Formal Specification and Verification

Deductive Verification: An introduction

26.01.2022

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 → 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 → 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 \((S, \text{Act}, \rightarrow, I, \text{AP}, L)\) where:

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

\( S \) and \( \text{Act} \) are either finite or countably infinite

**Notation:** \( s \xrightarrow{\alpha} s' \) instead of \((s, \alpha, s') \in \rightarrow\).
Programs and transition systems

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 $\text{Var}$
  - propositional logic formulas whose propositions are of the form “$x \in D$”
    - $(−3 < x \leq 5) \land (y = \text{green}) \land (x \leq 2 \ast x')$

- effect of the actions is formalized by means of a mapping:

  \[ \text{Effect} : \text{Act} \times \text{Eval}(\text{Var}) \rightarrow \text{Eval}(\text{Var}) \]

  - e.g., $\alpha \equiv x := y + 5$ and evaluation $\beta(x) = 17$ and $\beta(y) = -2$
  - $\text{Effect}(\alpha, \beta)(x) = \beta(y) + 5 = 3$,
  - $\text{Effect}(\alpha, \beta)(y) = \beta(y) = -2$
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
- $\rightarrow \subseteq Loc \times \left( Cond(Var) \times Act \right) \times Loc$, transition relation
- $g_0 \in Cond(Var)$ is the initial condition.

Notation: $l \xrightarrow{g^\cdot \alpha} l'$ denotes $(l, g, \alpha, l') \in \rightarrow$. 
From program graphs to transition systems

- Basic strategy: unfolding
  - state = location (current control) \( l \) + data valuation \( \beta \) \((l, \beta)\)
  - initial state = initial location + data valuation satisfying the initial condition \( g_0 \)

- Propositions and labeling
  - propositions: “at \( l \)” and “\( x \in D \)” for \( D \subseteq \text{dom}(x) \)
  - \( < l, \beta > \) is labeled with “at \( l \)” and all conditions that hold in \( \beta \).

- \( l \xrightarrow{g;\alpha} l' \) and \( g \) holds in \( \beta \) then \( < l, \beta > \xrightarrow{\alpha} < l', \text{Effect}(< l, \beta >) > \)
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 $l \xrightarrow{g: \alpha} l'$ and $\beta \models g$ then $< l, \beta > \xrightarrow{\alpha} < l', Effect(< l, \beta >) >$

- $I = \{ < l, \beta > | l \in Loc_0, \beta \models g_0 \}$
- $AP = Loc \cup Cond(Var)$ and
- $L(< l, \beta >) = \{ l \} \cup \{ g \in Cond(Var) | \beta \models g \}$. 
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, β), where l location and β 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 ≥ z: The set of all states (l, β) for which β(y) ≥ β(z) (i.e. β |⇒ y ≥ z)
Example

1: if (y \geq z) then skip else halt;
2: while (x < y) {
   x++;
}
3: if (x \geq 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:

Example:

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

Abbreviations:

\(move(l, l') := (pc = l \land pc' = l')\)
\(skip(v_1, \ldots, v_n) := (v'_1 = v_1 \land \cdots \land v'_n = v_n)\)
Programs as transition systems

Verification problem: Program + Description of the “bad” states

Succinct representation:

\[ P = (V, pc, Init, \mathcal{R}, \phi_{err}) \]

- \( V \) - 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 \( V' \)
- \( \phi_{err} \) - an error condition given by a formula over \( V \)
States, sets and relations

- Each program variable $x$ is assigned a domain of values $D_x$.
- Program state = function that assigns each program variable a value from its respective domain
- $S$ = set of program states
- Formula with free variables in $V$ = set of program states
- Formula with free variables in $V$ and $V'$ = binary relation over program states
  - First component of each pair refers to values of the variables $V$
  - Second component of the pair refers to values of the variables $V'$ (typically the new variables of the variables in $V$ after an instruction was executed)
States, sets and relations

- We identify formulas with the sets and relations that they represent.
- We identify the entailment relation between formulas $\models$ with set inclusion.
- We identify the satisfaction relation $|=\models$ between valuations and formulas, with the membership relation.
States, sets and relations

- We identify formulas with the sets and relations that they represent.
- We identify the entailment relation between formulas $\models$ with set inclusion.
- We identify the satisfaction relation $\models$ between valuations and formulas, with the membership relation.

