Rigorous Software Development
CSCI-GA 3033-009

Instructor: Thomas Wies

Spring 2013

Lecture 13
Invariant Generation

• Tools such as Dafny enable automated program verification by
  – automatically generating verification conditions and
  – automatically checking validity of the generated VCs.

• The user still needs to provide the invariants.
  – This is often the hardest part.

• Can we generate invariants automatically?
Axiomatic vs. Operational Semantics

\( A \) \( \{A\} P \{B\} \)

semantics

soundness

\( \vdash A \)

completeness

\( \vdash \{A\} P \{B\} \)

theorem proving

\( \vdash \{A\} P \{B\} \)
Programs as Systems of Constraints

1: `assume y ≥ z;
2: while x < y do
   x := x + 1;
3: `assert x ≥ z

\[ ρ_1 = \text{move}(ℓ_1, ℓ_2) \land y ≥ z \land \text{skip}(x,y,z) \]
\[ ρ_2 = \text{move}(ℓ_2, ℓ_2) \land x < y \land x' = x + 1 \land \text{skip}(y,z) \]
\[ ρ_3 = \text{move}(ℓ_2, ℓ_3) \land x ≥ y \land \text{skip}(x,y,z) \]
\[ ρ_4 = \text{move}(ℓ_3, ℓ_{\text{err}}) \land x < z \land \text{skip}(x,y,z) \]
\[ ρ_5 = \text{move}(ℓ_3, ℓ_{\text{exit}}) \land x ≥ z \land \text{skip}(x,y,z) \]

move(ℓ_1, ℓ_2) = pc = ℓ_1 \land pc' = ℓ_2
skip(x_1, ..., x_n) = x_1' = x_1 \land ... \land x_n' = x_n

- $V$ : finite set of program variables
- $\textit{init}$ : initiation condition given by a formula over $V$
- $R$ : a finite set of transition constraints
  - transition constraint $\rho \in R$ given by a formula over $V$ and their primed versions $V'$
  - we often think of $R$ as disjunction of its elements
    $$ R = \rho_1 \lor \ldots \lor \rho_n $$
- $\textit{error}$ : error condition given by a formula over $V$
Programs as Systems of Constraints

\[ P = (V, \text{init}, R, \text{error}) \]
\[ V = \{\text{pc}, x, y, z\} \]
\[ \text{init} = \text{pc} = \ell_1 \]
\[ R = \{\rho_1, \rho_2, \rho_3, \rho_4, \rho_5\} \text{ where} \]
\[ \rho_1 = \text{move}(\ell_1, \ell_2) \land y \geq z \land \text{skip}(x,y,z) \]
\[ \rho_2 = \text{move}(\ell_2, \ell_2) \land x < y \land x' = x + 1 \land \text{skip}(y,z) \]
\[ \rho_3 = \text{move}(\ell_2, \ell_3) \land x \geq y \land \text{skip}(x,y,z) \]
\[ \rho_4 = \text{move}(\ell_3, \ell_{\text{err}}) \land x < z \land \text{skip}(x,y,z) \]
\[ \rho_5 = \text{move}(\ell_3, \ell_{\text{exit}}) \land x \geq z \land \text{skip}(x,y,z) \]
\[ \text{error} = \text{pc} = \ell_{\text{err}} \]
Programs as Transition Systems

- **states** $Q$ are valuations of program variables $V$
- **initial states** $Q_{init}$ are the states satisfying the initiation condition $init$
  \[ Q_{init} = \{ q \in Q \mid q \models init \} \]
- **transition relation** $\rightarrow$ is the relation defined by the transition constraints in $R$
  \[ q_1 \rightarrow q_2 \quad \text{iff} \quad q_1, q_2' \models R \]
- **error states** $Q_{err}$ are the states satisfying the error condition $error$
  \[ Q_{err} = \{ q \in Q \mid q \models error \} \]
Partial Correctness of Programs

• a state $q$ is **reachable** in $P$ if it occurs in some computation of $P$

  \[ q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow \ldots \rightarrow q \quad \text{where } q_0 \in Q_{\text{init}} \]

