2' = x_2 + 1) \\
&\quad \land (x_2 = 3 \rightarrow x_2' = 0) \\
&\quad \land x_1' = x_2
\end{aligned}
$$

$$P(x_1, x_2) \stackrel{\text{def}}{=} 0 < x_1$$

✗ Inductive? ✗ Invariant?

base) $I(x_1, x_2) \models P(x_1, x_2)$? ✗

step) $P(x_1, x_2) \land T(x_1, x_2, x_1', x_2') \models P(x_1', x_2')$?

initial state $\{x_1 \mapsto 0, x_2 \mapsto 1\}$ is clearly reachable!

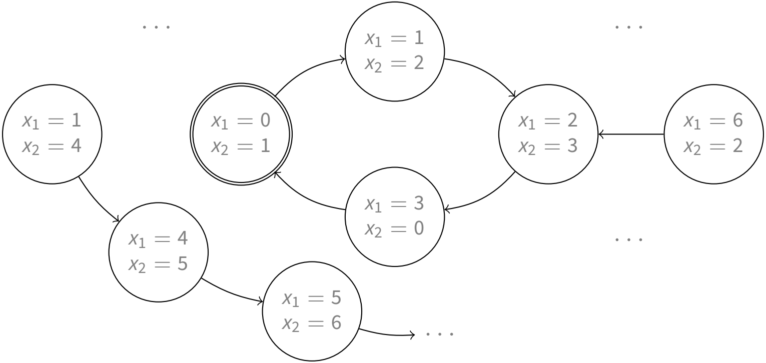
Example 1

$$\boldsymbol{x} = (x_1, x_2) \quad dom(x_1) = dom(x_2) = \{ 0, 1, 2, 3, 4, 5, 6, 7 \}$$

$$I(\boldsymbol{x}) \stackrel{\text{def}}{=} x_1 = 0 \land x_2 = 1$$

$$T(\boldsymbol{x}, \boldsymbol{x}') \stackrel{\text{def}}{=} x_1' = x_2 \land (x_2 \neq 3 \rightarrow x_2' = x_2 + 1) \land (x_2 = 3 \rightarrow x_2' = 0)$$

Transition graph fragment:



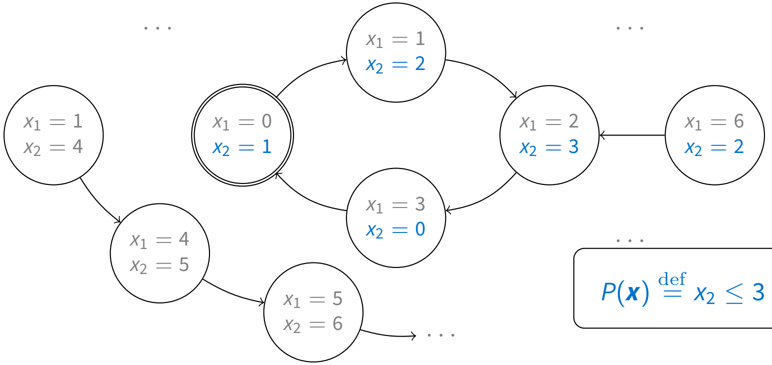
Example 1

$$\boldsymbol{x} = (x_1, x_2) \quad dom(x_1) = dom(x_2) = \{0, 1, 2, 3, 4, 5, 6, 7\}$$

$$I(\boldsymbol{x}) \stackrel{\text{def}}{=} x_1 = 0 \land x_2 = 1$$

$$T(\boldsymbol{x}, \boldsymbol{x}') \stackrel{\text{def}}{=} x_1' = x_2 \land (x_2 \neq 3 \rightarrow x_2' = x_2 + 1) \land (x_2 = 3 \rightarrow x_2' = 0)$$

