Static Program Analysis

Lecture 14: Abstract Interpretation V (Predicate Abstraction)

Summer Semester 2018

Thomas Noll
Software Modeling and Verification Group
RWTH Aachen University

https://moves.rwth-aachen.de/teaching/ss-18/spa/

Software Modeling and Verification Chair

RWTH AACHEN UNIVERSITY

Outline of Lecture 14

Recap: Abstraction of 16-Bit Multiplication

Abstract Interpretation of Multiplier

Example (Abstraction of 16-bit multiplier; cf. Example 13.3)

Abstract execution for

- $\texttt{f1} = 101_2\ (= 5)$, $\texttt{f2} = 1001010_2\ (= 74)$
- $\texttt{out}$, $\texttt{ovf}$ with arbitrary initial values
- $m = 5$

$$\vdots$$
$$\Downarrow$$
$$\langle [(\texttt{ovf},\texttt{out}) := (\texttt{out} : 17) + \texttt{f2}]^5; \ldots, \{[\texttt{f1} \mapsto 101_2, \texttt{f2} \mapsto 4, \texttt{out} \mapsto 0, \texttt{ovf} \mapsto 0]\}\rangle$$
$$\Downarrow$$
$$\langle \ldots, \{[\texttt{f1} \mapsto 101_2, \texttt{f2} \mapsto 4, \texttt{out} \mapsto 4, \texttt{ovf} \mapsto 0], \rangle$$
$$[\texttt{f1} \mapsto 101_2, \texttt{f2} \mapsto 4, \texttt{out} \mapsto 3, \texttt{ovf} \mapsto 1]\}$$
$$\Downarrow$$
$$\vdots$$

since $\texttt{out}' = (\texttt{out} + \texttt{f2} - 2^{16}) \bmod 5 = (0 + 4 - 1) \bmod 5 = 3$ in case of overflow

Recap: Abstraction of 16-Bit Multiplication

Extending the Specification

- **Observation:** the soundness property

$$\langle c, \sigma \rangle \to^+ \langle \downarrow, \sigma' \rangle \implies \sigma'(\texttt{out}) = \sigma(\texttt{f1}) \cdot \sigma(\texttt{f2}) \vee \sigma'(\texttt{ovf}) = 1$$

  is easily satisfied by always setting $\sigma'(\texttt{ovf}) = 1$
- Additionally required to ensure correctness in absence of overflows: reasoning about sizes (of bit string representations) of numbers

Recap: Abstraction of 16-Bit Multiplication

Extending the Specification

- **Observation:** the soundness property

$$\langle c, \sigma \rangle \to^+ \langle \downarrow, \sigma' \rangle \implies \sigma'(\texttt{out}) = \sigma(\texttt{f1}) \cdot \sigma(\texttt{f2}) \vee \sigma'(\texttt{ovf}) = 1$$

  is easily satisfied by always setting $\sigma'(\texttt{ovf}) = 1$
- Additionally required to ensure correctness in absence of overflows: reasoning about sizes (of bit string representations) of numbers

Logarithmic abstraction

- For $n \in \mathbb{N}$, let $s(n) := \lceil \log_2(n+1) \rceil$ be the minimal number of bits required to represent n (where $s(n) = 0$)
- It holds: $s(m \cdot n) \begin{cases} \leq 16 & \text{if } s(m) + s(n) \leq 16 \\ \geq 17 & \text{if } s(m) + s(n) \geq 18 \end{cases}$
- Strengthened (and satisfied) specification: if $\langle c, \sigma \rangle \to^+ \langle \downarrow, \sigma' \rangle$, then

$$\begin{cases} \sigma'(\texttt{out}) = \sigma(\texttt{f1}) \cdot \sigma(\texttt{f2}) \wedge \sigma'(\texttt{ovf}) = 0 & \text{if } s(\sigma(\texttt{f1})) + s(\sigma(\texttt{f2})) \leq 16 \\ \sigma'(\texttt{ovf}) = 1 & \text{if } s(\sigma(\texttt{f1})) + s(\sigma(\texttt{f2})) \geq 18 \\ \sigma'(\texttt{out}) = \sigma(\texttt{f1}) \cdot \sigma(\texttt{f2}) \vee \sigma'(\texttt{ovf}) = 1 & \text{otherwise} \end{cases}$$

