Program Analysis
Lecture 10

Rayna Dimitrova

WS 2016/2017
Abstraction

Abstract semantics: soundly approximate the concrete semantics in order to reason effectively and efficiently about program properties.

Possible outcomes of the analysis of the abstraction

- the abstraction satisfies the property: the concrete program also satisfies the property (soundness)
- the abstraction does not satisfy the property: the program might or might not satisfy the property (incompleteness)

Output of the analysis of the abstraction

- in the first case: a proof of correctness
- in the second case: an abstract counterexample, which might or might not correspond to an execution that violates the property
Abstract counterexample analysis

The abstract counterexample is an abstract execution that violates the property. Often we can algorithmically check whether

▶ the abstract counterexample corresponds to a concrete program execution that violates the property or
▶ it does not correspond to a concrete execution, and exists because of the imprecision of the abstraction

Idea: use the unconcretizable abstract counterexample to refine the abstraction and eliminate if from the newly constructed abstraction
Counterexample-Guided Abstraction Refinement (CEGAR)

program $c$, property $\varphi$

compute $c_{Abs}$

$c_{Abs} \models \varphi$?

no

is $r_{Abs}$ concretizable to an execution $r$ of $c$ such that $r \not\models \varphi$?

yes

refine the abstraction constructing $c_{Abs}$ where $r_{Abs}$ is not an execution

no

obtain abstract execution $r_{Abs}$ of $c_{Abs}$ such that $r_{Abs} \not\models \varphi$

Note: the abstraction refinement loop might not terminate
Counterexample-Guided Abstraction Refinement (CEGAR)

Input: a program $c$ and a property $\varphi$

1. construct a sound abstraction $c_{Abs}$ of $c$
2. check if $c_{Abs}$ satisfies the property $\varphi$
3. if the property holds for $c_{Abs}$ then in holds for $c$ (return $\checkmark$)
4. otherwise, obtain an abstract counterexample $r_{Abs}$
5. if $r_{Abs}$ corresponds to an execution in the concrete semantics, then the program $c$ does not satisfy the property $\varphi$ (return $\times$)
6. otherwise refine the abstraction in a way that $r_{Abs}$ is not an execution of the newly constructed $c_{Abs}$ and go to 2
A typical abstraction method used in many software verification tools (for example, SLAM at Microsoft) is **predicate abstraction**.

Use predicates (first-order formulas) to map states to bitvectors: track only predicates on data and remove data variables from model.

Sets of states are abstracted to Boolean formulas over a finite set $P$ of predicates. The abstraction is refined by adding predicates to $P$.

**Automatic generation of predicates:**
- extracted from the program (conditions of `if` and `while`)
- extracted from the abstract counterexamples
A **predicate** is a Boolean expression $b \in BExp$ (over $Vars$).

**Examples:** $x > 4$, $x < 3 \land x \geq y$, $x + y \leq 5$

A state $\sigma$ **satisfies** $b$ (written $\sigma \models b$) if $B[b](\sigma) = true$.

**Examples:** if $\sigma(x) = 6$, $\sigma(y) = 0$, then $\sigma \models x > 4$, $\sigma \not\models x + y \leq 5$

A predicate $p$ is **stronger** than a predicate $q$ (written $p \models q$) if for every state $\sigma \in States$ it holds that $\sigma \models p$ implies $\sigma \models q$.

We also say that the predicate $q$ is **weaker** than the predicate $p$.

Predicates $p$ and $q$ are called **logically equivalent** (written $p \models|q$) if both $p \models q$ and $q \models p$ hold.

**Examples:** $x > y \models x \geq y$, $x + y \leq 5 \models x \leq 2 + 3 - y$
Predicate abstraction lattice

The stronger-than relation $\models$ is not antisymmetric. To make $\models$ a partial order over Boolean expressions, we identify logically equivalent formulas (consider equivalence classes of $\models$).

For scalability of the abstraction, instead of arbitrary Boolean combinations over the predicates in a set of predicates $P$, a typical restriction is to consider conjunctions of predicates in $P$ and the negations of predicates in $P$ (called **Cartesian** abstraction).
Let $P = \{p_1, \ldots, p_n\}$ be a finite set of predicates.
Define $\neg P := \{\neg p_1, \ldots, \neg p_n\}$ to be the set of negated predicates.

A **cube** is a conjunction $\bigwedge Q$ of the elements of a set $Q \subseteq P \cup \neg P$.
We define $Cubes(P) := \{\bigwedge Q \mid Q \subseteq P \cup \neg P\}$.