Transition graph fragment:



$$P(\boldsymbol{x}) \stackrel{\text{def}}{=} x_2 \leq 3$$

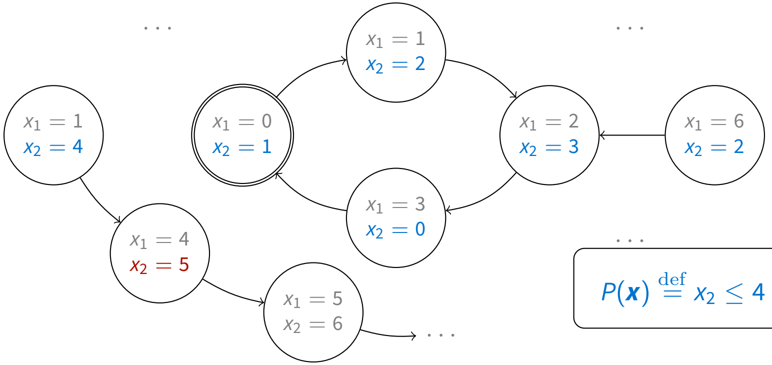
Example 1

$$\boldsymbol{x} = (x_1, x_2) \quad dom(x_1) = dom(x_2) = \{\, 0, 1, 2, 3, 4, 5, 6, 7 \,\}$$

$$I(\boldsymbol{x}) \stackrel{\text{def}}{=} x_1 = 0 \wedge x_2 = 1$$

$$T(\boldsymbol{x}, \boldsymbol{x}') \stackrel{\text{def}}{=} x_1' = x_2 \wedge (x_2 \neq 3 \rightarrow x_2' = x_2 + 1) \wedge (x_2 = 3 \rightarrow x_2' = 0)$$

Transition graph fragment:



$$P(\boldsymbol{x}) \stackrel{\text{def}}{=} x_2 \leq 4$$

Improving Induction's Applicability

1. $I(\boldsymbol{x}) \models P(\boldsymbol{x})$ 2. $P(\boldsymbol{x}) \wedge T(\boldsymbol{x}, \boldsymbol{x}_1') \models P(\boldsymbol{x}_1')$

A couple of options:

- *Inductive strengthening*: find an inductive property $Q(\boldsymbol{x})$ such that $Q(\boldsymbol{x}) \models P(\boldsymbol{x})$

  General solution but often expensive

- *k-induction*: Consider more than one transition step at a time

  Easy to automate although fairly weak in its basic form

Improving Induction's Applicability

1. $I(\boldsymbol{x}) \models P(\boldsymbol{x})$ 2. $P(\boldsymbol{x}) \wedge T(\boldsymbol{x}, \boldsymbol{x}_1') \models P(\boldsymbol{x}_1')$

A couple of options:

- *Inductive strengthening*: find an inductive property $Q(\boldsymbol{x})$ such that $Q(\boldsymbol{x}) \models P(\boldsymbol{x})$

  General solution but often expensive

- *k-induction*: Consider more than one transition step at a time

  Easy to automate although fairly weak in its basic form

Improving Induction's Applicability

1. $I(\boldsymbol{x}) \models P(\boldsymbol{x})$ 2. $P(\boldsymbol{x}) \wedge T(\boldsymbol{x}, \boldsymbol{x}'_1) \models P(\boldsymbol{x}'_1)$

A couple of options:

- *Inductive strengthening*: find an inductive property $Q(\boldsymbol{x})$ such that $Q(\boldsymbol{x}) \models P(\boldsymbol{x})$

  General solution but often expensive

- *k-induction*: Consider more than one transition step at a time

  Easy to automate although fairly weak in its basic form

Improving Induction's Applicability

1. $I(\boldsymbol{x}) \models P(\boldsymbol{x})$ 2. $P(\boldsymbol{x}) \wedge T(\boldsymbol{x}, \boldsymbol{x}_1') \models P(\boldsymbol{x}_1')$

A couple of options:

- *Inductive strengthening*: find an inductive property $Q(\boldsymbol{x})$ such that $Q(\boldsymbol{x}) \models P(\boldsymbol{x})$

  General solution but often expensive

- *k-induction*: Consider more than one transition step at a time

  Easy to automate although fairly weak in its basic form

Improving Induction's Applicability

1. $I(\boldsymbol{x}) \models P(\boldsymbol{x})$ 2. $P(\boldsymbol{x}) \wedge T(\boldsymbol{x}, \boldsymbol{x}_1') \models P(\boldsymbol{x}_1')$

