



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/>

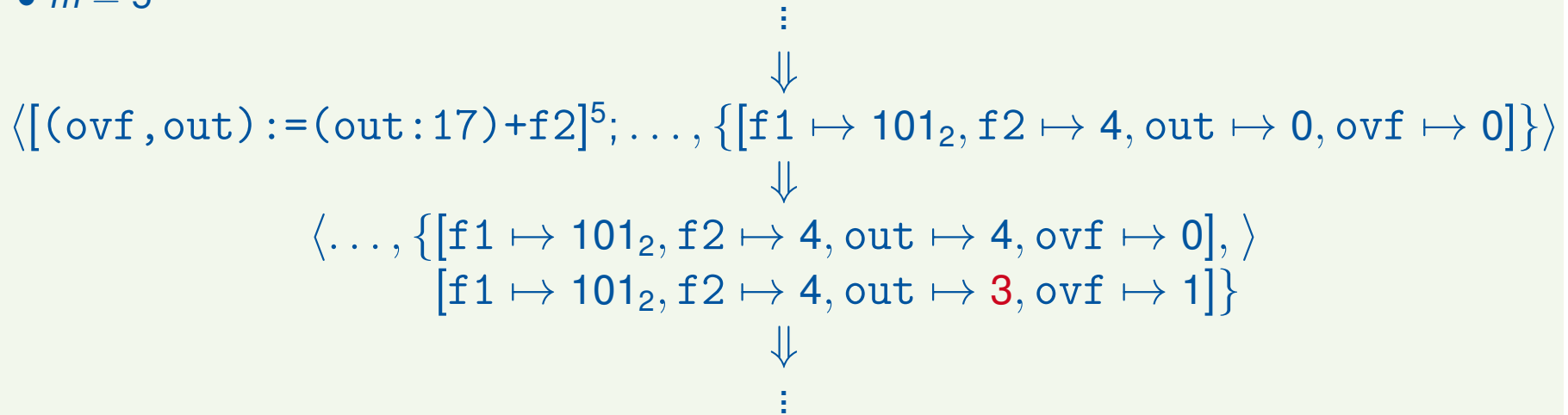
Recap: Abstraction of 16-Bit Multiplication

Abstract Interpretation of Multiplier

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

Abstract execution for

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



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

Recap: Abstraction of 16-Bit Multiplication

Extending the Specification

- **Observation:** the soundness property

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

is easily satisfied by always setting $\sigma'(\text{ovf}) = 1$

- 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 $\langle c, \sigma \rangle \rightarrow^+ \langle \downarrow, \sigma' \rangle$, then