The **predicate abstraction lattice** for $P$ is

$$Abs(P) := (Cubes(P), \models).$$
Predicate abstraction lattice

Lemma $Abs(P) = (Cubes(P), \models)$ is a complete lattice.

Proof

$\perp = false := \bigwedge\{p_1, \neg p_1\}$, $\top = true := \bigwedge\emptyset$

$q_1 \cap q_2 := q_1 \land q_2$

$q_1 \cup q_2 := q_1 \lor q_2$

where for a formula $b$ (not necessarily in $Abs(P)$), $\overline{b}$ is the strongest element of $Abs(P)$ that is weaker than $b$

$$\overline{b} \models \bigwedge\{q \in Cubes(P) \mid b \models q\}$$

$$\overline{b} \models \bigwedge\{l \in (P \cup \neg P) \mid b \models l\}$$
Example

Let \( P = \{p_1, p_2, p_3\} \).

Consider \( q_1 = p_1 \land p_2 \) and \( q_2 = p_1 \land \neg p_2 \). Then

\[
q_1 \sqcap q_2 = p_1 \land p_2 \land p_1 \land \neg p_2 \models false
\]

\[
q_1 \sqcup q_2 = \frac{(p_1 \land p_2) \lor (p_1 \land \neg p_2)}{p_1 \land (p_2 \lor \neg p_2) \models p_1}
\]

Consider \( r_1 = p_1 \land \neg p_2 \) and \( r_2 = \neg p_2 \land p_3 \). Then

\[
r_1 \sqcap r_2 = p_1 \land \neg p_2 \land \neg p_2 \land p_3 \models p_1 \land \neg p_2 \land p_3
\]

\[
r_1 \sqcup r_2 = \frac{(p_1 \land \neg p_2) \lor (\neg p_2 \land p_3)}{\neg p_2 \land (p_1 \lor p_3) \models \neg p_2 \land p_1}
\]

Suppose \( p_1 = (x > 4) \) and \( p_3 = (x > 6) \).

Then, \( p_1 \lor p_3 \models p_1 \), and thus \( r_1 \sqcup r_2 \models \neg p_2 \land p_1 \).
Galois connection for predicate abstraction

Given a finite set of predicates $P$ we define the Galois connection

$\alpha : \mathcal{P}(\text{States}) \to \text{Cubes}(P)$ and $\gamma : \text{Cubes}(P) \to \mathcal{P}(\text{States})$

where

$\gamma(q) := \{ \sigma \in \text{States} \mid \sigma \models q \}$ for $q \in \text{Cubes}(P)$

and

$\alpha(S) := \bigcup \{ q_\sigma \mid \sigma \in S \}$ for $S \subseteq \text{States}$

where for every $\sigma \in \text{States}$ we define $q_\sigma \in \text{Cubes}(P)$ by

$q_\sigma := \bigwedge \{ l \in (P \cup \neg P) \mid \sigma \models l \}$
Consider the set of predicates $P = \{ p_1, p_2, p_3 \}$ with $p_1 = (x \leq y)$, $p_2 = (x = y)$, $p_3 = (x > y)$ and the set of states $S = \{ \sigma_1, \sigma_2 \}$ where $\sigma_1(x) = 1$, $\sigma_1(y) = 2$, $\sigma_2(x) = 2$ and $\sigma_2(y) = 2$.

$$
\alpha(\{ \sigma_1, \sigma_2 \}) = q_{\sigma_1} \uplus q_{\sigma_2} \\
= (p_1 \land \neg p_2 \land \neg p_3) \uplus (p_1 \land p_2 \land \neg p_3) \\
= (p_1 \land \neg p_2 \land \neg p_3) \lor (p_1 \land p_2 \land \neg p_3) \\
\models (p_1 \land \neg p_3) \land (\neg p_2 \lor p_2) \\
\models p_1 \land \neg p_3 \\
\models p_1 \land \neg p_3
$$
Abstract semantics for predicate abstraction

Let $P$ be a finite set of predicates. Consider the abstract lattice $\text{Abs}(P) := (\text{Cubes}(P), \sqsubseteq)$ and the Galois connection $\mathcal{P}(\text{States}) \xleftarrow{\gamma} \text{Cubes}(P)$ defined above.