Outline of Lecture 14

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

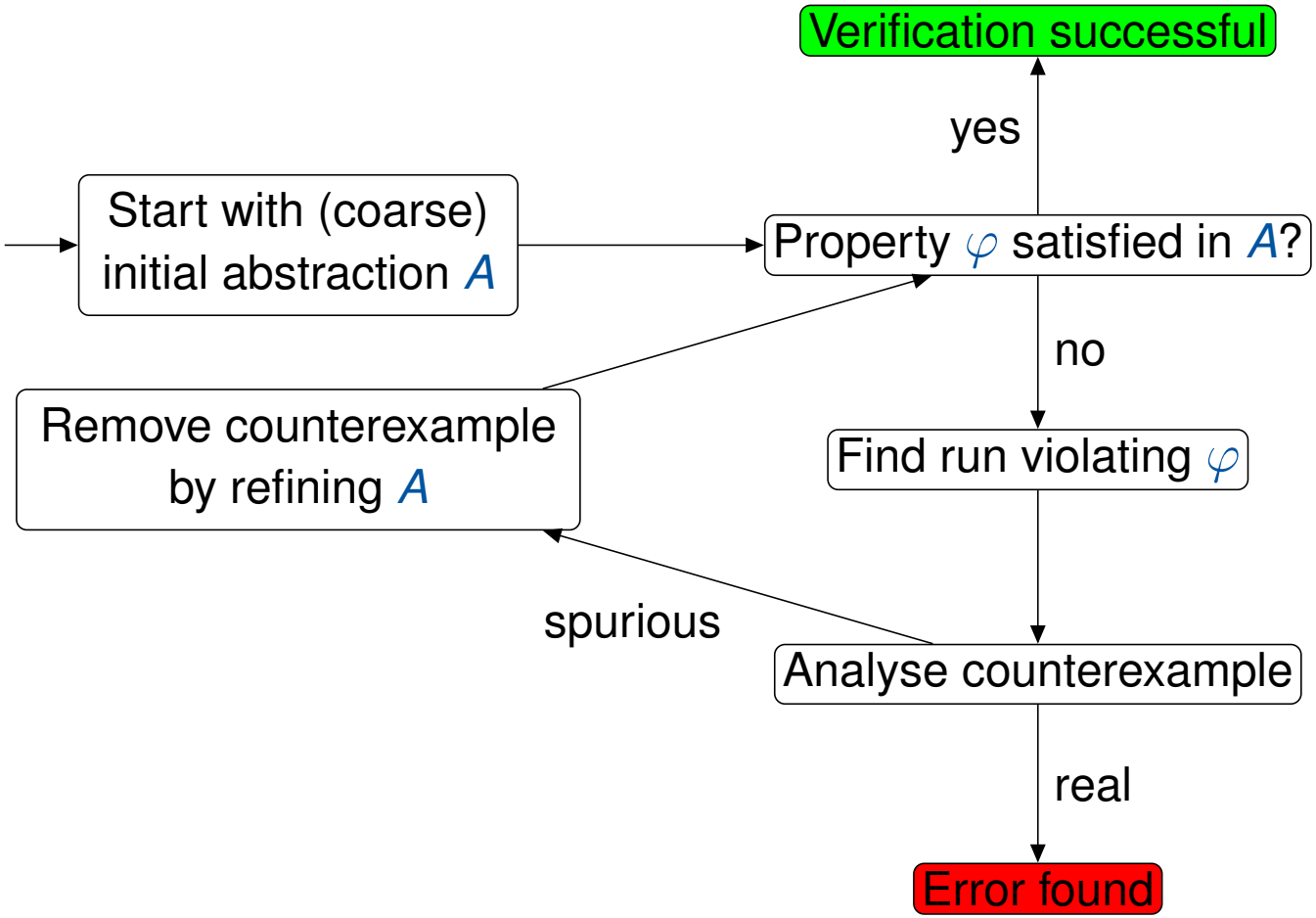
Abstraction Refinement

- **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
  - analysing counterexamples

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
  - analysing counterexamples
- **Difference** to standard abstract interpretation with fixed domain:
  abstraction parametrised by and specific to program

Counterexample-Guided Abstraction Refinement (CEGAR)



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; can entail non-termination)

Outline of Lecture 14

Predicate Abstraction I

Definition 14.1 (Predicate abstraction)

Let Var be a set of variables.