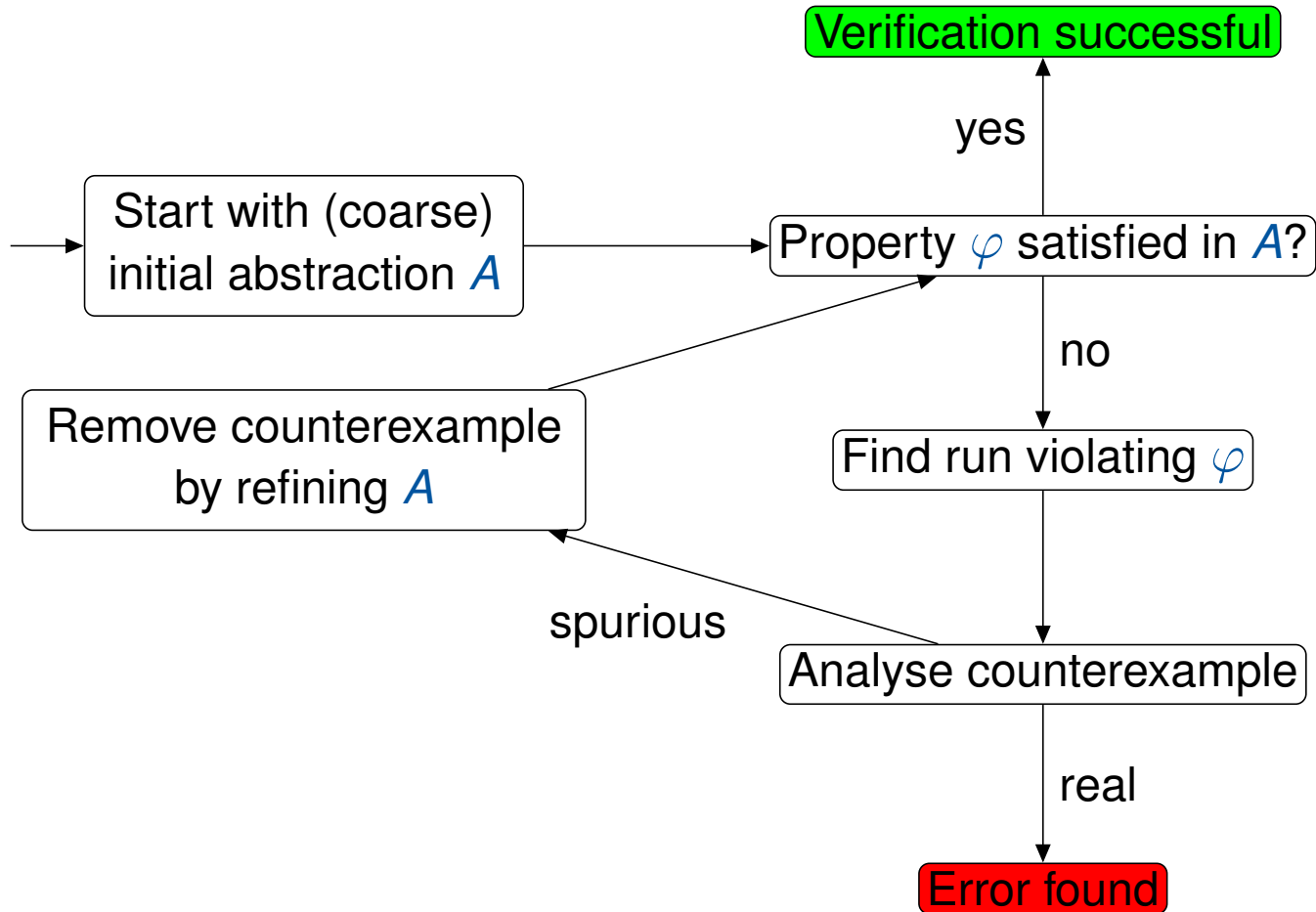
$$\begin{cases} \sigma'(\text{out}) = \sigma(f1) \cdot \sigma(f2) \wedge \sigma'(\text{ovf}) = 0 & \text{if } s(\sigma(f1)) + s(\sigma(f2)) \leq 16 \\ \sigma'(\text{ovf}) = 1 & \text{if } s(\sigma(f1)) + s(\sigma(f2)) \geq 18 \\ \sigma'(\text{out}) = \sigma(f1) \cdot \sigma(f2) \vee \sigma'(\text{ovf})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

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**

Abstraction Refinement Using Predicates

Counterexample-Guided Abstraction Refinement (CEGAR)



Abstraction Refinement Using Predicates

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)

The Predicate Abstraction Domain

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 $P = \{p_1, \dots, p_n\} \subseteq BExp$ be a finite set of predicates, and let $\neg P := \{\neg p_1, \dots, \neg p_n\}$. 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 $\text{false} \equiv \bigwedge \{p_i, \neg p_i, \dots\}$ if $P \neq \emptyset$)

The Predicate Abstraction Domain

Predicate Abstraction II

Lemma 14.2

$Abs(P)$ is a *complete lattice* with

- $\perp = \text{false}$, $\top = \text{true}$
- $Q_1 \sqcap Q_2 = \overline{Q_1 \wedge Q_2}$ where $\bar{b} := \bigwedge \{q \in P \cup \neg P \mid b \models q\}$
(i.e., \bar{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 $p_1 := (x = 1)$, $p_2 := (y = 2)$, $p_3 := (z = 3)$ (indep.).

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

$$Q_1 \sqcap Q_2 = \overline{Q_1 \wedge Q_2} = \overline{p_1 \wedge \neg p_2 \wedge p_3} \quad (= \bigwedge (Q_1 \cup Q_2))$$

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

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

$$Q_1 \sqcap Q_2 = \overline{Q_1 \wedge Q_2} = \overline{\text{false}} \quad (= \bigwedge (Q_1 \cup Q_2))$$

$$Q_1 \sqcup Q_2 = \overline{Q_1 \vee Q_2} \equiv \overline{p_1 \wedge (p_2 \vee \neg p_2)} = p_1 \quad (= \bigwedge (Q_1 \cap Q_2))$$

The Predicate Abstraction Domain

Predicate Abstraction III

Lemma 14.4

1. If predicates in P are pairwise independent, then

$$- Q_1 \sqcap Q_2 (= \overline{Q_1 \wedge Q_2}) = \wedge(Q_1 \cup Q_2)$$

$$- Q_1 \sqcup Q_2 (= \overline{Q_1 \vee Q_2}) = \wedge(Q_1 \cap Q_2)$$

2. Otherwise,

$$- Q_1 \sqcap Q_2 (= \overline{Q_1 \wedge Q_2}) \neq \wedge(Q_1 \cup Q_2) \text{ (but } \equiv \wedge(Q_1 \cup Q_2)\text{)}$$

$$- Q_1 \sqcup Q_2 (= \overline{Q_1 \vee Q_2}) \neq \wedge(Q_1 \cap Q_2) \text{ (and } \neq \wedge(Q_1 \cap Q_2)\text{)}$$