• denote by $Q_{\text{reach}}$ the set of all reachable states of $P$

• a program $P$ is **safe** if no error state is reachable in $P$

  \[ Q_{\text{reach}} \cap Q_{\text{err}} = \emptyset \]

  or, if $Q_{\text{reach}}$ is expressed as a formula $reach$ over $V$

  \[ \models reach \land error \Rightarrow \text{false} \]
Partial Correctness of Programs

- Initial states: $Q_{init}$
- Reachable states: $Q_{reach}$
- Error states: $Q_{err}$
- State space $Q$
Example: Reachable States of a Program

1: assume $y \geq z$;
2: while $x < y$ do
   $x := x + 1$;
3: assert $x \geq z$

Reachable states

reach = pc = $\ell_1$ \lor
        pc = $\ell_2 \land y \geq z$ \lor
        pc = $\ell_3 \land y \geq z \land x \geq y$ \lor
        pc = $\ell_{\text{exit}} \land y \geq z \land x \geq y$

What is the connection with invariants?
Can we compute reach?
Invariants of Programs

• an invariant $Q_I$ of a program $P$ is a superset of its reachable states:
  \[ Q_{reach} \subseteq Q_I \]

• an invariant $Q_I$ is safe if it does not contain any error states:
  \[ Q_I \land Q_{err} = \emptyset \]
or if $Q_I$ is expressed as a formula $I$ over $V$
  \[ \vdash I \land error \Rightarrow false \]

• $reach$ is the “smallest” invariant of $P$.
• In particular, if $P$ is safe then so is $reach$. 
Partial Correctness of Programs

- Initial states $Q_{init}$
- Reachable states $Q_{reach}$
- Safe invariant $Q_I$
- Error states $Q_{err}$

State space $Q$
Strongest Postconditions

• The strongest postcondition \( post(\rho, A) \) holds for any state \( q \) that is a \( \rho \)-successor state of some state satisfying \( A \):

\[
q' \models post(\rho, A) \quad \text{iff} \quad \exists q \in Q. \ q \models A \land q, q' \models \rho
\]

or equivalently

\[
post(\rho, A) = (\exists V. \ A \land \rho) [V/V']
\]

• Compute \( reach \) by applying \( post \) iteratively to \( init \)
Example: Application of post

- \( A = \text{pc} = \ell_2 \land y \geq z \)
- \( \rho = \text{move}(\ell_2, \ell_2) \land x < y \land x' = x + 1 \land \text{skip}(y, z) \)
- \( \text{post}(\rho, A) \)

\[
= (\exists V. A \land \rho) [V/V']
\]

\[
= (\exists \text{pc x y z. pc} = \ell_2 \land y \geq z \land \text{pc} = \ell_2 \land \text{pc'} = \ell_2 \land x < y \land x' = x + 1 \land y' = y \land z' = z) [\text{pc}/\text{pc'}, x/x', y/y', z/z']
\]

\[
= (y' \geq z' \land \text{pc'} = \ell_2 \land x' - 1 < y') [\text{pc}/\text{pc'}, x/x', y/y', z/z']
\]

\[
= y \geq z \land \text{pc} = \ell_2 \land x \leq y
\]
Iterating post

- $reach^i(\rho, A) = \begin{cases} A, & \text{if } i = 0 \\ post(post^{i-1}(\rho, A)) & \text{if } i > 0 \end{cases}$

- $reach = init \lor post(R, init) \lor post(R, post(R, init)) \lor \ldots = \bigvee_{i \geq 0} post^i(R, init)$

- $i^{th}$ disjunct of $reach$ represents all states reachable from $Q_{init}$ in $i$ computation steps.
Finite iteration of $post$ may suffice

