CS 511, Fall 2018, Handout 32

Brief Introduction to *Abstract Interpretation*

(one more approach to *Static Program Analysis*)

Assaf Kfoury

November 20, 2018 (adjusted: November 29, 2018)

PRELIMINARIES

- This handout on abstract interpretation complements Handout 12 on program unwinding + Handout 25 on program schemes.
- All three handouts cover material in a wider area: **Static Program Analysis** (SPA).
- **General Goal of SPA:** Given a property φ of program behavior (*e.g.*, related to *termination*, or *safety*, or *security*, or *correctness*, etc.), we want to define a formal methodology $\mathcal{M}_\varphi$ to decide for any program P whether every execution of P satisfies φ – *without compiling and running* P.
- The goal is to be achieved statically, *i.e.*, *just by examining the text of the program* P*, prior to any execution*.
- Ideally, methodology $\mathcal{M}_\varphi$ should be amenable to efficient implementation as an *algorithm/automated tool* that works directly on the text of P or indirectly on an abstraction of P (*e.g.*, a flowchart based on P, possibly omitting some aspects of P, in the datatypes used by P or in the control features of P, etc.).

PRELIMINARIES

- SPA is a worthy goal, but not achievable precisely for many properties:

  "Does P halt for every input that is a natural number?",
  "Does P return the expected/correct output on every execution?", etc.

- The difficulty is often bypassed/overcome in different ways:
  - by *limiting* the goal, *e.g.* to finite domains:

    "Does P halt for every input in the range $\{0, 1, \ldots, 100\}$?",
  - by *approximating* the goal:

    "If and when P halts, is its output correct/expected?",
  - by *randomizing* the goal:

    "If and when P halts, is its output *even* or *odd* with equal probability?",
  - by *abstracting* from the goal:

    "If and when P halts, is its output an *even* natural number?",
  - by *omitting programming features*:

    "Does P halt for every input if P is loop-free?",
  - and by other ways.

PRELIMINARIES

- Many good reasons for SPA, including:

  code optimization (to speed up execution),
  safety analysis (such as *type checking*),
  constant propagation,
  live variable analysis,
  strictness analysis,
  bug detection,
  . . .

ABSTRACT INTERPRETATION: small example

- A tiny programming language, $\mathscr{L}_0$, its abstract syntax in extended BNF:

$$\begin{aligned} n \in \text{num} &::= \cdots \mid -3 \mid -2 \mid -1 \mid 0 \mid 1 \mid 2 \mid 3 \mid \cdots \\ e \in \text{exp} &::= n \mid e_1 + e_2 \mid e_1 * e_2 \end{aligned}$$

- Standard (or concrete) interpretation, syntax-directed definition:

$$\begin{aligned} [\![n]\!]_{\text{std}} &\triangleq n && \text{for every } n \in \text{num} \\ [\![e_1 + e_2]\!]_{\text{std}} &\triangleq [\![e_1]\!]_{\text{std}} + [\![e_2]\!]_{\text{std}} && \text{for every } e_1, e_2 \in \text{exp} \\ [\![e_1 * e_2]\!]_{\text{std}} &\triangleq [\![e_1]\!]_{\text{std}} * [\![e_2]\!]_{\text{std}} && \text{for every } e_1, e_2 \in \text{exp} \end{aligned}$$

which imply $[\![\text{exp}]\!]_{\text{std}} \triangleq \left\{ [\![e]\!]_{\text{std}} \mid e \in \text{exp} \right\} = \mathbb{Z}$.

For simplicity in the notation, we have not distinguished between the symbols '$+$' and '$*$' (on the left of $\triangleq$) and their standard interpretations as *addition* and *multiplication* (on the right of $\triangleq$).

ABSTRACT INTERPRETATION: small example (continued)

- Abstract interpretation –

  just one instance of it, the so-called *Rule-of-Sign* interpretation, we choose other instances of *abstract interpretation* depending on the properties we want to study.

