Static Program Analysis

Lecture 13: Abstract Interpretation III (Abstract Semantics of WHILE)

Winter Semester 2016/17

Thomas Noll
Software Modeling and Verification Group
RWTH Aachen University

https://moves.rwth-aachen.de/teaching/ws-1617/spa/
Recap: Safe Approximation of Functions and Relations

Outline of Lecture 13

Recap: Safe Approximation of Functions and Relations

Systematic Construction of Abstractions

Abstract Semantics of WHILE

Correctness of Abstract Semantics
Recap: Safe Approximation of Functions and Relations

Safe Approximation of Functions I

Definition (Safe approximation)

Let \((\alpha, \gamma)\) be a Galois connection with \(\alpha : L \to M\) and \(\gamma : M \to L\), and let \(f : L^n \to L\) and \(f^\# : M^n \to M\) be functions of rank \(n \in \mathbb{N}\). Then \(f^\#\) is called a safe approximation of \(f\) if, whenever \(m_1, \ldots, m_n \in M\),

\[
\alpha(f(\gamma(m_1), \ldots, \gamma(m_n))) \sqsubseteq_M f^\#(m_1, \ldots, m_n).
\]

Moreover, \(f^\#\) is called most precise if the reverse inclusion is also true.

Abstract
\[
\begin{array}{c}
\vec{m} \\
\downarrow f^\#
\end{array}
\quad \rightarrow_{\gamma} \quad \begin{array}{c}
\gamma(\vec{m}) \\
\downarrow f
\end{array}
\]

Concrete
\[
\begin{array}{c}
f^\#(\vec{m}) \sqsubseteq M f(\gamma(\vec{m})) \\
\begin{array}{c}
\alpha(f(\gamma(\vec{m}))) \quad \leftrightarrow \quad f(\gamma(\vec{m}))
\end{array}
\end{array}
\]

**Interpretation:** the abstraction \(f^\#\) covers all concrete \(f\)-results

**Note:** monotonicity of \(f\) and/or \(f^\#\) is not required (but usually given; see Lemma 12.3)
Recap: Safe Approximation of Functions and Relations

Encoding Execution Relations by Transition Functions I

- **Reminder:** concrete semantics of WHILE
  - statements \( \text{skip} \mid x := a \mid c_1; c_2 \mid \text{if } b \text{ then } c_1 \text{ else } c_2 \text{ end} \mid \text{while } b \text{ do } c \text{ end} \in \text{Cmd} \)
  - states \( \Sigma := \{ \sigma \mid \sigma : \text{Var} \rightarrow \mathbb{Z} \} \) (Definition 11.6)
  - execution relation \( \rightarrow \subseteq (\text{Cmd} \times \Sigma) \times ((\text{Cmd} \cup \{ \downarrow \}) \times \Sigma) \) (Definition 11.9)

- Yields concrete domain \( L := (2^\Sigma, \subseteq) \) and concrete transition function:

**Definition (Concrete transition function)**

The concrete transition function of WHILE is defined by the family of functions

\[
\text{next}_{c,c'} : 2^\Sigma \rightarrow 2^\Sigma
\]

where \( c \in \text{Cmd}, c' \in \text{Cmd} \cup \{ \downarrow \} \) and, for every \( S \subseteq \Sigma \),

\[
\text{next}_{c,c'}(S) := \{ \sigma' \in \Sigma \mid \exists \sigma \in S : \langle c, \sigma \rangle \rightarrow \langle c', \sigma' \rangle \}.
\]
Recap: Safe Approximation of Functions and Relations

Encoding Execution Relations by Transition Functions II

Remarks: $\text{next}$ satisfies the following properties

- “Determinism” (cf. Theorem 11.11): 
  - for all $c \in \text{Cmd}$, $c' \in \text{Cmd} \cup \{\downarrow\}$ and $\sigma \in \Sigma$, $|\text{next}_{c,c'}(\{\sigma\})| \leq 1$
  - for all $c \in \text{Cmd}$ and $\sigma \in \Sigma$ there exists exactly one $c' \in \text{Cmd} \cup \{\downarrow\}$ such that $\text{next}_{c,c'}(\{\sigma\}) \neq \emptyset$

- When is $\text{next}_{c,c'}(S) = \emptyset$? Possible reasons:
  1. $S = \emptyset$
  2. $c'$ is not a possible successor statement of $c$, e.g.,
     - $c = (x := 0)$
     - $c' = \text{skip}$
  3. $c'$ is unreachable for all $\sigma \in S$, e.g.,
     - $c = (\text{if } x = 0 \text{ then } x := 1 \text{ else } \text{skip} \text{ end})$
     - $c' = \text{skip}$
     - $\sigma(x) = 0$ for each $\sigma \in S$
Recap: Safe Approximation of Functions and Relations