We will define a family of functions $\text{post}_c^\#$ and $\text{post}_{c,c'}^\#$ such that for all $c, c' \in \text{Progs}$ and $q \in \text{Cubes}(P)$ it holds that

\[
\text{post}_{c}^\#(q) \quad \models \quad \alpha(\text{post}_{c}(\gamma(q)))
\]
\[
\text{post}_{c,c'}^\#(q) \quad \models \quad \alpha(\text{post}_{c,c'}(\gamma(q)))
\]
Suppose that $x := a$ is the statement that leads from $c$ to $c'$.

For $q \in Cubes(P)$, the set $\gamma(q)$ is the set of states that satisfy $q$.

The **strongest postcondition** $sp(q, x := a)$ is a formula that characterizes the set of states $\text{post}_{c,c'}(\gamma(q))$, that is:

$$\text{post}_{c,c'}(\gamma(q)) = \{ \sigma \in States \mid \sigma \models sp(q, x := a) \}$$

We will define $\text{post}^\#_{c,c'}(q)$ to be $sp(q, x := a)$ and show that it has the desired properties.
Strongest postcondition and weakest precondition

A Hoare-triple $\{b\}c\{p\}$ consists of

- a predicate $b \in BExp$, called **precondition**
- a predicate $p \in BExp$, called **postcondition**
- a program $c$

The Hoare-triple $\{b\}c\{p\}$ is **valid** iff for every $\sigma \in States$ with $\sigma \models b$ and every $\sigma' \in States$ with $\sigma \models (c, \sigma) \rightarrow^* \sigma'$ it holds that $\sigma' \models p$.

**Intuition:** If $b$ is true before $c$ is executed, and if the execution of $c$ terminates, then $p$ is true afterwards.

**Note:** The condition is required to hold only for terminating executions of $c$, that is, we consider partial correctness.
Strongest postcondition and weakest precondition

Let \( c \) be a program.

The **strongest postcondition** of a predicate \( b \in BExp \) with respect to the program \( c \), denoted \( sp(b, c) \), is the strongest (w.r.t. \( \models \)) predicate \( p \in BExp \) such that the Hoare triple \( \{ b \} c \{ p \} \) is valid.

The **weakest precondition** of a predicate \( p \in BExp \) with respect to the program \( c \), denoted \( wp(c, p) \), is the weakest (w.r.t. \( \models \)) predicate \( b \in BExp \) such that the Hoare triple \( \{ b \} c \{ p \} \) is valid.

**Theorem (Dijkstra’76)**

\[
sp(b, x := a) \models \exists x': (b[x'/x] \land x = (a[x'/x])) \\
wp(x := a, p) \models p[a/x]
\]

where \( e[e'/x] \) is the expression obtained by the simultaneous replacement of all occurrences of \( x \) in the expression \( e \) by the expression \( e' \)

**Intuition:** \( x' \) in \( sp(b, x := a) \) corresponds to the old value of \( x \)
Strongest postcondition and weakest precondition

Let $c$ be a program.

The **strongest postcondition** of a predicate $b \in BExp$ with respect to the program $c$, denoted $sp(b, c)$, is the strongest (w.r.t. $|=\$) predicate $p \in BExp$ such that the Hoare triple $\{b\}c\{p\}$ is valid.

The **weakest precondition** of a predicate $p \in BExp$ with respect to the program $c$, denoted $wp(c, p)$, is the weakest (w.r.t. $|=\$) predicate $b \in BExp$ such that the Hoare triple $\{b\}c\{p\}$ is valid.

**Theorem (Dijkstra’76)**

$$ sp(b, x := a) \models \exists x' : (b[x'/x] \land x = (a[x'/x])) $$

$$ wp(x := a, p) \models p[a/x] $$

where $e[e'/x]$ is the expression obtained by the simultaneous replacement of all occurrences of $x$ in the expression $e$ by the expression $e'$

**Note:** Both formulas always exists and can be computed.

The formula for $sp(b, x := a)$ contains quantifiers.
Example

\[ sp(y > 5, x := y + 3) \models \]
\[ \exists x' : (y > 5)[x'/x] \land x = ((y + 3)[x'/x]) \models \]
\[ \exists x' : y > 5 \land x = y + 3 \models \]
\[ y > 5 \land x = y + 3 \]

\[ sp(y > 5 \land y > 3, x := x + y) \models \]
\[ \exists x' : (x > 5 \land y > 3)[x'/x] \land x = ((x + y)[x'/x]) \models \]
\[ \exists x' : x' > 5 \land y > 3 \land x = x' + y \models \]
\[ y > 3 \land x - y > 5 \]

\[ wp(x := y + 7, x > 5) \models \]
\[ (x > 5)[y + 7/x] \models \]
\[ y + 7 > 5 \models \]
\[ y > -2 \]
We now derive the abstract transition relation

\[ \Rightarrow \subseteq (\text{Progs} \times \text{Abs}(P)) \times ((\text{Progs} \times \text{Abs}(P)) \cup \text{Abs}(P)) \]

by the following rules
Abstract transition relation for predicate abstraction

**SKIP**

\[ ([\text{skip}]^l, q) \Rightarrow q \]