**Example:**

- Formula $y \geq z =$ set of program states in which the value of the variable $y$ is greater than the value of $z$.
- Formula $y' \geq z =$ binary relation over program states, $=$ set of pairs of program states $(s_1, s_2)$ in which the value of the variable $y$ in the second state $s_2$ is greater than the value of $z$ in the first state $s_1$. 
States, sets and relations

- We identify formulas with the sets and relations that they represent.
- We identify the entailment relation between formulas $\models$ with set inclusion.
- We identify the satisfaction relation $\models$ between valuations and formulas, with the membership relation.

**Example:**

- Formula $y \geq z$ = set of program states in which the value of the variable $y$ is greater than the value of $z$.
- Formula $y' \geq z$ = binary relation over program states, $=$ set of pairs of program states $(s_1, s_2)$ in which the value of the variable $y$ in the second state $s_2$ is greater than the value of $z$ in the first state $s_1$.
- If program state $s$ assigns 1, 3, 2, and $l_1$ to program variables $x, y, z,$ and $pc$, respectively, then $s \models y \geq z$.
States, sets and relations

- We identify formulas with the sets and relations that they represent.
- We identify the entailment relation between formulas $\models$ with set inclusion.
- We identify the satisfaction relation $|=\,$ between valuations and formulas, with the membership relation.

**Example:**
- Formula $y \geq z$ = set of program states in which the value of the variable $y$ is greater than the value of $z$.
- Formula $y' \geq z$ = binary relation over program states, $|=\,$ set of pairs of program states $(s_1, s_2)$ in which the value of the variable $y$ in the second state $s_2$ is greater than the value of $z$ in the first state $s_1$.
- If program state $s$ assigns 1, 3, 2, and $l_1$ to program variables $x, y, z$, and $pc$, respectively, then $s |= y \geq z$.
- Logical consequence: $y \geq z |= y + 1 \geq z$. 
Example Program

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 program

- Program variables $V = (pc, x, y, z)$
- Program counter $pc$
- Program variables $x, y,$ and $z$ range over integers: $D_x = D_y = D_z = \text{Int}$
  - Program counter $pc$ ranges over control locations: $D_{pc} = L$
- Set of control locations $L = \{l_1, l_2, l_3, l_4, l_5\}$
- Initiation condition $Init := (pc = l_1)$
- Error condition $\phi_{err} := (pc = l_5)$
- Program transitions $\mathcal{R} = \{\rho_1, \ldots, \rho_5\}$, where:
  \[
  \begin{align*}
  \rho_1 &= (move(l_1, l_2) \land y \geq z \land skip(x, y, z)) \\
  \rho_2 &= (move(l_2, l_2) \land x + 1 \leq y \land x' = x + 1 \land skip(y, z)) \\
  \rho_3 &= (move(l_2, l_3) \land x \geq y \land skip(x, y, z)) \\
  \rho_4 &= (move(l_3, l_4) \land x \geq z \land skip(x, y, z)) \\
  \rho_5 &= (move(l_3; l_5) \land x + 1 \leq z \land skip(x, y, z))
  \end{align*}
  \]
Initial state, error state, transition relation

- Each state that satisfies the initiation condition \( \text{Init} \) is called an initial state.
- Each state that satisfies the error condition \( \text{err} \) is called an error state.
- Program transition relation \( \rho_R \) is the union of the single-statement transition relations (formula representation: disjunction) i.e.,

\[
\rho_R = \bigvee_{\rho \in R} \rho
\]

- The state \( s \) has a transition to the state \( s' \) if the pair of states \( (s, s') \) lies in the program transition relation \( \rho_R \), i.e., if \( (s, s') \models \rho_R \):