Safe Approximation of Execution Relations

Reminder: abstraction determined by Galois connection \((\alpha, \gamma)\) with \(\alpha : L \rightarrow M\), \(\gamma : M \rightarrow L\)

- here: \(L := 2^\Sigma\), \(M\) not fixed
- usually \(M = \text{Var} \rightarrow \ldots\) (more efficient) or \(M = 2^\varrightarrow\ldots\) (more precise)
- write \(\text{Abs}\) in place of \(M\)
- thus \(\alpha : 2^\Sigma \rightarrow \text{Abs}\) and \(\gamma : \text{Abs} \rightarrow 2^\Sigma\)

Definition (Abstract semantics of WHILE)

Given \(\alpha : 2^\Sigma \rightarrow \text{Abs}\), an abstract semantics is defined by a family of functions

\[
\text{next}^\#_{c,c'} : \text{Abs} \rightarrow \text{Abs}
\]

where \(c \in \text{Cmd}, c' \in \text{Cmd} \cup \{\downarrow\}\), and each \(\text{next}^\#_{c,c'}\) is a safe approximation of \(\text{next}_{c,c'}\), i.e.,

\[
\alpha(\text{next}_{c,c'}(\gamma(\text{abs}))) \sqsubseteq_{\text{Abs}} \text{next}^\#_{c,c'}(\text{abs})
\]

for every \(\text{abs} \in \text{Abs}\) (notation: \(\langle c, \text{abs} \rangle \Rightarrow \langle c', \text{abs}' \rangle\) for \(\text{next}^\#_{c,c'}(\text{abs}) = \text{abs}'\)).
Systematic Construction of Abstractions

Outline of Lecture 13

Recap: Safe Approximation of Functions and Relations

Systematic Construction of Abstractions

Abstract Semantics of WHILE

Correctness of Abstract Semantics
Systematic Construction of Abstractions

Derivation of Abstract Semantics

- **Problem**: most precise safe approximation not always definable

Example 13.1 (Fermat’s Last Theorem)

Sign abstraction (cf. Example 11.3) on

\[
\langle \text{if } n > 2 \land x^n + y^n = z^n \text{ then } n := 1 \text{ else } n := -1 \text{ end, } \{[n, x, y, z \mapsto +]\} \rangle
\]

Result

\[
n \mapsto + \text{ possible iff there exist } n > 2 \text{ and } x, y, z \geq 1 \text{ such that } x^n + y^n = z^n
\]

Thus: most precise approximation yields

\[
\{[x, y, z \mapsto +], [n \mapsto -] \}
\]

iff equation unsolvable

Fermat’s Last Theorem: equation not solvable

Final proof by Andrew Wiles and Richard Taylor in 1995

More general: solvability of Diophantic equations even undecidable

Thus: resort to possibly imprecise safe approximations
Systematic Construction of Abstractions

Derivation of Abstract Semantics

- Problem: most precise safe approximation not always definable

Example 13.1 (Fermat’s Last Theorem)

Sign abstraction (cf. Example 11.3) on

\[
\langle \text{if } n > 2 \land x^n + y^n = z^n \text{ then } n := 1 \text{ else } n := -1 \text{ end}, \{[n, x, y, z \mapsto +]\} \rangle
\]

- Result \( n \mapsto + \) possible iff there exist \( n > 2 \) and \( x, y, z \geq 1 \) such that \( x^n + y^n = z^n \)
- Thus: most precise approximation yields \( \{[x, y, z \mapsto +, n \mapsto -]\} \) iff equation unsolvable
- Fermat’s Last Theorem: equation not solvable
- Final proof by Andrew Wiles and Richard Taylor in 1995
Systematic Construction of Abstractions

Derivation of Abstract Semantics

- **Problem:** most precise safe approximation not always definable

Example 13.1 (Fermat’s Last Theorem)

Sign abstraction (cf. Example 11.3) on

\[
\langle \text{if } n > 2 \land x^n + y^n = z^n \text{ then } n := 1 \text{ else } n := -1 \text{ end, } \{[n, x, y, z \mapsto +] \} \rangle
\]

