universität
innsbruck

# SAT and SMT Solving


**Sarah Winkler**

KRDB
Department of Computer Science
Free University of Bozen-Bolzano


lecture 12
WS 2022

# Outline

- Summary of Last Week
- Bounded Model Checking for Verification
- Quantifiers for SMT

## Definitions

- theory consists of
  - signature $\Sigma$: set of function and predicate symbols
  - axioms $T$: set of sentences in first-order logic in which only function and predicate symbols of $\Sigma$ appear
- theory is stably infinite if every satisfiable quantifier-free formula has model with infinite carrier set
- theory $T$ is convex if $F \vDash_T \bigvee_{i=1}^{n} u_i = v_i$ implies $F \vDash_T u_i = v_i$ for some $1 \leqslant i \leqslant n$ $\forall$ quantifier-free conjunction $F$ and variables $u_i, v_i$

## Definition

theory combination $T_1 \oplus T_2$: signature $\Sigma_1 \cup \Sigma_2$ and axioms $\mathcal{A}_1 \cup \mathcal{A}_2$

## Assumptions

two stably infinite theories $T_1$, $T_2$ over signatures $\Sigma_1$, $\Sigma_2$ such that

- $\Sigma_1 \cap \Sigma_2 = \{=\}$
- $T_i$-satisfiability of quantifier-free $\Sigma_i$-formulas is decidable for $i \in \{1, 2\}$

## Nelson-Oppen Method: Nondeterministic Version

Input quantifier-free conjunction $\varphi$ in theory combination $T_1 \oplus T_2$
*Output* satisfiable or unsatisfiable

1. purification

$$\varphi \approx \varphi_1 \wedge \varphi_2 \quad \text{for } \Sigma_1\text{-formula } \varphi_1 \text{ and } \Sigma_2\text{-formula } \varphi_2$$

2. guess and check
   - $V$ is set of shared variables in $\varphi_1$ and $\varphi_2$
   - guess equivalence relation $E$ on $V$
   - arrangement $\alpha(V, E)$ is formula

$$\bigwedge_{x\,E\,y} x = y \quad \wedge \bigwedge_{\neg(x\,E\,y)} x \neq y$$

   - if $\varphi_1 \wedge \alpha(V, E)$ is $T_1$-satisfiable and $\varphi_2 \wedge \alpha(V, E)$ is $T_2$-satisfiable then return satisfiable else return unsatisfiable

## Nelson-Oppen Method: Deterministic Version

Input quantifier-free conjunction $\varphi$ in combination $T_1 \oplus T_2$ of convex theories $T_1$ and $T_2$

Output satisfiable or unsatisfiable

1. purification $\varphi \approx \varphi_1 \wedge \varphi_2$ for $\Sigma_1$-formula $\varphi_1$ and $\Sigma_2$-formula $\varphi_2$
2. $V$: set of shared variables in $\varphi_1$ and $\varphi_2$

   $E$: already discovered equalities between variables in $V$
3. test satisfiability of $\varphi_1 \wedge E$ (and add implied equations)
   - if $\varphi_1 \wedge E$ is $T_1$-unsatisfiable then return unsatisfiable
   - else add new implied equalities to $E$
4. test satisfiability of $\varphi_2 \wedge E$ (and add implied equations)
   - if $\varphi_2 \wedge E$ is $T_2$-unsatisfiable then return unsatisfiable
   - else add new implied equalities to $E$
5. if $E$ has been extended in steps 3 or 4 then go to step 2

   else return satisfiable

# Outline

- Summary of Last Week
- Bounded Model Checking for Verification
- Quantifiers for SMT

# Disastrous Software Bugs

## Ariane 5 Flight 501 (1996)

- destroyed 37 seconds after launch
- software for Ariane 4 for was reused
- software error: data conversion from 64-bit floating point to 16-bit integer caused arithmetic overflow
- cost: 370 million $

http://en.wikipedia.org/wiki/Ariane_5_Flight_501

## Mars Exploration Rover "Spirit" (2004)

- landed on January 4
- stopped communicating on January 21
- software error: stuck in reboot loop
- reboot failed because of flash memory failure, ultimate problem: too many files

http://en.wikipedia.org/wiki/Spirit_(rover)

## Mars Exploration Rover “Spirit” (2004)

- landed on January 4
- stopped communicating on January 21
- software error: stuck in reboot loop
- reboot failed because of flash memory failure, ultimate problem: too many files

http://en.wikipedia.org/wiki/Spirit_(rover)

## Heathrow Terminal 5 Opening (2008)


- baggage system collapsed on opening day
- 42,000 bags not shipped with their owners, 500 flights cancelled
- software was tested but did not work properly with real-world load
- cost 50 million £

http://www.zdnet.com/article/it-failure-at-heathrow-t5-what-really-happened

## Trading Glitch at Knight Capital (2012)

- bug in trading software resulted in 45 minutes of uncontrolled buys
- company did 11% of US trading that year
- software was run in invalid configuration
- 440 million $ lost

http://en.wikipedia.org/wiki/Knight_Capital_Group

## Trading Glitch at Knight Capital (2012)

- bug in trading software resulted in 45 minutes of uncontrolled buys
- company did 11% of US trading that year
- software was run in invalid configuration
- 440 million $ lost

http://en.wikipedia.org/wiki/Knight_Capital_Group


## Death in Self-Driving Car Crash (2018)

- person died in accident with Uber's self-driving car
- victim was wrongly classified by software as non-obstacle

http://www.siliconrepublic.com/companies/uber-bug-crash

## Software is Ubiquitous in Critical Systems

transport, energy, medicine, communication, finance, embedded systems, ...

**Software is Ubiquitous in Critical Systems**
transport, energy, medicine, communication, finance, embedded systems, ...

## How to Ensure Correctness of Software?

- testing
  - \+ cheap, simple
  - – checks desired result only for given set of testcases
- verification
  - \+ can prove automatically that system meets specification, i.e., desired output is delivered for all inputs
  - – more costly

## Software is Ubiquitous in Critical Systems

transport, energy, medicine, communication, finance, embedded systems, ...

## How to Ensure Correctness of Software?

- testing
  - \+ cheap, simple
  - − checks desired result only for given set of testcases
- verification
  - \+ can prove automatically that system meets specification, i.e., desired output is delivered for all inputs
  - − more costly

## Model Checking

- widely used verification approach to
  - find bugs in software and hardware
  - prove correctness of models

**Software is Ubiquitous in Critical Systems**
transport, energy, medicine, communication, finance, embedded systems, ...

## How to Ensure Correctness of Software?

- testing
  - + cheap, simple
  - – checks desired result only for given set of testcases
- verification
  - + can prove automatically that system meets specification, i.e., desired output is delivered for all inputs
  - – more costly

## Model Checking

- widely used verification approach to
  - find bugs in software and hardware
  - prove correctness of models
- Turing Award 2007 for Clarke, Emerson, and Sifakis

## Software is Ubiquitous in Critical Systems

transport, energy, medicine, communication, finance, embedded systems, ...

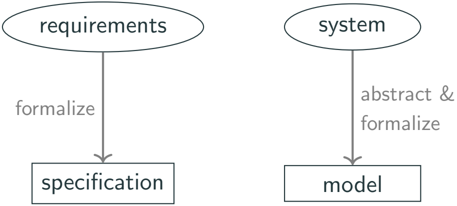
## How to Ensure Correctness of Software?

- testing
  - + cheap, simple
  - – checks desired result only for given set of testcases
- verification
  - + can prove automatically that system meets specification, i.e., desired output is delivered for all inputs
  - – more costly

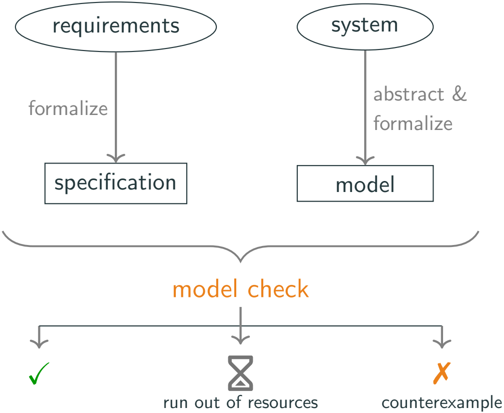
## Model Checking

- widely used verification approach to
  - find bugs in software and hardware
  - prove correctness of models
- Turing Award 2007 for Clarke, Emerson, and Sifakis
- bounded model checking can be reduced to SAT/SMT

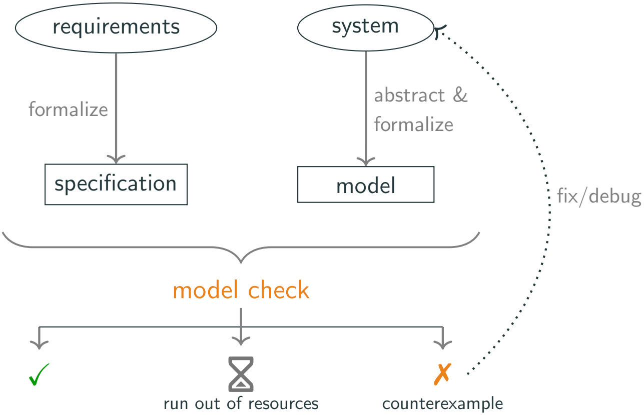
# Model Checking: Workflow



# Model Checking: Workflow



# Model Checking: Workflow



## Model Checking Example: Mutex (1)

- concurrent processes $P_0, P_1$ share some resource, access controlled by mutex

## Model Checking Example: Mutex (1)

- concurrent processes $P_0, P_1$ share some resource, access controlled by mutex
- program run by $P_0$, $P_1$ matches pattern

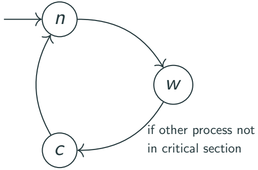
```
# non-critical section
while (other process critical) :
  wait ()
# critical section
# non-critical section
```

## Model Checking Example: Mutex (1)

- concurrent processes $P_0, P_1$ share some resource, access controlled by mutex
- program run by $P_0$, $P_1$ matches pattern

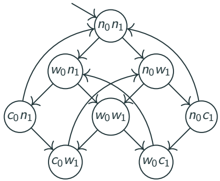
```
# non-critical section
while (other process critical) :
  wait ()
# critical section
# non-critical section
```

- process can be abstracted to model $\mathcal{M} = \langle S, R \rangle$ with states $S = \{n, w, c\}$ and transitions $R$:

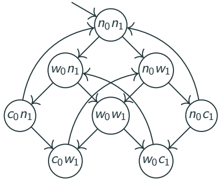


## Model Checking Example: Mutex (2)

- obtain model for 2 processes by product construction: write $s_0 s_1$ for $P_0$ being in state $s_0$ and $P_1$ in state $s_1$



## Model Checking Example: Mutex (2)

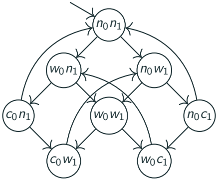



- obtain model for 2 processes by product construction: write $s_0 s_1$ for $P_0$ being in state $s_0$ and $P_1$ in state $s_1$

- desired properties:

| | |
|---|---|
| **safe:** | only one process is in its critical section at any time |
| **live:** | whenever any process wants to enter its critical section, it will eventually be permitted to do so |
| **non-blocking:** | a process can always request to enter its critical section |

# Model Checking Example: Mutex (2)




- obtain model for 2 processes by product construction: write $s_0 s_1$ for $P_0$ being in state $s_0$ and $P_1$ in state $s_1$

- desired properties:

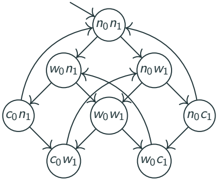
  **safe:** only one process is in its critical section at any time

  **live:** whenever any process wants to enter its critical section, it will eventually be permitted to do so

  **non-blocking:** a process can always request to enter its critical section

- how to formalize desired properties? temporal logic

## Model Checking Example: Mutex (2)




- obtain model for 2 processes by product construction: write $s_0 s_1$ for $P_0$ being in state $s_0$ and $P_1$ in state $s_1$

- desired properties:

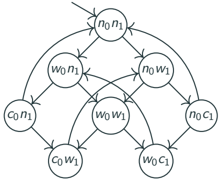
  **safe:** only one process is in its critical section at any time

  **live:** whenever any process wants to enter its critical section, it will eventually be permitted to do so

  **non-blocking:** a process can always request to enter its critical section

- how to formalize desired properties? temporal logic, e.g. LTL or CTL

## Model Checking Example: Mutex (2)




- obtain model for 2 processes by product construction:
  write $s_0 s_1$ for $P_0$ being in state $s_0$ and $P_1$ in state $s_1$

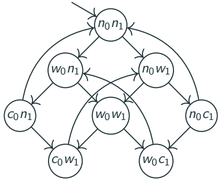
- desired properties:

| | |
|---|---|
| **safe:** | only one process is in its critical section at any time |
| **live:** | whenever any process wants to enter its critical section, it will eventually be permitted to do so |
| **non-blocking:** | a process can always request to enter its critical section |

- how to formalize desired properties? temporal logic, e.g. LTL or CTL

  **safe:** $\mathsf{G}\ \neg(c_0 \wedge c_1)$

## Model Checking Example: Mutex (2)




- obtain model for 2 processes by product construction:
  write $s_0 s_1$ for $P_0$ being in state $s_0$ and $P_1$ in state $s_1$

- desired properties:

**safe:** only one process is in its critical section at any time

**live:** whenever any process wants to enter its critical section, it will eventually be permitted to do so

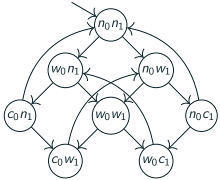
**non-blocking:** a process can always request to enter its critical section