A couple of options:

- *Inductive strengthening*: find an inductive property $Q(\boldsymbol{x})$ such that $Q(\boldsymbol{x}) \models P(\boldsymbol{x})$

  General solution but often expensive

- *k-induction*: Consider more than one transition step at a time

  Easy to automate although fairly weak in its basic form

Improving Induction's Applicability

1. $I(\boldsymbol{x}) \models P(\boldsymbol{x})$ 2. $P(\boldsymbol{x}) \land T(\boldsymbol{x}, \boldsymbol{x}_1') \models P(\boldsymbol{x}_1')$

A couple of options:

- *Inductive strengthening*: find an inductive property $Q(\boldsymbol{x})$ such that $Q(\boldsymbol{x}) \models P(\boldsymbol{x})$

  General solution but often expensive

- *k-induction*: Consider more than one transition step at a time

  Easy to automate although fairly weak in its basic form

Inductive Strengthening

Find an inductive property $Q(\boldsymbol{x})$ such that $Q(\boldsymbol{x}) \models P(\boldsymbol{x})$

Inductive Strengthening

Find an inductive property $Q(\boldsymbol{x})$ such that $Q(\boldsymbol{x}) \models P(\boldsymbol{x})$

Example 1

$x_2 \leq 4$ is not inductive

However, $x_2 \leq 3$ is inductive and $x_2 \leq 3 \models x_2 \leq 4$

Inductive Strengthening

Find an inductive property $Q(\boldsymbol{x})$ such that $Q(\boldsymbol{x}) \models P(\boldsymbol{x})$

Theorem 6
If $Q(\boldsymbol{x})$ is inductive for $\mathbb{S}$ and $Q(\boldsymbol{x}) \models P(\boldsymbol{x})$ then $\mathbb{S} \models \square P(\boldsymbol{x})$

Inductive Strengthening

Find an inductive property $Q(\boldsymbol{x})$ such that $Q(\boldsymbol{x}) \models P(\boldsymbol{x})$

Theorem 6
If $Q(\boldsymbol{x})$ *is inductive for* $\mathbb{S}$ *and* $Q(\boldsymbol{x}) \models P(\boldsymbol{x})$ *then* $\mathbb{S} \models \Box P(\boldsymbol{x})$

There is actually a Q that works for every P!

Inductive Strengthening

Find an inductive property $Q(\boldsymbol{x})$ such that $Q(\boldsymbol{x}) \models P(\boldsymbol{x})$

Theorem 6
If $Q(\boldsymbol{x})$ is inductive for $\mathbb{S}$ and $Q(\boldsymbol{x}) \models P(\boldsymbol{x})$ then $\mathbb{S} \models \square P(\boldsymbol{x})$

Consider smallest k such that $R_{\leq k}(\boldsymbol{x}) \equiv R_{\leq k+1}(\boldsymbol{x})$

Inductive Strengthening

Find an inductive property $Q(\boldsymbol{x})$ such that $Q(\boldsymbol{x}) \models P(\boldsymbol{x})$

Theorem 6
If $Q(\boldsymbol{x})$ is inductive for $\mathbb{S}$ and $Q(\boldsymbol{x}) \models P(\boldsymbol{x})$ then $\mathbb{S} \models \Box P(\boldsymbol{x})$

Consider smallest k such that $R_{\leq k}(\boldsymbol{x}) \equiv R_{\leq k+1}(\boldsymbol{x})$

Theorem 7
$R_{\leq k}(\boldsymbol{x})$ is the strongest inductive invariant for $\mathbb{S}$:

1. *$R_{\leq k}(\boldsymbol{x})$ is inductive for $\mathbb{S}$*
2. *$P(\boldsymbol{x})$ is invariant for $\mathbb{S}$ iff $R_{\leq k}(\boldsymbol{x}) \models P(\boldsymbol{x})$*

Inductive Strengthening

Find an inductive property $Q(\boldsymbol{x})$ such that $Q(\boldsymbol{x}) \models P(\boldsymbol{x})$

