# Static Program Analysis

## Lecture 12: Abstract Interpretation II (Safe Approximation of Functions and Relations)

Thomas Noll

Lehrstuhl für Informatik 2
(Software Modeling and Verification)

RWTH AACHEN UNIVERSITY

noll@cs.rwth-aachen.de

http://moves.rwth-aachen.de/teaching/ws-1415/spa/

Winter Semester 2014/15

# Outline

# Galois Connections I

**Definition (Galois connection)**

Let $(L, \sqsubseteq_L)$ and $(M, \sqsubseteq_M)$ be complete lattices. A pair $(\alpha, \gamma)$ of monotonic functions

$$\alpha : L \to M \quad \text{and} \quad \gamma : M \to L$$

is called a Galois connection if

$$\forall l \in L : l \sqsubseteq_L \gamma(\alpha(l)) \quad \text{and} \quad \forall m \in M : \alpha(\gamma(m)) \sqsubseteq_M m$$

Evariste Galois (1811–1832)

**Interpretation:**

- $L = \{\text{sets of concrete values}\}$, $M = \{\text{sets of abstract values}\}$
- $\alpha =$ abstraction function, $\gamma =$ concretization function
- $l \sqsubseteq_L \gamma(\alpha(l))$: $\alpha$ yields over-approximation
- $\alpha(\gamma(m)) \sqsubseteq_M m$: no loss of precision by abstraction after concretization
- Usually: $l \neq \gamma(\alpha(l))$, $\alpha(\gamma(m)) = m$

# Properties of Galois Connections

## Lemma

*Let $(\alpha, \gamma)$ be a Galois connection with $\alpha : L \to M$ and $\gamma : M \to L$, and let $l \in L$, $m \in M$, $L' \subseteq L$, $M' \subseteq M$.*

1. $\alpha(l) \sqsubseteq_M m \iff l \sqsubseteq_L \gamma(m)$
2. $\gamma$ *is uniquely determined by* $\alpha$ *as follows:*
$$\gamma(m) = \bigsqcup\{l \in L \mid \alpha(l) \sqsubseteq_M m\}$$
3. $\alpha$ *is uniquely determined by* $\gamma$ *as follows:*
$$\alpha(l) = \bigsqcap\{m \in M \mid l \sqsubseteq_L \gamma(m)\}$$
4. $\alpha$ *is completely distributive:* $\alpha(\bigsqcup L') = \bigsqcup\{\alpha(l) \mid l \in L'\}$
5. $\gamma$ *is completely multiplicative:* $\gamma(\bigsqcap M') = \bigsqcap\{\gamma(m) \mid m \in M'\}$

## Proof.

on the board □

# Outline

# Evaluation of Expressions

## Definition (Evaluation function)

Let $\sigma \in \Sigma$ be a state.

1. $val_\sigma : AExp \to \mathbb{Z} : a \to val_\sigma(a)$
   yields the value of $a$ in state $\sigma$
2. $val_\sigma : BExp \to \mathbb{B} : b \to val_\sigma(b)$
   yields the value of $b$ in state $\sigma$

## Example

Let $\sigma(\mathtt{x}) = 1$ and $\sigma(\mathtt{y}) = 2$.

1. $val_\sigma(\mathtt{2 * x + y}) = 4$
2. $val_\sigma(\neg(\mathtt{x + 1 > y})) = \mathsf{true}$

# Execution of Statements I

## Definition (Execution relation for statements)

If $c \in Cmd$ and $\sigma \in \Sigma$, then $\langle c, \sigma \rangle$ is called a configuration. The execution relation

$$\rightarrow \subseteq \ (Cmd \times \Sigma) \times ((Cmd \cup \{\downarrow\}) \times \Sigma)$$

is defined by the following rules:

$$\text{(skip)} \frac{}{\langle \texttt{skip}, \sigma \rangle \rightarrow \langle \downarrow, \sigma \rangle}$$

$$\text{(asgn)} \frac{}{\langle x \texttt{ := } a, \sigma \rangle \rightarrow \langle \downarrow, \sigma[x \mapsto val_\sigma(a)] \rangle}$$

$$\text{(seq1)} \frac{\langle c_1, \sigma \rangle \rightarrow \langle c_1', \sigma' \rangle \quad c_1' \neq \downarrow}{\langle c_1 ; c_2, \sigma \rangle \rightarrow \langle c_1' ; c_2, \sigma' \rangle}$$

$$\text{(seq2)} \frac{\langle c_1, \sigma \rangle \rightarrow \langle \downarrow, \sigma' \rangle}{\langle c_1 ; c_2, \sigma \rangle \rightarrow \langle c_2, \sigma' \rangle}$$

