Static Program Analysis

Lecture 17: Abstract Interpretation VI (Predicate Abstraction)

Thomas Noll

Lehrstuhl für Informatik 2
(Software Modeling and Verification)

RWTH Aachen University

noll@cs.rwth-aachen.de

http://www-i2.informatik.rwth-aachen.de/i2/spa11/

Summer Semester 2011

Informatik Sommerfest

Im Informatikzentrum
Ahornstraße 55

1. Juli 2011

16:00	**Beginn**
16:30	**Festakt** (Aula 2) **Verleihung der Zeugnisse**
danach	**Geselliges Feiern** (Parkplatz, Foyer E2)

INFORMATIK
RWTHAACHEN

Outline

Abstraction Refinement

- **Problem:** desired program property cannot be shown using current abstraction method

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

Abstraction Refinement

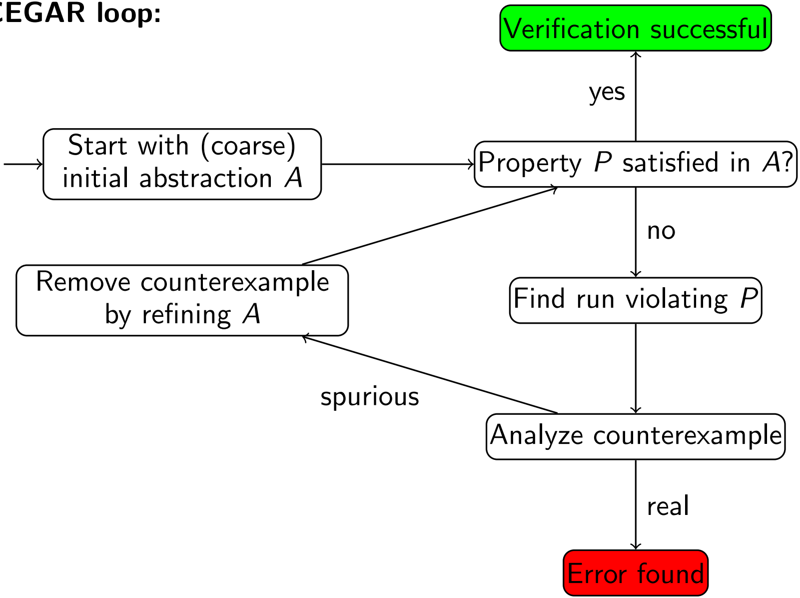
- **Problem:** desired program property cannot be shown using current abstraction method
- **Reasons:**
  1. program really violates property or
  2. current abstraction is too coarse
- **Solutions:**
  1. fix the problem
  2. refine abstraction

Abstraction Refinement

- **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
  - counterexamples

Counterexample-Guided Abstraction Refinement

CEGAR loop:



Abstraction Refinement for Predicates

1. Extract predicates (i.e., logical formulae) from counterexample

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)

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

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 17.1 (Predicate abstraction)

Let Var be a set of variables.

- A predicate is a Boolean expression $p \in BExp$ over Var.

Predicate Abstraction I

Definition 17.1 (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 $val_\sigma(p) = \text{true}$.

Definition 17.1 (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 $val_\sigma(p) = \text{true}$.
- 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).

Predicate Abstraction I

Definition 17.1 (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 $val_\sigma(p) = \mathsf{true}$.
- 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 $q \models p$.

Predicate Abstraction I

Definition 17.1 (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:

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

Predicate Abstraction I

Definition 17.1 (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 $val_\sigma(p) = \text{true}$.
- 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: true $:= \bigwedge \emptyset$, false $:= \ldots \wedge p_i \wedge \ldots \wedge \neg p_i \wedge \ldots$

Predicate Abstraction II

Lemma 17.2

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

- $\bot =$ false, $\top =$ true
- $q_1 \sqcap q_2 = q_1 \wedge q_2$
- $q_1 \sqcup q_2 = \overline{q_1 \vee q_2}$ *where* $\overline{b} := \bigwedge\{q \in Abs(p_1, \ldots, p_n) \mid b \models q\}$
  (i.e., strongest formula in $Abs(p_1, \ldots, p_n)$ that is implied by $q_1 \vee q_2$)

Predicate Abstraction II

Lemma 17.2

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

- $\bot = \text{false}$, $\top = \text{true}$
- $q_1 \sqcap q_2 = q_1 \wedge q_2$
- $q_1 \sqcup q_2 = \overline{q_1 \vee q_2}$ *where* $\overline{b} := \bigwedge\{q \in Abs(p_1, \ldots, p_n) \mid b \models q\}$
  (i.e., strongest formula in $Abs(p_1, \ldots, p_n)$ that is implied by $q_1 \vee q_2$)