- Result \( n \mapsto + \) possible iff there exist \( n > 2 \) and \( x, y, z \geq 1 \) such that \( x^n + y^n = z^n \)
- Thus: most precise approximation yields \( \{[x, y, z \mapsto +, n \mapsto -] \} \) iff equation unsolvable
- **Fermat’s Last Theorem:** equation not solvable
- Final proof by Andrew Wiles and Richard Taylor in 1995

- More general: solvability of Diophantic equations even undecidable
- Thus: resort to **possibly imprecise** safe approximations
Systematic Construction of Abstractions

Extraction Functions

- **Assumption**: abstraction determined by pointwise mapping of concrete values
- If $L = 2^C$ and $M = 2^A$ with $\sqsubseteq_L = \sqsubseteq_M = \subseteq$, then $\beta : C \rightarrow A$ is called an extraction function
- $\beta$ determines Galois connection $(\alpha, \gamma)$ where
  \[
  \alpha : L \rightarrow M : I \mapsto \beta(I) \quad (= \{ \beta(c) \mid c \in I \})
  \]
  \[
  \gamma : M \rightarrow L : m \mapsto \beta^{-1}(m) \quad (= \{ c \in C \mid \beta(c) \in m \})
  \]
Systematic Construction of Abstractions

Extraction Functions

- **Assumption:** abstraction determined by pointwise mapping of concrete values
- If $L = 2^C$ and $M = 2^A$ with $\sqsubseteq_L = \sqsubseteq_M = \subseteq$, then $\beta : C \to A$ is called an extraction function
- $\beta$ determines Galois connection $(\alpha, \gamma)$ where

  $\alpha : L \to M : l \mapsto \beta(l) = \{ \beta(c) \mid c \in l \}$
  $\gamma : M \to L : m \mapsto \beta^{-1}(m) = \{ c \in C \mid \beta(c) \in m \}$

Example 13.2

1. Parity abstraction (cf. Example 11.2): $\beta : \mathbb{Z} \to \{ \text{even}, \text{odd} \}$ where

   $\beta(z) := \begin{cases} 
   \text{even} & \text{if } z \text{ even} \\
   \text{odd} & \text{if } z \text{ odd}
   \end{cases}$
Systematic Construction of Abstractions

Extraction Functions

- **Assumption:** abstraction determined by pointwise mapping of concrete values
- If $L = 2^C$ and $M = 2^A$ with $\subseteq_L = \subseteq_M = \subseteq$, then $\beta : C \rightarrow A$ is called an extraction function
- $\beta$ determines Galois connection $(\alpha, \gamma)$ where

  \[
  \alpha : L \rightarrow M : l \mapsto \beta(l) \quad (= \{\beta(c) | c \in l\})
  \]

  \[
  \gamma : M \rightarrow L : m \mapsto \beta^{-1}(m) \quad (= \{c \in C | \beta(c) \in m\})
  \]

Example 13.2

1. Parity abstraction (cf. Example 11.2): $\beta : \mathbb{Z} \rightarrow \{\text{even, odd}\}$ where

   \[
   \beta(z) := \begin{cases} 
   \text{even} & \text{if } z \text{ even} \\
   \text{odd} & \text{if } z \text{ odd}
   \end{cases}
   \]

2. Sign abstraction (cf. Example 11.3): $\beta : \mathbb{Z} \rightarrow \{+, -, 0\}$ with $\beta = \text{sgn}$
Systematic Construction of Abstractions

Extraction Functions

- **Assumption:** abstraction determined by pointwise mapping of concrete values
- If \( L = 2^C \) and \( M = 2^A \) with \( \subseteq_L = \subseteq_M = \subseteq \), then \( \beta : C \to A \) is called an extraction function
- \( \beta \) determines Galois connection \((\alpha, \gamma)\) where
  \[
  \alpha : L \to M : l \mapsto \beta(l) \quad (= \{ \beta(c) \mid c \in l \})
  \]
  \[
  \gamma : M \to L : m \mapsto \beta^{-1}(m) \quad (= \{ c \in C \mid \beta(c) \in m \})
  \]

**Example 13.2**

1. Parity abstraction (cf. Example 11.2): \( \beta : \mathbb{Z} \to \{\text{even, odd}\} \) where
   \[
   \beta(z) := \begin{cases} 
   \text{even} & \text{if } z \text{ even} \\
   \text{odd} & \text{if } z \text{ odd}
   \end{cases}
   \]

2. Sign abstraction (cf. Example 11.3): \( \beta : \mathbb{Z} \to \{+, -, 0\} \) with \( \beta = \text{sgn} \)

3. Interval abstraction (cf. Example 11.4): not definable by extraction function (as \( \text{int} \) is not of the form \( 2^A \))
Safe Approximation by Extraction Functions