  Preliminary notions for *Rule-of-Sign*:

  a new domain, an abstraction of $\mathbb{Z}$:

  $$\textbf{SIGN} \triangleq \{\text{zero}, \text{pos}, \text{neg}, \top\}$$

  one unary map $sign : \mathbb{Z} \to \textbf{SIGN}$, to transfer values from $\mathbb{Z}$ to **SIGN**:

  $$sign(n) \triangleq \begin{cases} \text{zero} & \text{if } n = 0, \\ \text{pos} & \text{if } n > 0, \\ \text{neg} & \text{if } n < 0, \\ \top & \text{if “we don’t know” the sign of } n, \textit{ i.e.} \\ & n \text{ is some unknown number in } \mathbb{Z}. \end{cases}$$

ABSTRACT INTERPRETATION: small example (continued)

two binary operations, $\oplus$ and $\otimes$, corresponding to
$+ : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$ and $* : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$:

$\oplus$: **SIGN** $\times$ **SIGN** $\to$ **SIGN**

$\otimes$: **SIGN** $\times$ **SIGN** $\to$ **SIGN**

$\oplus$	zero	pos	neg	$\top$
zero	zero	pos	neg	$\top$
pos	pos	pos	$\top$	$\top$
neg	neg	$\top$	neg	$\top$
$\top$	$\top$	$\top$	$\top$	$\top$

$\otimes$	zero	pos	neg	$\top$
zero	zero	zero	zero	zero
pos	zero	pos	neg	$\top$
neg	zero	neg	pos	$\top$
$\top$	zero	$\top$	$\top$	$\top$

ABSTRACT INTERPRETATION: small example (continued)

- *Rule-of-Sign* interpretation – an instance of abstract interpretation:

$$\begin{array}{lll} [\![n]\!]_{\text{ros}} & \triangleq sign(n) & \text{for every } n \in \text{num} \\ [\![e_1 + e_2]\!]_{\text{ros}} & \triangleq [\![e_1]\!]_{\text{ros}} \oplus [\![e_2]\!]_{\text{ros}} & \text{for every } e_1, e_2 \in \text{exp} \\ [\![e_1 * e_2]\!]_{\text{ros}} & \triangleq [\![e_1]\!]_{\text{ros}} \otimes [\![e_2]\!]_{\text{ros}} & \text{for every } e_1, e_2 \in \text{exp} \end{array}$$

  which imply $[\![\text{exp}]\!]_{\text{ros}} \triangleq \left\{ [\![e]\!]_{\text{ros}} \mid e \in \text{exp} \right\} = \textbf{SIGN}$.

- for every expression in the original language,
  e.g. $e \triangleq -10 * (71 + 79)$, we have two interpretations:

$$[\![e]\!]_{\text{std}} = -1500 \quad \text{and} \quad [\![e]\!]_{\text{ros}} = \text{neg}$$

  How to relate *standard/concrete* interpretations and *abstract* interpretations? What makes the latter relevant? First, some exercises . . .

ABSTRACT INTERPRETATION: small example (continued)

Exercise. Consider a programming language $\mathscr{L}_1$ which extends $\mathscr{L}_0$ on slide 5 :

$$
\begin{array}{lll}
x \in \text{var} & ::= & \text{a} \mid \text{b} \mid \text{c} \mid \cdots \\
n \in \text{num} & ::= & \cdots \mid \text{-3} \mid \text{-2} \mid \text{-1} \mid 0 \mid 1 \mid 2 \mid 3 \mid \cdots \\
e \in \text{exp} & ::= & x \mid n \mid e_1 + e_2 \mid e_1 * e_2 \mid e_1 - e_2 \mid \textbf{if } e_1 \textbf{ then } e_2 \textbf{ else } e_3
\end{array}
$$

where '$+$', '$*$', and '$-$', are to be interpreted in the usual way, as 'addition', 'multiplication', and 'subtraction'. The evaluation of the conditional starts with the evaluation of e_1; if e_1 evaluates to any value $\neq 0$, then e_2 is evaluated, otherwise e_3 is evaluated.

