Static Program Analysis

Lecture 11: Abstract Interpretation II (Safe Approximation)

Summer Semester 2018

Thomas Noll
Software Modeling and Verification Group
RWTH Aachen University

https://moves.rwth-aachen.de/teaching/ss-18/spa/

Software Modeling and Verification Chair

RWTH AACHEN UNIVERSITY

Vortrag und Diskussion

Process Science and Data Science: A Match Made in Heaven!

Prof. Dr. Wil van der Aalst

Freitag | 08. Juni 2018 | 15.30 - 16.45 Uhr | Aula 2 der RWTH | Ahornstr. 55

Eintritt frei. Anmeldung nicht erforderlich. Der Vortrag findet im Rahmen des Sommerfests der Informatik der RWTH statt. Bitte pünktlich erscheinen.

The Process and Data Science (PADS) group, headed by prof.dr.ir. Wil van der Aalst, is a new research unit in RWTH's Department of Computer Science. The scope of PADS includes all activities where discrete processes are analyzed, reengineered, and/or supported in a data-driven manner. Process-centricity is combined with an array of Data Science techniques.

This talk will introduce Process Mining as a novel way to turn event data into valuable insights, predictions, and decisions. Events refer to activities executed by resources at particular times and for particular cases. Such event data are collected everywhere; in logistics, manufacturing, finance, healthcare, customer relationship management, e-learning, e-government, and many other domains. Process Mining can be used to discover the real processes, to detect deviations from normative processes, and to analyze bottlenecks and waste.

The interplay between Process Science and Data Science generates many interesting research problems. For example, how to deal with terabytes of event data scattered over dozens of database tables? How to use process mining responsibly (e.g., ensure fairness and confidentiality)? How to amalgamate Process Mining with Operations Research approaches (optimization, simulation, etc.)? These questions are scientifically interesting and practically relevant. This is illustrated by the 25+ commercial Process Mining tools based on the ideas developed by prof. van der Aalst and his colleagues. His talk will show different tools and their use in a variety of domains.

In Kooperation mit Fachgruppe Informatik, der Regionalgruppe der Gesellschaft für Informatik (RIA) und des Regionalen Industrieclubs Informatik Aachen (Regina e.V.)

RWTH AACHEN UNIVERSITY

Galois Connections

Definition (Galois connection)

Let $(L, \sqsubseteq_L)$ and $(M, \sqsubseteq_M)$ be complete lattices. A pair (α, γ) 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, $\gamma =$ concretisation function
- $l \sqsubseteq_L \gamma(\alpha(l))$: α yields over-approximation
- $\alpha(\gamma(m)) \sqsubseteq_M m$: no loss of precision by abstraction after concretisation
- Usually: $l \neq \gamma(\alpha(l))$, $\alpha(\gamma(m)) = m$ (“Galois insertion”)

Recap: Galois Connections

Properties of Galois Connections

Lemma

Let (α, γ) 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. γ *is uniquely determined by* α *as follows:* $\gamma(m) = \bigsqcup\{l \in L \mid \alpha(l) \sqsubseteq_M m\}$
3. α *is uniquely determined by* γ *as follows:* $\alpha(l) = \bigsqcap\{m \in M \mid l \sqsubseteq_L \gamma(m)\}$
4. α *is completely distributive: for every* $L' \subseteq L$, $\alpha(\bigsqcup L') = \bigsqcup\{\alpha(l) \mid l \in L'\}$
5. γ *is completely multiplicative: for every* $M' \subseteq M$, $\gamma(\bigsqcap M') = \bigsqcap\{\gamma(m) \mid m \in M'\}$

Proof.

on the board □

Execution of Statements I

Definition (Execution relation for statements)

If $c \in Cmd_\downarrow$ for $TCmd := Cmd \cup \{\downarrow\}$ and $\sigma \in \Sigma$, then $\langle c, \sigma \rangle$ is called a configuration. The execution relation

$$\rightarrow \subseteq (Cmd \times \Sigma) \times (Cmd_\downarrow \times \Sigma)$$

is defined by the following rules:

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

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

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

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

Execution of Statements II

Definition (Execution relation for statements; continued)

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

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

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

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

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

Recap: Concrete Semantics of WHILE Programs

Determinism Property of Execution Relation

This operational semantics is well defined in the following sense:

Theorem

The execution relation for statements is deterministic, i.e., whenever $c \in Cmd$, $\sigma \in \Sigma$ *and* $\kappa_1, \kappa_2 \in Cmd_\downarrow \times \Sigma$ *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 next summer semester

Safe Approximation of Functions I

Definition 11.1 (Safe approximation)