• Fixed point is reached after $n$ steps if

$$\forall i \leq n+1 \quad post^i(R, init) \Rightarrow \forall i \leq n \quad post^i(R, init)$$
Example Iteration

\[ \rho_1 = \text{move}(\ell_1, \ell_2) \land y \geq z \land \text{skip}(x,y,z) \]
\[ \rho_2 = \text{move}(\ell_2, \ell_2) \land x < y \land x' = x + 1 \land \text{skip}(y,z) \]
\[ \rho_3 = \text{move}(\ell_2, \ell_3) \land x \geq y \land \text{skip}(x,y,z) \]
\[ \rho_4 = \text{move}(\ell_3, \ell_{\text{err}}) \land x < z \land \text{skip}(x,y,z) \]
\[ \rho_5 = \text{move}(\ell_3, \ell_{\text{exit}}) \land x \geq z \land \text{skip}(x,y,z) \]

\[ \text{post}^0(R, \text{init}) = \text{init} = \text{pc} = \ell_1 \]
Example Iteration

\[\rho_1 = \text{move}(\ell_1, \ell_2) \wedge y \geq z \wedge \text{skip}(x, y, z)\]
\[\rho_2 = \text{move}(\ell_2, \ell_2) \wedge x < y \wedge x' = x + 1 \wedge \text{skip}(y, z)\]
\[\rho_3 = \text{move}(\ell_2, \ell_3) \wedge x \geq y \wedge \text{skip}(x, y, z)\]
\[\rho_4 = \text{move}(\ell_3, \ell_{\text{err}}) \wedge x < z \wedge \text{skip}(x, y, z)\]
\[\rho_5 = \text{move}(\ell_3, \ell_{\text{exit}}) \wedge x \geq z \wedge \text{skip}(x, y, z)\]

\[\text{post}^2(R, \text{init}) = \text{post}(\rho_2, \text{post}(R, \text{init})) \lor \text{post}(\rho_3, \text{post}(R, \text{init}))\]
\[= \text{pc} = \ell_2 \wedge y \geq z \wedge x \leq y \lor \text{pc} = \ell_3 \wedge y \geq z \wedge x \geq y\]

\[\text{post}^3(R, \text{init}) = \text{post}(\rho_2, \text{post}^2(R, \text{init})) \lor \text{post}(\rho_3, \text{post}^2(R, \text{init})) \lor \text{post}(\rho_4, \text{post}^2(R, \text{init})) \lor \text{post}(\rho_5, \text{post}^2(R, \text{init}))\]
\[= \text{pc} = \ell_2 \wedge y \geq z \wedge x \leq y \lor \text{pc} = \ell_3 \wedge y \geq z \wedge x = y \lor \text{pc} = \ell_{\text{exit}} \wedge y \geq z \wedge x \leq y \lor \text{false}\]
Example Iteration

\[ \text{post}^3(R, \text{init}) = \]
\[ = \text{pc} = \ell_2 \land y \geq z \land x \leq y \lor \text{pc} = \ell_3 \land y \geq z \land x \geq y \lor \]
\[ \text{pc} = \ell_{\text{exit}} \land y \geq z \land x \leq y \]
\[ \text{post}^4(R, \text{init}) = \text{post}^3(R, \text{init}) \]

Fixed point:

\[ \text{reach} \]
\[ = \text{post}^0(R, \text{init}) \lor \text{post}^1(R, \text{init}) \lor \text{post}^2(R, \text{init}) \lor \text{post}^3(R, \text{init}) \]
\[ = \text{pc} = \ell_1 \lor \]
\[ \text{pc} = \ell_2 \land y \geq z \lor \]
\[ \text{pc} = \ell_3 \land y \geq z \land x \geq y \lor \]
\[ \text{pc} = \ell_{\text{exit}} \land y \geq z \land x \leq y \]
Checking Safety

• An inductive invariant $I$ contains the initial states and is closed under successors:
  $$\models init \Rightarrow I \quad \text{and} \quad \models post(R, I) \Rightarrow I$$

• A program is safe if there exists a safe inductive invariant.

• $reach$ is the strongest inductive invariant.
Inductive Invariants for Example Program

• weakest inductive invariant: true
  – set of all states
  – contains error states
• strongest inductive invariant: *reach*
  \[ pc = \ell_1 \lor pc = \ell_2 \land y \geq z \lor \]
  \[ pc = \ell_3 \land y \geq z \land x \geq y \lor pc = \ell_{exit} \land y \geq z \land x \geq y \]
• slightly weaker inductive invariant:
  \[ pc = \ell_1 \lor pc = \ell_2 \land y \geq z \lor \]
  \[ pc = \ell_3 \land y \geq z \land x \geq y \lor pc = \ell_{exit} \]
• Can we drop another conjunct in one of the disjuncts?
Inductive Invariants for Example Program