1. Define the concrete semantics of $\mathscr{L}_1$. Because expressions in $\mathscr{L}_1$ may contain free variables, the concrete semantics of $\mathscr{L}_1$ must be a function $[\![\ ]\!]_{\text{std}}$ of two arguments, an expression $e \in \text{exp}$ together with a *state* $\sigma : \text{var} \to \mathbb{Z}$. Following common practice,[1] we write the first argument e inside the double square brackets and its second argument σ outside, as in $[\![e]\!]_{\text{std}}\sigma$.
2. Extend the abstract interpretation (Rule-of-Sign) of $\mathscr{L}_0$ on slides 6 , 7 , and 8 , to the language $\mathscr{L}_1$. The abstract interpretation must be a function which returns a value denoted $[\![e]\!]_{\text{ros}}\sigma'$ for an appropriately defined state σ'.

[1] A *state* σ here is the same as a *state* σ (abstract) or ρ (concrete) in Static Program Analysis by Moller and Schwartzbach. Elsewhere in the literature, a state is also called a *store* or an *environment*.

ABSTRACT INTERPRETATION: small example (continued)

Exercise. This continues the exercise on slide 9. Consider a programming language $\mathscr{L}_2$ which extends $\mathscr{L}_1$ (itself extending the earlier $\mathscr{L}_0$):

$$
\begin{array}{lll}
x \in \text{var} & ::= & \cdots \qquad \text{(same as before)} \\
n \in \text{num} & ::= & \cdots \qquad \text{(same as before)} \\
e \in \text{exp} & ::= & x \mid n \mid e_1 + e_2 \mid e_1 * e_2 \mid e_1 - e_2 \mid e_1 / e_2 \mid \textbf{if } e_1 \textbf{ then } e_2 \textbf{ else } e_3
\end{array}
$$

where the additional operation symbol '$/$' is to be interpreted in the usual way as 'integer division'. Again here, the evaluation of the conditional starts with the evaluation of e_1; if e_1 evaluates to any value $\neq 0$, then e_2 is evaluated, otherwise e_3 is evaluated.

1. Repeat Part 1 of the exercise on slide 9, now extended to $\mathscr{L}_2$.
2. Repeat Part 2 of the exercise on slide 9, extended to $\mathscr{L}_2$.

Hint: Since division by 0 is undefined, think of $[\![e]\!]_{\text{std}}$ as defining a *partial* (rather than *total*) function from the set of states $(\text{var} \to \mathbb{Z})$ to $\mathbb{Z}$, and consider adding a new abstract value $\perp$ (for 'undefined') to the abstract domain, *i.e.*, let $\textbf{SIGN}_\perp \triangleq \textbf{SIGN} \cup \{\perp\}$. You also need to define an abstract operation, call it '`ite`', corresponding to **if-then-else**.

The abstract operation '$\oslash$' corresponding to '$/$' is defined by a 5×5 table (not 4×4), because $\textbf{SIGN}_\perp$ contains 5 values. Similarly the tables for '$\oplus$', '$\otimes$', and '$\ominus$' are 5×5.

ABSTRACT INTERPRETATION: small example (continued)

Exercise. This continues the exercise on slide 10, specifically Part 2, which extends the abstract interpretation of $\mathscr{L}_0$ and $\mathscr{L}_1$ to an abstract interpretation for $\mathscr{L}_2$. In this exercise, we re-consider the language $\mathscr{L}_2$, but we want to define a finer abstract interpretation for it, by adding new abstract values to $\textbf{SIGN}_\perp$ to obtain a larger abstract domain $\textbf{SIGN}_\perp^\#$:

$$\textbf{SIGN}_\perp^\# \triangleq \textbf{SIGN}_\perp \cup \{\text{one}, \text{pos}^\#, \text{neg}^\#\} \quad \text{where}$$

'one' is the abstract value of the set $\{1\}$,

'$\text{pos}^\#$' is the abstract value of the set $\{n \in \mathbb{Z} | n \geqslant 0\}$,

'$\text{neg}^\#$' is the abstract value of the set $\{n \in \mathbb{Z} | n \leqslant 0\}$.