- how to formalize desired properties? temporal logic, e.g. LTL or CTL

**safe:** $\mathsf{G}\ \neg(c_0 \wedge c_1)$

**live:** $\mathsf{G}\ (w_0 \rightarrow \mathsf{F}\ c_0)$

# Model Checking Example: Mutex (2)




- obtain model for 2 processes by product construction:
  write $s_0 s_1$ for $P_0$ being in state $s_0$ and $P_1$ in state $s_1$

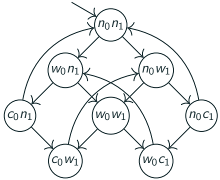
- desired properties:

  **safe:** only one process is in its critical section at any time
  **live:** whenever any process wants to enter its critical section, it will eventually be permitted to do so
  **non-blocking:** a process can always request to enter its critical section

- how to formalize desired properties? temporal logic, e.g. LTL or CTL

  **safe:** $\mathrm{G}\ \neg(c_0 \wedge c_1)$
  **live:** $\mathrm{G}\ (w_0 \rightarrow \mathrm{F}\ c_0)$
  **non-blocking:** $\mathrm{AG}\ (n_0 \rightarrow \mathrm{EX}\ w_0)$

# Model Checking Example: Mutex (2)




- obtain model for 2 processes by product construction: write $s_0 s_1$ for $P_0$ being in state $s_0$ and $P_1$ in state $s_1$
- desired properties:

| | |
|---|---|
| **safe:** | only one process is in its critical section at any time |
| **live:** | whenever any process wants to enter its critical section, it will eventually be permitted to do so |
| **non-blocking:** | a process can always request to enter its critical section |

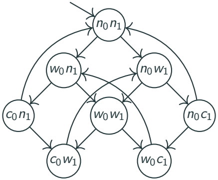
- how to formalize desired properties? temporal logic, e.g. LTL or CTL

**safe:** $\mathsf{G}\ \neg(c_0 \wedge c_1)$ ✓ as $c_0 c_1$ unreachable

**live:** $\mathsf{G}\ (w_0 \to \mathsf{F}\ c_0)$

**non-blocking:** $\mathsf{AG}\ (n_0 \to \mathsf{EX}\ w_0)$

# Model Checking Example: Mutex (2)




- obtain model for 2 processes by product construction: write $s_0 s_1$ for $P_0$ being in state $s_0$ and $P_1$ in state $s_1$
- desired properties:

| | |
|---|---|
| **safe:** | only one process is in its critical section at any time |
| **live:** | whenever any process wants to enter its critical section, it will eventually be permitted to do so |
| **non-blocking:** | a process can always request to enter its critical section |

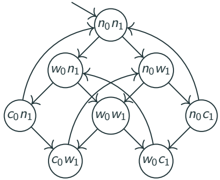
- how to formalize desired properties? temporal logic, e.g. LTL or CTL

**safe:** $\mathsf{G}\ \neg(c_0 \wedge c_1)$ ✓ as $c_0 c_1$ unreachable

**live:** $\mathsf{G}\ (w_0 \rightarrow \mathsf{F}\ c_0)$ ✗ e.g. 


**non-blocking:** $\mathsf{AG}\ (n_0 \rightarrow \mathsf{EX}\ w_0)$

# Model Checking Example: Mutex (2)




- obtain model for 2 processes by product construction: write $s_0 s_1$ for $P_0$ being in state $s_0$ and $P_1$ in state $s_1$

- desired properties:

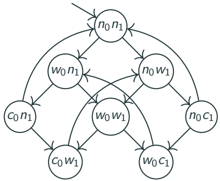
| | |
|---|---|
| **safe:** | only one process is in its critical section at any time |
| **live:** | whenever any process wants to enter its critical section, it will eventually be permitted to do so |
| **non-blocking:** | a process can always request to enter its critical section |

- how to formalize desired properties? temporal logic, e.g. LTL or CTL

| | | |
|---|---|---|
| **safe:** | $\mathsf{G}\ \neg(c_0 \wedge c_1)$ | ✓ as $c_0 c_1$ unreachable |
| **live:** | $\mathsf{G}\ (w_0 \to \mathsf{F}\ c_0)$ | ✗ e.g. $w_0 n_1 \to w_0 w_1 \to w_0 c_1 \to w_0 n_1 \to \cdots$ |
| **non-blocking:** | $\mathsf{AG}\ (n_0 \to \mathsf{EX}\ w_0)$ | ✓ |

## Model Checking Example: Mutex (2)

- obtain model for 2 processes by product construction:
  write $s_0 s_1$ for $P_0$ being in state $s_0$ and $P_1$ in state $s_1$




- desired properties:

**safe:** only one process is in its critical section at any time

**live:** whenever any process wants to enter its critical section, it will eventually be permitted to do so

**non-blocking:** a process can always request to enter its critical section

- how to formalize desired properties? temporal logic, e.g. LTL or CTL

**safe:** $\mathsf{G}\ \neg(c_0 \wedge c_1)$ ✓ as $c_0 c_1$ unreachable

**live:** $\mathsf{G}\ (w_0 \rightarrow \mathsf{F}\ c_0)$ ✗ e.g. $w_0 n_1 \rightarrow w_0 w_1 \rightarrow w_0 c_1 \rightarrow w_0 n_1 \rightarrow \cdots$

**non-blocking:** $\mathsf{AG}\ (n_0 \rightarrow \mathsf{EX}\ w_0)$ ✓

### Observation

model checking is feasible for this example because state space is finite and small

# Common Kinds of Properties

## Safety property

- "bad things don't happen"
- expressed as $\mathsf{G}\ \psi$, for some $\psi$ without temporal operators

# Common Kinds of Properties

## Safety property

- "bad things don't happen"
- expressed as $\mathsf{G}\ \psi$, for some $\psi$ without temporal operators

## Example

- safety properties
  - program never reaches an error state $\mathsf{G}(\neg \mathit{error})$

# Common Kinds of Properties

## Safety property

- "bad things don't happen"
- expressed as $\mathsf{G}\ \psi$, for some $\psi$ without temporal operators

## Example

- safety properties
  - program never reaches an error state $\mathsf{G}(\neg \mathit{error})$
  - programm does not violate access permissions $\mathsf{G}(\neg \mathit{violation})$

# Common Kinds of Properties

## Safety property

- "bad things don't happen"
- expressed as $\mathsf{G}\ \psi$, for some $\psi$ without temporal operators

## Example

- safety properties
  - program never reaches an error state $\mathsf{G}(\neg \mathit{error})$
  - programm does not violate access permissions $\mathsf{G}(\neg \mathit{violation})$
  - program never uses more than 1GB of RAM $\mathsf{G}(\mathit{mem} < 1\mathit{GB})$

# Common Kinds of Properties

**Safety property**

- "bad things don't happen"
- expressed as $\mathsf{G}\ \psi$, for some $\psi$ without temporal operators

**Liveness property**

- "good things happen eventually"
- expressed as $\mathsf{G}\ (\psi \to \mathsf{F}\chi)$, for some $\psi, \chi$ without temporal operators

**Example**

- safety properties
  - program never reaches an error state $\mathsf{G}(\neg \mathit{error})$
  - programm does not violate access permissions $\mathsf{G}(\neg \mathit{violation})$
  - program never uses more than 1GB of RAM $\mathsf{G}(\mathit{mem} < 1\mathit{GB})$

# Common Kinds of Properties

## Safety property

- "bad things don't happen"
- expressed as $\mathsf{G}\ \psi$, for some $\psi$ without temporal operators

## Liveness property

- "good things happen eventually"
- expressed as $\mathsf{G}\ (\psi \to \mathsf{F}\chi)$, for some $\psi, \chi$ without temporal operators

## Example

- safety properties
  - program never reaches an error state $\mathsf{G}(\neg \mathit{error})$
  - programm does not violate access permissions $\mathsf{G}(\neg \mathit{violation})$
  - program never uses more than 1GB of RAM $\mathsf{G}(\mathit{mem} < 1\mathit{GB})$
- liveness properties
  - every task is eventually processed $\mathsf{G}(\mathit{task\ created} \to \mathsf{F}\mathit{processed})$

# Common Kinds of Properties

## Safety property

- “bad things don’t happen”
- expressed as $\mathsf{G}\ \psi$, for some $\psi$ without temporal operators

## Liveness property

- “good things happen eventually”
- expressed as $\mathsf{G}\ (\psi \to \mathsf{F}\chi)$, for some $\psi, \chi$ without temporal operators

## Example

- safety properties
  - program never reaches an error state $\mathsf{G}(\neg \mathit{error})$
  - programm does not violate access permissions $\mathsf{G}(\neg \mathit{violation})$
  - program never uses more than 1GB of RAM $\mathsf{G}(\mathit{mem} < 1\mathit{GB})$
- liveness properties
  - every task is eventually processed $\mathsf{G}(\mathit{task\ created} \to \mathsf{F}\mathit{processed})$
  - the database is eventually consistent $\mathsf{G}(\mathit{change\ DB} \to \mathsf{F}\mathit{consistent})$

# Common Kinds of Properties

## Safety property

- "bad things don't happen"
- expressed as $\mathsf{G}\ \psi$, for some $\psi$ without temporal operators

## Liveness property

- "good things happen eventually"
- expressed as $\mathsf{G}\ (\psi \rightarrow \mathsf{F}\chi)$, for some $\psi, \chi$ without temporal operators

## Example

- safety properties
  - program never reaches an error state $\mathsf{G}(\neg \mathit{error})$
  - programm does not violate access permissions $\mathsf{G}(\neg \mathit{violation})$
  - program never uses more than 1GB of RAM $\mathsf{G}(\mathit{mem} < 1\mathit{GB})$
- liveness properties
  - every task is eventually processed $\mathsf{G}(\mathit{task\ created} \rightarrow \mathsf{F}\mathit{processed})$
  - the database is eventually consistent $\mathsf{G}(\mathit{change\ DB} \rightarrow \mathsf{F}\mathit{consistent})$
  - if user inputs *a*, program eventually does *b* $\mathsf{G}(a \rightarrow \mathsf{F}b)$

# Common Kinds of Properties

## Safety property

- "bad things don't happen"
- expressed as $\mathsf{G}\ \psi$, for some $\psi$ without temporal operators
- violated within finite number of steps

## Liveness property

- "good things happen eventually"
- expressed as $\mathsf{G}\ (\psi \to \mathsf{F}\chi)$, for some $\psi, \chi$ without temporal operators

## Example

- safety properties
  - program never reaches an error state $\mathsf{G}(\neg \mathit{error})$
  - programm does not violate access permissions $\mathsf{G}(\neg \mathit{violation})$
  - program never uses more than 1GB of RAM $\mathsf{G}(\mathit{mem} < 1\mathit{GB})$
- liveness properties
  - every task is eventually processed $\mathsf{G}(\mathit{task\ created} \to \mathsf{F}\mathit{processed})$
  - the database is eventually consistent $\mathsf{G}(\mathit{change\ DB} \to \mathsf{F}\mathit{consistent})$
  - if user inputs $a$, program eventually does $b$ $\mathsf{G}(a \to \mathsf{F}b)$

## Example: Can This Program Cause An Overflow?

```
void main() {
  int x = -1;
  int y = input();
  while (y<100) {
    y = y+x;
  }
}
```

## Example: Can This Program Cause An Overflow?

```
void main() {
  int x = -1;
  int y = input();
  while (y<100) {
    y = y+x;
  }
}
```

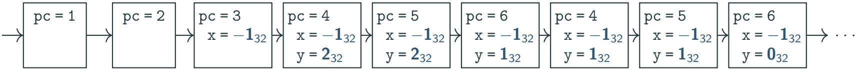
- model checking problem:
  - state consists of values for (x, y) + value of program counter pc

# Example: Can This Program Cause An Overflow?

```
1 void main() {
2   int x = -1;
3   int y = input();
4   while (y<100) {
5     y = y+x;
6   }
7 }
```

- model checking problem:
  - state consists of values for (x, y) + value of program counter pc

- (part of) model:

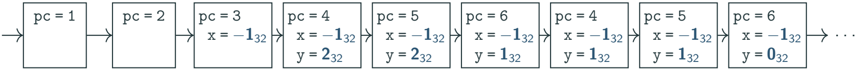


## Example: Can This Program Cause An Overflow?

```
1 void main() {
2   int x = -1;
3   int y = input();
4   while (y<100) {
5     y = y+x;
6   }
7 }
```

addition $x + y$ in line 5 does not over/underflow

- model checking problem:
  - state consists of values for (x, y) + value of program counter pc
  - safety property $\mathrm{G}\ ((x > \mathbf{0}_{32} \wedge x{+}y < y) \vee (x \leqslant \mathbf{0}_{32} \wedge x{+}y > y))$
- (part of) model:

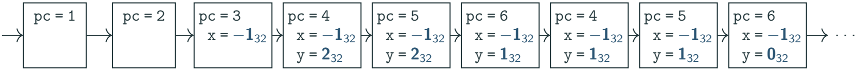


# Example: Can This Program Cause An Overflow?

```
1 void main() {
2   int x = -1;
3   int y = input();
4   while (y<100) {
5     y = y+x;
6   }
7 }
```

addition $x + y$ in line 5 does not over/underflow

- model checking problem:
  - state consists of values for (x, y) + value of program counter pc
  - safety property $\mathsf{G}\ ((x > \mathbf{0}_{32} \wedge x{+}y < y) \vee (x \leqslant \mathbf{0}_{32} \wedge x{+}y > y))$
- (part of) model:

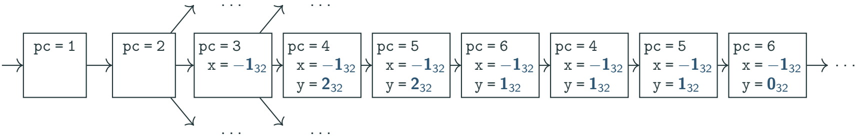


# Example: Can This Program Cause An Overflow?

```
1 void main() {
2   int x = -1;
3   int y = input();
4   while (y<100) {
5     y = y+x;
6   }
7 }
```

addition $x + y$ in line 5 does not over/underflow

- model checking problem:
  - state consists of values for (x, y) + value of program counter pc
  - safety property $\mathsf{G}\ ((x > \mathbf{0}_{32} \land x{+}y < y) \lor (x \leqslant \mathbf{0}_{32} \land x{+}y > y))$
- (part of) model:



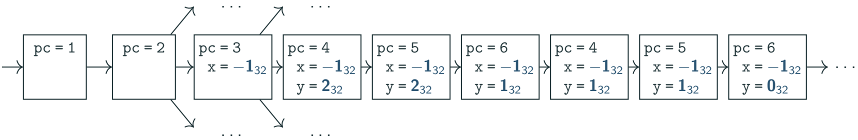

- but state space is very large: $(2^{32})^2 \cdot 7$ for bit width 32

# Example: Can This Program Cause An Overflow?

```
1 void main() {
2   int x = -1;
3   int y = input();
4   while (y<100) {
5     y = y+x;
6   }
7 }
```

addition $x + y$ in line 5 does not over/underflow

- model checking problem:
  - state consists of values for (x, y) + value of program counter pc
  - safety property $\mathsf{G}\ ((x > \mathbf{0}_{32} \wedge x{+}y < y) \vee (x \leqslant \mathbf{0}_{32} \wedge x{+}y > y))$
- (part of) model:




- but state space is very large: $(2^{32})^2 \cdot 7$ for bit width 32
- cannot check all possible values

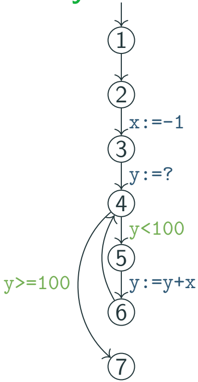
# Example: Can This Program Cause An Overflow? Next try.

```
void main() {
  int x = -1;
  int y = input();
  while (y<100) {
    y = y+x;
  }
}
```

# Example: Can This Program Cause An Overflow? Next try.

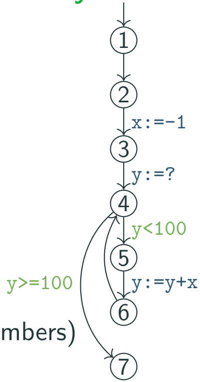
```
void main() {
  int x = -1;
  int y = input();
  while (y<100) {
    y = y+x;
  }
}
```

- construct program graph $G$



# Example: Can This Program Cause An Overflow? Next try.

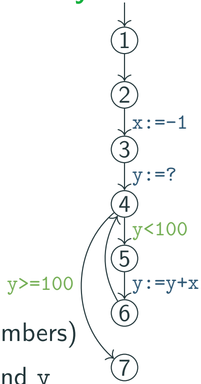
```
1 void main() {
2   int x = -1;
3   int y = input();
4   while (y<100) {
5     y = y+x;
6   }
7 }
```




- construct program graph $G$
- $\{1, \ldots, 7\}$ are possible values of program counter (line numbers)

## Example: Can This Program Cause An Overflow? Next try.

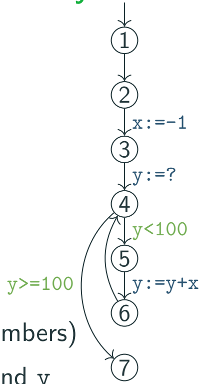
```
1 void main() {
2   int x = -1;
3   int y = input();
4   while (y<100) {
5     y = y+x;
6   }
7 }
```




- construct program graph $G$
- $\{1, \ldots, 7\}$ are possible values of program counter (line numbers)
- state is tuple $\langle \mathrm{pc}, x, y \rangle$ of values of program counter, `x`, and `y`

# Example: Can This Program Cause An Overflow? Next try.

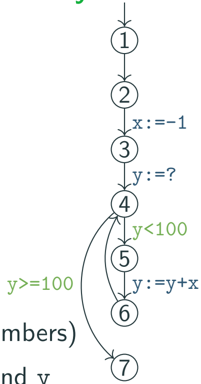
```
1 void main() {
2   int x = -1;
3   int y = input();
4   while (y<100) {
5     y = y+x;
6   }
7 }
```




- construct program graph $G$
- $\{1, \ldots, 7\}$ are possible values of program counter (line numbers)
- state is tuple $\langle \mathrm{pc}, x, y \rangle$ of values of program counter, x, and y
- state of form $\langle 1, \ldots, \ldots \rangle$ is initial state

# Example: Can This Program Cause An Overflow? Next try.

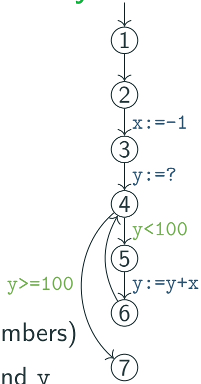
```
1 void main() {
2   int x = -1;
3   int y = input();
4   while (y<100) {
5     y = y+x;
6   }
7 }
```




- construct program graph $G$
- $\{1, \ldots, 7\}$ are possible values of program counter (line numbers)
- state is tuple $\langle \mathrm{pc}, x, y \rangle$ of values of program counter, `x`, and `y`
- state of form $\langle 1, \ldots, \ldots \rangle$ is initial state
- examples of state transitions according to $G$:
  - $\langle 4, -\mathbf{1}_{32}, \mathbf{10}_{32} \rangle \rightarrow \langle 5, -\mathbf{1}_{32}, \mathbf{10}_{32} \rangle$ is possible

## Example: Can This Program Cause An Overflow? Next try.

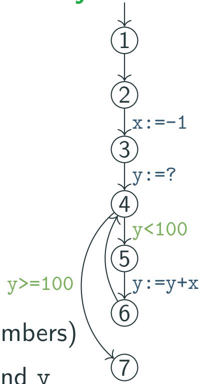
```
1 void main() {
2   int x = -1;
3   int y = input();
4   while (y<100) {
5     y = y+x;
6   }
7 }
```




- construct program graph $G$
- $\{1, \ldots, 7\}$ are possible values of program counter (line numbers)
- state is tuple $\langle \mathrm{pc}, x, y \rangle$ of values of program counter, `x`, and `y`
- state of form $\langle 1, \ldots, \ldots \rangle$ is initial state
- examples of state transitions according to $G$:
  - $\langle 4, -\mathbf{1}_{32}, \mathbf{10}_{32} \rangle \to \langle 5, -\mathbf{1}_{32}, \mathbf{10}_{32} \rangle$ is possible
  - $\langle 4, -\mathbf{1}_{32}, \mathbf{101}_{32} \rangle \to \langle 7, -\mathbf{1}_{32}, \mathbf{101}_{32} \rangle$ is possible

## Example: Can This Program Cause An Overflow? Next try.

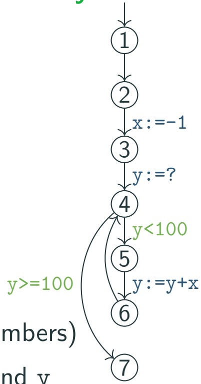
```
1 void main() {
2   int x = -1;
3   int y = input();
4   while (y<100) {
5     y = y+x;
6   }
7 }
```




- construct program graph $G$
- $\{1, \ldots, 7\}$ are possible values of program counter (line numbers)
- state is tuple $\langle \text{pc}, x, y \rangle$ of values of program counter, x, and y
- state of form $\langle 1, \ldots, \ldots \rangle$ is initial state
- examples of state transitions according to $G$:
  - $\langle 4, -\mathbf{1}_{32}, \mathbf{10}_{32} \rangle \to \langle 5, -\mathbf{1}_{32}, \mathbf{10}_{32} \rangle$ is possible
  - $\langle 4, -\mathbf{1}_{32}, \mathbf{101}_{32} \rangle \to \langle 7, -\mathbf{1}_{32}, \mathbf{101}_{32} \rangle$ is possible
  - $\langle 4, \mathbf{10}_{32}, \mathbf{101}_{32} \rangle \to \langle 5, \mathbf{10}_{32}, \mathbf{101}_{32} \rangle$ is not possible

# Example: Can This Program Cause An Overflow? Next try.

```
1 void main() {
2   int x = -1;
3   int y = input();
4   while (y<100) {
5     y = y+x;
6   }
7 }
```



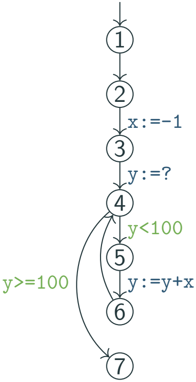

- construct program graph $G$
- $\{1, \ldots, 7\}$ are possible values of program counter (line numbers)
- state is tuple $\langle \mathrm{pc}, x, y \rangle$ of values of program counter, x, and y
- state of form $\langle 1, \ldots, \ldots \rangle$ is initial state
- examples of state transitions according to $G$:
  - $\langle 4, -\mathbf{1}_{32}, \mathbf{10}_{32} \rangle \to \langle 5, -\mathbf{1}_{32}, \mathbf{10}_{32} \rangle$ is possible
  - $\langle 4, -\mathbf{1}_{32}, \mathbf{101}_{32} \rangle \to \langle 7, -\mathbf{1}_{32}, \mathbf{101}_{32} \rangle$ is possible
  - $\langle 4, \mathbf{10}_{32}, \mathbf{101}_{32} \rangle \to \langle 5, \mathbf{10}_{32}, \mathbf{101}_{32} \rangle$ is not possible
  - $\langle 4, -\mathbf{1}_{32}, \mathbf{1}_{32} \rangle \to \langle 5, -\mathbf{1}_{32}, \mathbf{2}_{32} \rangle$ is not possible

# Idea

consider symbolic program executions with bounded length,
try to solve with SMT solver

## Example: Can This Program Cause An Overflow?

1. define predicates
   - $I(\langle pc, x, y \rangle) = (pc = 1)$ to characterize initial state

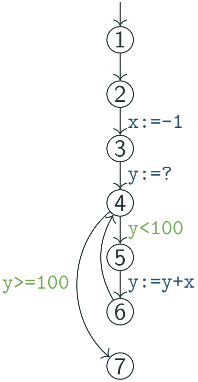


## Example: Can This Program Cause An Overflow?

1 define predicates

- $I(\langle pc, x, y\rangle) = (pc = 1)$ to characterize initial state
- to characterize possible state transitions:

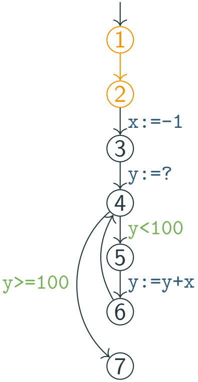
$$T(\langle pc, x, y\rangle, \langle pc', x', y'\rangle) =$$



## Example: Can This Program Cause An Overflow?

1. define predicates
   - $I(\langle pc, x, y\rangle) = (pc = 1)$ to characterize initial state
   - to characterize possible state transitions:

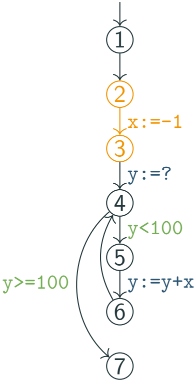
$$T(\langle pc, x, y\rangle, \langle pc', x', y'\rangle) =$$
$$(pc = 1 \land pc' = 2)$$



## Example: Can This Program Cause An Overflow?

1. define predicates
   - $I(\langle pc, x, y\rangle) = (pc = 1)$ to characterize initial state
   - to characterize possible state transitions:

$$T(\langle pc, x, y\rangle, \langle pc', x', y'\rangle) =$$
$$(pc = 1 \wedge pc' = 2) \vee (pc = 2 \wedge pc' = 3 \wedge x' = -1)$$

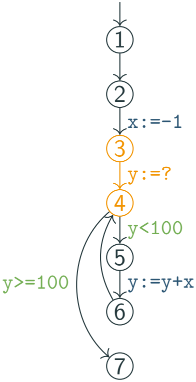


# Example: Can This Program Cause An Overflow?

1 define predicates

- $I(\langle pc, x, y\rangle) = (pc = 1)$ to characterize initial state
- to characterize possible state transitions:

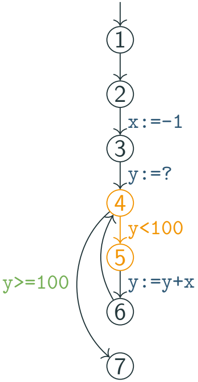
$$T(\langle pc, x, y\rangle, \langle pc', x', y'\rangle) =$$
$$(pc = 1 \wedge pc' = 2) \vee (pc = 2 \wedge pc' = 3 \wedge x' = -1) \vee$$
$$(pc = 3 \wedge pc' = 4 \wedge x = x')$$



## Example: Can This Program Cause An Overflow?

1. define predicates
   - $I(\langle pc, x, y\rangle) = (pc = 1)$ to characterize initial state
   - to characterize possible state transitions:

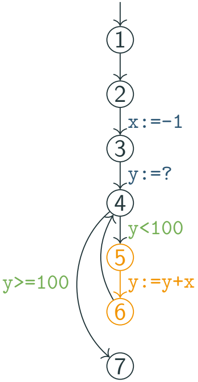
$$T(\langle pc, x, y\rangle, \langle pc', x', y'\rangle) =$$
$$(pc = 1 \wedge pc' = 2) \vee (pc = 2 \wedge pc' = 3 \wedge x' = -1) \vee$$
$$(pc = 3 \wedge pc' = 4 \wedge x = x') \vee$$
$$(pc = 4 \wedge pc' = 5 \wedge y < 100 \wedge x = x' \wedge y = y')$$



## Example: Can This Program Cause An Overflow?

1. define predicates
   - $I(\langle pc, x, y\rangle) = (pc = 1)$ to characterize initial state
   - to characterize possible state transitions:

$$
\begin{aligned}
&T(\langle pc, x, y\rangle, \langle pc', x', y'\rangle) = \\
&\quad (pc = 1 \wedge pc' = 2) \vee (pc = 2 \wedge pc' = 3 \wedge x' = -1) \vee \\
&\quad (pc = 3 \wedge pc' = 4 \wedge x = x') \vee \\
&\quad (pc = 4 \wedge pc' = 5 \wedge y < 100 \wedge x = x' \wedge y = y') \vee \\
&\quad (pc = 5 \wedge pc' = 6 \wedge y' = y + x \wedge x = x')
\end{aligned}
$$

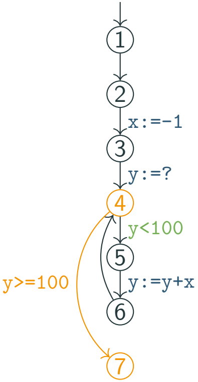


## Example: Can This Program Cause An Overflow?

1 define predicates

- $I(\langle pc, x, y \rangle) = (pc = 1)$ to characterize initial state
- to characterize possible state transitions:

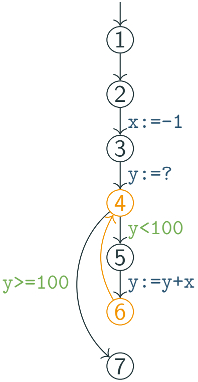
$$
\begin{aligned}
&T(\langle pc, x, y\rangle, \langle pc', x', y'\rangle) = \\
&\quad (pc = 1 \wedge pc' = 2) \vee (pc = 2 \wedge pc' = 3 \wedge x' = -1) \vee \\
&\quad (pc = 3 \wedge pc' = 4 \wedge x = x') \vee \\
&\quad (pc = 4 \wedge pc' = 5 \wedge y < 100 \wedge x = x' \wedge y = y') \vee \\
&\quad (pc = 5 \wedge pc' = 6 \wedge y' = y + x \wedge x = x') \vee \\
&\quad (pc = 4 \wedge pc' = 7 \wedge y \geqslant 100 \wedge x = x' \wedge y = y')
\end{aligned}
$$



# Example: Can This Program Cause An Overflow?

1. define predicates
   - $I(\langle pc, x, y\rangle) = (pc = 1)$ to characterize initial state
   - to characterize possible state transitions:

$$
\begin{aligned}
&T(\langle pc, x, y\rangle, \langle pc', x', y'\rangle) = \\
&\quad (pc = 1 \wedge pc' = 2) \vee (pc = 2 \wedge pc' = 3 \wedge x' = -1) \vee \\
&\quad (pc = 3 \wedge pc' = 4 \wedge x = x') \vee \\
&\quad (pc = 4 \wedge pc' = 5 \wedge y < 100 \wedge x = x' \wedge y = y') \vee \\
&\quad (pc = 5 \wedge pc' = 6 \wedge y' = y + x \wedge x = x') \vee \\
&\quad (pc = 4 \wedge pc' = 7 \wedge y \geqslant 100 \wedge x = x' \wedge y = y') \vee \\
&\quad (pc = 6 \wedge pc' = 4 \wedge x = x' \wedge y = y')
\end{aligned}
$$



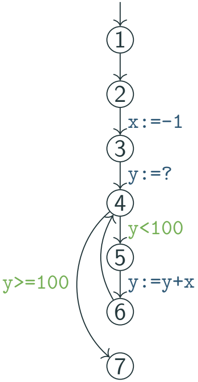
## Example: Can This Program Cause An Overflow?

1 define predicates

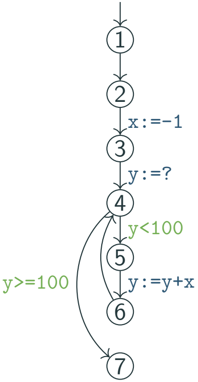
- $I(\langle pc, x, y\rangle) = (pc = 1)$ to characterize initial state
- to characterize possible state transitions:

$$
\begin{aligned}
&T(\langle pc, x, y\rangle, \langle pc', x', y'\rangle) = \\
&\quad (pc = 1 \wedge pc' = 2) \vee (pc = 2 \wedge pc' = 3 \wedge x' = -1) \vee \\
&\quad (pc = 3 \wedge pc' = 4 \wedge x = x') \vee \\
&\quad (pc = 4 \wedge pc' = 5 \wedge y < 100 \wedge x = x' \wedge y = y') \vee \\
&\quad (pc = 5 \wedge pc' = 6 \wedge y' = y + x \wedge x = x') \vee \\
&\quad (pc = 4 \wedge pc' = 7 \wedge y \geqslant 100 \wedge x = x' \wedge y = y') \vee \\
&\quad (pc = 6 \wedge pc' = 4 \wedge x = x' \wedge y = y')
\end{aligned}
$$

- $P(\langle pc, x, y\rangle) = (pc = 5) \wedge ((x > \mathbf{0}_{32} \wedge x + y \leqslant y) \vee (x \leqslant \mathbf{0}_{32} \wedge (y + x > y)))$



# Example: Can This Program Cause An Overflow?




1. define predicates
   - $I(\langle pc, x, y\rangle) = (pc = 1)$ to characterize initial state
   - to characterize possible state transitions:

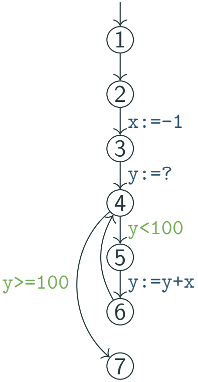
$$
\begin{aligned}
&T(\langle pc, x, y\rangle, \langle pc', x', y'\rangle) = \\
&\quad (pc = 1 \wedge pc' = 2) \vee (pc = 2 \wedge pc' = 3 \wedge x' = -1) \vee \\
&\quad (pc = 3 \wedge pc' = 4 \wedge x = x') \vee \\
&\quad (pc = 4 \wedge pc' = 5 \wedge y < 100 \wedge x = x' \wedge y = y') \vee \\
&\quad (pc = 5 \wedge pc' = 6 \wedge y' = y + x \wedge x = x') \vee \\
&\quad (pc = 4 \wedge pc' = 7 \wedge y \geqslant 100 \wedge x = x' \wedge y = y') \vee \\
&\quad (pc = 6 \wedge pc' = 4 \wedge x = x' \wedge y = y')
\end{aligned}
$$

   - $P(\langle pc, x, y\rangle) = (pc = 5) \wedge ((x > \mathbf{0}_{32} \wedge x + y \leqslant y) \vee (x \leqslant \mathbf{0}_{32} \wedge (y + x > y)))$

2. for states $s_0, \ldots, s_k$ formula $\varphi_k$ expresses overflow occurring within $k$ steps:

$$\varphi_k = I(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1}) \wedge \bigvee_{i=0}^{k} P(s_i)$$

## Example: Can This Program Cause An Overflow?




1. define predicates
   - $I(\langle pc, x, y\rangle) = (pc = 1)$ to characterize initial state
   - to characterize possible state transitions:

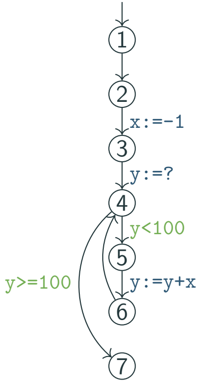
$$
\begin{aligned}
&T(\langle pc, x, y\rangle, \langle pc', x', y'\rangle) = \\
&\quad (pc = 1 \wedge pc' = 2) \vee (pc = 2 \wedge pc' = 3 \wedge x' = -1) \vee \\
&\quad (pc = 3 \wedge pc' = 4 \wedge x = x') \vee \\
&\quad (pc = 4 \wedge pc' = 5 \wedge y < 100 \wedge x = x' \wedge y = y') \vee \\
&\quad (pc = 5 \wedge pc' = 6 \wedge y' = y + x \wedge x = x') \vee \\
&\quad (pc = 4 \wedge pc' = 7 \wedge y \geqslant 100 \wedge x = x' \wedge y = y') \vee \\
&\quad (pc = 6 \wedge pc' = 4 \wedge x = x' \wedge y = y')
\end{aligned}
$$

   - $P(\langle pc, x, y\rangle) = (pc = 5) \wedge ((x > \mathbf{0}_{32} \wedge x + y \leqslant y) \vee (x \leqslant \mathbf{0}_{32} \wedge (y + x > y)))$

2. for states $s_0, \ldots, s_k$ formula $\varphi_k$ expresses overflow occurring within $k$ steps:

$$\varphi_k = I(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1}) \wedge \bigvee_{i=0}^{k} P(s_i)$$

## Example: Can This Program Cause An Overflow?




1. define predicates
   - $I(\langle pc, x, y\rangle) = (pc = 1)$ to characterize initial state
   - to characterize possible state transitions:

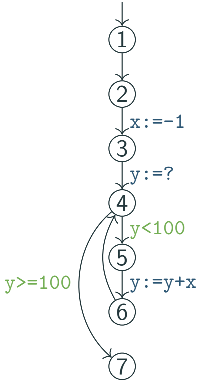
$$
\begin{aligned}
&T(\langle pc, x, y\rangle, \langle pc', x', y'\rangle) = \\
&\quad (pc = 1 \wedge pc' = 2) \vee (pc = 2 \wedge pc' = 3 \wedge x' = -1) \vee \\
&\quad (pc = 3 \wedge pc' = 4 \wedge x = x') \vee \\
&\quad (pc = 4 \wedge pc' = 5 \wedge y < 100 \wedge x = x' \wedge y = y') \vee \\
&\quad (pc = 5 \wedge pc' = 6 \wedge y' = y + x \wedge x = x') \vee \\
&\quad (pc = 4 \wedge pc' = 7 \wedge y \geqslant 100 \wedge x = x' \wedge y = y') \vee \\
&\quad (pc = 6 \wedge pc' = 4 \wedge x = x' \wedge y = y')
\end{aligned}
$$

   - $P(\langle pc, x, y\rangle) = (pc = 5) \wedge ((x > \mathbf{0}_{32} \wedge x + y \leqslant y) \vee (x \leqslant \mathbf{0}_{32} \wedge (y + x > y)))$

2. for states $s_0, \ldots, s_k$ formula $\varphi_k$ expresses overflow occurring within $k$ steps:

$$\varphi_k = I(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1}) \wedge \bigvee_{i=0}^{k} P(s_i)$$

## Example: Can This Program Cause An Overflow?




1. define predicates
   - $I(\langle pc, x, y\rangle) = (pc = 1)$ to characterize initial state
   - to characterize possible state transitions:

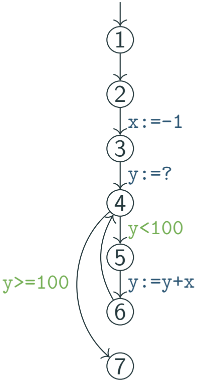
$$\begin{aligned}
&T(\langle pc, x, y\rangle, \langle pc', x', y'\rangle) = \\
&\quad (pc = 1 \wedge pc' = 2) \vee (pc = 2 \wedge pc' = 3 \wedge x' = -1) \vee \\
&\quad (pc = 3 \wedge pc' = 4 \wedge x = x') \vee \\
&\quad (pc = 4 \wedge pc' = 5 \wedge y < 100 \wedge x = x' \wedge y = y') \vee \\
&\quad (pc = 5 \wedge pc' = 6 \wedge y' = y + x \wedge x = x') \vee \\
&\quad (pc = 4 \wedge pc' = 7 \wedge y \geqslant 100 \wedge x = x' \wedge y = y') \vee \\
&\quad (pc = 6 \wedge pc' = 4 \wedge x = x' \wedge y = y')
\end{aligned}$$

   - $P(\langle pc, x, y\rangle) = (pc = 5) \wedge ((x > \mathbf{0}_{32} \wedge x + y \leqslant y) \vee (x \leqslant \mathbf{0}_{32} \wedge (y + x > y)))$

2. for states $s_0, \ldots, s_k$ formula $\varphi_k$ expresses overflow occurring within $k$ steps:

$$\varphi_k = I(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1}) \wedge \bigvee_{i=0}^{k} P(s_i)$$

# Example: Can This Program Cause An Overflow?




1. define predicates
   - $I(\langle pc, x, y\rangle) = (pc = 1)$ to characterize initial state
   - to characterize possible state transitions:

$$
\begin{aligned}
&T(\langle pc, x, y\rangle, \langle pc', x', y'\rangle) = \\
&\quad (pc = 1 \wedge pc' = 2) \vee (pc = 2 \wedge pc' = 3 \wedge x' = -1) \vee \\
&\quad (pc = 3 \wedge pc' = 4 \wedge x = x') \vee \\
&\quad (pc = 4 \wedge pc' = 5 \wedge y < 100 \wedge x = x' \wedge y = y') \vee \\
&\quad (pc = 5 \wedge pc' = 6 \wedge y' = y + x \wedge x = x') \vee \\
&\quad (pc = 4 \wedge pc' = 7 \wedge y \geqslant 100 \wedge x = x' \wedge y = y') \vee \\
&\quad (pc = 6 \wedge pc' = 4 \wedge x = x' \wedge y = y')
\end{aligned}
$$

