# Static Program Analysis

## Lecture 13: Abstract Interpretation III (Abstract Interpretation of WHILE Programs)

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

# Safe Approximation of Functions

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

| **Abstract** | | **Concrete** |
|---|---|---|
| $\vec{m}$ | $\xrightarrow{\gamma}$ | $\gamma(\vec{m})$ |
| $\downarrow f^\#$ | | $\downarrow f$ |
| $f^\#(\vec{m}) \sqsupseteq \alpha(f(\gamma(\vec{m})))$ | $\xleftarrow{\alpha}$ | $f(\gamma(\vec{m}))$ |

- **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 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 (Concrete transition function)**

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

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

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

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

# 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 $\sigma \in \Sigma$: $|\text{next}_{c,c'}(\{\sigma\})| \leq 1$
  - for all $c \in Cmd$ and $\sigma \in \Sigma$ there exists exactly one $c' \in Cmd \cup \{\downarrow\}$ such that $\text{next}_{c,c'}(\{\sigma\}) \neq \emptyset$
- When is $\text{next}_{c,c'}(S) = \emptyset$? Possible reasons:
  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 (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 $\text{next}^\#_{c,c'}(abs) = abs'$.

# Outline

## Example 13.1 (Hailstone Sequences)

```
[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;
```

- additional `skip` statements only for labels
- abstract transition system for $\sigma(\mathtt{n}) \in \mathbb{Z}_{\mathtt{odd}}$: on the board
- formal derivation later

- Collatz Conjecture: given any $n > 0$, the program finally returns 1 (that is, every Hailstone Sequence terminates with 1)
- see http://en.wikipedia.org/wiki/Collatz_conjecture
- AKA $3n + 1$ Conjecture, Ulam Conjecture, Kakutani's Problem, Thwaites' Conjecture, Hasse's Algorithm, or Syracuse Problem
- New proof attempt by Gerhard Opfer from Hamburg University (http://preprint.math.uni-hamburg.de/public/papers/hbam/hbam2011-09.pdf)

# Outline

# Derivation of Abstract Semantics

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

## Example 13.2 (Fermat's Last Theorem)

Sign abstraction (cf. Example 11.3) on

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

- Result $\texttt{n} = 1$ possible iff there exist $n > 2$ and $x, y, z \geq 1$ such that $x^n + y^n = z^n$
- Fermat's Last Theorem: equation not solvable
- Final proof by Andrew Wiles and Richard Taylor in 1995

- More general: solvability of Diophantic equations undecidable
- Thus: resort to possibly imprecise safe approximations

# Extraction Functions

- **Assumption:** abstraction determined by pointwise mapping of concrete elements
- 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 13.3**

1. Parity abstraction (cf. Example 11.2): $\beta : \mathbb{Z} \to \{\text{even}, \text{odd}\}$ where
$$\beta(z) := \begin{cases} \text{even} & \text{if } z \text{ even} \\ \text{odd} & \text{if } z \text{ odd} \end{cases}$$
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 $Int$ is not of the form $2^A$)

**Reminder:** safe approximation condition (Definition 12.3)

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

## Theorem 13.4

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

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

*is a safe approximation of* $f$.

## Proof.

on the board □

# Safe Approximation of Arithmetic Operations

## Example 13.5 (Sign abstraction)

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

| $+^\#$ | $\{+\}$ | $\{-\}$ | $\{0\}$ |
|---|---|---|---|
| $\{+\}$ | $\{+\}$ | $\{+, -, 0\}$ | $\{+\}$ |
| $\{-\}$ | $\{+, -, 0\}$ | $\{-\}$ | $\{-\}$ |
| $\{0\}$ | $\{+\}$ | $\{-\}$ | $\{0\}$ |

| $*^\#$ | $\{+\}$ | $\{-\}$ | $\{0\}$ |
|---|---|---|---|
| $\{+\}$ | $\{+\}$ | $\{-\}$ | $\{0\}$ |
| $\{-\}$ | $\{-\}$ | $\{+\}$ | $\{0\}$ |
| $\{0\}$ | $\{0\}$ | $\{0\}$ | $\{0\}$ |

and
$$\begin{aligned}\{+, 0\} *^\# \{-\} &= \{+\} *^\# \{-\} \cup \{0\} *^\# \{-\} \\ &= \{-\} \cup \{0\} \\ &= \{-, 0\}\end{aligned}$$

etc.

# Safe Approximation of Boolean Operations

## Example 13.6 (Sign abstraction)

1. Relational operations:
   - $C = \mathbb{Z} \cup \mathbb{B}$, $A = \{+, -, 0\} \cup \mathbb{B}$, $\beta = \mathsf{sgn}$
   -

| $=^\#$ | $\{+\}$ | $\{-\}$ | $\{0\}$ |
|---|---|---|---|
| $\{+\}$ | $\{\mathsf{true}, \mathsf{false}\}$ | $\{\mathsf{false}\}$ | $\{\mathsf{false}\}$ |
| $\{-\}$ | $\{\mathsf{false}\}$ | $\{\mathsf{true}, \mathsf{false}\}$ | $\{\mathsf{false}\}$ |
| $\{0\}$ | $\{\mathsf{false}\}$ | $\{\mathsf{false}\}$ | $\{\mathsf{true}\}$ |