Your task is to define the abstract operations $\oplus$, $\otimes$, $\ominus$, and $\oslash$, each being of the form $\textbf{SIGN}_\perp^\# \times \textbf{SIGN}_\perp^\# \to \textbf{SIGN}_\perp^\#$ and each requiring a table of dimension 8×8.

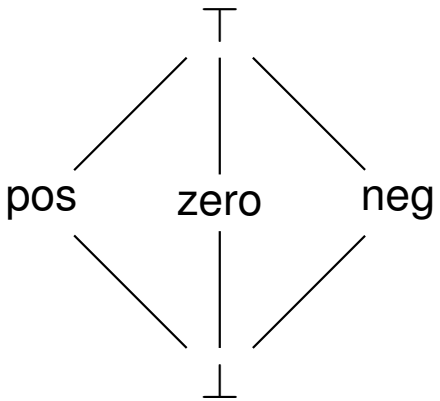
Hint: By introducing the new abstract value '$\text{pos}^\#$' we are able to write the equality:

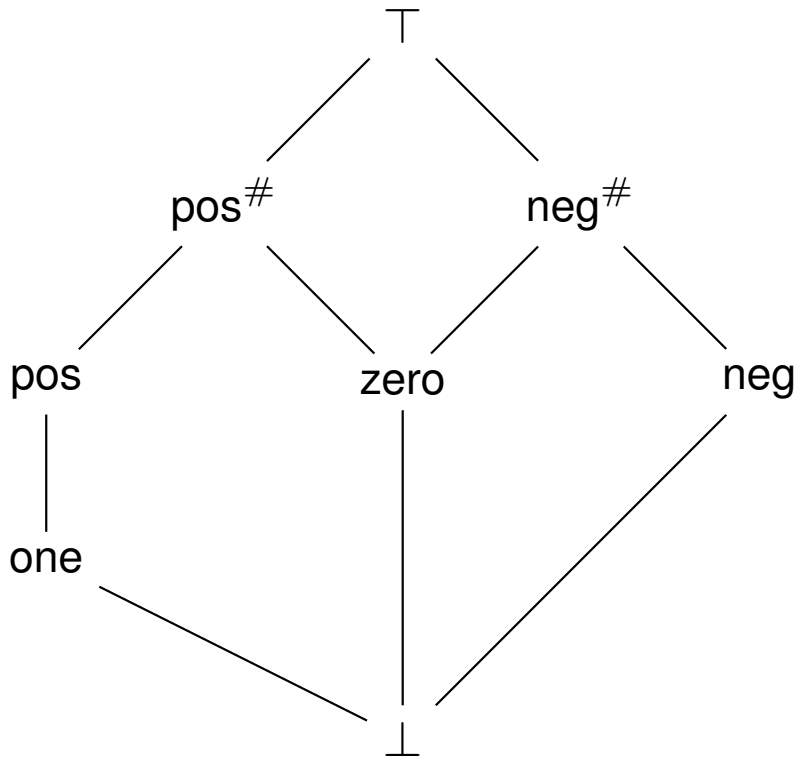
$$\text{pos} \oslash \text{pos} = \text{pos}^\# \qquad \text{instead of} \qquad \text{pos} \oslash \text{pos} = \top \ .$$

The latter equality is less informative than the former. In the absence of '$\text{pos}^\#$' only the latter holds, and not $\text{pos} \oslash \text{pos} = \text{pos}$, because if $0 < m < n$ then $m/n = 0$. The other abstract values 'one' and '$\text{neg}^\#$' are needed when we apply the other operators $\oplus$, $\otimes$, and $\ominus$, to '$\text{pos}^\#$'.