Theorem 6
If $Q(\boldsymbol{x})$ is inductive for $\mathbb{S}$ and $Q(\boldsymbol{x}) \models P(\boldsymbol{x})$ then $\mathbb{S} \models \Box P(\boldsymbol{x})$

Consider smallest k such that $R_{\leq k}(\boldsymbol{x}) \equiv R_{\leq k+1}(\boldsymbol{x})$

Example 1

$k = 3$

$R_{\leq 3}(\boldsymbol{x}) \equiv (x_2 = 0 \wedge x_1 = 3) \vee (x_2 \in \{1, 2, 3\} \wedge x_1 = x_2 - 1)$

$R_{\leq 3}(\boldsymbol{x}) \models x \leq 4$, hence $x \leq 4$ is invariant

Issues with Strongest Inductive Invariant

Computing $R = R_{\leq k}(\boldsymbol{x})$ with $R_{\leq k}(\boldsymbol{x}) \equiv R_{\leq k+1}(\boldsymbol{x})$ is expensive

Boolean encodings of R (as a QBF or a OBDD) can be exponentially large in the size of $\boldsymbol{x}$

Good News:

Computing R to prove some P invariant is overkill in many cases

There are practically efficient methods that compute an inductive overapproximation $\overline{R}$ of R that entails P

However, such methods are beyond the scope of this course

Issues with Strongest Inductive Invariant

Computing $R = R_{\leq k}(\boldsymbol{x})$ with $R_{\leq k}(\boldsymbol{x}) \equiv R_{\leq k+1}(\boldsymbol{x})$ is expensive

Boolean encodings of R (as a QBF or a OBDD) can be exponentially large in the size of $\boldsymbol{x}$

Good News:

Computing R to prove some P invariant is overkill in many cases

There are practically efficient methods that compute an inductive overapproximation $\overline{R}$ of R that entails P

However, such methods are beyond the scope of this course

Issues with Strongest Inductive Invariant

Computing $R = R_{\leq k}(\boldsymbol{x})$ with $R_{\leq k}(\boldsymbol{x}) \equiv R_{\leq k+1}(\boldsymbol{x})$ is expensive

Boolean encodings of R (as a QBF or a OBDD) can be exponentially large in the size of $\boldsymbol{x}$

Good News:

Computing R to prove some P invariant is overkill in many cases

There are practically efficient methods that compute an inductive overapproximation $\overline{R}$ of R that entails P

However, such methods are beyond the scope of this course

Issues with Strongest Inductive Invariant

Computing $R = R_{\leq k}(\boldsymbol{x})$ with $R_{\leq k}(\boldsymbol{x}) \equiv R_{\leq k+1}(\boldsymbol{x})$ is expensive

Boolean encodings of R (as a QBF or a OBDD) can be exponentially large in the size of $\boldsymbol{x}$

Good News:

Computing R to prove some P invariant is overkill in many cases

There are practically efficient methods that compute an inductive overapproximation $\overline{R}$ of R that entails P

However, such methods are beyond the scope of this course

k-Induction

Consider more than one transition step at a time

Check that P is *k-inductive* for the system represented by I and T

If

$I(\mathbf{x}_0) \wedge T(\mathbf{x}_0, \mathbf{x}_1) \wedge \cdots \wedge T(\mathbf{x}_{i-1}, \mathbf{x}_i) \not\models P(\mathbf{x}_i)$ for some $i \geq 0$

then

P is not invariant

If, for some $k \geq 0$,

$I(\mathbf{x}_0) \wedge T(\mathbf{x}_0, \mathbf{x}_1) \wedge \cdots \wedge T(\mathbf{x}_{i-1}, \mathbf{x}_i) \models P(\mathbf{x}_i)$ for $i = 0, \ldots, k$

and

$P(\mathbf{x}_0) \wedge \cdots \wedge P(\mathbf{x}_k) \wedge T(\mathbf{x}_0, \mathbf{x}_1) \wedge \cdots \wedge T(\mathbf{x}_k, \mathbf{x}_{k+1}) \models P(\mathbf{x}_{k+1})$

