Static Program Analysis

Lecture 15: Abstract Interpretation V (Numerical & Predicate Abstraction)

Thomas Noll

Lehrstuhl für Informatik 2
(Software Modeling and Verification)

RWTH AACHEN UNIVERSITY

noll@cs.rwth-aachen.de

http://moves.rwth-aachen.de/teaching/ws-1415/spa/

Winter Semester 2014/15

Oral Exam in Static Program Analysis

- Options:
  - Thu 12 March
  - Tue 24 March
  - Thu 26 March
  - Wed 08 April
- Registration via https://terminplaner2.dfn.de/foodle/Exam-Static-Program-Analysis-54991 (accessible through http://moves.rwth-aachen.de/teaching/ws-1415/spa/)

Outline

Non-Relational Abstraction Domains

In non-relational domains, abstract values are independently referring to single variables:

- Signs (cf. Example 11.3): $\text{sgn}(x) = s$ ($x \in Var$, $s \in \{+, -, 0\}$)
- Intervals (cf. Example 11.4): $x \in J$
  ($x \in Var$, $J \in (\mathbb{Z} \cup \{-\infty\}) \times (\mathbb{Z} \cup \{+\infty\}) \cup \{\emptyset\}$)
- Parities (cf. Example 11.2): $x \in \mathbb{Z}_p$ ($x \in Var$, $p \in \{\text{even}, \text{odd}\}$)
- Congruences (cf. Lemma 14.4): $x \bmod m = k$
  ($x \in Var$, $m > 1$, $k \in \{0, \ldots, m-1\}$)

Observations

- Expressive power:
  - Signs < Intervals (since $+ \cong [1, \infty]$, ...)
  - Parities < Congruences (since x even $\iff$ $x \bmod 2 = 0$, ...)
  - Intervals and Congruences are incomparable
- Congruences can prove disequalities but not inequalities
  - e.g., $x \bmod m \neq y \bmod m \implies$ no zero division in $1/(x - y)$
- Mutual dependencies like $x \leq y$ generally not representable
- Non-relational domains efficient to represent and manipulate

Relational Abstraction Domains

In relational domains, interdependencies between variables are captured:

- Difference Bound Matrices (DBMs):
  conjunctions of $x - y \leq c$ and $\pm x \leq c$ ($x, y \in Var$, $c \in \mathbb{Z}$)
- Octagons: conjunctions of $ax + by \leq c$
  ($x, y \in Var$, $a, b \in \{-1, 0, 1\}$, $c \in \mathbb{Z}$)
- Octahedra: conjunctions of $a_1x_1 + \ldots + a_nx_n \leq c$
  ($x_i \in Var$, $a_i \in \{-1, 0, 1\}$, $c \in \mathbb{Z}$)
- Polyhedra: conjunctions of $a_1x_1 + \ldots + a_nx_n \leq c$
  ($x_i \in Var$, $a_i \in \mathbb{Z}$, $c \in \mathbb{Z}$)

Observations

- Expressive power:
  - DBMs < Octagons < Octahedra < Polyhedra
  - Intervals < DBMs (since $x \in [c_1, c_2] \iff -x \leq -c_1 \wedge x \leq c_2$)
- Can prove inequalities but not (general) disequalities
- Representation and manipulation generally more involved
  - Polyhedra require computation of convex hulls (exponential in $|Var|$)

Combining Non-Relational and Relational Domains

Linear Congruences combine features of Congruences and Polyhedra:

- given by conjunctions of

$$(a_1 x_1 + \ldots + a_n x_n) \bmod m = z$$

  ($x_i \in Var$, $a_i \in \mathbb{Z}$, $m > 1$, $z \in \mathbb{Z}$)

- typical application:

  $2x + 1 \bmod m \neq y \bmod m \implies$ no zero division in $1/(2x + 1 - y)$

- Again usable for proving disequalities but not inequalities

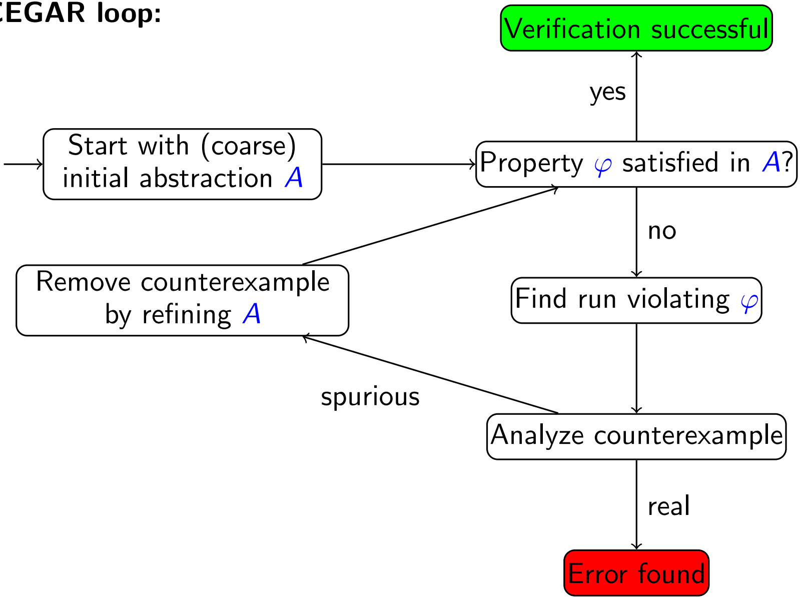
Outline

Abstraction Refinement

- **Problem:** desired program property cannot be shown using current abstraction method
- **Reasons:**
  1. program really violates property or
  2. current abstraction is too coarse
- **Solutions:**
  1. fix the problem
  2. refine abstraction
- **Abstraction refinement:** most successful (automatic) method based on
  - predicate abstraction and
  - analyzing counterexamples
- **Difference** to standard abstract interpretation:
  abstraction parametrised by and specific to program

Counterexample-Guided Abstraction Refinement

CEGAR loop:



Abstraction Refinement for Predicates

1. Extract predicates (i.e., logical formulae) from counterexample
2. Use Galois connection that classifies program states according to validity of predicates (predicate abstraction)
3. Compute new abstract semantics and search for new counterexamples
4. Iterate until property satisfied or real counterexample found (with increasing set of predicates)

Outline

Predicate Abstraction I

Definition 15.1 (Predicate abstraction)

Let Var be a set of variables.

- A predicate is a Boolean expression $p \in BExp$ over Var.
- A state $\sigma \in \Sigma$ satisfies $p \in BExp$ ($\sigma \models p$) if $val_\sigma(p) = \mathrm{true}$.
- p implies q ($p \models q$) if $\sigma \models q$ whenever $\sigma \models p$ (or: p is stronger than q, q is weaker than p).
- p and q are equivalent ($p \equiv q$) if $p \models q$ and $q \models p$.
- Let $P = \{p_1, \ldots, p_n\} \subseteq BExp$ be a finite set of predicates, and let $\neg P := \{\neg p_1, \ldots, \neg p_n\}$. An element of $P \cup \neg P$ is called a literal. The predicate abstraction lattice is defined by:

$$Abs(p_1, \ldots, p_n) := \left(\left\{\bigwedge Q \mid Q \subseteq P \cup \neg P\right\}, \models\right).$$

Abbreviations: $\mathrm{true} := \bigwedge \emptyset$, $\mathrm{false} := \bigwedge \{p_i, \neg p_i, \ldots\}$

Predicate Abstraction II

Lemma 15.2

$Abs(p_1, \ldots, p_n)$ is a complete lattice with

- $\bot = \text{false}$, $\top = \text{true}$
- $Q_1 \sqcap Q_2 = \overline{Q_1 \wedge Q_2}$
- $Q_1 \sqcup Q_2 = \overline{Q_1 \vee Q_2}$ *where* $\overline{b} := \bigwedge\{q \in P \cup \neg P \mid b \models q\}$
  (i.e., strongest formula in $Abs(p_1, \ldots, p_n)$ that is implied by $Q_1 \vee Q_2$)

Example 15.3

Let $P := \{p_1, p_2, p_3\}$.