1: assume $y \geq z$;
2: while $x < y$ do
   $x := x + 1$;
3: assert $x \geq z$

Safe inductive invariant:
$$pc = \ell_1$$
$$pc = \ell_2 \land y \geq z$$
$$pc = \ell_3 \land y \geq z \land x \geq y$$
$$pc = \ell_{exit}$$
Computing Inductive Invariants

- We can compute the strongest inductive invariants by iterating \textit{post} on \textit{init}.
- Can we ensure that this process terminates?
- In general no: checking safety of programs is undecidable.
- But we can compute weaker inductive invariants
  - conservatively abstract the behavior of the program
  - iterate an abstraction of \textit{post} that is guaranteed to terminate.
Abstracting \textit{post}

• instead of iteratively applying post, use over-approximation \textit{post}^# such that always

\[
\text{post}(\rho, F) \models \text{post}^#(\rho, F)
\]

• decompose computation of \textit{post}^# into two steps:
  – first, apply \textit{post} and
  – then, over-approximate the result

• define abstraction function \( \alpha \) such that

\[
F \models \alpha(F)
\]

• for a given abstraction function \( \alpha \) define

\[
\text{post}^#(\rho, F) = \alpha(\text{post}(\rho, F))
\]
Abstracting \textit{reach} by \textit{reach\#}

- instead of computing \textit{reach}, compute \textit{reach\#} such that
  \[
  \text{reach} \models \text{reach\#}
  \]
- check whether \textit{reach\#} contains an error state
  if \( \models \text{reach\#} \land \text{error} \Rightarrow \text{false} \) then
    \( \models \text{reach} \land \text{error} \Rightarrow \text{false} \), i.e. program is safe
- compute \textit{reach\#} by applying iteration
  \[
  \text{reach\#} = \alpha(\text{init}) \lor \text{post\#}(R, \alpha(\text{init})) \lor \text{post\#}(R, \text{post\#}(R, \alpha(\text{init}))) \lor \ldots
  \]
  \[
  = \bigvee_{i \geq 0} (\text{post\#})^i(R, \text{init})
  \]
- consequence: \( \text{reach} \models \text{reach\#} \)
Predicate Abstraction

• construct abstraction $\alpha$ using a given set of building blocks, so-called predicates
• predicate = formula over program variables $V$
• fix finite set of predicates $Preds = \{p_1, ..., p_n\}$
• over-approximate $F$ by conjunction of predicates in $Preds$

$$\alpha(F) = \bigwedge\{ p \in Preds \mid F \models p \}$$

• computation of $\alpha(F)$ requires $n$ theorem prover calls ($n =$ number of predicates)
Predicate Abstraction

\[ p_1 \equiv x \leq 0 \quad p_2 \equiv y > 0 \quad \ldots \]

reachable states
reach

\[ p_1 \wedge p_2 \wedge \ldots \]
\[ x:0, y:5 \]
\[ x:-1, y:3 \]

invariant
reach\#

\[ x:0, y:3 \]
\[ x:1, y:5 \]
\[ \neg p_1 \wedge p_2 \wedge \ldots \]

state space \( Q \)

error states
error
Example: compute
\[ \alpha(\text{pc} = \ell_2 \land y \geq z \land x + 1 \leq y) \]

- \( Preds = \{\text{pc} = \ell_1, \ldots, \text{pc} = \ell_{\text{err}}, y \geq z, x \leq y\} \)

<table>
<thead>
<tr>
<th></th>
<th>pc = \ell_1</th>
<th>pc = \ell_2</th>
<th>pc = \ell_3</th>
<th>pc = \ell_{\text{exit}}</th>
<th>pc = \ell_{\text{err}}</th>
<th>y \geq z</th>
<th>x \leq y</th>
</tr>
</thead>
<tbody>
<tr>
<td>( pc = \ell_2 \land y \geq z \land x + 1 \leq y )</td>
<td>( \not\models )</td>
<td>( \models )</td>
<td>( \not\models )</td>
<td>( \not\models )</td>
<td>( \not\models )</td>
<td>( \models )</td>
<td>( \models )</td>
</tr>
</tbody>
</table>