then

P is k-inductive and hence invariant

k-Induction, Main Idea

Check that P is *k-inductive* for the system represented by I and T

If

$I(\boldsymbol{x}_0) \wedge T(\boldsymbol{x}_0, \boldsymbol{x}_1) \wedge \cdots \wedge T(\boldsymbol{x}_{i-1}, \boldsymbol{x}_i) \not\models P(\boldsymbol{x}_i)$ for some $i \geq 0$

then

P is not invariant

If, for some $k \geq 0$,

$I(\boldsymbol{x}_0) \wedge T(\boldsymbol{x}_0, \boldsymbol{x}_1) \wedge \cdots \wedge T(\boldsymbol{x}_{i-1}, \boldsymbol{x}_i) \models P(\boldsymbol{x}_i)$ for $i = 0, \ldots, k$

and

$P(\boldsymbol{x}_0) \wedge \cdots \wedge P(\boldsymbol{x}_k) \wedge T(\boldsymbol{x}_0, \boldsymbol{x}_1) \wedge \cdots \wedge T(\boldsymbol{x}_k, \boldsymbol{x}_{k+1}) \models P(\boldsymbol{x}_{k+1})$

then

P is k-inductive and hence invariant

k-Induction, Main Idea

Check that P is *k-inductive* for the system represented by I and T

If

$$I(\boldsymbol{x}_0) \wedge T(\boldsymbol{x}_0, \boldsymbol{x}_1) \wedge \cdots \wedge T(\boldsymbol{x}_{i-1}, \boldsymbol{x}_i) \not\models P(\boldsymbol{x}_i) \text{ for some } i \geq 0$$

then

P is not invariant

If, for some $k \geq 0$,

$$I(\boldsymbol{x}_0) \wedge T(\boldsymbol{x}_0, \boldsymbol{x}_1) \wedge \cdots \wedge T(\boldsymbol{x}_{i-1}, \boldsymbol{x}_i) \models P(\boldsymbol{x}_i) \text{ for } i = 0, \ldots, k$$

and

$$P(\boldsymbol{x}_0) \wedge \cdots \wedge P(\boldsymbol{x}_k) \wedge T(\boldsymbol{x}_0, \boldsymbol{x}_1) \wedge \cdots \wedge T(\boldsymbol{x}_k, \boldsymbol{x}_{k+1}) \models P(\boldsymbol{x}_{k+1})$$

then

P is k-inductive and hence invariant

k-Induction is Sufficient for Invariance

Theorem 7

Every state property P *that is* k*-inductive for some* $k \geq 0$ *for a transition system* $\mathbb{S}$ *is invariant for* $\mathbb{S}$*, i.e.,* $\mathbb{S} \models \square P$.

k-Induction is Sufficient for Invariance

Theorem 7

Every state property P that is k-inductive for some $k \geq 0$ for a transition system $\mathbb{S}$ is invariant for $\mathbb{S}$, i.e., $\mathbb{S} \models \Box P$.

Example 1

$P(\boldsymbol{x}) = x_2 \leq 4$ is not inductive but is 1-inductive:

$$x_{2,0} \leq 4 \wedge x_{2,1} \leq 4 \wedge T(\boldsymbol{x}_0, \boldsymbol{x}_1) \wedge T(\boldsymbol{x}_1, \boldsymbol{x}_2) \models x_{2,2} \leq 4$$

k-Induction is Sufficient for Invariance

Theorem 7
Every state property P that is k-inductive for some $k \geq 0$ for a transition system $\mathbb{S}$ is invariant for $\mathbb{S}$, i.e., $\mathbb{S} \models \square P$.

Example 1

$P(\boldsymbol{x}) = x_2 \leq 4$ is not inductive but is 1-inductive:

$$x_{2,0} \leq 4 \wedge x_{2,1} \leq 4 \wedge T(\boldsymbol{x}_0, \boldsymbol{x}_1) \wedge T(\boldsymbol{x}_1, \boldsymbol{x}_2) \models x_{2,2} \leq 4$$

Path $(1, 4) \to (4, 5)$ is not a counterexample for 1-induction

k-Induction is Sufficient for Invariance

Theorem 7

Every state property P that is k-inductive for some $k \geq 0$ for a transition system $\mathbb{S}$ is invariant for $\mathbb{S}$, i.e., $\mathbb{S} \models \Box P$.