   - $P(\langle pc, x, y\rangle) = (pc = 5) \wedge ((x > \mathbf{0}_{32} \wedge x + y \leqslant y) \vee (x \leqslant \mathbf{0}_{32} \wedge (y + x > y)))$

2. for states $s_0, \ldots, s_k$ formula $\varphi_k$ expresses overflow occurring within $k$ steps:

$$\varphi_k = I(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1}) \wedge \bigvee_{i=0}^{k} P(s_i)$$

3. if $\varphi_k$ satisfiable then overflow can occur within $k$ steps, e.g. for $k = 5$

# Bounded Model Checking

- find counterexamples to desired property of transition system (bugs)
- counterexamples are bounded in size

## Bounded Model Checking

- find counterexamples to desired property of transition system (bugs)
- counterexamples are bounded in size

## Definition (Transition System)

transition system $\mathcal{T} = (S, \rightarrow, S_0, L)$ where

- $S$ is set of states
- $\rightarrow \subseteq S \times S$ is transition relation
- $S_0 \subseteq S$ is set of initial states
- $A$ is a set of propositional atoms
- $L : S \rightarrow 2^A$ is labeling function associating state with subset of $A$

## Bounded Model Checking

- find counterexamples to desired property of transition system (bugs)
- counterexamples are bounded in size

## Definition (Transition System)

transition system $\mathcal{T} = (S, \rightarrow, S_0, L)$ where

- $S$ is set of states
- $\rightarrow \subseteq S \times S$ is transition relation
- $S_0 \subseteq S$ is set of initial states
- $A$ is a set of propositional atoms
- $L : S \rightarrow 2^A$ is labeling function associating state with subset of $A$

## Remark

$S$ and $A$ may be (countably) infinite

# Bounded Model Checking: Safety Properties

## Idea

given transition system and property $\mathsf{G}\ \psi$, look for counterexamples in $\leqslant k$ steps

$$\to s_0 \to s_1 \to s_2 \to \cdots \to s_k$$

$$\neg\psi \quad \vee \quad \neg\psi \quad \vee \quad \neg\psi \quad \vee \quad \cdots \quad \vee \quad \neg\psi$$

# Bounded Model Checking: Safety Properties

## Idea

given transition system and property G $\psi$, look for counterexamples in $\leqslant k$ steps

$$\rightarrow s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \cdots \rightarrow s_k$$

$$\neg\psi \;\vee\; \neg\psi \;\vee\; \neg\psi \;\vee\; \cdots \;\vee\; \neg\psi$$

## SAT/SMT Encoding

given transition system $\mathcal{T}$ and safety property G $\psi$

- use encoding $\langle s \rangle$ of state $s \in S$ by set of SAT/SMT variables

# Bounded Model Checking: Safety Properties

## Idea

given transition system and property G $\psi$, look for counterexamples in $\leqslant k$ steps

$$\to s_0 \to s_1 \to s_2 \to \cdots \to s_k$$

$$\neg\psi \;\lor\; \neg\psi \;\lor\; \neg\psi \;\lor\; \cdots \;\lor\; \neg\psi$$

## SAT/SMT Encoding

given transition system $\mathcal{T}$ and safety property G $\psi$

- use encoding $\langle s \rangle$ of state $s \in S$ by set of SAT/SMT variables
- use predicates
  - $I$ for initial states such that use $I(\langle s \rangle)$ is true iff $s \in S_0$

# Bounded Model Checking: Safety Properties

## Idea

given transition system and property G $\psi$, look for counterexamples in $\leqslant k$ steps

$$\to s_0 \to s_1 \to s_2 \to \cdots \to s_k$$

$$\neg\psi \;\vee\; \neg\psi \;\vee\; \neg\psi \;\vee\; \cdots \;\vee\; \neg\psi$$

## SAT/SMT Encoding

given transition system $\mathcal{T}$ and safety property G $\psi$

- use encoding $\langle s \rangle$ of state $s \in S$ by set of SAT/SMT variables
- use predicates
  - $I$ for initial states such that use $I(\langle s \rangle)$ is true iff $s \in S_0$
  - $T$ for transitions such that $T(\langle s \rangle, \langle s' \rangle)$ is true iff $s \to s'$ in $\mathcal{T}$

# Bounded Model Checking: Safety Properties

## Idea

given transition system and property G $\psi$, look for counterexamples in $\leqslant k$ steps

$$\to s_0 \to s_1 \to s_2 \to \cdots \to s_k$$

$$\neg\psi \;\vee\; \neg\psi \;\vee\; \neg\psi \;\vee\; \cdots \;\vee\; \neg\psi$$

## SAT/SMT Encoding

given transition system $\mathcal{T}$ and safety property G $\psi$

- use encoding $\langle s \rangle$ of state $s \in S$ by set of SAT/SMT variables
- use predicates
  - $I$ for initial states such that use $I(\langle s \rangle)$ is true iff $s \in S_0$
  - $T$ for transitions such that $T(\langle s \rangle, \langle s' \rangle)$ is true iff $s \to s'$ in $\mathcal{T}$
  - $P$ such that $P(\langle s \rangle)$ is true iff $\psi$ holds in $s$

# Bounded Model Checking: Safety Properties

## Idea

given transition system and property G $\psi$, look for counterexamples in $\leqslant k$ steps

$$\to s_0 \to s_1 \to s_2 \to \cdots \to s_k$$

$$\neg\psi \;\vee\; \neg\psi \;\vee\; \neg\psi \;\vee\; \cdots \;\vee\; \neg\psi$$

## SAT/SMT Encoding

given transition system $\mathcal{T}$ and safety property G $\psi$

- use encoding $\langle s \rangle$ of state $s \in S$ by set of SAT/SMT variables
- use predicates
  - $I$ for initial states such that use $I(\langle s \rangle)$ is true iff $s \in S_0$
  - $T$ for transitions such that $T(\langle s \rangle, \langle s' \rangle)$ is true iff $s \to s'$ in $\mathcal{T}$
  - $P$ such that $P(\langle s \rangle)$ is true iff $\psi$ holds in $s$
- use different fresh variables for $k+1$ states $\langle s_0 \rangle, \ldots, \langle s_k \rangle$

# Bounded Model Checking: Safety Properties

## Idea

given transition system and property G $\psi$, look for counterexamples in $\leqslant k$ steps

$$\to (s_0) \to (s_1) \to (s_2) \to \cdots \to (s_k)$$

$$\neg\psi \;\vee\; \neg\psi \;\vee\; \neg\psi \;\vee\; \cdots \;\vee\; \neg\psi$$

## SAT/SMT Encoding

given transition system $\mathcal{T}$ and safety property G $\psi$

- use encoding $\langle s \rangle$ of state $s \in S$ by set of SAT/SMT variables
- use predicates
  - $I$ for initial states such that use $I(\langle s \rangle)$ is true iff $s \in S_0$
  - $T$ for transitions such that $T(\langle s \rangle, \langle s' \rangle)$ is true iff $s \to s'$ in $\mathcal{T}$
  - $P$ such that $P(\langle s \rangle)$ is true iff $\psi$ holds in $s$
- use different fresh variables for $k + 1$ states $\langle s_0 \rangle, \dots, \langle s_k \rangle$
- check satisfiability of

$$I(\langle s_0 \rangle) \wedge \bigwedge_{i=0}^{k-1} T(\langle s_i \rangle, \langle s_{i+1} \rangle) \wedge \bigvee_{i=0}^{k} \neg P(\langle s_i \rangle)$$

# Bounded Model Checking: Safety Properties

## Idea

given transition system and property G $\psi$, look for counterexamples in $\leqslant k$ steps

$$\to s_0 \to s_1 \to s_2 \to \cdots \to s_k$$

$$\neg\psi \;\lor\; \neg\psi \;\lor\; \neg\psi \;\lor\; \cdots \;\lor\; \neg\psi$$

## SAT/SMT Encoding

given transition system $\mathcal{T}$ and safety property G $\psi$

- use encoding $\langle s \rangle$ of state $s \in S$ by set of SAT/SMT variables
- use predicates
  - $I$ for initial states such that use $I(\langle s \rangle)$ is true iff $s \in S_0$
  - $T$ for transitions such that $T(\langle s \rangle, \langle s' \rangle)$ is true iff $s \to s'$ in $\mathcal{T}$
  - $P$ such that $P(\langle s \rangle)$ is true iff $\psi$ holds in $s$
- use different fresh variables for $k+1$ states $\langle s_0 \rangle, \ldots, \langle s_k \rangle$
- check satisfiability of

$$I(\langle s_0 \rangle) \wedge \bigwedge_{i=0}^{k-1} T(\langle s_i \rangle, \langle s_{i+1} \rangle) \wedge \bigvee_{i=0}^{k} \neg P(\langle s_i \rangle)$$

# Bounded Model Checking: Safety Properties

## Idea

given transition system and property G $\psi$, look for counterexamples in $\leqslant k$ steps

$$\to s_0 \to s_1 \to s_2 \to \cdots \to s_k$$

$$\neg\psi \;\lor\; \neg\psi \;\lor\; \neg\psi \;\lor\; \cdots \;\lor\; \neg\psi$$

## SAT/SMT Encoding

given transition system $\mathcal{T}$ and safety property G $\psi$

- use encoding $\langle s\rangle$ of state $s \in S$ by set of SAT/SMT variables
- use predicates
  - $I$ for initial states such that use $I(\langle s\rangle)$ is true iff $s \in S_0$
  - $T$ for transitions such that $T(\langle s\rangle, \langle s'\rangle)$ is true iff $s \to s'$ in $\mathcal{T}$
  - $P$ such that $P(\langle s\rangle)$ is true iff $\psi$ holds in $s$
- use different fresh variables for $k+1$ states $\langle s_0\rangle, \dots, \langle s_k\rangle$
- check satisfiability of

$$I(\langle s_0\rangle) \land \bigwedge_{i=0}^{k-1} T(\langle s_i\rangle, \langle s_{i+1}\rangle) \land \bigvee_{i=0}^{k} \neg P(\langle s_i\rangle)$$

# Bounded Model Checking: Liveness Properties

## Idea

- counterexample to liveness property $\mathsf{G}\,(\psi \rightarrow \mathsf{F}\chi)$ requires infinite path

# Bounded Model Checking: Liveness Properties

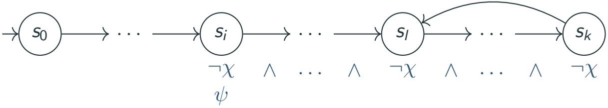
## Idea

- counterexample to liveness property $\mathsf{G}\,(\psi \to \mathsf{F}\chi)$ requires infinite path
- look for counterexamples in $\leqslant k$ steps of lasso shape:

$$\begin{tikzcd} {} \arrow[r] & s_0 \arrow[r] & \cdots \arrow[r] & s_i \arrow[r] & \cdots \arrow[r] & s_l \arrow[r] & \cdots \arrow[r] & s_k \arrow[ll, bend right] \end{tikzcd}$$

## Idea

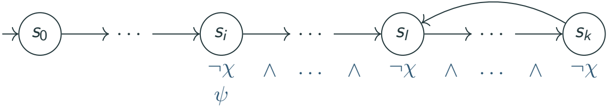
- counterexample to liveness property G $(\psi \to \mathsf{F}\chi)$ requires infinite path
- look for counterexamples in $\leqslant k$ steps of lasso shape:



# Bounded Model Checking: Liveness Properties

## Idea

- counterexample to liveness property G $(\psi \to \mathsf{F}\chi)$ requires infinite path
- look for counterexamples in $\leqslant k$ steps of lasso shape:




## SAT/SMT Encoding

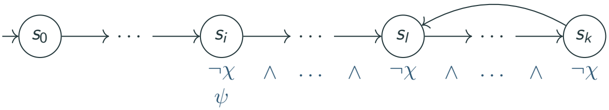
given transition system $\mathcal{T}$ and liveness property G $(\psi \to \mathsf{F}\chi)$

- use encoding of states, predicates $I$ and $T$ as for safety properties

# Bounded Model Checking: Liveness Properties

## Idea

- counterexample to liveness property G $(\psi \to \mathsf{F}\chi)$ requires infinite path
- look for counterexamples in $\leqslant k$ steps of lasso shape:




## SAT/SMT Encoding

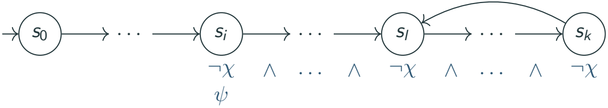
given transition system $\mathcal{T}$ and liveness property G $(\psi \to \mathsf{F}\chi)$

- use encoding of states, predicates $I$ and $T$ as for safety properties
- predicate $P$ such that $P(\langle s \rangle)$ is true iff $\psi$ holds in $s$

# Bounded Model Checking: Liveness Properties

## Idea

- counterexample to liveness property G $(\psi \to \mathsf{F}\chi)$ requires infinite path
- look for counterexamples in $\leqslant k$ steps of lasso shape:




## SAT/SMT Encoding

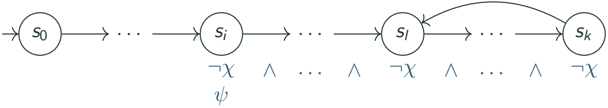
given transition system $\mathcal{T}$ and liveness property G $(\psi \to \mathsf{F}\chi)$

- use encoding of states, predicates $I$ and $T$ as for safety properties
- predicate $P$ such that $P(\langle s \rangle)$ is true iff $\psi$ holds in $s$
- predicate $C$ such that $C(\langle s \rangle)$ is true iff $\chi$ holds in $s$