Let (α, γ) 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		**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 $f^\#$ covers all concrete f-results
- **Note:** monotonicity of f or $f^\#$ is *not* required (but usually given; see Lemma 11.3)

Safe Approximation of Functions II

Example 11.2 (Safeness: $\alpha(f(\gamma(m_1), \ldots, \gamma(m_n))) \sqsubseteq_M f^\#(m_1, \ldots, m_n)$)

1. Parity abstraction (cf. Example 10.2): $L = (2^{\mathbb{Z}}, \subseteq)$, $M = (2^{\{\text{even},\text{odd}\}}, \subseteq)$
   - $n = 0$: for $f = \text{one} \subseteq 2^{\mathbb{Z}} : () \mapsto \{1\}$,
     - $\text{one}^\#() = \{\text{odd}\}$ is most precise: $\alpha(\{1\}) = \{\text{odd}\} = \text{one}^\#()$
     - $\text{one}^\#() = \{\text{even}, \text{odd}\}$ is (only) safe: $\alpha(\{1\}) = \{\text{odd}\} \subsetneq \{\text{even}, \text{odd}\} = \text{one}^\#()$
     - $\text{one}^\#() = \{\text{even}\}$ is unsafe: $\alpha(\{1\}) = \{\text{odd}\} \not\subseteq \{\text{even}\} = \text{one}^\#()$
   - $n = 1$: for $f = \text{dec} : 2^{\mathbb{Z}} \to 2^{\mathbb{Z}} : Z \mapsto \{z - 1 \mid z \in Z\}$,
     - $\text{dec}^\#(\{\text{even}\}) = \{\text{odd}\}$ is most precise: $\alpha(\text{dec}(\gamma(\{\text{even}\}))) = \{\text{odd}\} = \text{dec}^\#(\{\text{even}\})$
     - $\text{dec}^\#(\{\text{even}\}) = \{\text{odd}, \text{even}\}$ is (only) safe: $\alpha(\text{dec}(\gamma(\{\text{even}\}))) = \{\text{odd}\} \subsetneq \{\text{odd}, \text{even}\} = \text{dec}^\#(\{\text{even}\})$
     - $\text{dec}^\#(\{\text{even}\}) = \emptyset$ is unsafe: $\alpha(\text{dec}(\gamma(\{\text{even}\}))) = \{\text{odd}\} \not\subseteq \emptyset = \text{dec}^\#(\{\text{even}\})$
   - $n = 2$: for $f = + : 2^{\mathbb{Z}} \times 2^{\mathbb{Z}} \to 2^{\mathbb{Z}} : (z_1, z_2) \mapsto z_1 + z_2$,
     - $\{\text{even}\} +^\# \{\text{odd}\} = \{\text{odd}\}$ is m.p.: $\alpha(\gamma(\{\text{even}\}) + \gamma(\{\text{odd}\})) = \{\text{odd}\} = \{\text{even}\} +^\# \{\text{odd}\}$
     - $\{\text{even}\} +^\# \{\text{odd}\} = \{\text{even}, \text{odd}\}$ is (only) safe: $\alpha(\gamma(\{\text{even}\}) + \gamma(\{\text{odd}\})) = \{\text{odd}\} \subsetneq \{\text{even}, \text{odd}\} = \{\text{even}\} +^\# \{\text{odd}\}$
     - $\{\text{even}\} +^\# \{\text{odd}\} = \{\text{even}\}$ is unsafe: $\alpha(\gamma(\{\text{even}\}) + \gamma(\{\text{odd}\})) = \{\text{odd}\} \not\subseteq \{\text{even}\} = \{\text{even}\} +^\# \{\text{odd}\}$

Safe Approximation of Functions III

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

Example 11.2 (continued)

Most precise approximations (with $L = (2^{\mathbb{Z}}, \subseteq)$):

2. Sign abstraction (cf. Example 10.3): $M = (2^{\{+,-,0\}}, \subseteq)$
   - $n = 0$: $\text{one}^\#() = \{+\}$
   - $n = 1$: $\text{dec}^\#(\{+\}) = \{+, 0\}$, $-^\#(\{+\}) = \{-\}$
   - $n = 2$: $\{+\} +^\# \{+\} = \{+\}$, $\{+\} -^\# \{+\} = \{+, -, 0\}$, $\{+\} \cdot^\# \{-\} = \{-\}$
3. Interval abstraction (cf. Example 10.4): $M = ((\mathbb{Z} \cup \{-\infty\}) \times (\mathbb{Z} \cup \{+\infty\}) \cup \{\emptyset\}, \subseteq)$
   - $n = 0$: $\text{one}^\#() = [1, 1]$
   - $n = 1$: $\text{dec}^\#([z_1, z_2]) = [z_1 - 1, z_2 - 1]$, $-^\#([z_1, z_2]) = [-z_2, -z_1]$
   - $n = 2$: $[y_1, y_2] +^\# [z_1, z_2] = [y_1 + z_1, y_2 + z_2]$
     $[y_1, y_2] -^\# [z_1, z_2] = [y_1 - z_2, y_2 - z_1]$
     $[y_1, y_2] \cdot^\# [z_1, z_2] = [\bigsqcap\{y_1z_1, y_1z_2, y_2z_1, y_2z_2\}, \bigsqcup\{y_1z_1, y_1z_2, y_2z_1, y_2z_2\}]$