Reminder: safe approximation condition (Definition 12.1)

\[ \alpha(f(\gamma(m_1), \ldots, \gamma(m_n))) \subseteq_M f^#(m_1, \ldots, m_n). \]
Systematic Construction of Abstractions

Safe Approximation by Extraction Functions

Reminder: safe approximation condition (Definition 12.1)

\[ \alpha(f(\gamma(m_1), \ldots, \gamma(m_n))) \sqsubseteq_M f^#(m_1, \ldots, m_n). \]

Theorem 13.3

Let \( L = 2^C \) and \( M = 2^A \) with \( \sqsubseteq_L = \sqsubseteq_M = \subseteq \), \( \beta : C \rightarrow A \) be an extraction function, and \( f : C^n \rightarrow C \). Then

\[ f^# : M^n \rightarrow M : (m_1, \ldots, m_n) \mapsto \{ \beta(f(c_1, \ldots, c_n)) \mid \forall i \in \{1, \ldots, n\} : c_i \in \beta^{-1}(m_i) \} \]

is a safe approximation of \( f \).
Systematic Construction of Abstractions

Safe Approximation by Extraction Functions

Reminder: safe approximation condition (Definition 12.1)

\[ \alpha(f(\gamma(m_1), \ldots, \gamma(m_n))) \sqsubseteq_M f^#(m_1, \ldots, m_n). \]

Theorem 13.3

Let \( L = 2^C \) and \( M = 2^A \) with \( \sqsubseteq_L = \sqsubseteq_M = \subseteq \), \( \beta : C \rightarrow A \) be an extraction function, and \( f : C^n \rightarrow C \). Then

\[ f^# : M^n \rightarrow M : (m_1, \ldots, m_n) \mapsto \{ \beta(f(c_1, \ldots, c_n)) \mid \forall i \in \{1, \ldots, n\} : c_i \in \beta^{-1}(m_i) \} \]

is a safe approximation of \( f \).

Proof.

on the board
Systematic Construction of Abstractions

Safe Approximation of Arithmetic Operations

Example 13.4 (Sign abstraction)

For $C = \mathbb{Z}$, $A = \{+,-,0\}$, $\beta = \text{sgn}$:

<table>
<thead>
<tr>
<th></th>
<th>$+$</th>
<th>$-$</th>
<th>$0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$+$</td>
<td>$+$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$-$</td>
<td>$+, -, 0$</td>
<td>$-$</td>
<td>$-$</td>
</tr>
<tr>
<td>$0$</td>
<td>$+$</td>
<td>$-$</td>
<td>$0$</td>
</tr>
</tbody>
</table>

and $\{+, 0\} \# \{-\} = \{+, \} \# \{-\} \cup \{0\} \# \{-\}$

<table>
<thead>
<tr>
<th></th>
<th>$+$</th>
<th>$-$</th>
<th>$0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$+$</td>
<td>$+$</td>
<td>$-$</td>
<td>$0$</td>
</tr>
<tr>
<td>$-$</td>
<td>$+, +$</td>
<td>$+, +$</td>
<td>$0$</td>
</tr>
<tr>
<td>$0$</td>
<td>$0$</td>
<td>$0$</td>
<td>$0$</td>
</tr>
</tbody>
</table>

$= \{-\} \cup \{0\}$

$= \{-, 0\}$

etc.
### Systematic Construction of Abstracts

### Safe Approximation of Boolean Operations

**Example 13.5 (Sign abstraction)**

1. Relational operations: for \( C = \mathbb{Z} \cup \mathbb{B} \), \( A = \{+, -, 0\} \cup \mathbb{B} \), \( \beta = \text{sgn} \):

<table>
<thead>
<tr>
<th></th>
<th>{+}</th>
<th>{-}</th>
<th>{0}</th>
</tr>
</thead>
<tbody>
<tr>
<td>{+}</td>
<td>{true, false}</td>
<td>{false}</td>
<td>{false}</td>
</tr>
<tr>
<td>{-}</td>
<td>{false}</td>
<td>{true, false}</td>
<td>{false}</td>
</tr>
<tr>
<td>{0}</td>
<td>{false}</td>
<td>{false}</td>
<td>{true}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>{+}</th>
<th>{-}</th>
<th>{0}</th>
</tr>
</thead>
<tbody>
<tr>
<td>{+}</td>
<td>{true, false}</td>
<td>{true}</td>
<td>{true}</td>
</tr>
<tr>
<td>{-}</td>
<td>{false}</td>
<td>{true, false}</td>
<td>{false}</td>
</tr>
<tr>
<td>{0}</td>
<td>{false}</td>
<td>{true}</td>
<td>{false}</td>
</tr>
</tbody>
</table>

and \((\{+, 0\} = \# \{0\}) = (\{+\} = \# \{0\} \cup \{0\} = \# \{0\}) = \{false\} \cup \{true\} = \{true, false\}\) etc.
### Example 13.5 (Sign abstraction)