Example 1

$P(\boldsymbol{x}) = x_2 \leq 4$ is not inductive but is 1-inductive:

$$x_{2,0} \leq 4 \wedge x_{2,1} \leq 4 \wedge T(\boldsymbol{x}_0, \boldsymbol{x}_1) \wedge T(\boldsymbol{x}_1, \boldsymbol{x}_2) \models x_{2,2} \leq 4$$

Path $(1, 4) \rightarrow (4, 5)$ is not a counterexample for 1-induction

$P(\boldsymbol{x}) = x_2 \leq 5$ is not 1-inductive but is 2-inductive

k-Induction is Sufficient for Invariance

Theorem 7
Every state property P that is k-inductive for some $k \geq 0$ for a transition system $\mathbb{S}$ is invariant for $\mathbb{S}$, i.e., $\mathbb{S} \models \Box P$.

Note:

- inductive = 0-inductive
- k-inductive implies $(k+1)$-inductive
- k-induction is not necessary for invariance:
  some invariants are not k-inductive for any k

Basic k-Induction

procedure *kInduction*(I, T, P)
input: formulas $I(\boldsymbol{x}), T(\boldsymbol{x}, \boldsymbol{x}'), F(\boldsymbol{x})$
output: “yes” or “no” output
begin
  $k := 0$; $\hat{T} := \top$; $\hat{P} := P(\boldsymbol{x}_0)$
  loop
    if $I(\boldsymbol{x}_0) \wedge \hat{T} \wedge \neg P(\boldsymbol{x}_k)$ is satisfiable **then return** “no”
    if $\hat{P} \wedge \hat{T} \wedge T(\boldsymbol{x}_k, \boldsymbol{x}_{k+1}) \wedge \neg P(\boldsymbol{x}_{k+1})$ is unsatisfiable **then return** “yes”
    $k := k + 1$
    $\hat{T} := \hat{T} \wedge T(\boldsymbol{x}_{k-1}, \boldsymbol{x}_k)$ // $\hat{T} = \top \wedge T(\boldsymbol{x}_0, \boldsymbol{x}_1) \wedge \cdots \wedge T(\boldsymbol{x}_{k-1}, \boldsymbol{x}_k)$
    $\hat{P} := \hat{P} \wedge P(\boldsymbol{x}_k)$ // $\hat{P} = P(\boldsymbol{x}_0) \wedge \cdots \wedge P(\boldsymbol{x}_k)$
  end loop
end

Basic k-Induction

procedure $kInduction(I, T, P)$
input: formulas $I(\boldsymbol{x}), T(\boldsymbol{x}, \boldsymbol{x}'), F(\boldsymbol{x})$
output: "yes" or "no" output
begin
$k := 0$; $\hat{T} := \top$; $\hat{P} := P(\boldsymbol{x}_0)$
loop
if $I(\boldsymbol{x}_0) \land \hat{T} \land \neg P(\boldsymbol{x}_k)$ is satisfiable **then return** "no"
if $\hat{P} \land \hat{T} \land T(\boldsymbol{x}_k, \boldsymbol{x}_{k+1}) \land \neg P(\boldsymbol{x}_{k+1})$ is unsatisfiable **then return** "yes"
$k := k + 1$
$\hat{T} := \hat{T} \land T(\boldsymbol{x}_{k-1}, \boldsymbol{x}_k)$ // $\hat{T} = \top \land T(\boldsymbol{x}_0, \boldsymbol{x}_1) \land \cdots \land T(\boldsymbol{x}_{k-1}, \boldsymbol{x}_k)$
$\hat{P} := \hat{P} \land P(\boldsymbol{x}_k)$ // $\hat{P} = P(\boldsymbol{x}_0) \land \cdots \land P(\boldsymbol{x}_k)$
end loop
end