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

Predicate Abstraction I

Definition 14.1 (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 $val_\sigma(p) = \text{true}$.

Predicate Abstraction I

Definition 14.1 (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 14.1 (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 $q \models p$.

Predicate Abstraction I

Definition 14.1 (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 $q \models p$.
- p and q are independent if $p \not\models q$ and $q \not\models p$, otherwise interdependent.

Predicate Abstraction I

Definition 14.1 (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 $q \models p$.
- p and q are independent if $p \not\models q$ and $q \not\models p$, otherwise interdependent.
- 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:

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

Predicate Abstraction I

Definition 14.1 (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 $q \models p$.
- p and q are independent if $p \not\models q$ and $q \not\models p$, otherwise interdependent.
- 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:

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

Abbreviation: $\text{true} := \bigwedge \emptyset$ (and $\text{false} \equiv \bigwedge \{p_i, \neg p_i, \ldots\}$ if $P \neq \emptyset$)

Predicate Abstraction II

Lemma 14.2

$Abs(P)$ is a complete lattice with

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

Predicate Abstraction II

Lemma 14.2

$Abs(P)$ is a complete lattice with

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

Example 14.3

Let $P := \{p_1, p_2, p_3\}$ with $p_1 := (\mathrm{x} = 1)$, $p_2 := (\mathrm{y} = 2)$, $p_1 := (\mathrm{z} = 3)$ (indep.).

1. For $Q_1 := p_1 \wedge \neg p_2$ and $Q_2 := \neg p_2 \wedge p_3$, we obtain

$$Q_1 \sqcap Q_2 = \overline{Q_1 \wedge Q_2} = p_1 \wedge \neg p_2 \wedge p_3 \qquad (= \bigwedge(Q_1 \cup Q_2))$$
$$Q_1 \sqcup Q_2 = \overline{Q_1 \vee Q_2} \equiv \overline{\neg p_2 \wedge (p_1 \vee p_3)} = \neg p_2 \qquad (= \bigwedge(Q_1 \cap Q_2))$$

Predicate Abstraction II

Lemma 14.2

$Abs(P)$ is a complete lattice with

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

Example 14.3

Let $P := \{p_1, p_2, p_3\}$ with $p_1 := (\mathrm{x} = 1)$, $p_2 := (\mathrm{y} = 2)$, $p_1 := (\mathrm{z} = 3)$ (indep.).

1. For $Q_1 := p_1 \wedge \neg p_2$ and $Q_2 := \neg p_2 \wedge p_3$, we obtain

$$Q_1 \sqcap Q_2 = \overline{Q_1 \wedge Q_2} = p_1 \wedge \neg p_2 \wedge p_3 \qquad (= \bigwedge(Q_1 \cup Q_2))$$
$$Q_1 \sqcup Q_2 = \overline{Q_1 \vee Q_2} \equiv \overline{\neg p_2 \wedge (p_1 \vee p_3)} = \neg p_2 \qquad (= \bigwedge(Q_1 \cap Q_2))$$

2. For $Q_1 := p_1 \wedge p_2$ and $Q_2 := p_1 \wedge \neg p_2$, we obtain

$$Q_1 \sqcap Q_2 = \overline{Q_1 \wedge Q_2} = \text{false} \qquad (= \bigwedge(Q_1 \cup Q_2))$$
$$Q_1 \sqcup Q_2 = \overline{Q_1 \vee Q_2} \equiv \overline{p_1 \wedge (p_2 \vee \neg p_2)} = p_1 \qquad (= \bigwedge(Q_1 \cap Q_2))$$

Predicate Abstraction III

Lemma 14.4

1. *If predicates in P are pairwise independent, then*
   - $Q_1 \sqcap Q_2 \left(= \overline{Q_1 \wedge Q_2}\right) = \bigwedge(Q_1 \cup Q_2)$
   - $Q_1 \sqcup Q_2 \left(= \overline{Q_1 \vee Q_2}\right) = \bigwedge(Q_1 \cap Q_2)$
2. *Otherwise,*
   - $Q_1 \sqcap Q_2 \left(= \overline{Q_1 \wedge Q_2}\right) \neq \bigwedge(Q_1 \cup Q_2)$ *(but* $\equiv \bigwedge(Q_1 \cup Q_2)$*)*
   - $Q_1 \sqcup Q_2 \left(= \overline{Q_1 \vee Q_2}\right) \neq \bigwedge(Q_1 \cap Q_2)$ *(and* $\not\equiv \bigwedge(Q_1 \cap Q_2)$*)*

