Predicate Abstraction

Andreas Podelski
MPI Saarbrücken

Januray 30, 2002

- Goal: verification of safety properties of C programs

- properties reducible to:
  program label "ℓ_{error}" not reachable

- equivalent: no state $\langle \ell_{error}, \ldots \rangle$ in accumulating semantics

- Method:
  compute $acc^{\#}$ abstraction of accumulating semantics
  if no state $\langle \ell_{error}, \ldots \rangle$ in meaning of $acc^{\#}$ then done
  otherwise: don't know

- Idea:
  transform C program P into "abstract" program P' with Boolean variables only

- P' finite-state, hence
  $acc(P')$, the accumulating semantics of P' is computable

- $acc(P') = acc^{\#}(P)$, i.e. the abstract program's semantics is the original program's abstract semantics

- abstract program has, say, n Boolean variables $b_1, \ldots, b_n$

- stand for predicates $p_1, \ldots, p_n$ over states of P

- e.g. p_1 holds of state s if in s, value of program variable z is 0

- similarly, p_2 if values of program variables x and y coincide

- recall that a state maps program variables (including the program counter pc) to values
  state can be written as a tuple, given a fixed order of variables
  e.g. $\langle \ell_{\mathsf{init}}, 1, 1, 0, 13 \rangle$ for order pc, x, y, z, w

- What is the abstraction corresponding to the “source-to-source” transformation $P \mapsto P'$, for a given set of n predicates?

- Once we have define the transformation: can we compute it?

- The abstraction needs to be semantics-based.

- First define the abstract semantics of program P;
  the program P' is just one way to represent abstract semantics.

- Accumulating semantics of P is
  least fixpoint of operator post over sets of states
  $\mathsf{post} : 2^{\mathsf{States}} \to 2^{\mathsf{States}},\ S \mapsto \{s' \in \mathsf{States} \mid \exists s \in S :\ s \rightsquigarrow_P s'\}$

- accumulating semantics of $P' =$
  abstract accumulating semantics of P is
  least fixpoint of operator $\mathsf{post}^\#$ over sets of bitvectors
  $\mathsf{post}^\# : 2^{\{0,1\}^n} \to 2^{\{0,1\}^n},\ B \mapsto \{b' \in \{0,1\}^n \mid \exists b \in B :\ b \rightsquigarrow_{P'} b'\}$

- the question is whether we can compute $\mathsf{post}^\#$

- $\mathsf{post}^{\#} = \alpha \circ \mathsf{post} \circ \gamma$

- $\alpha : 2^{\mathsf{States}} \to 2^{\{0,1\}^n},\ S \mapsto \{\langle p_1(s), \dots, p_n(s)\rangle \mid s \in S\}$

- $\alpha(S) = \bigcap\{B \subseteq \{0,1\}^n \mid \forall s \in S :\ \varphi_B(s)\}$
  where
  $\varphi_B \ \equiv\ \bigvee_{j=1\dots,m} \bigwedge_{i=1,\dots,n} b_{ij} \cdot p_i$ if $B = \{\langle b_1^j, \dots, b_n^j \mid j = 1 \dots, m\rangle\}$

- $\gamma : 2^{\{0,1\}^n} \to 2^{\mathsf{States}},\ B \mapsto \{s \in \mathsf{States} \mid \varphi_B(s)\}$

- elements of concrete domain represented by formulas φ built up from some infinite set of atomic formulas, say in disjunctive normal form
  $\varphi \;\equiv\; \bigvee_{i \in I} \bigwedge_{j \in J} \varphi_{ij}$
  where φ_{ij} not necessarily from given finite set of predicates

- compute $\mathsf{post}^{\#}(B)$ via γ and α

- representation used for $\gamma(B)$: formula φ

- how do we obtain the formula $\varphi' \;\equiv\; \mathsf{post}(\varphi)$

- write C program in form of sequence of conditional goto statements

- if guard then action

- guard = formula in program variables including program counter

- example: $pc = \ell_1 7 \wedge x \neq y$

- action = logical version of imperative-style assignment

- example: x'=x+y

- formula in primed variable x' and unprimed variables x and y translates assigment $x := x + y$

- primed variable stands for value of (unprimed) variable after execution of statement

- $\mathsf{post}(\varphi) = (\exists x, y, z, \ldots (\varphi \wedge \mathsf{guard} \wedge action))[x/x', y/y', z/z', \ldots]$

- to compute formula representing successor states of (set of states represented by) formula φ,

- take its conjunction with guard and action formulas, obtaining a formula in primed and unprimed variables,

- quantify existentially all unprimed variables, obtaining a formula in primed variables only,

- rename all primed variables by their unprimed version, obtaining a formula in unprimed variables only.

- example: $\mathsf{post}(pc = \ell_1 7 \wedge x = 1 \wedge y = 2)$ wrt. guard $pc = \ell_1 7 \wedge x \neq y$ and action $x' = x + y$

- goal: compute $\mathsf{post}^{\#}(B)$ via $\mathsf{post}^{\#} = \alpha \circ \mathsf{post}\gamma$

- $\gamma(B)$ represented by formula φ built up with predicates from $p_1, \ldots, p_n$

- $\varphi' = \mathsf{post}(\varphi)$ new formula, generally not built up from those predicates

- next step: compute $\alpha(\varphi')$

- use $\alpha(S) = \bigcap\{B \subseteq \{0,1\}^n \mid \forall s \in S : \ \varphi_B(s)\}$
  where
  $\varphi_B \ \equiv \ \bigvee_{j=1\ldots,m} \bigwedge_{i=1,\ldots,n} b_{ij} \cdot p_i$ if $B = \{\langle b_1^j, \ldots, b_n^j \mid j = 1 \ldots, m\rangle\}$

- simplified: $\alpha(\varphi') = \bigwedge\{\psi$ built up from $p_1, \ldots, p_n \mid \varphi' \models \psi\}$