  - \( s : V \rightarrow \bigcup_{x \in V} D_x, \ s(x) \in D_x \) for all \( x \in V \)
  - \( s' : V \rightarrow \bigcup_{x \in V} D_x, \ s'(x) \in D_x \) for all \( x \in V \)
  - \( \beta : V \cup V' \rightarrow \bigcup_{x \in X} D_x \) defined for every \( x \in V \) by
    \[
    \beta(x) = s(x), \ \beta(x') = s'(x)
    \]
    has the property that \( \beta \models \rho_R \).
A program computation is a sequence of states $s_1s_2\ldots$ such that:

- The first element is an initial state, i.e., $s_1 \models \text{Init}$
- Each pair of consecutive states $(s_i, s_{i+1})$ is connected by a program transition, i.e., $(s_i, s_{i+1}) \models \rho \mathcal{R}$.
- If the sequence is finite then the last element does not have any successors i.e., if the last element is $s_n$, then there is no state $s$ such that $(s_n, s) \models \rho \mathcal{R}$.
Example Program

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 of a computation:

(l₁, 1, 3, 2), (l₂, 1, 3, 2), (l₂, 2, 3, 2), (l₂, 3, 3, 2), (l₃, 3, 3, 2), (l₄, 3, 3, 2)

• sequence of transitions ρ₁, ρ₂, ρ₂, ρ₃, ρ₄
• state = tuple of values of program variables pc, x, y, and z
• last program state does not any successors
Correctness: Safety

• a state is reachable if it occurs in some program computation

• a program is safe if no error state is reachable

• ... if and only if no error state lies in $\phi_{reach}$,

\[ \phi_{err} \land \phi_{reach} \models \bot \]

where $\phi_{reach} =$ set of program states which are reachable from some initial state

• ... if and only if no initial state lies in $\phi_{reach}^{-1}$,

\[ Init \land \phi_{reach}^{-1}(\phi_{err}) \models \bot \]

where $\phi_{reach}^{-1}(\phi_{err}) =$ set of program states from which some state in $\phi_{err}$ is reachable
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

Set of reachable states:

$$\phi_{\text{reach}} = (pc = l_1 \lor (pc = l_2 \land y \geq z) \lor (pc = l_3 \land y \geq z \land x \geq y) \lor (pc = l_4 \land y \geq z \land x \geq y))$$
Post operator

Let $\phi$ be a formula over $V$

Let $\rho$ be a formula over $V$ and $V'$

Define a post-condition function $post$ by:

$$post(\phi, \rho) = \exists V'' : \phi[V''/V] \land \rho[V''/V][V/V']$$

An application $post(\phi, \rho)$ computes the image of the set $\phi$ under the relation $\rho$. 
Let $\phi$ be a formula over $V$

Let $\rho$ be a formula over $V$ and $V'$

Define a post-condition function $post$ by:

$$post(\phi, \rho) = \exists V'': \phi[V''/V] \land \rho[V''/V][V/V']$$

An application $post(\phi, \rho)$ computes the image of the set $\phi$ under the relation $\rho$.

$post$ distributes over disjunction wrt. each argument:

- $post(\phi, \rho_1 \lor \rho_2) = post(\phi, \rho_1) \lor post(\phi, \rho_2)$
- $post(\phi_1 \lor \phi_2, \rho) = post(\phi_1, \rho) \lor post(\phi_2, \rho)$
Application of post in example program

Set of states $\phi := (pc = l_2 \land y \geq z)$

Transition relation $\rho := \rho_2$

$$\rho_2 = (move(l_2, l_2) \land x + 1 \leq y \land x' = x + 1 \land \text{skip}(y, z))$$

$$\text{post}(\phi, \rho) = \exists V''(pc = l_2 \land y \geq x)[V''/V] \land \rho_2[V''/V][V/V']$$
$$= \exists V''(pc'' = l_2 \land y'' \geq x'')\land$$
$$\quad (pc'' = l_2 \land pc' = l_2 \land x'' + 1 \leq y'' \land x' = x'' + 1 \land y' = y'' \land z' = z'')$$
$$= \exists V''(pc'' = l_2 \land y'' \geq x'')\land$$
$$\quad (pc'' = l_2 \land pc = l_2 \land x'' + 1 \leq y'' \land x = x'' + 1 \land y = y'' \land z = z'')$$
$$= (pc = l_2 \land y \leq z \land x \leq y)$$
Application of post in example program