# Bounded Model Checking: Liveness Properties

## Idea

- counterexample to liveness property $\mathsf{G}\,(\psi \to \mathsf{F}\chi)$ requires infinite path
- look for counterexamples in $\leqslant k$ steps of lasso shape:




## SAT/SMT Encoding

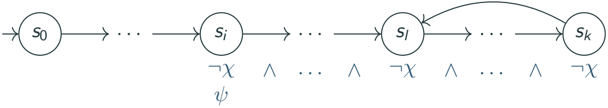
given transition system $\mathcal{T}$ and liveness property $\mathsf{G}\,(\psi \to \mathsf{F}\chi)$

- use encoding of states, predicates $I$ and $T$ as for safety properties
- predicate $P$ such that $P(\langle s \rangle)$ is true iff $\psi$ holds in $s$
- predicate $C$ such that $C(\langle s \rangle)$ is true iff $\chi$ holds in $s$
- check satisfiability of

$$I(\langle s_0 \rangle) \wedge \bigwedge_{i=0}^{k-1} T(\langle s_i \rangle, \langle s_{i+1} \rangle) \wedge \bigvee_{i=0}^{k} \left( P(\langle s_i \rangle) \wedge \bigwedge_{j=i}^{k} \neg C(\langle s_j \rangle) \wedge \bigvee_{l=i}^{k} T(\langle s_k \rangle, \langle s_l \rangle) \right)$$

## Idea

- counterexample to liveness property $\mathsf{G}\,(\psi \to \mathsf{F}\chi)$ requires infinite path
- look for counterexamples in $\leqslant k$ steps of lasso shape:




## SAT/SMT Encoding

given transition system $\mathcal{T}$ and liveness property $\mathsf{G}\,(\psi \to \mathsf{F}\chi)$

- use encoding of states, predicates $I$ and $T$ as for safety properties
- predicate $P$ such that $P(\langle s \rangle)$ is true iff $\psi$ holds in $s$
- predicate $C$ such that $C(\langle s \rangle)$ is true iff $\chi$ holds in $s$
- check satisfiability of

$$I(\langle s_0 \rangle) \wedge \bigwedge_{i=0}^{k-1} T(\langle s_i \rangle, \langle s_{i+1} \rangle) \wedge \bigvee_{i=0}^{k} \left( P(\langle s_i \rangle) \wedge \bigwedge_{j=i}^{k} \neg C(\langle s_j \rangle) \wedge \bigvee_{l=i}^{k} T(\langle s_k \rangle, \langle s_l \rangle) \right)$$

# Bounded Model Checking: Liveness Properties

## Idea

- counterexample to liveness property $\mathsf{G}\,(\psi \to \mathsf{F}\chi)$ requires infinite path
- look for counterexamples in $\leqslant k$ steps of lasso shape:

$$\begin{tikzcd}
{} \arrow[r] & s_0 \arrow[r] & \cdots \arrow[r] & s_i \arrow[r] & \cdots \arrow[r] & s_l \arrow[r] & \cdots \arrow[r] & s_k \arrow[ll, bend right] \\
 & & & \neg\chi & \wedge \quad \cdots \quad \wedge & \neg\chi & \wedge \quad \cdots \quad \wedge & \neg\chi \\
 & & & \psi & & & &
\end{tikzcd}$$

## SAT/SMT Encoding

given transition system $\mathcal{T}$ and liveness property $\mathsf{G}\,(\psi \to \mathsf{F}\chi)$

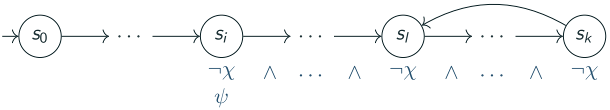
- use encoding of states, predicates $I$ and $T$ as for safety properties
- predicate $P$ such that $P(\langle s\rangle)$ is true iff $\psi$ holds in $s$
- predicate $C$ such that $C(\langle s\rangle)$ is true iff $\chi$ holds in $s$
- check satisfiability of

$$I(\langle s_0\rangle) \wedge \bigwedge_{i=0}^{k-1} T(\langle s_i\rangle, \langle s_{i+1}\rangle) \wedge \bigvee_{i=0}^{k} \left( P(\langle s_i\rangle) \wedge \bigwedge_{j=i}^{k} \neg C(\langle s_i\rangle) \wedge \bigvee_{l=i}^{k} T(\langle s_k\rangle, \langle s_l\rangle) \right)$$

# Bounded Model Checking: Liveness Properties

## Idea

- counterexample to liveness property $\mathsf{G}\,(\psi \to \mathsf{F}\chi)$ requires infinite path
- look for counterexamples in $\leqslant k$ steps of lasso shape:




## SAT/SMT Encoding

given transition system $\mathcal{T}$ and liveness property $\mathsf{G}\,(\psi \to \mathsf{F}\chi)$

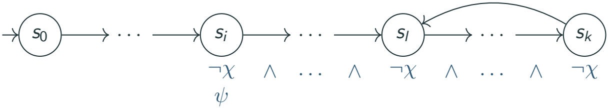
- use encoding of states, predicates $I$ and $T$ as for safety properties
- predicate $P$ such that $P(\langle s \rangle)$ is true iff $\psi$ holds in $s$
- predicate $C$ such that $C(\langle s \rangle)$ is true iff $\chi$ holds in $s$
- check satisfiability of

$$I(\langle s_0 \rangle) \wedge \bigwedge_{i=0}^{k-1} T(\langle s_i \rangle, \langle s_{i+1} \rangle) \wedge \bigvee_{i=0}^{k} \left( P(\langle s_i \rangle) \wedge \bigwedge_{j=i}^{k} \neg C(\langle s_i \rangle) \wedge \bigvee_{l=i}^{k} T(\langle s_k \rangle, \langle s_l \rangle) \right)$$

# Bounded Model Checking: Liveness Properties

## Idea

- counterexample to liveness property $\mathsf{G}\,(\psi \to \mathsf{F}\chi)$ requires infinite path
- look for counterexamples in $\leqslant k$ steps of lasso shape:




## SAT/SMT Encoding

given transition system $\mathcal{T}$ and liveness property $\mathsf{G}\,(\psi \to \mathsf{F}\chi)$

- use encoding of states, predicates $I$ and $T$ as for safety properties
- predicate $P$ such that $P(\langle s\rangle)$ is true iff $\psi$ holds in $s$
- predicate $C$ such that $C(\langle s\rangle)$ is true iff $\chi$ holds in $s$
- check satisfiability of

$$I(\langle s_0\rangle) \wedge \bigwedge_{i=0}^{k-1} T(\langle s_i\rangle, \langle s_{i+1}\rangle) \wedge \bigvee_{i=0}^{k} \left( P(\langle s_i\rangle) \wedge \bigwedge_{j=i}^{k} \neg C(\langle s_i\rangle) \wedge \bigvee_{l=i}^{k} T(\langle s_k\rangle, \langle s_l\rangle) \right)$$

## Transition System $\mathcal{T}(P)$ from Program $P$

- state $\langle pc, v_0, \ldots, v_n \rangle$ consists of
  - value for program counter $\mathtt{pc}$, i.e. line number in $P$
  - assignment for variables in scope $\mathtt{v}_0, \ldots, \mathtt{v}_n$

## Transition System $\mathcal{T}(P)$ from Program $P$

- state $\langle pc, v_0, \ldots, v_n \rangle$ consists of
  - value for program counter $\mathtt{pc}$, i.e. line number in $P$
  - assignment for variables in scope $\mathtt{v}_0, \ldots, \mathtt{v}_n$
- there is step $s \to s'$ for $s = \langle pc, v_0, \ldots, v_n \rangle$ and $s' = \langle pc', v_0', \ldots, v_n' \rangle$ iff $P$ admits step from $s$ to $s'$

## Transition System $\mathcal{T}(P)$ from Program $P$

- state $\langle pc, v_0, \ldots, v_n \rangle$ consists of
  - value for program counter $\mathtt{pc}$, i.e. line number in $P$
  - assignment for variables in scope $\mathtt{v}_0, \ldots, \mathtt{v}_n$
- there is step $s \to s'$ for $s = \langle pc, v_0, \ldots, v_n \rangle$ and $s' = \langle pc', v_0', \ldots, v_n' \rangle$ iff $P$ admits step from $s$ to $s'$
- $S_0$ consists of initial program states

## Transition System $\mathcal{T}(P)$ from Program $P$

- state $\langle pc, v_0, \dots, v_n \rangle$ consists of
  - value for program counter $\mathrm{pc}$, i.e. line number in $P$
  - assignment for variables in scope $\mathrm{v}_0, \dots, \mathrm{v}_n$
- there is step $s \to s'$ for $s = \langle pc, v_0, \dots, v_n \rangle$ and $s' = \langle pc', v_0', \dots, v_n' \rangle$ iff $P$ admits step from $s$ to $s'$
- $S_0$ consists of initial program states
- atom set $A$ consists of all propositional formulas over $pc, v_0, \dots, v_n$

## Transition System $\mathcal{T}(P)$ from Program $P$

- state $\langle pc, v_0, \ldots, v_n \rangle$ consists of
  - value for program counter $\mathrm{pc}$, i.e. line number in $P$
  - assignment for variables in scope $\mathrm{v}_0, \ldots, \mathrm{v}_n$
- there is step $s \to s'$ for $s = \langle pc, v_0, \ldots, v_n \rangle$ and $s' = \langle pc', v_0', \ldots, v_n' \rangle$ iff $P$ admits step from $s$ to $s'$
- $S_0$ consists of initial program states
- atom set $A$ consists of all propositional formulas over $pc, v_0, \ldots, v_n$
- labeling $L(s)$ is set of all atoms $A$ which hold in $s = \langle pc, v_0, \ldots, v_n \rangle$

## Transition System $\mathcal{T}(P)$ from Program $P$

- state $\langle pc, v_0, \ldots, v_n \rangle$ consists of
  - value for program counter $\mathrm{pc}$, i.e. line number in $P$
  - assignment for variables in scope $\mathrm{v}_0, \ldots, \mathrm{v}_n$
- there is step $s \to s'$ for $s = \langle pc, v_0, \ldots, v_n \rangle$ and $s' = \langle pc', v_0', \ldots, v_n' \rangle$ iff $P$ admits step from $s$ to $s'$
- $S_0$ consists of initial program states
- atom set $A$ consists of all propositional formulas over $pc, v_0, \ldots, v_n$
- labeling $L(s)$ is set of all atoms $A$ which hold in $s = \langle pc, v_0, \ldots, v_n \rangle$

## Program Graph

- nodes are line numbers
- edge from line $l$ to line $l'$ if program counter can go from line $l$ to $l'$
- two kinds of edge labels:
  - conditions for program counter to take this path
  - assignments of updated variables

## Transition System $\mathcal{T}(P)$ from Program $P$

- state $\langle pc, v_0, \ldots, v_n \rangle$ consists of
  - value for program counter $\mathrm{pc}$, i.e. line number in $P$
  - assignment for variables in scope $\mathrm{v}_0, \ldots, \mathrm{v}_n$
- there is step $s \rightarrow s'$ for $s = \langle pc, v_0, \ldots, v_n \rangle$ and $s' = \langle pc', v'_0, \ldots, v'_n \rangle$ iff $P$ admits step from $s$ to $s'$
- $S_0$ consists of initial program states
- atom set $A$ consists of all propositional formulas over $pc, v_0, \ldots, v_n$
- labeling $L(s)$ is set of all atoms $A$ which hold in $s = \langle pc, v_0, \ldots, v_n \rangle$

## Program Graph

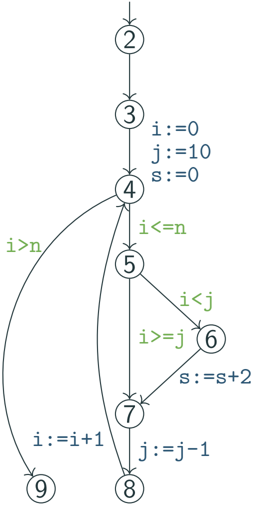
- nodes are line numbers
- edge from line $l$ to line $l'$ if program counter can go from line $l$ to $l'$
- two kinds of edge labels:
  - conditions for program counter to take this path
  - assignments of updated variables
- program graph is useful to derive encoding of $\mathcal{T}(P)$

## Checking an Explicit Assertion

```c
int n;
int main() {
  int i=0, j=10, s=0;
  for(i=0; i<=n; i++) {
    if (i<j)
      s = s + 2;
    j--;
  }
  assert(s==n*2 || s == 0);
}
```

## Checking an Explicit Assertion

```c
int n;
int main() {
  int i=0, j=10, s=0;
  for(i=0; i<=n; i++) {
    if (i<j)
      s = s + 2;
    j--;
  }
  assert(s==n*2 || s == 0);
}
```

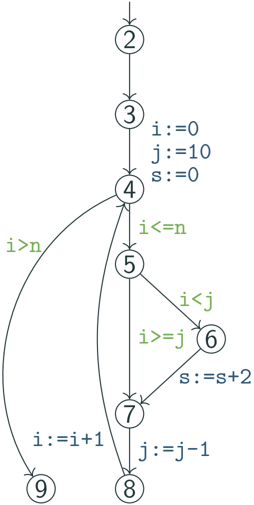



- construct program graph

# Checking an Explicit Assertion

```c
int n;
int main() {
  int i=0, j=10, s=0;
  for(i=0; i<=n; i++) {
    if (i<j)
      s = s + 2;
    j--;
  }
  assert(s==n*2 || s == 0);
}
```