Predicate Abstraction III

Lemma 14.4

1. *If predicates in P are pairwise independent, then*
   - $Q_1 \sqcap Q_2 \left(= \overline{Q_1 \wedge Q_2}\right) = \bigwedge(Q_1 \cup Q_2)$
   - $Q_1 \sqcup Q_2 \left(= \overline{Q_1 \vee Q_2}\right) = \bigwedge(Q_1 \cap Q_2)$
2. *Otherwise,*
   - $Q_1 \sqcap Q_2 \left(= \overline{Q_1 \wedge Q_2}\right) \neq \bigwedge(Q_1 \cup Q_2)$ *(but* $\equiv \bigwedge(Q_1 \cup Q_2)$*)*
   - $Q_1 \sqcup Q_2 \left(= \overline{Q_1 \vee Q_2}\right) \neq \bigwedge(Q_1 \cap Q_2)$ *(and* $\not\equiv \bigwedge(Q_1 \cap Q_2)$*)*

Example 14.5

1. - $p_1 := (\texttt{x <= y})$, $p_2 := (\texttt{x >= y})$, $p_3 := (\texttt{x = y})$
   - $Q_1 := p_1$, $Q_2 := p_2$

   $\Rightarrow Q_1 \sqcap Q_2 = \overline{Q_1 \wedge Q_2} = p_1 \wedge p_2 \wedge p_3 \not\equiv \bigwedge(Q_1 \cup Q_2) = p_1 \wedge p_2$

Predicate Abstraction III

Lemma 14.4

1. *If predicates in P are pairwise independent, then*
   - $Q_1 \sqcap Q_2 \left(= \overline{Q_1 \wedge Q_2}\right) = \bigwedge(Q_1 \cup Q_2)$
   - $Q_1 \sqcup Q_2 \left(= \overline{Q_1 \vee Q_2}\right) = \bigwedge(Q_1 \cap Q_2)$
2. *Otherwise,*
   - $Q_1 \sqcap Q_2 \left(= \overline{Q_1 \wedge Q_2}\right) \neq \bigwedge(Q_1 \cup Q_2)$ *(but* $\equiv \bigwedge(Q_1 \cup Q_2)$*)*
   - $Q_1 \sqcup Q_2 \left(= \overline{Q_1 \vee Q_2}\right) \neq \bigwedge(Q_1 \cap Q_2)$ *(and* $\not\equiv \bigwedge(Q_1 \cap Q_2)$*)*

Example 14.5

1. – $p_1 := (\texttt{x <= y})$, $p_2 := (\texttt{x >= y})$, $p_3 := (\texttt{x = y})$
   – $Q_1 := p_1$, $Q_2 := p_2$
   $\Rightarrow Q_1 \sqcap Q_2 = \overline{Q_1 \wedge Q_2} = p_1 \wedge p_2 \wedge p_3 \not\equiv \bigwedge(Q_1 \cup Q_2) = p_1 \wedge p_2$
2. – $p_1 := (\texttt{x > y})$, $p_2 := (\texttt{x >= y})$, $p_3 := (\texttt{x = y})$
   – $Q_1 := p_1 \wedge p_2 \wedge \neg p_3 \; (\equiv \texttt{x > y})$, $Q_2 := p_3$
   $\Rightarrow Q_1 \sqcup Q_2 = \overline{Q_1 \vee Q_2} = p_2 \not\equiv \bigwedge(Q_1 \cap Q_2) = \text{true}$

Predicate Abstraction III

Lemma 14.4

1. *If predicates in P are pairwise independent, then*
   - $Q_1 \sqcap Q_2 \left(= \overline{Q_1 \wedge Q_2}\right) = \bigwedge(Q_1 \cup Q_2)$
   - $Q_1 \sqcup Q_2 \left(= \overline{Q_1 \vee Q_2}\right) = \bigwedge(Q_1 \cap Q_2)$
2. *Otherwise,*
   - $Q_1 \sqcap Q_2 \left(= \overline{Q_1 \wedge Q_2}\right) \neq \bigwedge(Q_1 \cup Q_2)$ *(but* $\equiv \bigwedge(Q_1 \cup Q_2)$*)*
   - $Q_1 \sqcup Q_2 \left(= \overline{Q_1 \vee Q_2}\right) \neq \bigwedge(Q_1 \cap Q_2)$ *(and* $\not\equiv \bigwedge(Q_1 \cap Q_2)$*)*

