Abstraction

Andreas Podelski

December 12, 2011

abstraction of *post* by $post^{\#}$

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

$$post(\varphi, \rho) \models post^{\#}(\varphi, \rho)$$

- decompose computation of $post^{\#}$ into two steps: first, apply *post* and then, over-approximate result
- define abstraction function α such that always

$$\varphi \models \alpha(\varphi) \ .$$

- for a given abstraction function α, define $post^{\#}$:

$$post^{\#}(\varphi, \rho) \ = \ \alpha(post(\varphi, \rho))$$

abstraction of φ_{reach} by $\varphi^{\#}_{reach}$

- instead of computing φ_{reach},
  compute over-approximation $\varphi^{\#}_{reach}$ such that $\varphi^{\#}_{reach} \supseteq \varphi_{reach}$
- check whether $\varphi^{\#}_{reach}$ contains any error states

  if $\varphi^{\#}_{reach} \wedge \varphi_{err} \models \mathit{false}$
  then $\varphi_{reach} \wedge \varphi_{err} \models \mathit{false}$, i.e., program is safe
- compute $\varphi^{\#}_{reach}$ by applying iteration

$$
\begin{aligned}
\varphi^{\#}_{reach} = {} & \alpha(\varphi_{init}) \vee \\
& \mathit{post}^{\#}(\alpha(\varphi_{init}), \rho_{\mathcal{R}}) \vee \\
& \mathit{post}^{\#}(\mathit{post}^{\#}(\alpha(\varphi_{init}), \rho_{\mathcal{R}}), \rho_{\mathcal{R}}) \vee \dots \\
= {} & \bigvee_{i \geq 0} (\mathit{post}^{\#})^{i}(\alpha(\varphi_{init}), \rho_{\mathcal{R}})
\end{aligned}
$$

- consequence: $\varphi_{reach} \models \varphi^{\#}_{reach}$

predicate abstraction

- construct abstraction $\alpha(\varphi)$ using a given set of building blocks, so-called predicates
- predicate = formula over the program variables V
- fix finite set of predicates $\mathit{Preds} = \{p_1, \ldots, p_n\}$
- over-approximation of φ by conjunction of predicates in *Preds*

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

- computation of $\alpha(\varphi)$ requires n entailment checks (n = number of predicates)

example: compute $\alpha(at\_\ell_2 \wedge y \geq z \wedge x + 1 \leq y)$

- $Preds = \{at\_\ell_1, \dots, at\_\ell_5, y \geq z, x \geq y\}$

1. to compute $\alpha(\varphi)$, check logical consequence between φ and each of the predicates:

	$y \geq z$	$x \geq y$	at_ℓ_1	at_ℓ_2	at_ℓ_3	at_ℓ_4	at_ℓ_5
$at\_\ell_2 \wedge$ $y \geq z \wedge$ $x + 1 \leq y$	$\models$	$\not\models$	$\not\models$	$\models$	$\not\models$	$\not\models$	$\not\models$

2. result of abstraction = conjunction over entailed predicates

$$\alpha\left(\begin{array}{l} at\_\ell_2 \wedge \\ y \geq z \ \wedge \ x + 1 \leq y \end{array} \right) \ = \ at\_\ell_2 \wedge y \geq z$$

trivial abstraction $\alpha(\varphi) = \mathit{true}$

- result of applying predicate abstraction is *true* if none of the predicates is entailed by φ ("predicates are too specific")
  ... always the case if $\mathit{Preds} = \emptyset$