- result of abstraction = conjunction of implied predicates

\[ \alpha(\text{pc} = \ell_2 \land y \geq z \land x + 1 \leq y) = \text{pc} = \ell_2 \land y \geq z \land x \leq y \]
Trivial Abstraction

• Result of applying predicate abstraction is \textit{true} if none of the predicates is implied by \( F \)
  \[ \alpha(F) = true \]
  “predicates are too specific”

• This is always the case if \( \text{Preds} = \emptyset \)
Algorithm AbstReach

begin
  $\alpha := \lambda F. \land \{ p \in Preds | \models F \Rightarrow p \}$
  $post^# := \lambda \rho F. \alpha(post (\rho, F))$
  $reach^# := \alpha(init)$
  $Tree := \emptyset$
  $Worklist := \{reach^#\}$

  while $Worklist \neq \emptyset$ do
    $F := \text{choose from} Worklist$
    $Worklist := Worklist \setminus \{F\}$
    for each $\rho \in R$ do
      $F' := post^#(\rho, F)$
      if $F' \not\models reach^#$ then
        $reach^# := reach^# \lor F'$
        $Worklist := Worklist \cup \{F'\}$
        $Tree := Tree \cup \{(F', \rho, F)\}$
      end
    end
  end

return $(reach^#, Tree)$
end
Abstract Reachability Graph

- \( Perts = \{false, \text{pc} = \ell_1, ..., \text{pc} = \ell_{\text{err}}, y \geq z, x \leq y\} \)
- nodes \( F_1, ..., F_4 \in Q^\#_{\text{reach}} \)
- labeled edges \( \in Tree \)
- dotted edge: entailment relation (here: \( post^\#(\rho_2, F_2) \models F_2 \))

\( F_1: \text{pc} = \ell_1 \)

\( F_2: \text{pc} = \ell_2 \land y \geq z \)

\( F_3: \text{pc} = \ell_3 \land y \geq z \land x \geq y \)

\( F_4: \text{pc} = \ell_{\text{exit}} \land y \geq z \land x \geq y \)

\( F_1 = \alpha(\text{init}) \)

\( F_2 = post^\#(\rho_1, F_1) \)

\( post^\#(\rho_2, F_2) \models F_2 \)

\( F_3 = post^\#(\rho_3, F_2) \)

\( F_4 = post^\#(\rho_5, F_3) \)
Abstract Reachability Graph

\[ p_1 \equiv x \leq 0 \quad p_2 \equiv y > 0 \quad \ldots \]

reachable states

reach

\[ p_1 \land p_2 \land \ldots \]

invariant

\[ \text{reach}^# \]

\[ x:0, y:5 \]

\[ x := x + 1 \]

\[ x:1, y:5 \]

\[ \neg p_1 \land p_2 \land \ldots \]

state space \( Q \)

error states

error
Example: Computing \( reach^\# \)

- \( Preds = \{ \text{false}, \ pc = \ell_1, \ldots, \ pc = \ell_{\text{err}}, \ y \geq z, \ x \leq y \} \)

- over-approximation of the set of initial states \( init \):
  \[
  F_1 = \alpha(init) = pc = \ell_1
  \]

- apply \( post^\# \) on \( F_1 \) and each program transition \( \rho_i \):
  \[
  F_2 = post^\#(\rho_1, F_1) = \alpha(pc = \ell_2 \land y \geq z) = pc = \ell_2 \land y \geq z
  \]
  \[
  post(\rho_1, F_1)
  \]
  \[
  post^\#(\rho_2, F_1) = \ldots = post^\#(\rho_5, F_1) = \bigwedge\{\text{false}, \ldots\} = \text{false}
  \]
Example: Computing \( {\text{reach}}^\# \)

- application of \( \rho_1, \rho_4, \text{and} \rho_5 \) on \( F_2 \) results in \text{false} (since \( \rho_1, \rho_4, \rho_5 \) are applicable only if \( \text{pc} = \ell_1 \text{ or } \text{pc} = \ell_3 \) holds)