Set of states $\phi := (pc = l_2 \land y \geq z)$

Transition relation $\rho := \rho_2$

$$\rho_2 = (\text{move}(l_2, l_2) \land x + 1 \leq y \land x' = x + 1 \land \text{skip}(y, z))$$

$$post(\phi, \rho) = \exists V'' (pc = l_2 \land y \geq x)[V''/V] \land \rho_2[V''/V][V/V']$$

$$= \exists V'' (pc'' = l_2 \land y'' \geq x'') \land$$

$$(pc'' = l_2 \land pc' = l_2 \land x'' + 1 \leq y'' \land x' = x'' + 1 \land y' = y'' \land z' = z'')[V/V']$$

$$= \exists V'' (pc'' = l_2 \land y'' \geq x'') \land$$

$$(pc'' = l_2 \land pc = l_2 \land x'' + 1 \leq y'' \land x = x'' + 1 \land y = y'' \land z = z'')$$

$$= (pc = l_2 \land y \leq z \land x \leq y)$$

[Renamed] program variables:

$V = (pc, x, y, z)$, $V' = (pc', x', y', z')$, $V'' = (pc'', x'', y'', z'')$
Iteration of post

\[ \text{post}^n(\phi, \rho) = n\text{-fold application of post to } \phi \text{ under } \rho \]

\[ \text{post}^n(\phi, \rho) = \begin{cases} 
\phi & \text{if } n = 0 \\
\text{post}(\text{post}^{n-1}(\phi, \rho)), \rho) & \text{otherwise}
\end{cases} \]

Characterize \( \phi_{\text{reach}} \) using iterates of post:

\[ \phi_{\text{reach}} = \text{Init} \lor \text{post} \text{ (Init, } \rho_{\mathcal{R}} \text{)} \lor \text{post} \text{ (post} \text{ (Init, } \rho_{\mathcal{R}} \text{)}, \rho_{\mathcal{R}} \text{)} \lor \ldots \]
\[ = \bigvee_{i \geq 0} \text{post}^i \text{ (Init, } \rho_{\mathcal{R}} \text{)} \]

Disjuncts = iterates for every natural number \( n \) ("\( \omega \)-iteration")
Finite iteration post may suffice

Fixpoint reached in \( n \) steps if \( \bigvee_{i=1}^{n} \text{post}^i(\text{Init}, \rho_R) = \bigvee_{i=1}^{n+1} \text{post}^i(\text{Init}, \rho_R) \)

Then \( \bigvee_{i=1}^{n} \text{post}^i(\text{Init}, \rho_R) = \bigvee_{i \geq 0} \text{post}^i(\text{Init}, \rho_R) \)
Forward reachability analysis

Compute $\bigvee_{i=1}^{n} post^{i}(Init, \rho_{\mathcal{R}})$, $n \geq 0$.

If there exists $m \in \mathbb{N}$ such that

$$\bigvee_{i=0}^{m} post^{i}(Init, \rho_{\mathcal{R}}) = \bigvee_{i=0}^{m+1} post^{i}(Init, \rho_{\mathcal{R}})$$

then fixpoint reached.

Let $\phi_{reach} := \bigvee_{i=1}^{m} post^{i}(Init, \rho_{\mathcal{R}})$

If $\phi_{reach} \cap \phi_{err} = \emptyset$ then safety is guaranteed.
Backward reachability analysis

Another possibility: Start from a bad state and compute states from which the bad state can be reached.

If the initial states are not among these states then safety is guaranteed.
Let $\phi$ be a formula over $V$

Let $\rho$ be a formula over $V$ and $V'$

Define a pre-condition function $pre$ by:

$$pre(\phi, \rho) = \exists V' : \rho \land \phi[V'/V]$$

An application $pre(\phi, \rho)$ computes the preimage of the set $\phi$ under the relation $\rho$.

Computation of $pre^n$ similar.
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

Set of states from which $l_5$ is reachable

$$\phi_{\text{reach}}^{-1} = pc = l_5 \lor (pc = l_3 \land x < z) \lor (pc = l_2 \land x < y \land x + 1 < z) \lor (pc = l_2 \land x < y \land x + 2 < z) \lor \ldots$$