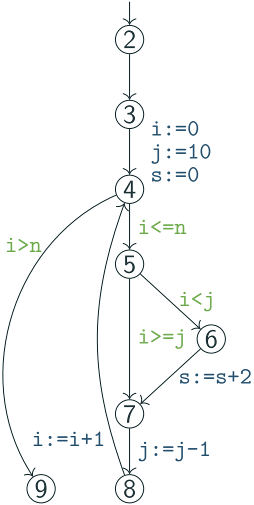
- construct program graph
- states are of form $\langle \mathrm{pc}, i, j, n, s \rangle$



## Checking an Explicit Assertion

```
1  int n;
2  int main() {
3    int i=0, j=10, s=0;
4    for(i=0; i<=n; i++) {
5      if (i<j)
6        s = s + 2;
7      j--;
8    }
9    assert(s==n*2 || s == 0);
10 }
```

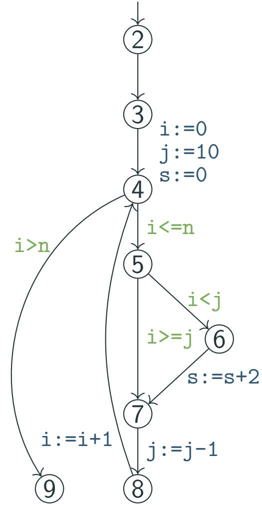
- construct program graph
- states are of form $\langle \mathrm{pc}, i, j, n, s \rangle$
- safety property to check is
  $\mathrm{G}\ (\mathrm{pc} = 9 \rightarrow (s = 2n \vee s = 0))$



# Checking an Explicit Assertion

```
1 int n;
2 int main() {
3   int i=0, j=10, s=0;
4   for(i=0; i<=n; i++) {
5     if (i<j)
6       s = s + 2;
7     j--;
8   }
9   assert(s==n*2 || s == 0);
10 }
```

- construct program graph
- states are of form $\langle \mathrm{pc}, i, j, n, s \rangle$
- safety property to check is
  $\mathrm{G}\ (\mathrm{pc} = 9 \rightarrow (s = 2n \lor s = 0))$
- see `verification.py`



## Software Verification Competition (SV-COMP)

- annual competition
  `https://sv-comp.sosy-lab.org/2018/`
- industrial (and crafted) benchmarks
  `https://github.com/sosy-lab/sv-benchmarks`
- many tools use SMT solvers

## Software Verification Competition (SV-COMP)

- annual competition
  `https://sv-comp.sosy-lab.org/2018/`
- industrial (and crafted) benchmarks
  `https://github.com/sosy-lab/sv-benchmarks`
- many tools use SMT solvers

## Common Safety Properties

- no overflow in addition: $(\mathtt{x} > 0 \land \mathtt{x} + \mathtt{y} \geqslant \mathtt{y}) \lor (\mathtt{x} \leqslant 0 \land \mathtt{x} + \mathtt{y} \leqslant \mathtt{y})$

## Software Verification Competition (SV-COMP)

- annual competition
  `https://sv-comp.sosy-lab.org/2018/`
- industrial (and crafted) benchmarks
  `https://github.com/sosy-lab/sv-benchmarks`
- many tools use SMT solvers

## Common Safety Properties

- no overflow in addition: $(\mathtt{x} > 0 \land \mathtt{x} + \mathtt{y} \geqslant \mathtt{y}) \lor (\mathtt{x} \leqslant 0 \land \mathtt{x} + \mathtt{y} \leqslant \mathtt{y})$
- array accesses in bounds: $0 \leqslant \mathtt{i} < \mathit{size}(\mathtt{a})$ for all accesses $\mathtt{a}[\mathtt{i}]$

## Software Verification Competition (SV-COMP)

- annual competition
  `https://sv-comp.sosy-lab.org/2018/`
- industrial (and crafted) benchmarks
  `https://github.com/sosy-lab/sv-benchmarks`
- many tools use SMT solvers

## Common Safety Properties

- no overflow in addition: $(\mathtt{x} > 0 \wedge \mathtt{x} + \mathtt{y} \geqslant \mathtt{y}) \vee (\mathtt{x} \leqslant 0 \wedge \mathtt{x} + \mathtt{y} \leqslant \mathtt{y})$
- array accesses in bounds: $0 \leqslant \mathtt{i} < \mathit{size}(\mathtt{a})$ for all accesses `a[i]`
- memory safety: set predicate $\mathit{ok}(\mathit{addr})$ when memory allocated, check $\mathit{ok}(\mathtt{p})$ for every dereference $*\mathtt{p}$

## Software Verification Competition (SV-COMP)

- annual competition
  `https://sv-comp.sosy-lab.org/2018/`
- industrial (and crafted) benchmarks
  `https://github.com/sosy-lab/sv-benchmarks`
- many tools use SMT solvers

## Common Safety Properties

- no overflow in addition: $(\mathtt{x} > 0 \land \mathtt{x} + \mathtt{y} \geqslant \mathtt{y}) \lor (\mathtt{x} \leqslant 0 \land \mathtt{x} + \mathtt{y} \leqslant \mathtt{y})$
- array accesses in bounds: $0 \leqslant \mathtt{i} < \mathit{size}(\mathtt{a})$ for all accesses $\mathtt{a}[\mathtt{i}]$
- memory safety: set predicate $\mathit{ok}(\mathit{addr})$ when memory allocated, check $\mathit{ok}(\mathtt{p})$ for every dereference $*\mathtt{p}$
- explicit assertions

# Outline

- Summary of Last Week
- Bounded Model Checking for Verification
- Quantifiers for SMT
  - Skolemization

# Applications of Quantifiers in SMT

## Example (Homework)

Imagine a village of monkeys where each monkey owns at least two bananas. As the monkeys are well-organised, each tree contains exactly three monkeys. Monkeys are also very friendly, so every monkey has a partner.

# Applications of Quantifiers in SMT

## Example (Homework)

Imagine a village of monkeys where each monkey owns at least two bananas. As the monkeys are well-organised, each tree contains exactly three monkeys. Monkeys are also very friendly, so every monkey has a partner.

### Example (Homework)

Imagine a village of monkeys where each monkey owns at least two bananas. As the monkeys are well-organised, each tree contains exactly three monkeys. Monkeys are also very friendly, so every monkey has a partner.

### More important applications

- automated theorem proving
  $\forall x\ y\ z.\ \mathsf{inv}(x) \cdot x = 0 \wedge 0 \cdot x = x \wedge x \cdot (y \cdot z) = (x \cdot y) \cdot z$

# Applications of Quantifiers in SMT

**Example (Homework)**

Imagine a village of monkeys where each monkey owns at least two bananas. As the monkeys are well-organised, each tree contains exactly three monkeys. Monkeys are also very friendly, so every monkey has a partner.

## More important applications

- automated theorem proving
  $\forall x\, y\, z.\ \mathsf{inv}(x) \cdot x = 0 \wedge 0 \cdot x = x \wedge x \cdot (y \cdot z) = (x \cdot y) \cdot z$
- software verification
  $\forall x.\ \mathsf{pre}(x) \longrightarrow \mathsf{post}(x)$

# Applications of Quantifiers in SMT

### Example (Homework)

Imagine a village of monkeys where each monkey owns at least two bananas. As the monkeys are well-organised, each tree contains exactly three monkeys. Monkeys are also very friendly, so every monkey has a partner.

### More important applications

- automated theorem proving
  $\forall x\ y\ z.\ \mathrm{inv}(x) \cdot x = 0 \wedge 0 \cdot x = x \wedge x \cdot (y \cdot z) = (x \cdot y) \cdot z$
- software verification
  $\forall x.\ \mathrm{pre}(x) \longrightarrow \mathrm{post}(x)$
- function synthesis
  $\forall \mathit{input}.\ \exists \mathit{output}.\ \mathrm{F}(\mathit{input}, \mathit{output})$

# Applications of Quantifiers in SMT

## Example (Homework)

Imagine a village of monkeys where each monkey owns at least two bananas. As the monkeys are well-organised, each tree contains exactly three monkeys. Monkeys are also very friendly, so every monkey has a partner.

## More important applications

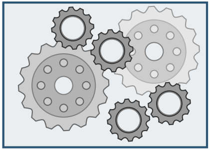
- automated theorem proving
  $\forall x\ y\ z.\ \mathrm{inv}(x) \cdot x = 0 \wedge 0 \cdot x = x \wedge x \cdot (y \cdot z) = (x \cdot y) \cdot z$
- software verification
  $\forall x.\ \mathrm{pre}(x) \longrightarrow \mathrm{post}(x)$
- function synthesis
  $\forall input.\ \exists output.\ \mathrm{F}(input, output)$
- planning
  $\exists plan.\ \forall time.\ \mathrm{spec}(plan, time)$

# SMT Solving with Quantifiers

## SMT solver

$\varphi$ $\longrightarrow$

$f(a) = a \vee (f(b) \neq a \wedge a = b)$



SAT ($v$) $f(a) = a$

UNSAT

## Decision Properties

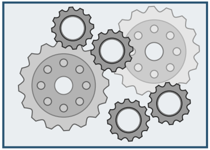
- SMT solvers can decide propositional logic + LIA/LRA/EUF/BV/...

# SMT Solving with Quantifiers

## SMT solver

$\varphi$

$\forall x.\ \mathsf{f}(x) \neq x \wedge \mathsf{f}(a) = b \wedge a = b$



SAT ($v$)

unknown

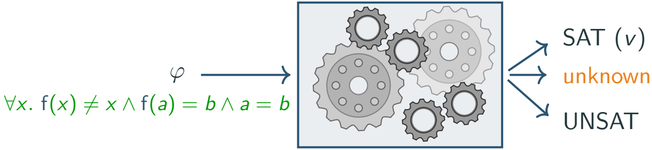
UNSAT

## Decision Properties

- SMT solvers can decide propositional logic + LIA/LRA/EUF/BV/...
- many SMT solvers also have support for quantifiers

# SMT Solving with Quantifiers

## SMT solver




## Decision Properties

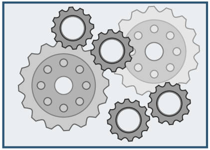
- SMT solvers can decide propositional logic + LIA/LRA/EUF/BV/...
- many SMT solvers also have support for quantifiers,
  but have in general no decision procedure for theories + quantifiers

# SMT Solving with Quantifiers

## SMT solver

$\varphi$ $\longrightarrow$

$\forall x.\ \mathsf{f}(x) \neq x \wedge \mathsf{f}(a) = b \wedge a = b$



SAT ($v$)

unknown

UNSAT

## Decision Properties

first-order logic is undecidable!

- SMT solvers can decide propositional logic + LIA/LRA/EUF/BV/...
- many SMT solvers also have support for quantifiers,
  but have in general no decision procedure for theories + quantifiers

# Skolemization

Thoralf Skolem

## Getting rid of $\exists$ quantifiers

- replace $\exists x.\ \mathrm{P}(x)$ by $\mathrm{P}(\mathrm{a})$

# Skolemization

name witness for existential quantifier

## Getting rid of $\exists$ quantifiers

- replace $\exists x.\ \mathrm{P}(x)$ by $\mathrm{P}(\mathrm{a})$

Thoralf Skolem

# Skolemization

name witness for existential quantifier

Thoralf Skolem

## Getting rid of $\exists$ quantifiers

- replace $\exists x.\ P(x)$ by $P(a)$
- replace $\forall y\ \exists x.\ P(x)$ by $\forall y\ P(f(y))$

# Skolemization

name witness for existential quantifier

## Getting rid of $\exists$ quantifiers

- replace $\exists x.\ \mathrm{P}(x)$ by $\mathrm{P}(\mathrm{a})$
- replace $\forall y\, \exists x.\ \mathrm{P}(x)$ by $\forall y\ \mathrm{P}(\mathrm{f}(y))$
- replace $\forall z\, \forall y\, \exists x.\ \mathrm{R}(x)$ by $\forall z\, \forall y\ \mathrm{R}(\mathrm{f}(y, z))$

Thoralf Skolem

# Skolemization

name witness for existential quantifier

## Getting rid of $\exists$ quantifiers

- replace $\exists x.\ \mathrm{P}(x)$ by $\mathrm{P}(\mathrm{a})$
- replace $\forall y\ \exists x.\ \mathrm{P}(x)$ by $\forall y\ \mathrm{P}(\mathrm{f}(y))$
- replace $\forall z\ \forall y\ \exists x.\ \mathrm{R}(x)$ by $\forall z\ \forall y\ \mathrm{R}(\mathrm{f}(y, z))$

Thoralf Skolem

## Definitions

- $\varphi$ is in prenex form if $\varphi = Q_1 x_1 \ldots Q_n x_n\ \psi$ for $\psi$ quantifier-free and $Q_i \in \{\forall, \exists\}$

# Skolemization

name witness for existential quantifier

## Getting rid of $\exists$ quantifiers

- replace $\exists x.\ \mathrm{P}(x)$ by $\mathrm{P}(\mathrm{a})$
- replace $\forall y\, \exists x.\ \mathrm{P}(x)$ by $\forall y\ \mathrm{P}(\mathrm{f}(y))$
- replace $\forall z\, \forall y\, \exists x.\ \mathrm{R}(x)$ by $\forall z\, \forall y\ \mathrm{R}(\mathrm{f}(y, z))$

Thoralf Skolem

## Definitions

- $\varphi$ is in prenex form if $\varphi = Q_1 x_1 \dots Q_n x_n\ \psi$ for $\psi$ quantifier-free and $Q_i \in \{\forall, \exists\}$
- $\varphi$ is in Skolem form if in prenex form without existential quantifier

name witness for existential quantifier

Thoralf Skolem

## Getting rid of $\exists$ quantifiers

- replace $\exists x.\ \mathrm{P}(x)$ by $\mathrm{P}(\mathrm{a})$
- replace $\forall y\ \exists x.\ \mathrm{P}(x)$ by $\forall y\ \mathrm{P}(\mathrm{f}(y))$
- replace $\forall z\ \forall y\ \exists x.\ \mathrm{R}(x)$ by $\forall z\ \forall y\ \mathrm{R}(\mathrm{f}(y, z))$