1. For $Q_1 := p_1 \wedge \neg p_2$ and $Q_2 := \neg p_2 \wedge p_3$, we obtain
$$Q_1 \sqcap Q_2 = \overline{Q_1 \wedge Q_2} \equiv \overline{p_1 \wedge \neg p_2 \wedge p_3}$$
$$Q_1 \sqcup Q_2 = \overline{Q_1 \vee Q_2} \equiv \overline{\neg p_2 \wedge (p_1 \vee p_3)} \equiv \neg p_2$$
2. For $Q_1 := p_1 \wedge p_2$ and $Q_2 := p_1 \wedge \neg p_2$, we obtain
$$Q_1 \sqcap Q_2 = \overline{Q_1 \wedge Q_2} \equiv \overline{\text{false}}$$
$$Q_1 \sqcup Q_2 = \overline{Q_1 \vee Q_2} \equiv \overline{p_1 \wedge (p_2 \vee \neg p_2)} \equiv p_1$$

Predicate Abstraction III

Definition 15.4 (Galois connection for predicate abstraction)

The Galois connection for predicate abstraction is determined by

$$\alpha : 2^\Sigma \to Abs(p_1, \ldots, p_n) \quad \text{and} \quad \gamma : Abs(p_1, \ldots, p_n) \to 2^\Sigma$$

with

$$\alpha(S) := \bigsqcup \{Q_\sigma \mid \sigma \in S\} \quad \text{and} \quad \gamma(Q) := \{\sigma \in \Sigma \mid \sigma \models Q\}$$

where $Q_\sigma := \bigwedge(\{p_i \mid 1 \leq i \leq n, \sigma \models p_i\} \cup \{\neg p_i \mid 1 \leq i \leq n, \sigma \not\models p_i\})$.

Example 15.5

- Let $Var := \{\mathtt{x}, \mathtt{y}\}$
- Let $P := \{p_1, p_2, p_3\}$ where $p_1 := (\mathtt{x<=y})$, $p_2 := (\mathtt{x=y})$, $p_3 := (\mathtt{x>y})$
- If $S = \{\sigma_1, \sigma_2\} \subseteq \Sigma$ with $\sigma_1 = [\mathtt{x} \mapsto 1, \mathtt{y} \mapsto 2]$, $\sigma_2 = [\mathtt{x} \mapsto 2, \mathtt{y} \mapsto 2]$, then

$$\begin{aligned} \alpha(S) &= Q_{\sigma_1} \sqcup Q_{\sigma_2} \\ &= \underline{(p_1 \wedge \neg p_2 \wedge \neg p_3) \sqcup (p_1 \wedge p_2 \wedge \neg p_3)} \\ &= (p_1 \wedge \neg p_2 \wedge \neg p_3) \vee (p_1 \wedge p_2 \wedge \neg p_3) \\ &\equiv p_1 \wedge \neg p_3 \end{aligned}$$

- If $Q = p_1 \wedge \neg p_2 \in Abs(p_1, \ldots, p_n)$, then $\gamma(Q) = \{\sigma \in \Sigma \mid \sigma(\mathtt{x}) < \sigma(\mathtt{y})\}$

Outline

Abstract Semantics for Predicate Abstraction I

Definition 15.6 (Execution relation for predicate abstraction)