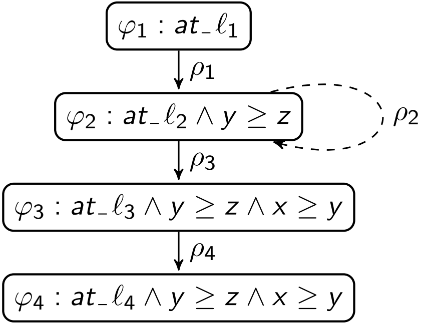
algorithm AbstReach

begin

$\alpha := \lambda\varphi . \bigwedge\{p \in Preds \mid \varphi \models p\}$
$post^{\#} := \lambda(\varphi,\rho) . \alpha(post(\varphi,\rho))$
$ReachStates^{\#} := \{\alpha(\varphi_{init})\}$
$Parent := \emptyset$
$Worklist := ReachStates^{\#}$
while $Worklist \neq \emptyset$ **do**
  $\varphi :=$ choose from $Worklist$
  $Worklist := Worklist \setminus \{\varphi\}$
  for each $\rho \in \mathcal{R}$ **do**
    $\varphi' := post^{\#}(\varphi,\rho)$
    if $\varphi' \notin ReachStates^{\#}$ **then**
      $ReachStates^{\#} := \{\varphi'\} \cup ReachStates^{\#}$
      $Parent := \{(\varphi,\rho,\varphi')\} \cup Parent$
      $Worklist := \{\varphi'\} \cup Worklist$
return $(ReachStates^{\#}, Parent)$

end

Abstract Reachability Graph



$$\varphi_1 = \alpha(\varphi_{init})$$
$$\varphi_2 = post^{\#}(\varphi_1, \rho_1)$$
$$post^{\#}(\varphi_2, \rho_2) \models \varphi_2$$
$$\varphi_3 = post^{\#}(\varphi_2, \rho_3)$$
$$\varphi_4 = post^{\#}(\varphi_3, \rho_4)$$

- $Preds = \{false, at\_\ell_1, \ldots, at\_\ell_5, y \geq z, x \geq y\}$
- nodes $\varphi_1, \ldots, \varphi_4 \in ReachStates^{\#}$
- labeled edges $\in$ *Parent*
- dotted edge : entailment relation (here, $post^{\#}(\varphi_2, \rho_2) \models \varphi_2$)

example: predicate abstraction to compute $\varphi^{\#}_{reach}$

- $Preds = \{false, at\_\ell_1, \dots, at\_\ell_5, y \geq z, x \geq y\}$
- over-approximation of the set of initial states φ_{init}:

$$\varphi_1 = \alpha(at\_\ell_1) = at\_\ell_1$$

- apply $post^{\#}$ on φ_1 wrt. each program transition:

$$\varphi_2 = post^{\#}(\varphi_1, \rho_1) = \alpha(\underbrace{at\_\ell_2 \wedge y \geq z}_{post(\varphi_1, \rho_1)}) = at\_\ell_2 \wedge y \geq z$$

$$post^{\#}(\varphi_1, \rho_2) = \dots = post^{\#}(\varphi_1, \rho_5) = \bigwedge\{false, \dots\} = false$$

apply $post^\#$ to $\varphi_2 \;=\; (at\_\ell_2 \wedge y \geq z)$

- application of ρ_1, ρ_4, and ρ_5 on φ_2 results in *false* (since ρ_1, ρ_4, and ρ_5 are applicable only if either at_ℓ_1 or at_ℓ_3 hold)
- for ρ_2 we obtain

  $$post^\#(\varphi_2, \rho_2) = \alpha(at\_\ell_2 \wedge y \geq z \wedge x \leq y) = at\_\ell_2 \wedge y \geq z$$

  result is φ_2 which is already in $ReachStates^\#$: nothing to do
- for ρ_3 we obtain

  $$\begin{aligned} post^\#(\varphi_2, \rho_3) &= \alpha(at\_\ell_3 \wedge y \geq z \wedge x \geq y) \\ &= at\_\ell_3 \wedge y \geq z \wedge x \geq y \\ &= \varphi_3 \end{aligned}$$

  new node φ_3 in $ReachStates^\#$, new edge in *Parent*

apply $post^{\#}$ to $\varphi_3 = (at\_\ell_3 \wedge y \geq z \wedge x \geq y)$

- application of ρ_1, ρ_2, and ρ_3 on φ_3 results in *false*
- for ρ_4 we obtain:

$$\begin{aligned} post^{\#}(\varphi_3, \rho_4) &= \alpha(at\_\ell_4 \wedge y \geq z \wedge x \geq y \wedge x \geq z) \\ &= at\_\ell_4 \wedge y \geq z \wedge x \geq y \\ &= \varphi_4 \end{aligned}$$

  new node φ_4 in $ReachStates^{\#}$, new edge in *Parent*

- for ρ_5 (assertion violation) we obtain:

$$\begin{aligned} post^{\#}(\varphi_3, \rho_5) &= \alpha(at\_\ell_5 \wedge y \geq z \wedge x \geq y \wedge x + 1 \leq z) \\ &= false \end{aligned}$$