1. **Relational operations:** for \( C = \mathbb{Z} \cup \mathbb{B} \), \( A = \{+, -, 0\} \cup \mathbb{B} \), \( \beta = \text{sgn} \):

<table>
<thead>
<tr>
<th></th>
<th>(+)</th>
<th>(-)</th>
<th>(0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(#)</td>
<td>true, false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>(+)</td>
<td>true, false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>(-)</td>
<td>false</td>
<td>true, false</td>
<td>false</td>
</tr>
<tr>
<td>(0)</td>
<td>false</td>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>(+)</th>
<th>(-)</th>
<th>(0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&gt;)</td>
<td>{true}</td>
<td>{false}</td>
<td>{false}</td>
</tr>
<tr>
<td>(+)</td>
<td>{true}</td>
<td>{false}</td>
<td>{false}</td>
</tr>
<tr>
<td>(-)</td>
<td>{false}</td>
<td>{true}</td>
<td>{false}</td>
</tr>
<tr>
<td>(0)</td>
<td>{false}</td>
<td>{true}</td>
<td>{false}</td>
</tr>
</tbody>
</table>

and \((\{+, 0\} = \# \{0\}) = (\{+\} = \# \{0\} \cup \{0\} = \# \{0\}) = \{\text{false}\} \cup \{\text{true}\} = \{\text{true, false}\} \) etc.

2. **Boolean connectives:** for \( C = A = \mathbb{B} \):

\[-\# = \neg \quad \land\# = \land \quad \lor\# = \lor\]

and \( \{\text{true, false}\} \land\# \{\text{true}\} = \{\text{true}\} \land\# \{\text{true}\} \cup \{\text{false}\} \land\# \{\text{true}\} \)

\( = \{\text{true}\} \cup \{\text{false}\} \)

\( = \{\text{true, false}\} \)

etc.
Abstract Semantics of WHILE

Outline of Lecture 13

Recap: Safe Approximation of Functions and Relations

Systematic Construction of Abstractions

Abstract Semantics of WHILE

Correctness of Abstract Semantics
Abstract Semantics of WHILE

Abstract Program States

Now: take values of variables into account

Definition 13.6 (Abstract program state)

Let $\beta : \mathbb{Z} \rightarrow A$ be an extraction function.

- An abstract (program) state is an element of the set

$$\{\rho \mid \rho : \text{Var} \rightarrow A\},$$

called the abstract state space.
Abstract Semantics of WHILE

Abstract Program States

Now: take values of variables into account

Definition 13.6 (Abstract program state)

Let $\beta : \mathbb{Z} \rightarrow A$ be an extraction function.

- An abstract (program) state is an element of the set 
  \[ \{ \rho \mid \rho : \text{Var} \rightarrow A \}, \]
  called the abstract state space.
- The abstract domain is denoted by $\text{Abs} := 2^{\text{Var} \rightarrow A}$. 

Abstract Semantics of WHILE

Abstract Program States

Now: take values of variables into account

**Definition 13.6 (Abstract program state)**

Let $\beta : \mathbb{Z} \rightarrow A$ be an extraction function.

- An abstract (program) state is an element of the set
  $$\{\rho \mid \rho : \text{Var} \rightarrow A\},$$
  called the abstract state space.
- The abstract domain is denoted by $Abs := 2^{\text{Var} \rightarrow A}$.
- The abstraction function $\alpha : 2^\Sigma \rightarrow Abs$ is given by
  $$\alpha(S) := \{\beta \circ \sigma \mid \sigma \in S\}$$
  for every $S \subseteq \Sigma$. 
### Abstract Semantics of WHILE

### Abstract Evaluation of Expressions

**Definition 13.7 (Abstract evaluation functions)**

Let $\rho : Var \rightarrow A$ be an abstract state.

