# Formal Specification and Verification

Deductive Verification: An introduction

7.02.2017

Viorica Sofronie-Stokkermans

e-mail: sofronie@uni-koblenz.de

# Overview

- **Model checking:**

  Finite transition systems / CTL properties

  States are "entities" (no precise description, except for labelling functions)

  No precise description of actions (only $\rightarrow$ important)

# Overview

- **Model checking:**

  Finite transition systems / CTL properties

  States are "entities" (no precise description, except for labelling functions)

  No precise description of actions (only $\rightarrow$ important)

Extensions in two possible directions:

- More precise description of the actions/events
  - Propositional Dynamic Logic (last time)
  - Hoare logic (not discussed in this lecture)
- More precise description of states (and possibly also of actions)
  - succinct representation: formulae represent a set of states
  - deductive verification (today)

# Transition systems (Reminder)

- Model to describe the behaviour of systems
- Digraphs where nodes represent states, and edges model transitions
- **State:** Examples
  - the current colour of a traffic light
  - the current values of all program variables + the program counter
  - the current value of the registers together with the values of the input bits
- **Transition** ("state change"): Examples
  - a switch from one colour to another
  - the execution of a program statement
  - the change of the registers and output bits for a new input

# Transition systems

**Definition.**

A transition system $TS$ is a tuple $(S, Act, \rightarrow, I, AP, L)$ where:

- $S$ is a set of states
- $Act$ is a set of actions
- $\rightarrow \subseteq S \times Act \times S$ is a transition relation
- $I \subseteq S$ is a set of initial states
- $AP$ is a set of atomic propositions
- $L : S \rightarrow 2^{AP}$ is a labeling function

$S$ and $Act$ are either finite or countably infinite

**Notation:** $s \stackrel{\alpha}{\rightarrow} s'$ instead of $(s, \alpha, s') \in \rightarrow$.

# Programs and transition systems

Program graph representation

# Program graph representation

Some preliminaries

- typed variables with a valuation that assigns values in a fixed structure to variables
  - e.g., $\beta(x) = 17$ and $\beta(y) = -2$
- Boolean conditions: set of formulae over *Var*
  - propositional logic formulas whose propositions are of the form "$x \in D$"
  - $(-3 < x \leq 5) \wedge (y = green) \wedge (x \leq 2 * x')$
- effect of the actions is formalized by means of a mapping:

$$Effect : Act \times Eval(Var) \rightarrow Eval(Var)$$

  - e.g., $\alpha \equiv x := y + 5$ and evaluation $\beta(x) = 17$ and $\beta(y) = -2$
  - $Effect(\alpha, \beta)(x) = \beta(y) + 5 = 3$,
  - $Effect(\alpha, \beta)(y) = \beta(y) = -2$

# Program graph representation

## Program graphs

A program graph $PG$ over set $Var$ of typed variables is a tuple

$$(Loc, Act, Effect, \rightarrow, Loc_0, g_0)$$

where

- $Loc$ is a set of locations with initial locations $Loc_0 \subseteq Loc$
- $Act$ is a set of actions
- $Effect : Act \times Eval(Var) \rightarrow Eval(Var)$ is the effect function
- $\rightarrow \ \subseteq \ Loc \times (\underbrace{Cond(Var)}_{\text{Boolean conditions on } Var} \times Act) \times Loc$, transition relation
- $g_0 \in Cond(Var)$ is the initial condition.

Notation: $l \overset{g:\alpha}{\rightarrow} l'$ denotes $(l, g, \alpha, l') \in \rightarrow$.

# From program graphs to transition systems

- Basic strategy: unfolding
  - state = location (current control) $l$ + data valuation $\beta$ $\qquad (l, \beta)$
  - initial state = initial location + data valuation satisfying the initial condition $g_0$
- Propositions and labeling
  - propositions: "at $l$" and "$x \in D$" for $D \subseteq dom(x)$
  - $< l, \beta >$ is labeled with "at $l$" and all conditions that hold in $\beta$.
- $l \stackrel{g:\alpha}{\rightarrow} l'$ and $g$ holds in $\beta$ then $< l, \beta > \stackrel{\alpha}{\rightarrow} < l', Effect(< l, \beta >) >$

# Transition systems for program graphs

The transition system $TS(PG)$ of program graph

$$PG = (Loc, Act, Effect, \rightarrow, Loc_0, g_0)$$

over set $Var$ of variables is the tuple $(S, Act, \rightarrow, I, AP, L)$ where:

- $S = Loc \times Eval(Var)$
- $\rightarrow S \times Act \times S$ is defined by the rule:
  If $l \overset{g:\alpha}{\rightarrow} l'$ and $\beta \models g$ then $< l, \beta > \overset{\alpha}{\rightarrow} < l', Effect(< l, \beta >) >$
- $I = \{< l, \beta >| \; l \in \mathrm{Loc}_0, \beta \models g_0\}$
- $AP = Loc \cup Cond(Var)$ and
- $L(< l, \beta >) = \{l\} \cup \{g \in Cond(Var) \mid \beta \models g\}$.