ABSTRACT INTERPRETATION: lattices of abstract domains

The abstract domain for the exercise on slide 10 is pictured on the left,
and the abstract domain for the exercise on slide 11 is pictured on the right:





Under the abstraction functions $\alpha_1 : \mathcal{P}(\mathbb{Z}) \to \textbf{SIGN}_\perp$ and $\alpha_2 : \mathcal{P}(\mathbb{Z}) \to \textbf{SIGN}^\#_\perp$
(defined on the next slide 13), every edge in the lattices of $\textbf{SIGN}_\perp$ on the left
and $\textbf{SIGN}^\#_\perp$ on the right, respectively, depicts the subset relation '$\subseteq$'.

abstraction functions and concretization functions

abstraction $\alpha_1 : \mathcal{P}(\mathbb{Z}) \to \textbf{SIGN}_\bot$ and concretization $\gamma_1 : \textbf{SIGN}_\bot \to \mathcal{P}(\mathbb{Z})$:

$$\alpha_1(S) \triangleq \begin{cases} \bot & \text{if } S = \varnothing, \\ \text{zero} & \text{if } S = \{0\}, \\ \text{pos} & \text{if } S \subseteq \{n \in \mathbb{Z} | n > 0\}, \\ \text{neg} & \text{if } S \subseteq \{n \in \mathbb{Z} | n < 0\}, \\ \top & \text{otherwise.} \end{cases} \qquad \gamma_1(a) \triangleq \begin{cases} \varnothing & \text{if } a = \bot, \\ \{0\} & \text{if } a = \text{zero}, \\ \{n \in \mathbb{Z} | n > 0\} & \text{if } a = \text{pos}, \\ \{n \in \mathbb{Z} | n < 0\} & \text{if } a = \text{neg}, \\ \mathbb{Z} & \text{if } a = \top. \end{cases}$$

abstraction $\alpha_2 : \mathcal{P}(\mathbb{Z}) \to \textbf{SIGN}^\#_\bot$ and concretization $\gamma_2 : \textbf{SIGN}^\#_\bot \to \mathcal{P}(\mathbb{Z})$:

$$\alpha_2(S) \triangleq \begin{cases} \bot & \text{if } S = \varnothing, \\ \text{zero} & \text{if } S = \{0\}, \\ \text{one} & \text{if } S = \{1\}, \\ \text{pos} & \text{if } S \subseteq \{n \in \mathbb{Z} | n > 0\}, \\ \text{neg} & \text{if } S \subseteq \{n \in \mathbb{Z} | n < 0\}, \\ \text{pos}^\# & \text{if } S \subseteq \{n \in \mathbb{Z} | n \geqslant 0\}, \\ \text{neg}^\# & \text{if } S \subseteq \{n \in \mathbb{Z} | n \leqslant 0\}, \\ \top & \text{otherwise.} \end{cases} \quad \gamma_2(a) \triangleq \begin{cases} \varnothing & \text{if } a = \bot, \\ \{0\} & \text{if } a = \text{zero}, \\ \{1\} & \text{if } a = \text{one}, \\ \{n \in \mathbb{Z} | n > 0\} & \text{if } a = \text{pos}, \\ \{n \in \mathbb{Z} | n < 0\} & \text{if } a = \text{neg}, \\ \{n \in \mathbb{Z} | n \geqslant 0\} & \text{if } a = \text{pos}^\#, \\ \{n \in \mathbb{Z} | n \leqslant 0\} & \text{if } a = \text{neg}^\#, \\ \mathbb{Z} & \text{if } a = \top. \end{cases}$$

(Read cases of α_2 top-down because of the overlaps in cases 4+6 and 5+7.)

Galois connections

If $(C, \subseteq)$ and $(A, \sqsubseteq)$ are *partially ordered sets*, in particular *lattices*, and $\alpha : C \to A$ and $\gamma : A \to C$ are a pair of functions such that:

1. $\boldsymbol{\gamma \circ \alpha}$ is ***extensive***, *i.e.*, $c \subseteq \gamma(\alpha(c))$ for all $c \in C$, and
2. $\boldsymbol{\alpha \circ \gamma}$ is ***reductive***, *i.e.*, $\alpha(\gamma(a)) \sqsubseteq a$ for all $a \in A$,

then we call the pair (α, γ) a *Galois connection*.