Example 14.5

1. – $p_1 := (\texttt{x <= y})$, $p_2 := (\texttt{x >= y})$, $p_3 := (\texttt{x = y})$
   – $Q_1 := p_1$, $Q_2 := p_2$
   $\Rightarrow Q_1 \sqcap Q_2 = \overline{Q_1 \wedge Q_2} = p_1 \wedge p_2 \wedge p_3 \not\equiv \bigwedge(Q_1 \cup Q_2) = p_1 \wedge p_2$
2. – $p_1 := (\texttt{x > y})$, $p_2 := (\texttt{x >= y})$, $p_3 := (\texttt{x = y})$
   – $Q_1 := p_1 \wedge p_2 \wedge \neg p_3 \; (\equiv \texttt{x > y})$, $Q_2 := p_3$
   $\Rightarrow Q_1 \sqcup Q_2 = \overline{Q_1 \vee Q_2} = p_2 \not\equiv \bigwedge(Q_1 \cap Q_2) = \text{true}$

Note: If interdependencies are ignored, then (computationally simpler) Cartesian Abstraction is performed.

Predicate Abstraction IV

Definition 14.6 (Galois connection for predicate abstraction)

The Galois connection for predicate abstraction is determined by

$$\alpha : 2^{\Sigma} \to \mathit{Abs}(P) \qquad \gamma : \mathit{Abs}(P) \to 2^{\Sigma}$$

with $\alpha(S) := \bigsqcup\{Q_\sigma \mid \sigma \in S\}$ $\qquad \gamma(Q) := \{\sigma \in \Sigma \mid \sigma \models Q\}$

where $Q_\sigma := \bigwedge(\{p \in P \mid \sigma \models p\} \cup \{\neg p \in \neg P \mid \sigma \not\models p\})$

Predicate Abstraction IV

Definition 14.6 (Galois connection for predicate abstraction)

The Galois connection for predicate abstraction is determined by

$$\begin{array}{rll} & \alpha : 2^{\Sigma} \to \mathit{Abs}(P) & \gamma : \mathit{Abs}(P) \to 2^{\Sigma} \\ \text{with} & \alpha(S) := \bigsqcup\{Q_\sigma \mid \sigma \in S\} & \gamma(Q) := \{\sigma \in \Sigma \mid \sigma \models Q\} \\ \text{where} & Q_\sigma := \bigwedge(\{p \in P \mid \sigma \models p\} \cup \{\neg p \in \neg P \mid \sigma \not\models p\}) & \end{array}$$

Example 14.7

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

Predicate Abstraction IV

Definition 14.6 (Galois connection for predicate abstraction)

The Galois connection for predicate abstraction is determined by

$$\alpha : 2^{\Sigma} \to \mathit{Abs}(P) \qquad \gamma : \mathit{Abs}(P) \to 2^{\Sigma}$$

with $\alpha(S) := \bigsqcup\{Q_\sigma \mid \sigma \in S\}$ $\qquad \gamma(Q) := \{\sigma \in \Sigma \mid \sigma \models Q\}$

where $Q_\sigma := \bigwedge(\{p \in P \mid \sigma \models p\} \cup \{\neg p \in \neg P \mid \sigma \not\models p\})$

Example 14.7

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

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

Predicate Abstraction IV

Definition 14.6 (Galois connection for predicate abstraction)

The Galois connection for predicate abstraction is determined by

$$\alpha : 2^{\Sigma} \to \mathit{Abs}(P) \qquad \gamma : \mathit{Abs}(P) \to 2^{\Sigma}$$

with $\alpha(S) := \bigsqcup\{Q_\sigma \mid \sigma \in S\}$ $\qquad \gamma(Q) := \{\sigma \in \Sigma \mid \sigma \models Q\}$

where $Q_\sigma := \bigwedge(\{p \in P \mid \sigma \models p\} \cup \{\neg p \in \neg P \mid \sigma \not\models p\})$