Will diverge if P is not k-inductive for any k

Basic k-Induction with Termination Check

procedure *kInduction*(I, T, P)
input: formulas $I(\boldsymbol{x}), T(\boldsymbol{x}, \boldsymbol{x}'), P(\boldsymbol{x})$
output: "yes" or "no" output
begin
$k := 0$; $\hat{T} := \top$; $\hat{P} := P(\boldsymbol{x}_0)$
loop
if $I(\boldsymbol{x}_0) \wedge \hat{T} \wedge \neg P(\boldsymbol{x}_k)$ is satisfiable **then return** "no"
if $\hat{P} \wedge \hat{T} \wedge T(\boldsymbol{x}_k, \boldsymbol{x}_{k+1}) \wedge \neg P(\boldsymbol{x}_{k+1})$ is unsatisfiable **then return** "yes"
$k := k + 1$
$\hat{T} := \hat{T} \wedge T(\boldsymbol{x}_{k-1}, \boldsymbol{x}_k)$ // $\hat{T} = \top \wedge T(\boldsymbol{x}_0, \boldsymbol{x}_1) \wedge \cdots \wedge T(\boldsymbol{x}_{k-1}, \boldsymbol{x}_k)$
$\hat{P} := \hat{P} \wedge P(\boldsymbol{x}_k)$ // $\hat{P} = P(\boldsymbol{x}_0) \wedge \cdots \wedge P(\boldsymbol{x}_k)$
if $I(\boldsymbol{x}_0) \wedge \hat{T} \wedge \bigwedge_{0 \leq i < j \leq k} \boldsymbol{x}_i \neq \boldsymbol{x}_j$ is unsatisfiable **then return** "yes"
end loop
end

Basic k-Induction with Termination Check

procedure *kInduction*(I, T, P)
input: formulas $I(\boldsymbol{x}), T(\boldsymbol{x}, \boldsymbol{x}'), P(\boldsymbol{x})$
output: “yes” or “no” output
begin
$k := 0$; $\hat{T} := \top$; $\hat{P} := P(\boldsymbol{x}_0)$
loop
if $I(\boldsymbol{x}_0) \wedge \hat{T} \wedge \neg P(\boldsymbol{x}_k)$ is satisfiable **then return** “no”
if $\hat{P} \wedge \hat{T} \wedge T(\boldsymbol{x}_k, \boldsymbol{x}_{k+1}) \wedge \neg P(\boldsymbol{x}_{k+1})$ is unsatisfiable **then return** “yes”
$k := k + 1$
$\hat{T} := \hat{T} \wedge T(\boldsymbol{x}_{k-1}, \boldsymbol{x}_k)$ // $\hat{T} = \top \wedge T(\boldsymbol{x}_0, \boldsymbol{x}_1) \wedge \cdots \wedge T(\boldsymbol{x}_{k-1}, \boldsymbol{x}_k)$
$\hat{P} := \hat{P} \wedge P(\boldsymbol{x}_k)$ // $\hat{P} = P(\boldsymbol{x}_0) \wedge \cdots \wedge P(\boldsymbol{x}_k)$
if $I(\boldsymbol{x}_0) \wedge \hat{T} \wedge \bigwedge_{0 \leq i < j \leq k} \boldsymbol{x}_i \neq \boldsymbol{x}_j$ is unsatisfiable **then return** “yes”
end loop
end

Guaranteed to terminate with finite-state systems

Extensions of Model Checking

- There are model-checking algorithms for temporal properties other than reachability and invariance

- There is a general model-checking algorithm for arbitrary LTL properties

- There are extensions of model-checking techniques for infinite-state systems as well

- They will not be considered in this course

Extensions of Model Checking

- There are model-checking algorithms for temporal properties other than reachability and invariance
- There is a general model-checking algorithm for arbitrary LTL properties
- There are extensions of model-checking techniques for infinite-state systems as well
- They will not be considered in this course

Extensions of Model Checking

- There are model-checking algorithms for temporal properties other than reachability and invariance
- There is a general model-checking algorithm for arbitrary LTL properties
- There are extensions of model-checking techniques for infinite-state systems as well
- They will not be considered in this course

Extensions of Model Checking

- There are model-checking algorithms for temporal properties other than reachability and invariance
- There is a general model-checking algorithm for arbitrary LTL properties
- There are extensions of model-checking techniques for infinite-state systems as well
- They will not be considered in this course