1. $val^\#_\rho : AExp \rightarrow 2^A$ is determined by $(f$ arithmetic operation$)$
   
   $val^\#_\rho(z) := \{\beta(z)\}$
   $val^\#_\rho(x) := \{\rho(x)\}$
   $val^\#_\rho(f(a_1, \ldots, a_n)) := f^\#(val^\#_\rho(a_1), \ldots, val^\#_\rho(a_n))$

2. $val^\#_\rho : BExp \rightarrow 2^B$ is determined by $(g/h$ relational/Boolean operation$)$
   
   $val^\#_\rho(t) := \{t\}$
   $val^\#_\rho(g(a_1, \ldots, a_n)) := g^\#(val^\#_\rho(a_1), \ldots, val^\#_\rho(a_n))$
   $val^\#_\rho(h(b_1, \ldots, b_n)) := h^\#(val^\#_\rho(b_1), \ldots, val^\#_\rho(b_n))$

**Example 13.8 (Sign abstraction)**

Let $\rho(x) = +$ and $\rho(y) = -$.

1. $val^\#_\rho(2 * x + y) = \{+,-,0\}$
2. $val^\#_\rho(\neg(x + 1 > y)) = \{\text{false}\}$
Abstract Semantics of WHILE

Abstract Evaluation of Expressions

Definition 13.7 (Abstract evaluation functions)

Let $\rho : \text{Var} \rightarrow A$ be an abstract state.

1. $\text{val}^\#_\rho : \text{AExp} \rightarrow 2^A$ is determined by ($f$ arithmetic operation)
   
   \[
   \begin{align*}
   \text{val}^\#_\rho(z) & := \{\beta(z)\} \\
   \text{val}^\#_\rho(x) & := \{\rho(x)\} \\
   \text{val}^\#_\rho(f(a_1, \ldots, a_n)) & := f^\#(\text{val}^\#_\rho(a_1), \ldots, \text{val}^\#_\rho(a_n))
   \end{align*}
   \]

2. $\text{val}^\#_\rho : \text{BExp} \rightarrow 2^\mathbb{B}$ is determined by ($g/h$ relational/Boolean operation)

   \[
   \begin{align*}
   \text{val}^\#_\rho(t) & := \{t\} \\
   \text{val}^\#_\rho(g(a_1, \ldots, a_n)) & := g^\#(\text{val}^\#_\rho(a_1), \ldots, \text{val}^\#_\rho(a_n)) \\
   \text{val}^\#_\rho(h(b_1, \ldots, b_n)) & := h^\#(\text{val}^\#_\rho(b_1), \ldots, \text{val}^\#_\rho(b_n))
   \end{align*}
   \]

Example 13.8 (Sign abstraction)

Let $\rho(x) = +$ and $\rho(y) = -$.

1. $\text{val}^\#_\rho(2 \times x + y) = \{+, -, 0\}$

2. $\text{val}^\#_\rho(\neg(x + 1 > y)) = \{\text{false}\}$
Abstract Semantics of WHILE

Abstract Evaluation of Expressions

Definition 13.7 (Abstract evaluation functions)

Let $\rho : \text{Var} \rightarrow A$ be an abstract state.

1. $\text{val}^\#: A\text{Exp} \rightarrow 2^A$ is determined by ($f$ arithmetic operation)
   
   $\text{val}^\#(z) := \{\beta(z)\}$

   $\text{val}^\#(x) := \{\rho(x)\}$

   $\text{val}^\#(f(a_1, \ldots, a_n)) := f^\#(\text{val}^\#(a_1), \ldots, \text{val}^\#(a_n))$

2. $\text{val}^\#: B\text{Exp} \rightarrow 2^B$ is determined by ($g/h$ relational/Boolean operation)
   
   $\text{val}^\#(t) := \{t\}$

   $\text{val}^\#(g(a_1, \ldots, a_n)) := g^\#(\text{val}^\#(a_1), \ldots, \text{val}^\#(a_n))$

   $\text{val}^\#(h(b_1, \ldots, b_n)) := h^\#(\text{val}^\#(b_1), \ldots, \text{val}^\#(b_n))$

Example 13.8 (Sign abstraction)

Let $\rho(x) = +$ and $\rho(y) = -$.