# Execution of Statements II

**Definition (Execution relation for statements; continued)**

$$\text{(if1)}\ \frac{val_\sigma(b) = \text{true}}{\langle \texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2, \sigma\rangle \to \langle c_1, \sigma\rangle}$$

$$\text{(if2)}\ \frac{val_\sigma(b) = \text{false}}{\langle \texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2, \sigma\rangle \to \langle c_2, \sigma\rangle}$$

$$\text{(wh1)}\ \frac{val_\sigma(b) = \text{true}}{\langle \texttt{while } b \texttt{ do } c, \sigma\rangle \to \langle c\texttt{;while } b \texttt{ do } c, \sigma\rangle}$$

$$\text{(wh2)}\ \frac{val_\sigma(b) = \text{false}}{\langle \texttt{while } b \texttt{ do } c, \sigma\rangle \to \langle \downarrow, \sigma\rangle}$$

**Remark:** $\downarrow$ indicates successful termination of the program

# Outline

# An Execution Example

## Example 12.1

- $c := \underbrace{\texttt{y := 1; while} \underbrace{\neg\texttt{(x=1)}}_{b} \texttt{do} \underbrace{\underbrace{\texttt{y := y*x}}_{c_1} \texttt{;} \underbrace{\texttt{x := x-1}}_{c_2}}_{c_0}}$
- Claim: $\langle c, \sigma \rangle \to^+ \langle \downarrow, \sigma_{1,6} \rangle$ for every $\sigma \in \Sigma$ with $\sigma(\texttt{x}) = 3$
- Notation: $\sigma_{i,j}$ means $\sigma(\texttt{x}) = i$, $\sigma(\texttt{y}) = j$
- Derivation: on the board

# Determinism Property of Execution Relation

This operational semantics is well defined in the following sense:

**Theorem 12.2**

*The execution relation for statements is deterministic, i.e., whenever $c \in Cmd$, $\sigma \in \Sigma$ and $\kappa_1, \kappa_2 \in (Cmd \cup \{\downarrow\}) \times \Sigma$ such that $\langle c, \sigma \rangle \to \kappa_1$ and $\langle c, \sigma \rangle \to \kappa_2$, then $\kappa_1 = \kappa_2$.*

**Proof.**

omitted □

More on formal semantics of programming languages:
*Semantics and Verification of Software* in forthcoming summer semester

# Outline

# Safe Approximation of Functions I

## Definition 12.3 (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 it is called most precise safe approximation if the reverse inclusion is also true.

$$\begin{array}{ccc} \textbf{Abstract} & & \textbf{Concrete} \\ \vec{m} & \xrightarrow{\gamma} & \gamma(\vec{m}) \\ \downarrow f^\# & & \downarrow f \\ f^\#(\vec{m}) \sqsupseteq \alpha(f(\gamma(\vec{m}))) & \xleftarrow{\alpha} & f(\gamma(\vec{m})) \end{array}$$

- **Interpretation:** the abstraction $f^\#$ of $f$ covers all concrete results
- **Note:** monotonicity of $f$ and/or $f^\#$ is *not* required (but usually given; see Lemma 12.5)

# Safe Approximation of Functions II

**Reminder:** $\alpha(f(\gamma(m_1), \dots, \gamma(m_n))) \sqsubseteq_M f^\#(m_1, \dots, m_n)$

## Example 12.4

1. Parity abstraction (cf. Example 11.2): most precise approximations
   - $n = 0$: $1^\# = \{\text{odd}\}$
   - $n = 1$: $-^\#(P) = P$, $(-1)^\#(\{\text{even}\}) = \{\text{odd}\}$
   - $n = 2$: $\{\text{even}\} +^\# \{\text{odd}\} = \{\text{odd}\}$, $\{\text{even}\} \cdot^\# \{\text{odd}\} = \{\text{even}\}$
2. Sign abstraction (cf. Example 11.3): most precise approximations
   - $n = 0$: $1^\# = \{+\}$
   - $n = 1$: $-^\#(\{+\}) = \{-\}$, $(-1)^\#(\{+\}) = \{+, 0\}$
   - $n = 2$: $\{+\} +^\# \{+\} = \{+\}$
     $\{+\} +^\# \{-\} = \{+, -, 0\}$
     $\{+\} \cdot^\# \{-\} = \{-\}$
3. Interval abstraction (cf. Example 11.4): most precise approximations
   - $n = 0$: $z^\# = [z, z]$
   - $n = 1$: $-^\#([z_1, z_2]) = [-z_2, -z_1]$, $(-1)^\#([z_1, z_2]) = [z_1 - 1, z_2 - 1]$
   - $n = 2$: $[y_1, y_2] +^\# [z_1, z_2] = [y_1 + z_1, y_2 + z_2]$
     $[y_1, y_2] -^\# [z_1, z_2] = [y_1 - z_2, y_2 - z_1]$

# Safe Approximation of Functions III

## Lemma 12.5

*If $f : L^n \to L$ and $f^\# : M^n \to M$ are monotonic, then $f^\#$ is a safe approximation of $f$ iff, for all $l_1, \ldots, l_n \in L$,*

$$\alpha(f(l_1, \ldots, l_n)) \sqsubseteq_M f^\#(\alpha(l_1), \ldots, \alpha(l_n)).$$