Example 14.5

1. $- p_1 := (x \leq y), p_2 := (x \geq y), p_3 := (x = y)$

$$- Q_1 := p_1, Q_2 := p_2$$

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

2. $- p_1 := (x > y), p_2 := (x \geq y), p_3 := (x = y)$

$$- Q_1 := p_1 \wedge p_2 \wedge \neg p_3 (\equiv x > y), Q_2 := p_3$$

$$\Rightarrow Q_1 \sqcup Q_2 = \overline{Q_1 \vee Q_2} = p_2 \neq \wedge(Q_1 \cap Q_2) = \text{true}$$

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

The Predicate Abstraction Domain

Predicate Abstraction IV

Definition 14.6 (Galois connection for predicate abstraction)

The **Galois connection for predicate abstraction** is determined by

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

Example 14.7

- Let $\text{Var} := \{x, y\}$ and $P := \{p_1, p_2, p_3\}$ where $p_1 := (x \leq y)$, $p_2 := (x = y)$, $p_3 := (x > y)$
- If $S = \{\sigma_1, \sigma_2\} \subseteq \Sigma$ with $\sigma_1 = [x \mapsto 1, y \mapsto 2]$, $\sigma_2 = [x \mapsto 2, y \mapsto 2]$,
then $\alpha(S) = Q_{\sigma_1} \sqcup Q_{\sigma_2}$
$$\begin{aligned} &= (p_1 \wedge \neg p_2 \wedge \neg p_3) \sqcup (p_1 \wedge p_2 \wedge \neg p_3) \\ &= \frac{(p_1 \wedge \neg p_2 \wedge \neg p_3) \vee (p_1 \wedge p_2 \wedge \neg p_3)}{\equiv p_1 \wedge \neg p_3} \end{aligned}$$
- If $Q = p_1 \wedge \neg p_2 \in \text{Abs}(P)$, then $\gamma(Q) = \{\sigma \in \Sigma \mid \sigma(x) < \sigma(y)\}$

Abstract Semantics for Predicate Abstraction

Abstract Semantics for Predicate Abstraction

Definition 14.8 (Execution relation for predicate abstraction)

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

$$\begin{array}{c} \text{(skip)} \frac{}{\langle \text{skip}, Q \rangle \Rightarrow \langle \downarrow, Q \rangle} \quad \text{(asgn)} \frac{}{\langle x := a, Q \rangle \Rightarrow \langle \downarrow, \bigsqcup \{ Q_{\sigma[x \mapsto \text{val}_{\sigma}(a)]} \mid \sigma \models Q \} \rangle} \\ \text{(seq1)} \frac{\langle c_1, Q \rangle \Rightarrow \langle c'_1, Q' \rangle \quad c'_1 \neq \downarrow}{\langle c_1; c_2, Q \rangle \Rightarrow \langle c'_1; c_2, Q' \rangle} \quad \text{(seq2)} \frac{\langle c_1, Q \rangle \Rightarrow \langle \downarrow, Q' \rangle}{\langle c_1; c_2, Q \rangle \Rightarrow \langle c_2, Q' \rangle} \\ \text{(if1)} \frac{}{\langle \text{if } b \text{ then } c_1 \text{ else } c_2 \text{ end}, Q \rangle \Rightarrow \langle c_1, \overline{Q \wedge b} \rangle} \\ \text{(if2)} \frac{}{\langle \text{if } b \text{ then } c_1 \text{ else } c_2 \text{ end}, Q \rangle \Rightarrow \langle c_2, \overline{Q \wedge \neg b} \rangle} \\ \text{(wh1)} \frac{}{\langle \text{while } b \text{ do } c \text{ end}, Q \rangle \Rightarrow \langle c; \text{while } b \text{ do } c \text{ end}, \overline{Q \wedge b} \rangle} \\ \text{(wh2)} \frac{}{\langle \text{while } b \text{ do } c \text{ end}, Q \rangle \Rightarrow \langle \downarrow, \overline{Q \wedge \neg b} \rangle} \end{array}$$

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
 - $\rho_1 := (x > y)$
 - $\rho_2 := (x \geq y)$
- **Abstract transition system:** on the board
- **Remark:** $\rho_1 := (x > y)$ alone not sufficient to prove loop invariant
(as not necessarily valid after label 3)

Computation of Postconditions

Computation of Postconditions

Problem: $\bar{b} = \bigwedge \{q \in P \cup \neg P \mid b \models q\}$ (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 $i \in \{1, \dots, n\}$, try to prove $b \models p_i$ and $b \models \neg p_i$
 - approximate \bar{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 \bar{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 $n \implies$ solution: Binary Decision Diagrams (BDDs)