1. $\text{val}^\#(2 * x + y) = \{+, -, 0\}$

2. $\text{val}^\#(\neg(x + 1 > y)) = \{\text{false}\}$
Abstract Semantics of WHILE

Reminder: abstract domain is $Abs := 2^{\mathit{Var} \rightarrow A}$

Definition 13.9 (Abstract execution relation for statements)

If $c \in \mathit{Cmd}$ and $abs \in Abs$, then $\langle c, abs \rangle$ is called an abstract configuration. The abstract execution relation is defined by the following rules:

\[
\begin{align*}
\langle \mathit{skip}, abs \rangle & \Rightarrow \langle \Downarrow, abs \rangle \\
\langle x := a, abs \rangle & \Rightarrow \langle \Downarrow, \{ \rho[x \mapsto a'] \mid \rho \in abs, a' \in \mathit{val}^\#(a) \} \rangle \\
\langle c_1, abs \rangle & \Rightarrow \langle c'_1, abs' \rangle \quad c'_1 \neq \Downarrow \\
\langle c_1 ; c_2, abs \rangle & \Rightarrow \langle c'_1 ; c_2, abs' \rangle \\
\langle c_1, abs \rangle & \Rightarrow \langle \Downarrow, abs' \rangle \\
\langle c_1 ; c_2, abs \rangle & \Rightarrow \langle c_2, abs' \rangle
\end{align*}
\]
### Abstract Semantics of WHILE II

**Definition 13.9 (Abstract execution relation for statements; continued)**

1. **(if1)**
   
   \[
   \exists \rho \in \text{abs} : \text{true} \in \text{val}_\rho^#(b) \\
   \langle \text{if } b \text{ then } c_1 \text{ else } c_2 \text{ end}, \text{abs} \rangle \Rightarrow \langle c_1, \text{abs} \setminus \{\rho \in \text{abs} \mid \text{val}_\rho^#(b) = \{\text{false}\}\} \rangle
   \]

2. **(if2)**
   
   \[
   \exists \rho \in \text{abs} : \text{false} \in \text{val}_\rho^#(b) \\
   \langle \text{if } b \text{ then } c_1 \text{ else } c_2 \text{ end}, \text{abs} \rangle \Rightarrow \langle c_2, \text{abs} \setminus \{\rho \in \text{abs} \mid \text{val}_\rho^#(b) = \{\text{true}\}\} \rangle
   \]

3. **(wh1)**
   
   \[
   \exists \rho \in \text{abs} : \text{true} \in \text{val}_\rho^#(b) \\
   \langle \text{while } b \text{ do } c \text{ end}, \text{abs} \rangle \Rightarrow \langle c;\text{while } b \text{ do } c \text{ end}, \text{abs} \setminus \{\rho \in \text{abs} \mid \text{val}_\rho^#(b) = \{\text{false}\}\} \rangle
   \]

4. **(wh2)**
   
   \[
   \exists \rho \in \text{abs} : \text{false} \in \text{val}_\rho^#(b) \\
   \langle \text{while } b \text{ do } c \text{ end}, \text{abs} \rangle \Rightarrow \langle \downarrow, \text{abs} \setminus \{\rho \in \text{abs} \mid \text{val}_\rho^#(b) = \{\text{true}\}\} \rangle
   \]
Abstract Semantics of WHILE

Abstract Semantics of WHILE III

Definition 13.10 (Abstract transition function)

The abstract transition function is defined by the family of mappings

$$\text{next}_{c,c'} : \text{Abs} \rightarrow \text{Abs},$$

given by

$$\text{next}_{c,c'}(\text{abs}) := \bigcup \{ \text{abs}' \in \text{Abs} \mid \langle c, \text{abs} \rangle \Rightarrow \langle c', \text{abs}' \rangle \}$$
Abstract Semantics of WHILE

Abstract Semantics of WHILE III

Definition 13.10 (Abstract transition function)

The abstract transition function is defined by the family of mappings

$$\text{next}^\#_{c,c'} : \text{Abs} \to \text{Abs},$$

given by

$$\text{next}^\#_{c,c'}(abs) := \bigcup \{ abs' \in \text{Abs} \mid \langle c, abs \rangle \Rightarrow \langle c', abs' \rangle \}$$

Example 13.11 (Hailstone Sequences; cf. Example 12.7)

```
[skip]^1;
while [¬(n = 1)]^2 do
  if [even(n)]^3 then
    [n := n / 2]^4; [skip]^5
  else
    [n := 3 * n + 1]^6; [skip]^7
  end
end
```