## Definitions

- $\varphi$ is in prenex form if $\varphi = Q_1 x_1 \dots Q_n x_n\ \psi$ for $\psi$ quantifier-free and $Q_i \in \{\forall, \exists\}$
- $\varphi$ is in Skolem form if in prenex form without existential quantifier

## Skolemization

1. bring formula into prenex form

# Skolemization

name witness for existential quantifier

### Getting rid of $\exists$ quantifiers

- replace $\exists x.\ \mathrm{P}(x)$ by $\mathrm{P}(\mathrm{a})$
- replace $\forall y\ \exists x.\ \mathrm{P}(x)$ by $\forall y\ \mathrm{P}(\mathrm{f}(y))$
- replace $\forall z\ \forall y\ \exists x.\ \mathrm{R}(x)$ by $\forall z\ \forall y\ \mathrm{R}(\mathrm{f}(y, z))$

Thoralf Skolem

### Definitions

- $\varphi$ is in prenex form if $\varphi = Q_1 x_1 \dots Q_n x_n\ \psi$ for $\psi$ quantifier-free and $Q_i \in \{\forall, \exists\}$
- $\varphi$ is in Skolem form if in prenex form without existential quantifier

### Skolemization

1. bring formula into prenex form
2. replace $\forall x_1, \dots, x_k \exists y\ \psi[y]$ by $\forall x_1, \dots, x_k\ \psi[\mathrm{f}(x_1, \dots, x_k)]$ for fresh f

   until no existential quantifiers left

# Skolemization

name witness for existential quantifier

## Getting rid of $\exists$ quantifiers

- replace $\exists x.\ \mathrm{P}(x)$ by $\mathrm{P}(\mathrm{a})$
- replace $\forall y\ \exists x.\ \mathrm{P}(x)$ by $\forall y\ \mathrm{P}(\mathrm{f}(y))$
- replace $\forall z\ \forall y\ \exists x.\ \mathrm{R}(x)$ by $\forall z\ \forall y\ \mathrm{R}(\mathrm{f}(y, z))$

Thoralf Skolem

## Definitions

- $\varphi$ is in prenex form if $\varphi = Q_1 x_1 \dots Q_n x_n\ \psi$ for $\psi$ quantifier-free and $Q_i \in \{\forall, \exists\}$
- $\varphi$ is in Skolem form if in prenex form without existential quantifier

## Skolemization

1. bring formula into prenex form
2. replace $\forall x_1, \dots, x_k \exists y\ \psi[y]$ by $\forall x_1, \dots, x_k\ \psi[\mathrm{f}(x_1, \dots, x_k)]$ for fresh f

   until no existential quantifiers left

## Theorem

if $\varphi'$ is skolemization of $\varphi$ then $\varphi$ and $\varphi'$ are equisatisfiable

# Skolemization

name witness for existential quantifier

Thoralf Skolem

### Getting rid of $\exists$ quantifiers

- replace $\exists x.\ \mathrm{P}(x)$ by $\mathrm{P}(\mathrm{a})$
- replace $\forall y\ \exists x.\ \mathrm{P}(x)$ by $\forall y\ \mathrm{P}(\mathrm{f}(y))$
- replace $\forall z\ \forall y\ \exists x.\ \mathrm{R}(x)$ by $\forall z\ \forall y\ \mathrm{R}(\mathrm{f}(y, z))$

### Definitions

- $\varphi$ is in prenex form if $\varphi = Q_1 x_1 \dots Q_n x_n\ \psi$ for $\psi$ quantifier-free and $Q_i \in \{\forall, \exists\}$
- $\varphi$ is in Skolem form if in prenex form without existential quantifier

### Skolemization

1. bring formula into prenex form
2. replace $\forall x_1, \dots, x_k \exists y\ \psi[y]$ by $\forall x_1, \dots, x_k\ \psi[\mathrm{f}(x_1, \dots, x_k)]$ for fresh f

until no existential quantifiers left

### Theorem

can consider formulas of shape $\forall x_1, \dots, x_n\ \varphi[x_1, \dots, x_n]$

if $\varphi'$ is skolemization of $\varphi$ then $\varphi$ and $\varphi'$ are equisatisfiable

## Definition

Herbrand instance of Skolem formula $\forall x_1, \ldots, x_n \; \varphi[x_1, \ldots, x_n]$ is $\varphi[t_1, \ldots, t_n]$ where $t_i$ is term over signature of $\varphi$

**Definition**

set of function symbols and constants

Herbrand instance of Skolem formula $\forall x_1, \ldots, x_n\ \varphi[x_1, \ldots, x_n]$ is $\varphi[t_1, \ldots, t_n]$ where $t_i$ is term over signature of $\varphi$

## Definition

Herbrand instance of Skolem formula $\forall x_1, \ldots, x_n\ \varphi[x_1, \ldots, x_n]$ is $\varphi[t_1, \ldots, t_n]$ where $t_i$ is term over signature of $\varphi$

## Remark

Herbrand instances are ground formulas, i.e., without (quantified) variables

Jacques Herbrand

### Definition

Herbrand instance of Skolem formula $\forall x_1, \ldots, x_n\ \varphi[x_1, \ldots, x_n]$ is $\varphi[t_1, \ldots, t_n]$ where $t_i$ is term over signature of $\varphi$

### Remark

Herbrand instances are ground formulas, i.e., without (quantified) variables

### Theorem (Herbrand)

*Skolem formula $\varphi$ is unsatisfiable $\Longleftrightarrow$*
*there exists finite unsatisfiable set of Herbrand instances of $\varphi$*

Jacques Herbrand

## Definition

Herbrand instance of Skolem formula $\forall x_1, \ldots, x_n\ \varphi[x_1, \ldots, x_n]$ is $\varphi[t_1, \ldots, t_n]$ where $t_i$ is term over signature of $\varphi$

## Remark

Herbrand instances are ground formulas, i.e., without (quantified) variables

## Theorem (Herbrand)

*Skolem formula $\varphi$ is unsatisfiable $\iff$*
*there exists finite unsatisfiable set of Herbrand instances of $\varphi$*

Jacques Herbrand

## Caveats

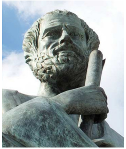
- at least one constant required per sort
- holds for pure first order logic, not necessarily in presence of theories

## Example: Is this syllogism correct?

All humans are mortal.
All Greeks are humans.

---

So all Greeks are mortal.

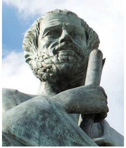


Aristotle

## Example: Is this syllogism correct?

$$\frac{\text{All humans are mortal.} \quad \text{All Greeks are humans.}}{\text{So all Greeks are mortal.}}$$

$$\frac{\forall x.\ H(x) \longrightarrow M(x) \qquad \forall x.\ G(x) \longrightarrow H(x)}{\forall x.\ G(x) \longrightarrow M(x)}$$

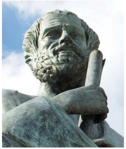


Aristotle

- translate to first-order logic

## Example: Is this syllogism correct?

$$\begin{array}{l} \text{All humans are mortal.} \\ \text{All Greeks are humans.} \\ \hline \text{So all Greeks are mortal.} \end{array} \qquad \begin{array}{l} \forall x.\ H(x) \longrightarrow M(x) \\ \forall x.\ G(x) \longrightarrow H(x) \\ \hline \forall x.\ G(x) \longrightarrow M(x) \end{array}$$



Aristotle

- translate to first-order logic
- check validity of

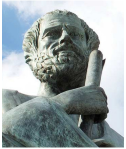
$$((\forall x.\ H(x) \longrightarrow M(x)) \wedge (\forall x.\ G(x) \longrightarrow H(x))) \longrightarrow (\forall x.\ G(x) \longrightarrow M(x))$$

## Example: Is this syllogism correct?

All humans are mortal.
All Greeks are humans.

So all Greeks are mortal.

$$\frac{\forall x.\ H(x) \longrightarrow M(x) \qquad \forall x.\ G(x) \longrightarrow H(x)}{\forall x.\ G(x) \longrightarrow M(x)}$$



Aristotle

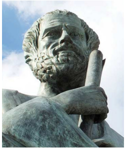
- translate to first-order logic
- check validity of

cannot be answered by SMT solver

$$((\forall x.\ H(x) \longrightarrow M(x)) \wedge (\forall x.\ G(x) \longrightarrow H(x))) \longrightarrow (\forall x.\ G(x) \longrightarrow M(x))$$

## Example: Is this syllogism correct?

| | |
|---|---|
| All humans are mortal. | $\forall x.\ H(x) \longrightarrow M(x)$ |
| All Greeks are humans. | $\forall x.\ G(x) \longrightarrow H(x)$ |
| So all Greeks are mortal. | $\forall x.\ G(x) \longrightarrow M(x)$ |



Aristotle

- translate to first-order logic
- check validity of

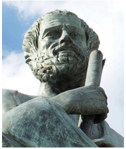
$$((\forall x.\ H(x) \longrightarrow M(x)) \wedge (\forall x.\ G(x) \longrightarrow H(x))) \longrightarrow (\forall x.\ G(x) \longrightarrow M(x))$$

- check unsatisfiability of

$$\forall x.\ H(x) \longrightarrow M(x), \quad \forall x.\ G(x) \longrightarrow H(x), \quad \exists x.\ G(x) \wedge \neg M(x)$$

## Example: Is this syllogism correct?

| | |
|---|---|
| All humans are mortal. | $\forall x.\ H(x) \longrightarrow M(x)$ |
| All Greeks are humans. | $\forall x.\ G(x) \longrightarrow H(x)$ |
| So all Greeks are mortal. | $\forall x.\ G(x) \longrightarrow M(x)$ |



Aristotle

- translate to first-order logic
- check validity of

$$((\forall x.\ H(x) \longrightarrow M(x)) \wedge (\forall x.\ G(x) \longrightarrow H(x))) \longrightarrow (\forall x.\ G(x) \longrightarrow M(x))$$

- check unsatisfiability of

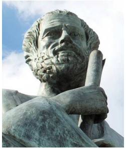
$$\forall x.\ H(x) \longrightarrow M(x), \quad \forall x.\ G(x) \longrightarrow H(x), \quad \exists x.\ G(x) \wedge \neg M(x)$$

- skolemize

$$\forall x.\ H(x) \longrightarrow M(x), \quad \forall x.\ G(x) \longrightarrow H(x), \quad G(a) \wedge \neg M(a)$$

## Example: Is this syllogism correct?

$$\frac{\begin{array}{l}\text{All humans are mortal.}\\ \text{All Greeks are humans.}\end{array}}{\text{So all Greeks are mortal.}} \qquad \frac{\begin{array}{l}\forall x.\ H(x) \longrightarrow M(x)\\ \forall x.\ G(x) \longrightarrow H(x)\end{array}}{\forall x.\ G(x) \longrightarrow M(x)}$$



Aristotle

- translate to first-order logic
- check validity of

$$((\forall x.\ H(x) \longrightarrow M(x)) \wedge (\forall x.\ G(x) \longrightarrow H(x))) \longrightarrow (\forall x.\ G(x) \longrightarrow M(x))$$

- check unsatisfiability of

$$\forall x.\ H(x) \longrightarrow M(x), \quad \forall x.\ G(x) \longrightarrow H(x), \quad \exists x.\ G(x) \wedge \neg M(x)$$

- skolemize

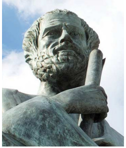
$$\forall x.\ H(x) \longrightarrow M(x), \quad \forall x.\ G(x) \longrightarrow H(x), \quad G(a) \wedge \neg M(a)$$

- already unsatisfiable when replacing quantified formulas by Herbrand instances

$$H(a) \longrightarrow M(a), \quad G(a) \longrightarrow H(a), \quad G(a) \wedge \neg M(a)$$

## Example: Is this syllogism correct?

| | |
|---|---|
| All humans are mortal. | $\forall x.\ H(x) \longrightarrow M(x)$ |
| All Greeks are humans. | $\forall x.\ G(x) \longrightarrow H(x)$ |
| So all Greeks are mortal. | $\forall x.\ G(x) \longrightarrow M(x)$ |



Aristotle

- translate to first-order logic
- check validity of

$$((\forall x.\ H(x) \longrightarrow M(x)) \wedge (\forall x.\ G(x) \longrightarrow H(x))) \longrightarrow (\forall x.\ G(x) \longrightarrow M(x))$$

- check unsatisfiability of

$$\forall x.\ H(x) \longrightarrow M(x), \quad \forall x.\ G(x) \longrightarrow H(x), \quad \exists x.\ G(x) \wedge \neg M(x)$$

- skolemize

$$\forall x.\ H(x) \longrightarrow M(x),$$

when adding right Herbrand instances
unsatisfiability can be detected by SMT solver

- already unsatisfiable when replacing quantified formulas by Herbrand instances

$$H(a) \longrightarrow M(a), \quad G(a) \longrightarrow H(a), \quad G(a) \wedge \neg M(a)$$

# Bibliography

Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Ofer Strichman, and Yunshan Zhu.
**Bounded Model Checking**
Advances in Computers 58, pp 117–148, 2003.

Armin Biere.
**Bounded Model Checking.**
Chapter 14 in: Handbook of Satisfiability, IOS Press, pp. 457–481, 2009.