- any further application of program transitions does not compute any additional reachable states
- thus, $\varphi^{\#}_{reach} = \varphi_1 \vee \ldots \vee \varphi_4$
- since $\varphi^{\#}_{reach} \wedge at\_\ell_5 \models false$, the program is proven safe

abstraction $\alpha(\varphi)$

- monotonicity

$$\varphi_1 \models \varphi_2 \quad \text{implies} \quad \alpha(\varphi_1) \models \alpha(\varphi_2)$$

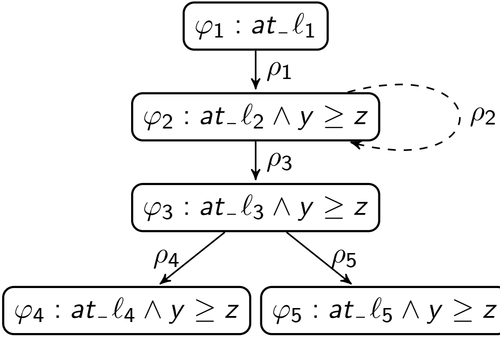
- idempotency

$$\alpha(\alpha(\varphi_1)) = \alpha(\varphi_1)$$

- extensiveness

$$\varphi_1 \models \alpha(\varphi_1)$$

Abstract reachability computation with $Preds = \{false, at\_\ell_1, \dots, at\_\ell_5, y \geq z\}$



$$\varphi_1 = \alpha(\varphi_{init})$$
$$\varphi_2 = post^{\#}(\varphi_1, \rho_1)$$
$$post^{\#}(\varphi_2, \rho_2) \models \varphi_2$$
$$\varphi_3 = post^{\#}(\varphi_2, \rho_3)$$
$$\varphi_4 = post^{\#}(\varphi_3, \rho_4)$$
$$\varphi_5 = post^{\#}(\varphi_3, \rho_5)$$

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

$$\varphi^{\#}_{reach} \wedge \varphi_{err} \not\models \mathit{false}$$

that is, AbstReach without the predicate $x \geq y$ fails to prove safety

counterexample path

- *Parent* relation records sequence leading to φ_5
  - apply ρ_1 to φ_1 and obtain φ_2
  - apply ρ_3 to φ_2 and obtain φ_3
  - apply ρ_5 to φ_3 and obtain φ_5
- counterexample path:
  sequence of program transitions ρ_1, ρ_3, and ρ_5
- Using this path and the functions α and $post^\#$ corresponding to the current set of predicates we obtain

$$\varphi_5 = post^\#(post^\#(post^\#(\alpha(\varphi_{init}), \rho_1), \rho_3), \rho_5)$$

  that is, φ_5 is equal to 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

$$
\begin{aligned}
& post(post(post(\varphi_{init}, \rho_1), \rho_3), \rho_5) \\
= \; & post(post(at\_\ell_2 \land y \geq z, \rho_3), \rho_5) \\
= \; & post(at\_\ell_3 \land y \geq z \land x \geq y, \rho_5) \\
= \; & false \; .
\end{aligned}
$$