Example 17.3

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

$$\begin{aligned} q_1 \sqcap q_2 &= q_1 \wedge q_2 \equiv p_1 \wedge \neg p_2 \wedge p_3 \\ q_1 \sqcup q_2 &= \overline{q_1 \vee q_2} \equiv \overline{\neg p_2 \wedge (p_1 \vee p_3)} \equiv \neg p_2 \end{aligned}$$

Predicate Abstraction II

Lemma 17.2

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

- $\bot = \text{false}$, $\top = \text{true}$
- $q_1 \sqcap q_2 = q_1 \wedge q_2$
- $q_1 \sqcup q_2 = \overline{q_1 \vee q_2}$ *where* $\overline{b} := \bigwedge\{q \in Abs(p_1, \ldots, p_n) \mid b \models q\}$
  (i.e., strongest formula in $Abs(p_1, \ldots, p_n)$ that is implied by $q_1 \vee q_2$)

Example 17.3

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
$$q_1 \sqcap q_2 = q_1 \wedge q_2 \equiv p_1 \wedge \neg p_2 \wedge p_3$$
$$q_1 \sqcup q_2 = \overline{q_1 \vee q_2} \equiv \overline{\neg p_2 \wedge (p_1 \vee p_3)} \equiv \neg p_2$$
2. For $q_1 := p_1 \wedge p_2$ and $q_2 := p_1 \wedge \neg p_2$, we obtain
$$q_1 \sqcap q_2 = q_1 \wedge q_2 \equiv \text{false}$$
$$q_1 \sqcup q_2 = \overline{q_1 \vee q_2} \equiv \overline{p_1 \wedge (p_2 \vee \neg p_2)} \equiv p_1$$

Predicate Abstraction III

Definition 17.4 (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 $q_\sigma := \bigwedge(\{p_i \mid 1 \leq i \leq n, \sigma \models p_i\} \cup \{\neg p_i \mid 1 \leq i \leq n, \sigma \not\models p_i\})$.

Predicate Abstraction III

Definition 17.4 (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 17.5

- Let $Var := \{\mathtt{x}, \mathtt{y}\}$
- Let $P := \{p_1, p_2, p_3\}$ where $p_1 := (\mathtt{x<=y})$, $p_2 := (\mathtt{x=y})$, $p_3 := (\mathtt{x>y})$

Predicate Abstraction III

Definition 17.4 (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 $q_\sigma := \bigwedge(\{p_i \mid 1 \le i \le n, \sigma \models p_i\} \cup \{\neg p_i \mid 1 \le i \le n, \sigma \not\models p_i\})$.

Example 17.5

- 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

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

Predicate Abstraction III

Definition 17.4 (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 17.5

- 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
$$\begin{aligned} \alpha(S) &= q_{\sigma_1} \sqcup q_{\sigma_2} \\ &= \underline{(p_1 \wedge \neg p_2 \wedge \neg p_3) \sqcup (p_1 \wedge p_2 \wedge \neg p_3)} \\ &\equiv (p_1 \wedge \neg p_2 \wedge \neg p_3) \vee (p_1 \wedge p_2 \wedge \neg p_3) \\ &\equiv p_1 \wedge \neg p_3 \end{aligned}$$
- 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 17.6 (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:

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

$$\text{(seq1)}\ \frac{\langle c_1, q \rangle \Rightarrow \langle c_1', q' \rangle}{\langle c_1 \texttt{;} c_2, q \rangle \Rightarrow \langle c_1' \texttt{;} c_2, q' \rangle} \qquad \text{(seq2)}\ \frac{\langle c_1, q \rangle \Rightarrow q'}{\langle c_1 \texttt{;} c_2, q \rangle \Rightarrow \langle c_2, q' \rangle}$$

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

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

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

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

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:

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

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:

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

- In Rules (if1, (if2), (wh1), (wh2): if $b = p_i$ for some $i \in \{1, \ldots, n\}$, then $q \wedge [\neg]b \in Abs(p_1, \ldots, p_n)$, and thus $\overline{q \wedge [\neg]b} = q \wedge [\neg]b$

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:

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

- In Rules (if1, (if2), (wh1), (wh2): if $b = p_i$ for some $i \in \{1, \ldots, n\}$, then $q \wedge [\neg]b \in Abs(p_1, \ldots, p_n)$, and thus $\overline{q \wedge [\neg]b} = q \wedge [\neg]b$
- An abstract configuration of the form $\langle c, \text{false}\rangle$ represents an unreachable configuration (as there is no $\sigma \in \Sigma$ such that $\sigma \models$ false) and can therefore be omitted

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:

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

- In Rules (if1, (if2), (wh1), (wh2): if $b = p_i$ for some $i \in \{1, \ldots, n\}$, then $q \wedge [\neg]b \in Abs(p_1, \ldots, p_n)$, and thus $\overline{q \wedge [\neg]b} = q \wedge [\neg]b$
- An abstract configuration of the form $\langle c, \text{false}\rangle$ represents an unreachable configuration (as there is no $\sigma \in \Sigma$ such that $\sigma \models \text{false}$) and can therefore be omitted
- If $P = \emptyset$ (and thus $Abs(P) = \{\text{true}, \text{false}\}$) and if no $b \in BExp_c$ is a contradiction, then the abstract transition system corresponds to the control flow graph of c

Abstract Semantics for Predicate Abstraction III

Example 17.7

```
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;
```

Example 17.7

```
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 $p_1 := (\texttt{x > y})$ and $p_2 := (\texttt{x >= y})$
- **Abstract transition system:** on the board

Abstract Semantics for Predicate Abstraction III

Example 17.7

```
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 (as not necessarily valid after label 3)

Abstract Semantics for Predicate Abstraction IV

Problem: q' generally not computable in

$$\text{(asgn)}\,\frac{}{\langle x \texttt{ := } a, q\rangle \Rightarrow \underbrace{\bigsqcup\{q_{\sigma[x \mapsto val_\sigma(a)]} \mid \sigma \models q\}}_{q'}}$$

(due to undecidability of implication in certain logics)

Abstract Semantics for Predicate Abstraction IV

Problem: q' generally not computable in

$$\text{(asgn)}\ \frac{}{\langle x \texttt{ := } a, q\rangle \Rightarrow \underbrace{\bigsqcup\{q_{\sigma[x\mapsto val_\sigma(a)]} \mid \sigma \models q\}}_{q'}}$$

(due to undecidability of implication in certain logics)

Solutions:

- Over-approximation: fall back to non-strongest postconditions
  - in practice, (automatic) theorem proving
  - for every $i \in \{1, \ldots, n\}$, try to prove $q' \models p_i$ and $q' \models \neg p_i$
  - approximate q' by conjunction of all provable literals

Abstract Semantics for Predicate Abstraction IV

Problem: q' generally not computable in

$$\text{(asgn)}\ \frac{}{\langle x := a, q\rangle \Rightarrow \underbrace{\bigsqcup\{q_{\sigma[x\mapsto val_\sigma(a)]} \mid \sigma \models q\}}_{q'}}$$

(due to undecidability of implication in certain logics)

Solutions:

- Over-approximation: fall back to non-strongest postconditions
  - in practice, (automatic) theorem proving
  - for every $i \in \{1, \ldots, n\}$, try to prove $q' \models p_i$ and $q' \models \neg p_i$
  - approximate q' by conjunction of all provable literals
- Restriction of programs:
  - $\models$ decidable for certain logics
  - example: Presburger arithmetic (first-order theory of $\mathbb{N}$ with $+$)
  - thus q' computable for WHILE programs without multiplication

Abstract Semantics for Predicate Abstraction IV

Problem: q' generally not computable in

$$\text{(asgn)}\ \frac{}{\langle x := a, q\rangle \Rightarrow \underbrace{\bigsqcup\{q_{\sigma[x \mapsto val_\sigma(a)]} \mid \sigma \models q\}}_{q'}}$$

(due to undecidability of implication in certain logics)

Solutions:

- Over-approximation: fall back to non-strongest postconditions
  - in practice, (automatic) theorem proving
  - for every $i \in \{1, \ldots, n\}$, try to prove $q' \models p_i$ and $q' \models \neg p_i$
  - approximate q' by conjunction of all provable literals
- Restriction of programs:
  - $\models$ decidable for certain logics
  - example: Presburger arithmetic (first-order theory of $\mathbb{N}$ with $+$)
  - thus q' computable for WHILE programs without multiplication
- Restriction to finite domains:
  - for example, binary numbers of fixed size
  - thus everything (domain, Galois connection, ...) exactly computable
  - problem: exponential blowup $\Longrightarrow$ solution: Binary Decision Diagrams