If $c \in Cmd$ and $Q \in Abs(p_1, \ldots, p_n)$, then $\langle c, Q \rangle$ is called an abstract configuration. The execution relation for predicate abstraction is defined by the following rules:

$$\text{(skip)}\ \frac{}{\langle \texttt{skip}, Q \rangle \Rightarrow \langle \downarrow, Q \rangle} \qquad \text{(asgn)}\ \frac{}{\langle x \texttt{ := } a, Q \rangle \Rightarrow \langle \downarrow, \bigsqcup\{ Q_{\sigma[x \mapsto val_\sigma(a)]} \mid \sigma \models Q \} \rangle}$$

$$\text{(seq1)}\ \frac{\langle c_1, Q \rangle \Rightarrow \langle c_1', Q' \rangle \quad c_1' \neq \downarrow}{\langle c_1 ; c_2, Q \rangle \Rightarrow \langle c_1' ; c_2, Q' \rangle} \qquad \text{(seq2)}\ \frac{\langle c_1, Q \rangle \Rightarrow \langle \downarrow, Q' \rangle}{\langle c_1 ; c_2, Q \rangle \Rightarrow \langle c_2, Q' \rangle}$$

$$\text{(if1)}\ \frac{}{\langle \texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2, Q \rangle \Rightarrow \langle c_1, \overline{Q \wedge b} \rangle}$$

$$\text{(if2)}\ \frac{}{\langle \texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2, Q \rangle \Rightarrow \langle c_2, \overline{Q \wedge \neg b} \rangle}$$

$$\text{(wh1)}\ \frac{}{\langle \texttt{while } b \texttt{ do } c, Q \rangle \Rightarrow \langle c ; \texttt{while } b \texttt{ do } c, \overline{Q \wedge b} \rangle}$$

$$\text{(wh2)}\ \frac{}{\langle \texttt{while } b \texttt{ do } c, Q \rangle \Rightarrow \langle \downarrow, \overline{Q \wedge \neg b} \rangle}$$

Abstract Semantics for Predicate Abstraction II

Remarks:

- In Rule (asgn), $\bigsqcup\{Q_{\sigma[x \mapsto val_\sigma(a)]} \mid \sigma \models Q\}$ denotes the strongest postcondition of Q w.r.t. statement $x := a$. It covers all states that are obtained from a state satisfying Q by applying the assignment $x := a$:

  $$\begin{array}{llcl} \text{Abstract:} & \langle x := a, Q\rangle & \Rightarrow & \langle \downarrow, \bigsqcup\{Q_{\sigma[x \mapsto val_\sigma(a)]} \mid \sigma \models Q\}\rangle \\ & \downarrow \gamma & & \uparrow \alpha \\ \text{Concrete:} & \langle x := a, \{\sigma \in \Sigma \mid \sigma \models Q\}\rangle & \rightarrow & \langle \downarrow, \{\sigma[x \mapsto val_\sigma(a)] \mid \sigma \models Q\}\rangle \end{array}$$

- An abstract configuration of the form $\langle c, \mathsf{false}\rangle$ represents an unreachable configuration (as there is no $\sigma \in \Sigma$ such that $\sigma \models \mathsf{false}$) and can therefore be omitted
- If $P = \emptyset$ (and thus $Abs(P) = \{\mathsf{true}, \mathsf{false}\}$) and if no $b \in BExp_c$ is a tautology or contradiction (i.e., resp. equivalent to true or false), then the abstract transition system corresponds to the control flow graph of c

Abstract Semantics for Predicate Abstraction III

Example 15.7

```
if [x > y]^1 then
  while [¬(y = 0)]^2 do
    [x := x - 1;]^3;
    [y := y - 1;]^4;
  if [x > y]^5 then
    [skip]^6;
  else
    [skip]^7;
else
  [skip]^8;
```

- **Claim:** label 7 not reachable (as `x > y` is a loop invariant)
- **Proof:** by predicate abstraction with $p_1 := (x > y)$ and $p_2 := (x >= y)$
- **Abstract transition system:** on the board
- **Remark:** $p_1 := (x > y)$ alone not sufficient to prove loop invariant (as not necessarily valid after label 3)