- by executing the program transitions ρ_1, ρ_3, and ρ_5 is not possible to reach any error
- conclude that the over-approximation is too coarse when dealing with the above path

need for refinement of abstraction

- need a more precise over-approximation that will prevent $\varphi^{\#}_{reach}$ from including error states

need for refinement of abstraction

- need a more precise over-approximation that will prevent $\varphi^{\#}_{reach}$ from including error states
- need a more precise over-approximation that will prevent α from including states that lead to error states along the path ρ_1, ρ_3, and ρ_5

need for refinement of abstraction

- need a more precise over-approximation that will prevent $\varphi^{\#}_{reach}$ from including error states
- need a more precise over-approximation that will prevent α from including states that lead to error states along the path ρ_1, ρ_3, and ρ_5
- need a refined abstraction function α 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

$$post^{\#}(post^{\#}(post^{\#}(\alpha(\varphi_{init}), \rho_1), \rho_3), \rho_5) \wedge \varphi_{err} \models false \ .$$

over-approximation along counterexample path

- goal:

$$post^{\#}(post^{\#}(post^{\#}(\alpha(\varphi_{init}), \rho_1), \rho_3), \rho_5) \land \varphi_{err} \models \mathit{false} \ .$$

- define sets of states $\psi_1, \ldots, \psi_4$ such that

$$\begin{aligned}
\varphi_{init} &\models \psi_1 \\
post(\psi_1, \rho_1) &\models \psi_2 \\
post(\psi_2, \rho_3) &\models \psi_3 \\
post(\psi_3, \rho_5) &\models \psi_4 \\
\psi_4 \land \varphi_{err} &\models \mathit{false}
\end{aligned}$$

- thus, $\psi_1, \ldots, \psi_4$ guarantee that no error state can be reached
  may approximate / still allow additional states
- example choice for $\psi_1, \ldots, \psi_4$

ψ_1	ψ_2	ψ_3	ψ_4
at_ℓ_1	$at\_\ell_2 \land y \geq z$	$at\_\ell_3 \land x \geq z$	false

refinement of predicate abstraction

- given sets of states $\psi_1, \ldots, \psi_4$ such that

$$\varphi_{init} \models \psi_1$$
$$post(\psi_1, \rho_1) \models \psi_2$$
$$post(\psi_2, \rho_3) \models \psi_3$$
$$post(\psi_3, \rho_5) \models \psi_4$$
$$\psi_4 \wedge \varphi_{err} \models \mathit{false}$$

- add $\psi_1, \ldots, \psi_4$ to the set of predicates *Preds*
- formal property (discussed later) guarantees:

$$\alpha(\varphi_{init}) \models \psi_1$$
$$post^{\#}(\psi_1, \rho_1) \models \psi_2$$
$$post^{\#}(\psi_2, \rho_3) \models \psi_3$$
$$post^{\#}(\psi_3, \rho_5) \models \psi_4$$
$$\psi_4 \wedge \varphi_{err} \models \mathit{false}$$

proves: no error state reachable along path ρ_1, ρ_3, and ρ_5

next . . .

- approach for analysing counterexample computed by ABSTREACH
- algorithms MAKEPATH, FEASIBLEPATH, and REFINEPATH

path computation

function MAKEPATH
input
  ψ - reachable abstract state
  Parent - predecessor relation
begin
  path $:=$ empty sequence
  φ' $:=$ ψ
  while exist φ and ρ such that $(\varphi, \rho, \varphi') \in$ *Parent* **do**
    path $:=$ ρ . *path*
    φ' $:=$ φ
  return *path*
end

path computation

- input: rechable abstract state ψ + *Parent* relation
- view *Parent* as a tree where ψ occurs as a node
- output: sequence of program transitions that labels the tree edges on path from root to ψ
- sequence is constructed iteratively by a backward traversal starting from the input node
- variable *path* keeps track of the construction
- in example, call $\textsc{MakePath}(\varphi_5, \mathit{Parent})$
- *path*, initially empty, is extended with transitions ρ_5, ρ_3, ρ_1
- corresponding edges: $(\varphi_3, \rho_5, \varphi_5)$, $(\varphi_2, \rho_3, \varphi_3)$, $(\varphi_1, \rho_1, \varphi_1)$
- output: $\mathit{path} = \rho_1 \rho_3 \rho_5$