Example 14.7

- Let $\mathit{Var} := \{\mathtt{x}, \mathtt{y}\}$ and $P := \{p_1, p_2, p_3\}$ where $p_1 := (\mathtt{x} <= \mathtt{y})$, $p_2 := (\mathtt{x} = \mathtt{y})$, $p_3 := (\mathtt{x} > \mathtt{y})$
- 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
$$\begin{aligned}\alpha(S) &= Q_{\sigma_1} \sqcup Q_{\sigma_2} \\ &= (p_1 \wedge \neg p_2 \wedge \neg p_3) \sqcup (p_1 \wedge p_2 \wedge \neg p_3) \\ &= (p_1 \wedge \neg p_2 \wedge \neg p_3) \vee (p_1 \wedge p_2 \wedge \neg p_3) \\ &\equiv p_1 \wedge \neg p_3\end{aligned}$$
- If $Q = p_1 \wedge \neg p_2 \in \mathit{Abs}(P)$, then $\gamma(Q) = \{\sigma \in \Sigma \mid \sigma(\mathtt{x}) < \sigma(\mathtt{y})\}$

Outline of Lecture 14

Abstract Semantics for Predicate Abstraction

Definition 14.8 (Execution relation for predicate abstraction)

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

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

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

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

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

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

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

Abstract Semantics for Predicate Abstraction

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 $x := a$:

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

Abstract Semantics for Predicate Abstraction

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 $x := a$:

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

- In Rules (if1), (if2), (wh1), (wh2), the fact that $b = p_i$ for some $i \in \{1, \ldots, n\}$ implies that $\overline{Q \wedge [\neg]b} \equiv Q \wedge [\neg]b$

Abstract Semantics for Predicate Abstraction

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 $x := a$:

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

- In Rules (if1), (if2), (wh1), (wh2), the fact that $b = p_i$ for some $i \in \{1, \ldots, n\}$ implies that $\overline{Q \wedge [\neg]b} \equiv Q \wedge [\neg]b$
- 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

Abstract Semantics for Predicate Abstraction

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 $x := a$:

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

- In Rules (if1), (if2), (wh1), (wh2), the fact that $b = p_i$ for some $i \in \{1, \ldots, n\}$ implies that $\overline{Q \wedge [\neg]b} \equiv Q \wedge [\neg]b$
- 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 tautology or contradiction (i.e., resp. equivalent to true or false), then the abstract transition system with initial configuration $\langle c, \text{true}\rangle$ corresponds to the control flow graph of c

Abstract Semantics for Predicate Abstraction

An Example

Example 14.9

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

Abstract Semantics for Predicate Abstraction

An Example

Example 14.9

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

- **Claim:** label 7 not reachable
  (as `x > y` is a loop invariant)
- **Proof:** by predicate abstraction with
  - $p_1 := (\texttt{x > y})$
  - $p_2 := (\texttt{x >= y})$
- **Abstract transition system:** on the board

Abstract Semantics for Predicate Abstraction

An Example

Example 14.9

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

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

Outline of Lecture 14

Computation of Postconditions

Problem: $\overline{b} = \bigwedge \{ q \in P \cup \neg P \mid b \models q \}$ (i.e., the strongest formula in $Abs(P)$ that is implied by b) is generally not computable (due to undecidability of implication in certain logics)

Computation of Postconditions

Problem: $\overline{b} = \bigwedge \{q \in P \cup \neg P \mid b \models q\}$ (i.e., the strongest formula in $Abs(P)$ that is implied by b) is generally not computable (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 $b \models p_i$ and $b \models \neg p_i$
  - approximate $\overline{b}$ by conjunction of all provable literals

Computation of Postconditions

Problem: $\overline{b} = \bigwedge \{q \in P \cup \neg P \mid b \models q\}$ (i.e., the strongest formula in $Abs(P)$ that is implied by b) is generally not computable (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 $b \models p_i$ and $b \models \neg p_i$
  - approximate $\overline{b}$ 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 $\overline{b}$ computable for WHILE programs without multiplication

Computation of Postconditions

Problem: $\overline{b} = \bigwedge\{q \in P \cup \neg P \mid b \models q\}$ (i.e., the strongest formula in $Abs(P)$ that is implied by b) is generally not computable (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 $b \models p_i$ and $b \models \neg p_i$
  - approximate $\overline{b}$ 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 $\overline{b}$ 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 in n $\Longrightarrow$ solution: Binary Decision Diagrams (BDDs)