**ASSIGN**

\[ ([x := a]^l, q) \Rightarrow sp(q, x := a) \]

**READ**

\[ ([\text{read}(x)]^l, q) \Rightarrow sp(q, \text{read}(x)) \]

**PRINT**

\[ ([\text{print}(x)]^l, q) \Rightarrow q \]

**SEQ\textsubscript{1}**

\[ (c_1, q) \Rightarrow q' \]

\[ (c_1; c_2, q) \Rightarrow (c_2, q') \]

**SEQ\textsubscript{2}**

\[ (c_1, q) \Rightarrow (c_1', q') \]

\[ (c_1; c_2, q) \Rightarrow (c_1'; c_2, q') \]
Abstract transition relation for predicate abstraction

**IF**_{true}\
\[
\text{(if } [b]^l \text{ then } \{c_1\} \text{ else } \{c_2\}, q) \Rightarrow (c_1, q \land b)
\]

**IF**_{false}\
\[
\text{(if } [b]^l \text{ then } \{c_1\} \text{ else } \{c_2\}, q) \Rightarrow (c_2, q \land \neg b)
\]

**WHILE**_{true}\
\[
\text{(while } [b]^l \text{ do } \{c\}, q) \Rightarrow (c; \text{while } [b]^l \text{ do } \{c\}, q \land b)
\]

**WHILE**_{false}\
\[
\text{(while } [b]^l \text{ do } \{c\}, q) \Rightarrow q \land \neg b
\]

**Note:** For a conditionals $b$, $q \land b$ is the strongest postcondition for the positive branch, and $q \land \neg b$ is the one for the negative branch. Typically, the guards of the conditionals and the loops of the analyzed program are included in the set of abstraction predicates $P$. 
Abstract transition relation for predicate abstraction

**IF**<sub>true</sub>

\[(\text{if } [b]^l \text{ then } \{c_1\} \text{ else } \{c_2\}, q) \Rightarrow (c_1, q \land b)\]

**IF**<sub>false</sub>

\[(\text{if } [b]^l \text{ then } \{c_1\} \text{ else } \{c_2\}, q) \Rightarrow (c_2, q \land \neg b)\]

**WHILE**<sub>true</sub>

\[
(\text{while } [b]^l \text{ do } \{c\}, q) \Rightarrow (c; \text{while } [b]^l \text{ do } \{c\}, q \land b)\]

**WHILE**<sub>false</sub>

\[(\text{while } [b]^l \text{ do } \{c\}, q) \Rightarrow q \land \neg b\]

**Note:** Abstract configurations of the form \((c, false)\) do not correspond to reachable configurations in the concrete semantics (since for no concrete state \(\sigma\) we have \(\sigma \models false\)) and can be eliminated.
Abstract predicate transformers

For all $c, c' \in \text{Progs}$ we define $\text{post}^c : \text{Cubes}(P) \rightarrow \text{Cubes}(P)$ and $\text{post}^{c, c'} : \text{Cubes}(P) \rightarrow \text{Cubes}(P)$ by

$$\text{post}^c(q) := \begin{cases} q' & \text{if } (c, q) \Rightarrow q' \\ false & \text{otherwise} \end{cases}$$

$$\text{post}^{c, c'}(q) := \begin{cases} q' & \text{if } (c, q) \Rightarrow (c', q') \\ false & \text{otherwise} \end{cases}$$

Note: Since in $\text{post}^{c, c'}$, the programs $c$ and $c'$ are fixed, $\text{post}^{c, c'}$ is a function.
Example

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

Prove that location 7 in the program is not reachable.

\(x > y\) is a loop invariant.

In order to establish the invariant \(x > y\), we need to also add the predicate \(x \geq y\).

\(P = \{p_1, p_2\}\), where

\[
\begin{align*}
p_1 &= x > y \\
p_2 &= x \geq y
\end{align*}
\]