feasibility of a path

function FEASIBLEPATH
input
  $\rho_1 \ldots \rho_n$ - path
begin
  $\varphi \; := \; post(\varphi_{init}, \rho_1 \circ \ldots \circ \rho_n)$
  if $\varphi \wedge \varphi_{err} \not\models$ *false* **then**
    return *true*
  else
    return *false*
end

feasibility of a path

- input: sequence of program transitions $\rho_1 \dots \rho_n$
- checks if there is a computation that produced by this sequence
- check uses the post-condition function and the relational composition of transition
- apply FEASIBLEPATH on example path $\rho_1\rho_3\rho_5$
- relational composition of transitions yields

$$\rho_1 \circ \rho_3 \circ \rho_5 = \mathit{false} \ .$$

- FEASIBLEPATH sets φ to *false* and then returns *false*

counterexample-guided discovery of predicates

function REFINEPATH
input
  $\rho_1 \ldots \rho_n$ - path
begin
  $\varphi_0, \ldots, \varphi_n$:= compute such that
    $(\varphi_{init} \models \varphi_0) \wedge$
    $(post(\varphi_0, \rho_1) \models \varphi_1) \wedge \ldots \wedge (post(\varphi_{n-1}, \rho_n) \models \varphi_n) \wedge$
    $(\varphi_n \wedge \varphi_{err} \models false)$
  return $\{\varphi_0, \ldots, \varphi_n\}$
end

- omitted: particular algorithm for finding $\varphi_0, \ldots, \varphi_n$

counterexample guided discovery of predicates

- input: sequence of program transitions $\rho_1 \ldots \rho_n$
- output: sets of states $\varphi_0, \ldots, \varphi_n$ such that
  - $\varphi_{init} \models \varphi_0$
  - $post(\varphi_{i-1}, \rho_i) \models \varphi_i$
  - $\varphi_n \wedge \varphi_{err} \models false$ for $i \in 1..n$
- if $\varphi_0, \ldots, \varphi_n$ are added to *Preds*
  then the resulting α and $post^\#$ guarantee that

$$
\begin{aligned}
&\alpha(\varphi_{init}) \models \varphi_0 \\
&post^\#(\varphi_0, \rho_1) \models \varphi_1 \\
&\ldots \\
&post^\#(\varphi_{n-1}, \rho_n) \models \varphi_n \\
&\varphi_n \wedge \varphi_{err} \models false\ .
\end{aligned}
$$

- in example, application of RefinePath on $\rho_1\rho_3\rho_5$ yields
  sequence of sets of states $\psi_1, \ldots, \psi_4$

next . . .

- algorithm for counterexample-guided abstraction refinement
- put together all building blocks into an algorithm AbstRefineLoop that verifies safety using predicate abstraction and counterexample guided refinement

predicate abstraction and refinement loop

function AbstRefineLoop
begin
    Preds := $\emptyset$
    repeat
        $(ReachStates^{\#}, Parent)$:= AbstReach(*Preds*)
        if exists $\psi \in ReachStates^{\#}$ such that $\psi \wedge \varphi_{err} \not\models false$
then
            path := MakePath(ψ, *Parent*)
            if FeasiblePath(*path*) **then**
                return "counterexample path: *path* "
            else
                Preds := RefinePath(*path*) $\cup$ *Preds*
        else
            return "program is correct"
end.

algorithm $\textsc{AbstRefineLoop}$

- input: program, output: proof or counterexample
- compute $\varphi^{\#}_{reach}$ using an abstraction defined wrt. set of predicates *Preds* (initially empty)
- over-approximation $\varphi^{\#}_{reach}$: set of formulas $\mathit{ReachStates}^{\#}$ where each formula represents a set of states
- if set of error states disjoint from over-approximation: stop
- otherwise, consider a formula ψ in $\mathit{ReachStates}^{\#}$ that witnesses overlap with error states
- refinement is only possible if overlap is caused by imprecision
- construct *path*, sequence of program transitions leading to ψ
- analyze *path* using $\textsc{FeasiblePath}$
- if *path* feasible: stop
- otherwise (*path* is not feasible), compute a set of predicates that refines the abstraction function

that's it!