- for \( \rho_2 \) we obtain
  
  \[ \text{post}^\# (\rho_2, F_2) = \alpha(\text{pc} = \ell_2 \land y \geq z \land x \leq y) = \text{pc} = \ell_2 \land y \geq z \land x \leq y \]

  result is \( F_2 \), which is already subsumed by \( {\text{reach}}^\# \)

- for \( \rho_3 \) we obtain
  
  \[ \text{post}^\# (\rho_3, F_2) = \alpha(\text{pc} = \ell_3 \land y \geq z \land x \geq y) \]
  
  \[ = \text{pc} = \ell_3 \land y \geq z \land x \geq y \]
  
  \[ = F_3 \]

  add new node \( F_3 \) to \( {\text{reach}}^\# \), new edge to \( \text{Tree} \)
Example: Computing $reach^#$

- application of $\rho_1$, $\rho_2$, and $\rho_3$ on $F_3$ results in $false$
- for $\rho_5$ we obtain

$$post^# (\rho_5, F_3) = \alpha(pc = \ell_{exit} \land y \geq z \land x \geq y)$$

$$= pc = \ell_{exit} \land y \geq z \land x \geq y$$

$$= F_4$$

new node $F_4$ in $reach^#$, new edge in $Tree$
- for $\rho_4$ (assertion violation) we obtain

$$post^# (\rho_4, F_3) = \alpha(pc = \ell_{err} \land y \geq z \land x \geq y \land x < z) = false$$

- any further application of program transitions does not compute any additional reachable states
- thus, $reach^# = F_1 \lor F_2 \lor F_3 \lor F_4$
- since $reach^# \land pc = \ell_{err} \models false$ the program is proved safe.
Abstract Reachability Graph

with $Preds = \{false, pc = \ell_1, \ldots, pc = \ell_{err}, y \geq z\}$

\begin{align*}
F_1 & : \quad pc = \ell_1 \\
F_2 & : \quad pc = \ell_2 \land y \geq z \\
F_3 & : \quad pc = \ell_3 \land y \geq z \\
F_4 & : \quad pc = \ell_{err} \land y \geq z \\
F_5 & : \quad pc = \ell_{exit} \land y \geq z
\end{align*}

$F_1 = \alpha(init)$
$F_2 = post^#(\rho_1, F_1)$
$\text{post}^#(\rho_2, F_2) \models F_2$
$F_3 = post^#(\rho_3, F_2)$
$F_4 = post^#(\rho_4, F_3)$
$F_5 = post^#(\rho_5, F_3)$
Too Coarse Abstraction

<table>
<thead>
<tr>
<th>reachable states $reach$</th>
<th>invariant $reach^#$</th>
<th>state space $Q$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>error states error</td>
</tr>
</tbody>
</table>
Finding the Right Predicates

• omitting just one predicate (in the example: \( x \geq y \)) may lead to an over-approximation \( \text{reach}^\# \) such that

\[
\text{reach}^\# \land \text{error} \not\equiv \text{false}
\]

that is, algorithm AbstReach fails to prove safety of the program without the predicate \( x \geq y \).

• How can we find the right predicates?
Counterexample Path

- Tree relation records sequence of transitions leading to $F_4$
  - apply $\rho_1$ to $F_1$ and obtain $F_2$
  - apply $\rho_3$ to $F_2$ and obtain $F_3$
  - apply $\rho_4$ to $F_3$ and obtain $F_4$
- counterexample path: sequence of transitions $\rho_1, \rho_3, \rho_4$
- Using this path and the functions $\alpha$ and $\text{post}^\#$ for the current set of predicates we obtain
  $$F_4 = \text{post}^\#(\rho_4, \text{post}^\#(\rho_3, \text{post}^\#(\rho_1, \alpha(\text{init}))))$$
- that is, $F_4$ is the over-approximation of the post-condition computed along the counterexample path.
Analysis of Counterexample Path

- check if the counterexample path also leads to the error states when no over-approximation is applied
- compute

\[
\text{post}(\rho_4, \text{post}(\rho_3, \text{post}(\rho_1, \text{init})))
\]

\[
= \text{post}(\rho_4, \text{post}(\rho_3, \text{pc} = \ell_2 \land y \geq z))
\]

\[
= \text{post}(\rho_4, \text{pc} = \ell_2 \land y \geq z \land x \geq y)
\]

\[
= \text{false}
\]
- by executing the program transitions $\rho_1$, $\rho_3$, and $\rho_4$ it is not possible to reach any error state.
- conclude that the over-approximation is too coarse when dealing with the above path.
Refinement of Abstraction

• need a more precise over-approximation that will prevent $reach^\#$ from including error states.