   -

| $>^\#$ | $\{+\}$ | $\{-\}$ | $\{0\}$ |
|---|---|---|---|
| $\{+\}$ | $\{\mathsf{true}, \mathsf{false}\}$ | $\{\mathsf{true}\}$ | $\{\mathsf{true}\}$ |
| $\{-\}$ | $\{\mathsf{false}\}$ | $\{\mathsf{true}, \mathsf{false}\}$ | $\{\mathsf{false}\}$ |
| $\{0\}$ | $\{\mathsf{false}\}$ | $\{\mathsf{true}\}$ | $\{\mathsf{false}\}$ |

   - $\{+, 0\} =^\# \{0\} = \{+\} =^\# \{0\} \cup \{0\} =^\# \{0\} = \{\mathsf{false}\} \cup \{\mathsf{true}\} = \{\mathsf{true}, \mathsf{false}\}$ etc.
2. Boolean connectives:
   - $C = A = \mathbb{B}$, $\neg^\# = \neg$, $\wedge^\# = \wedge$, ...
   - $\{\mathsf{true}, \mathsf{false}\} \wedge^\# \{\mathsf{true}\} = \{\mathsf{true}\} \wedge^\# \{\mathsf{true}\} \cup \{\mathsf{false}\} \wedge^\# \{\mathsf{true}\} = \{\mathsf{true}\} \cup \{\mathsf{false}\} = \{\mathsf{true}, \mathsf{false}\}$ etc.

# Abstract Program States

**Now:** take values of variables into account

### Definition 13.7 (Abstract program state)

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

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

# Abstract Evaluation of Expressions

## Definition 13.8 (Abstract evaluation functions)

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

1. $val^\#_\rho : AExp \to 2^A$ is determined by ($f$ arithmetic operation)

$$
\begin{aligned}
val^\#_\rho(z) &:= \{\beta(z)\} \\
val^\#_\rho(x) &:= \{\rho(x)\} \\
val^\#_\rho(f(a_1, \ldots, a_n)) &:= f^\#(val^\#_\rho(a_1), \ldots, val^\#_\rho(a_n))
\end{aligned}
$$

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

$$
\begin{aligned}
val^\#_\rho(t) &:= \{t\} \\
val^\#_\rho(g(a_1, \ldots, a_n)) &:= g^\#(val^\#_\rho(a_1), \ldots, val^\#_\rho(a_n)) \\
val^\#_\rho(h(b_1, \ldots, b_n)) &:= h^\#(val^\#_\rho(b_1), \ldots, val^\#_\rho(b_n))
\end{aligned}
$$

## Example 13.9 (Sign abstraction)

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

1. $val^\#_\rho(\mathtt{2 * x + y}) = \{+, -, 0\}$
2. $val^\#_\rho(\neg\mathtt{(x + 1 > y)}) = \{\mathsf{false}\}$

# Outline

# Abstract Semantics of WHILE I

**Reminder:** abstract domain is $Abs := 2^{Var \to A}$

## Definition 13.1 (Abstract execution relation for statements)

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

$$\text{(skip)}\ \frac{}{\langle \texttt{skip}, abs \rangle \Rightarrow \langle \downarrow, abs \rangle}$$

$$\text{(asgn)}\ \frac{}{\langle x \texttt{ := } a, abs \rangle \Rightarrow \langle \downarrow, \{\rho[x \mapsto a'] \mid \rho \in abs, a' \in val^{\#}_{\rho}(a)\} \rangle}$$

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

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

# Abstract Semantics of WHILE II

## Definition 13.1 (Abstract execution relation for statements; cont.)

$$\text{(if1)}\ \frac{\exists \rho \in abs : \text{true} \in val^{\#}_{\rho}(b)}{\begin{array}{l}\langle \texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2, abs\rangle \\ \Rightarrow \langle c_1, abs \setminus \{\rho \in abs \mid val^{\#}_{\rho}(b) = \{\text{false}\}\}\rangle\end{array}}$$

$$\text{(if2)}\ \frac{\exists \rho \in abs : \text{false} \in val^{\#}_{\rho}(b)}{\begin{array}{l}\langle \texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2, abs\rangle \\ \Rightarrow \langle c_2, abs \setminus \{\rho \in abs \mid val^{\#}_{\rho}(b) = \{\text{true}\}\}\rangle\end{array}}$$

$$\text{(wh1)}\ \frac{\exists \rho \in abs : \text{true} \in val^{\#}_{\rho}(b)}{\begin{array}{l}\langle \texttt{while } b \texttt{ do } c, abs\rangle \\ \Rightarrow \langle c\texttt{;while } b \texttt{ do } c, abs \setminus \{\rho \in abs \mid val^{\#}_{\rho}(b) = \{\text{false}\}\}\rangle\end{array}}$$

$$\text{(wh2)}\ \frac{\exists \rho \in abs : \text{false} \in val^{\#}_{\rho}(b)}{\langle \texttt{while } b \texttt{ do } c, abs\rangle \Rightarrow \langle \downarrow, abs \setminus \{\rho \in abs \mid val^{\#}_{\rho}(b) = \{\text{true}\}\}\rangle}$$

# Abstract Semantics of WHILE III

## Definition 13.2 (Abstract transition function)

The abstract transition function is defined by the family of mappings

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

given by

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

## Example 13.3 (Hailstone Sequences; cf. Example 13.1)

```
[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;
```

Execution relation with parity abstraction: see following slide (courtesy B. König)

# Abstrakte Interpretation von HAILSTONE