## Proof.

on the board □

# Outline

# Safe Approximation of Execution Relation I

- **Reminder:** concrete semantics of WHILE
  - states $\Sigma := \{\sigma \mid \sigma : Var \to \mathbb{Z}\}$ (Definition 11.6)
  - execution relation $\to \subseteq (Cmd \times \Sigma) \times ((Cmd \cup \{\downarrow\}) \times \Sigma)$ (Definition 11.9)
- Yields concrete domain $L := 2^\Sigma$ and concrete transition function:

**Definition 12.6 (Concrete transition function)**

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

$$\mathrm{next}_{c,c'} : 2^\Sigma \to 2^\Sigma$$

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

$$\mathrm{next}_{c,c'}(S) := \{\sigma' \in \Sigma \mid \exists \sigma \in S : \langle c, \sigma \rangle \to \langle c', \sigma' \rangle\}.$$

# Safe Approximation of Execution Relation II

**Remarks:** next satisfies the following properties

- "Determinism" (cf. Theorem 12.2):
  - for all $c \in Cmd$, $c' \in Cmd \cup \{\downarrow\}$ and $\sigma \in \Sigma$: $|\mathsf{next}_{c,c'}(\{\sigma\})| \leq 1$
  - for all $c \in Cmd$ and $\sigma \in \Sigma$ there exists exactly one $c' \in Cmd \cup \{\downarrow\}$ such that $|\mathsf{next}_{c,c'}(\{\sigma\})| \neq \emptyset$
- When is $\mathsf{next}_{c,c'}(S) = \emptyset$? Possibilities:
  1. $S = \emptyset$
  2. $c'$ not a possible successor statement of $c$, e.g.,
     - $c = (\texttt{x := 0})$
     - $c' = \texttt{skip}$
  3. $c'$ unreachable for all $\sigma \in S$, e.g.,
     - $c = (\texttt{if x = 0 then x := 1 else skip})$
     - $c' = \texttt{skip}$
     - $\sigma(\texttt{x}) = 0$ for each $\sigma \in S$

# Safe Approximation of Execution Relation III

- **Reminder:** abstraction determined by Galois connection $(\alpha, \gamma)$ with $\alpha : L \to M$ and $\gamma : M \to L$
  - here: $L := 2^{\Sigma}$, $M$ not fixed (usually $M = Var \to \ldots$ or $M = 2^{Var \to \ldots}$)
  - write $Abs$ in place of $M$
  - thus $\alpha : 2^{\Sigma} \to Abs$ and $\gamma : Abs \to 2^{\Sigma}$
- Yields abstract semantics:

**Definition 12.7 (Abstract semantics of WHILE)**

Given $\alpha : 2^{\Sigma} \to Abs$, an abstract semantics is defined by a family of functions

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

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

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

for every $abs \in Abs$.
Notation: $\langle c, abs \rangle \Rightarrow \langle c', abs' \rangle$ for $\text{next}^{\#}_{c,c'}(abs) = abs'$.

## Example 12.8 (Parity abstraction (cf. Example 11.2))

- $Abs = 2^{Var \to \{\text{even}, \text{odd}\}}$
- $Var = \{\texttt{n}\}$
- Notation: $[\texttt{n} \mapsto p] \in abs \in Abs$ for $p \in \{\text{even}, \text{odd}\}$
- Some abstract (non-)transitions:

$$\begin{array}{rr} & \langle \texttt{n := 3 * n + 1}, \{[\texttt{n} \mapsto \text{odd}]\}\rangle \\ \Rightarrow & \langle \downarrow, \{[\texttt{n} \mapsto \text{even}]\}\rangle \end{array}$$

$$\begin{array}{rr} & \langle \texttt{n := 2 * n + 1}, \{[\texttt{n} \mapsto \text{even}], [\texttt{n} \mapsto \text{odd}]\}\rangle \\ \Rightarrow & \langle \downarrow, \{[\texttt{n} \mapsto \text{odd}]\}\rangle \end{array}$$

$$\begin{array}{rr} & \langle \texttt{while } \neg\texttt{(n=1) do } c, \{[\texttt{n} \mapsto \text{odd}]\}\rangle \\ \Rightarrow & \langle \downarrow, \{[\texttt{n} \mapsto \text{odd}]\}\rangle \end{array}$$

$$\begin{array}{rr} & \langle \texttt{while } \neg\texttt{(n=1) do } c, \{[\texttt{n} \mapsto \text{odd}]\}\rangle \\ \Rightarrow & \langle c\texttt{; while } \neg\texttt{(n=1) do } c, \{[\texttt{n} \mapsto \text{odd}]\}\rangle \end{array}$$

$$\begin{array}{rr} & \langle \texttt{while } \neg\texttt{(n=1) do } c, \{[\texttt{n} \mapsto \text{even}]\}\rangle \\ \not\Rightarrow & \langle \downarrow, \{[\texttt{n} \mapsto \text{even}]\}\rangle \end{array}$$

$$\begin{array}{rr} & \langle \texttt{while } \neg\texttt{(n=1) do } c, \{[\texttt{n} \mapsto \text{even}]\}\rangle \\ \Rightarrow & \langle c\texttt{; while } \neg\texttt{(n=1) do } c, \{[\texttt{n} \mapsto \text{even}]\}\rangle \end{array}$$