• need a more precise over-approximation that will prevent $\alpha$ from including states that lead to error states along the path $\rho_1, \rho_3, \rho_4$.

• need a refined abstraction function and a corresponding $post^\#$ such that the execution of AbstReach along the counterexample path does not compute a set of states that contains some error states

$$post^\#(\rho_4, post^\#(\rho_3, post^\#(\rho_1, \alpha(init)))) \land \text{error} \models \text{false}$$
Over-Approximation along Counterexample Path

• goal: \( post^{#}(\rho_4, post^{#}(\rho_3, post^{#}(\rho_1, \alpha(init)))) \) \& error \models false

• find formulas \( F_1, F_2, F_3, F_4 \) such that

\[
\begin{align*}
init & \models F_1 \\
post(\rho_1, F_1) & \models F_2 \\
post(\rho_3, F_2) & \models F_3 \\
post(\rho_4, F_3) & \models F_4 \\
F_4 \& error & \models false
\end{align*}
\]

• thus, \( F_1, \ldots, F_4 \) guarantee that no error state can be reached but may still approximate, i.e., allow additional states

• example choice for \( F_1, \ldots, F_4 \)

\[
\begin{align*}
F_1 &= pc = l_1 \\
F_2 &= pc = l_2 \& y \geq z, \\
F_3 &= pc = l_3 \& x \geq z \\
F_4 &= false
\end{align*}
\]
Refinement of Predicate Abstraction

• given formulas $F_1, F_2, F_3, F_4$ such that

\[
\begin{align*}
\text{init} & \models F_1 \\
\text{post}(\rho_1, F_1) & \models F_2 \\
\text{post}(\rho_3, F_2) & \models F_3 \\
\text{post}(\rho_4, F_3) & \models F_4 \\
F_4 \land \text{error} & \models \text{false}
\end{align*}
\]

• add atoms of $F_1, ..., F_4$ to $\text{Preds}$.

• refinement guarantees that counterexample path $\rho_1, \rho_3, \rho_4$ is eliminated.
CEGAR: Counter-Example Guided Abstraction Refinement Loop

function AbstRefineLoop

begin

    Preds := ∅;
    repeat
        (reach#, Tree) := AbstReach(Preds)
        if exists F ∈ reach# such that F ∧ error \not\equiv false then
            path := MakePath(F, Tree)
            if FeasiblePath(path) then
                return "counterexample path: path"
            else
                Preds := Preds ∪ RefinePath(path)
            end
        else
            return "program is safe"
        end
    end
end
Path Computation

**function** MakePath

**input**
- $F_{err}$ - reachable abstract error state formula
- Tree – abstract reachability tree

**begin**
- path := empty sequence
- $F' := F_{err}$
- while exist F and $\rho$ such that $(F, \rho, F') \in Tree$ do
  - path := $\rho \cdot$ path
  - $F' := F$
- return path

**end**
Feasibility of a Path

function FeasiblePath
    input $\rho_1 \ldots \rho_n$ - path
    begin
        $F := post(\rho_1 \circ \ldots \circ \rho_n, \text{init})$
        if $F \land \text{error} \not\models false$ then
            return true
        else
            return false
        end
    end
Counterexample-Guided
Predicate Discovery

function RefinePath
    input
    $\rho_1 \ldots \rho_n$ – infeasible path
    begin
    $F_1, \ldots, F_{n+1}$ := compute such that
    $init \models F_1$ and
    $post(\rho_1, F_1) \models F_2$ and $\ldots$ $post(\rho_n, F_n \models F_{n+1}$ and
    $F_{n+1} \land error \models false$
    return $\{F_1, \ldots, F_{n+1}\}$
    end

omitted: particular algorithm for finding the $F_1, \ldots, F_{n+1}$