Exercise. Show that the pairs (α_1, γ_1) and (α_2, γ_2) on slide 13 are each a Galois connection.

relating CONCRETE and ABSTRACT interpretations

The following theorem expresses a *consistency* property of abstract interpretations – also called a *safety* or *soundness* property – here stated for language $\mathscr{L}_1$ or $\mathscr{L}_2$, on slides 9 and 10 respectively, and for any of the pairs (α, γ) on page 14 :

Theorem

For every expression $e \in \text{exp}$ *and every state* $\sigma : \text{var} \to \mathbb{Z}$,

$$\left\{ [\![e]\!]_{\text{std}}\, \sigma \right\} \subseteq \gamma\Big([\![e]\!]_{\text{ros}}\, \sigma' \Big) \quad \textit{where} \quad \sigma' \triangleq \left\{ x \mapsto \alpha(\{\sigma(x)\}) \;\middle|\; x \in \text{var} \right\}.$$

Exercise. Prove the preceding theorem.

Hint: For simplicity, first prove the theorem for $\mathscr{L}_1$, and then consider $\mathscr{L}_2$, and both relative to the first pair (α_1, γ_1) at the top of page 13 .

relating CONCRETE and ABSTRACT interpretations

Exercise. A *Galois insertion* is a stronger relationship than a *Galois connection*.
The Galois connection (α, γ) forms a *Galois insertion* iff

- $\alpha(\gamma(a)) = a$ for every a in the abstract domain A, *i.e.*, $\alpha \circ \gamma$ is the identity on A.

Show that each of the pairs (α_1, γ_1) and (α_2, γ_2) on page 13 is a Galois insertion.

Exercise. Give an example of a Galois connection which is not a Galois insertion.

ABSTRACT INTERPRETATION: how to handle iteration

This builds on the material introduced in the exercises on slide 9 and 10. Consider a programming language $\mathscr{L}_3$ which extends $\mathscr{L}_2$ on slide 10:

$$
\begin{array}{lll}
x \in \text{var} & ::= & \cdots \quad \text{(same as before)} \\
n \in \text{num} & ::= & \cdots \quad \text{(same as before)} \\
e \in \text{exp} & ::= & \cdots \quad \text{(same as before)} \\
C \in \text{comm} & ::= & x := e \mid C_1; C_2 \mid \textbf{if } e \textbf{ then } C_1 \textbf{ else } C_2 \mid \textbf{while } e \textbf{ do } C \textbf{ od}
\end{array}
$$

What is new here is an additional category in the syntax, namely 'comm' (for 'command'). The evaluation of the program phrase '**while** e **do** C **od**' starts with the evaluation of e; if the test e evaluates to any value $\neq 0$, then C is evaluated and the loop is executed repeatedly, until e evaluates to 0.

ABSTRACT INTERPRETATION: how to handle iteration

- How should we understand the semantics $[\![C]\!]_{\text{std}}$ of a command C?
- Abstract value '$\perp$' already represents 'failure' resulting from division by 0. Should '$\perp$' also represent the outcome of a diverging computation? Or should these two kinds of failure be separated?
- We can handle the situation in one of two approaches:
  1. We can choose to *not* distinguish between the two kinds of failure, and let '$\perp$' in the abstract domain denote the outcome of either.
  2. We can choose to deal with the two differently, by introducing a new value **wrong** in the concrete domain denoting the outcome of failure from a division by 0, and by taking $[\![C]\!]_{\text{std}}$ to be a partial function from states to states.

  For simplicity, we choose the first approach – which we use in the next exercise.

Exercise. This extends the exercise on slide 10 to the language $\mathscr{L}_3$ on slide 17:

1. Set up the inductive syntax-driven definition of the concrete semantics of $\mathscr{L}_3$.
2. Set up the inductive syntax-driven definition of the abstract semantics of $\mathscr{L}_3$.

Hint: We ask you to only set up the defining equations. One of the defining equations is recursive, resulting from the presence of the **while** construct, which can be solved using an appropriate fixed-point theorem. We do not ask you to solve the recursive equation.

(THIS PAGE INTENTIONALLY LEFT BLANK)