Execution relation with parity abstraction: see following slide (courtesy B. König)
Abstrakte Interpretation von HAILSTONE

\[
\langle \text{skip} \rangle^1; \ldots, \{n \mapsto \text{odd}\} \quad \rightarrow \quad \{n \mapsto \text{odd}\} \leftarrow
\]

\[
\langle \text{while } [n \neq 1]^2 \ldots, \{n \mapsto \text{odd}\} \rangle \quad \rightarrow \quad \langle \text{while } [n \neq 1]^2 \ldots, \{n \mapsto \text{even}\} \rangle
\]

\[
\langle \text{if } \text{even}(n)^3 \ldots, \{n \mapsto \text{odd}\} \rangle \quad \rightarrow \quad \langle \text{if } \text{even}(n)^3 \ldots, \{n \mapsto \text{even}\} \rangle
\]

\[
\rightarrow \langle n := 3n + 1 \rangle^6; \ldots, \{n \mapsto \text{odd}\} \quad \rightarrow \quad \langle n := n/2 \rangle^4; \ldots, \{n \mapsto \text{even}\}
\]

\[
\langle \text{skip} \rangle^7; \ldots, \{n \mapsto \text{even}\} \quad \rightarrow \quad \langle \text{skip} \rangle^5; \ldots, \{n \mapsto \text{even}, n \mapsto \text{odd}\}
\]

\[
\langle \text{while } [n \neq 1]^2 \ldots, \{n \mapsto \text{even}\}, [n \mapsto \text{odd}] \rangle \quad \rightarrow \quad \langle \text{while } [n \neq 1]^2 \ldots, \{n \mapsto \text{even}, n \mapsto \text{odd}\} \rangle
\]

\[
\langle \text{if } \text{even}(n)^3 \ldots, \{n \mapsto \text{even}, n \mapsto \text{odd}\} \rangle
\]
Correctness of Abstract Semantics

Outline of Lecture 13

Recap: Safe Approximation of Functions and Relations

Systematic Construction of Abstractions

Abstract Semantics of WHILE

Correctness of Abstract Semantics
Correctness of Abstract Semantics

Theorem 13.12 (Soundness of abstract semantics)

For each $c \in \text{Cmd}$ and $c' \in \text{Cmd} \cup \{\downarrow\}$, $\text{next}^{\#}_{c,c'}$ is a safe approximation of $\text{next}_{c,c'}$, i.e., for every $\text{abs} \in \text{Abs}$, $\alpha(\text{next}_{c,c'}(\gamma(\text{abs}))) \subseteq \text{next}^{\#}_{c,c'}(\text{abs})$. 

Proof (Lemma 13.13).

Proof (Theorem 13.12).
Correctness of Abstract Semantics

Theorem 13.12 (Soundness of abstract semantics)

For each \( c \in \text{Cmd} \) and \( c' \in \text{Cmd} \cup \{\downarrow\} \), \( \text{next}_{c,c'}^\# \) is a safe approximation of \( \text{next}_{c,c'} \), i.e., for every \( \text{abs} \in \text{Abs} \), \( \alpha(\text{next}_{c,c'}(\gamma(\text{abs}))) \subseteq \text{next}_{c,c'}^\#(\text{abs}) \).

The soundness proof employs the following auxiliary lemma.

Lemma 13.13 (Soundness of abstract evaluation)

Let \( \beta : \mathbb{Z} \to A \) be an extraction function.

1. For every \( a \in \text{AExp} \) and \( \sigma \in \Sigma \), \( \beta(\text{val}_\sigma(a)) \in \text{val}_{\beta_\sigma}^\#(a) \).
2. For every \( b \in \text{BExp} \) and \( \sigma \in \Sigma \), \( \text{val}_\sigma(b) \in \text{val}_{\beta_\sigma}^\#(b) \).

Proof (Lemma 13.13).

omitted
Correctness of Abstract Semantics

**Theorem 13.12 (Soundness of abstract semantics)**

For each \( c \in \text{Cmd} \) and \( c' \in \text{Cmd} \cup \{ \downarrow \} \), \( \text{next}_{c,c'}^{\text{#}} \) is a **safe approximation** of \( \text{next}_{c,c'} \), i.e., for every \( \text{abs} \in \text{Abs} \), \( \alpha(\text{next}_{c,c'}(\gamma(\text{abs})))) \subseteq \text{next}_{c,c'}^{\text{#}}(\text{abs}) \).

The soundness proof employs the following auxiliary lemma.

**Lemma 13.13 (Soundness of abstract evaluation)**

Let \( \beta : \mathbb{Z} \rightarrow \text{A} \) be an extraction function.

1. For every \( a \in \text{AExp} \) and \( \sigma \in \Sigma \), \( \beta(\text{val}_\sigma(a)) \in \text{val}_{\beta \circ \sigma}^{\#}(a) \).
2. For every \( b \in \text{BExp} \) and \( \sigma \in \Sigma \), \( \text{val}_\sigma(b) \in \text{val}_{\beta \circ \sigma}^{\#}(b) \).

**Proof (Lemma 13.13).**

omitted

**Proof (Theorem 13.12).**

on the board