   (thus, $+^\#/-^\#/\cdot^\#$ = $\oplus/\ominus/\odot$ from Slide 7.8)

Safe Approximation of Functions IV

Lemma 11.3

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 □

Encoding Execution Relations by Transition Functions I

- **Reminder:** concrete semantics of WHILE
  - statements $\texttt{skip} \mid x := a \mid c_1 ; c_2 \mid \texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2 \texttt{ end} \mid \texttt{while } b \texttt{ do } c \texttt{ end} \in \mathit{Cmd}$
  - states $\Sigma := \{\sigma \mid \sigma : \mathit{Var} \to \mathbb{Z}\}$ (Definition 10.6)
  - execution relation $\to \subseteq (\mathit{Cmd} \times \Sigma) \times ((\mathit{Cmd} \cup \{\downarrow\}) \times \Sigma)$ (Definition 10.9)
- Yields concrete domain $L := (2^\Sigma, \subseteq)$ and concrete transition function:

Definition 11.4 (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 $c \in \mathit{Cmd}$, $c' \in \mathit{Cmd} \cup \{\downarrow\}$ and, for every $S \subseteq \Sigma$,

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

Encoding Execution Relations by Transition Functions II

Remarks: next satisfies the following properties

- "Determinism" (cf. Theorem 10.11):
  - for all $c \in \mathit{Cmd}$, $c' \in \mathit{Cmd} \cup \{\downarrow\}$ and $\sigma \in \Sigma$, $|\mathrm{next}_{c,c'}(\{\sigma\})| \leq 1$
  - for all $c \in \mathit{Cmd}$ and $\sigma \in \Sigma$ there exists exactly one $c' \in \mathit{Cmd} \cup \{\downarrow\}$ such that $\mathrm{next}_{c,c'}(\{\sigma\}) \neq \emptyset$
- When is $\mathrm{next}_{c,c'}(S) = \emptyset$? Possible reasons:
  1. $S = \emptyset$
  2. c' is not a possible successor statement of c, e.g.,
     - $c = (\texttt{x := 0})$
     - $c' = \texttt{skip}$
  3. c' is unreachable for all $\sigma \in S$, e.g.,
     - $c = (\texttt{if x = 0 then x := 1 else skip end})$
     - $c' = \texttt{skip}$
     - $\sigma(\texttt{x}) = 0$ for each $\sigma \in S$

Safe Approximation of Execution Relations

Reminder: abstraction determined by Galois connection (α, γ) with $\alpha : L \to M$, $\gamma : M \to L$

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

Definition 11.5 (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: Parity Abstraction

Example 11.6 (Parity abstraction (cf. Example 10.2))

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

$$\langle \mathtt{n := 3 * n + 1}, \{[\mathtt{n} \mapsto \text{odd}]\}\rangle \Rightarrow \langle\downarrow, \{[\mathtt{n} \mapsto \text{even}]\}\rangle$$

$$\langle \mathtt{n := 2 * n + 1}, \{[\mathtt{n} \mapsto \text{even}], [\mathtt{n} \mapsto \text{odd}]\}\rangle \Rightarrow \langle\downarrow, \{[\mathtt{n} \mapsto \text{odd}]\}\rangle$$

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

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

$$\langle \mathtt{while}\ \neg(\mathtt{n=1})\ \mathtt{do}\ c\ \mathtt{end}, \{[\mathtt{n} \mapsto \text{even}]\}\rangle \Rightarrow \langle\downarrow, \emptyset\rangle$$

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

Example: Hailstone Sequences

Example 11.7 (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
  end
end
```

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

- Collatz Conjecture: given any $n > 0$, the program finally returns 1 (that is, every Hailstone Sequence terminates)
- aka $3n + 1$ *Conjecture*, *Ulam Conjecture*, *Kakutani's Problem*, *Thwaites' Conjecture*, *Hasse's Algorithm*, or *Syracuse Problem*
- Generally assumed to be true (experimental evidence, heuristic arguments)
- Latest (faulty) proof attempt by Gerhard Opfer from Hamburg University (2011)