# Problem

Set of states: $S = Loc \times Eval(Var)$

$Eval(Var)$ can be very large
(some variables can have values in large data domains e.g. integers)

Therefore it is also difficult to concretely represent $\rightarrow$
(the relation usually very large as well)

# Solution

**Succinct representation of sets of states and of transitions between states**

- Set of states: Formula (property of all states in the set)
- Transitions: Formulae (relation between the old values of the variables and the new values of the variables)

# Example

```
1: if (y >= z) then skip else halt;
2: while (x < y) {
      x++;
   }
3: if (x >= z) then skip else goto 5;
4: exit
5: error
```

# Example

```
1: if (y >= z) then skip else halt;
2: while (x < y) {
       x++;
   }
3: if (x >= z) then skip else goto 5;
4: exit
5: error
```

## States:

$(l, \beta)$, where $l$ location and $\beta$ assignment of values to the variables.

# Example

```
1: if (y >= z) then skip else halt;
2: while (x < y) {
      x++;
   }
3: if (x >= z) then skip else goto 5;
4: exit
5: error
```

**States:**

$(l, \beta)$, where $l$ location and $\beta$ assignment of values to the variables.

Idea: Take into account an additional variable pc (program counter), having as domain the set of locations.

State: assignment of values to the variables and to pc

# Example

```
1: if (y >= z) then skip else halt;
2: while (x < y) {
       x++;
   }
3: if (x >= z) then skip else goto 5;
4: exit
5: error
```

**States:**

$(l, \beta)$, where $l$ location and $\beta$ assignment of values to the variables.

Idea: Take into account an additional variable pc (program counter), having as domain the set of locations.

State: assignment of values to the variables and to pc

**Set of states:** Logical formula

Example:

$y \geq z$: The set of all states $(l, \beta)$ for which $\beta(y) \geq \beta(z)$ (i.e. $\beta \models y \geq z$)

# Example

```
1: if (y >= z) then skip else halt;
2: while (x < y) {
      x++;
   }
3: if (x >= z) then skip else goto 5;
4: exit
5: error
```

**Transition relation:** $(l, \beta) \rightarrow (l', \beta')$

# Example

```
1: if (y >= z) then skip else halt;
2: while (x < y) {
      x++;
   }
3: if (x >= z) then skip else goto 5;
4: exit
5: error
```

**Transition relation:** $(l, \beta) \rightarrow (l', \beta')$

**Expressed by logical formulae:** Formula containing primed and unprimed variables.

Example:

- $\rho_1 = (move(l_1, l_2) \wedge y \geq z \wedge skip(x, y, z))$
- $\rho_2 = (move(l_2, l_2) \wedge x + 1 \leq y \wedge x' = x + 1 \wedge skip(y, z))$
- $\rho_3 = (move(l_2, l_3) \wedge x \geq y \wedge skip(x, y, z))$
- $\rho_4 = (move(l_3, l_4) \wedge x \geq z \wedge skip(x, y, z))$
- $\rho_5 = (move(l_3; \ l_5) \wedge x + 1 \leq z \wedge skip(x, y, z))$

Abbreviations:

$move(l, l') := (pc = l \wedge pc' = l')$

$skip(v_1, \ldots, v_n) := (v_1' = v_1 \wedge \cdots \wedge v_n' = v_n)$

# Programs as transition systems

**Verification problem: Program + Description of the "bad" states**

**Succinct representation:**

$$P = (Var, pc, Init, \mathcal{R}) \qquad \phi_{\text{err}}$$

- $V$ - finite (ordered) set of program variables
- $pc$ - program counter variable ($pc$ included in $V$)
- $Init$ - initiation condition given by formula over $V$
- $\mathcal{R}$ - a finite set of transition relations

  Every transition relation $\rho \in \mathcal{R}$ is given by a formula over the variables $V$ and their primed versions $V'$

- $\phi_{\text{err}}$ - an error condition given by a formula over $V$

More details: next time.