SOS-style abstract semantics for statements

$[ass_{DSOS}]$ $\quad \langle x := a, ps \rangle \Rightarrow ps[x \mapsto DA[\![a]\!](ps)]$

$[skip_{DSOS}]$ $\quad \langle skip, ps \rangle \Rightarrow ps$

$$[comp^1_{DSOS}] \quad \frac{\langle S_1, ps \rangle \Rightarrow \langle S_1', ps' \rangle}{\langle S_1; S_2, ps \rangle \Rightarrow \langle S_1'; S_2, ps' \rangle}$$

$$[comp^2_{DSOS}] \quad \frac{\langle S_1, ps \rangle \Rightarrow ps'}{\langle S_1; S_2, ps \rangle \Rightarrow \langle S_2, ps' \rangle}$$

$[if^{TT}_{DSOS}]$ $\quad \langle \underline{if}\ b\ \underline{then}\ S_1\ \underline{else}\ S_2, ps \rangle \Rightarrow \langle S_1, ps \rangle$ $\quad$ if $DB[\![b]\!](ps) = TT$

$[if^{FF}_{DSOS}]$ $\quad \langle \underline{if}\ b\ \underline{then}\ S_1\ \underline{else}\ S_2, ps \rangle \Rightarrow \langle S_2, ps \rangle$ $\quad$ if $DB[\![b]\!](ps) = FF$

$[if^{ANY,1}_{DSOS}]$ $\quad \langle \underline{if}\ b\ \underline{then}\ S_1\ \underline{else}\ S_2, ps \rangle \Rightarrow \langle S_1, ps \rangle$ $\quad$ if $DB[\![b]\!](ps) = ANY$

$[if^{ANY,2}_{DSOS}]$ $\quad \langle \underline{if}\ b\ \underline{then}\ S_1\ \underline{else}\ S_2, ps \rangle \Rightarrow \langle S_2, ps \rangle$ $\quad$ if $DB[\![b]\!](ps) = ANY$

$[while_{DSOS}]$ $\quad \langle \underline{while}\ b\ \underline{do}\ S, ps \rangle \Rightarrow \langle \underline{if}\ b\ \underline{then}\ (S; \underline{while}\ b\ \underline{do}\ S)\ \underline{else}\ skip, ps \rangle$

Example : Execute **while** $\neg(x=0)$ **do** $x := x-2$ from ps such that $ps(x) = POS$. Draw the config. graph.

$\Downarrow$

$\langle$ **while** $\neg(x=0)$ **do** $x := x-2$, $ps\rangle$

$\Downarrow$

$\langle$ **if** $\neg(x=0)$ **then** $(x := x-2$; **while** $\neg(x=0)$ **do** $x := x-2)$ **else** skip , $ps\rangle$

$\Downarrow$

$\langle x := x-2$; **while** $\neg(x=0)$ **do** $x := x-2$, $ps\rangle$

$\Downarrow$

$\langle$ **while** $\neg(x=0)$ **do** $x := x-2$, $ps[x \mapsto ANY]\rangle$

$\Downarrow$

$\langle$ **if** $\neg(x=0)$ **then** $(x := x-2$; **while** $\neg(x=0)$ **do** $x := x-2)$ **else** skip, $ps[x \mapsto ANY]\rangle$

$\Downarrow$ $\qquad$ $\Downarrow$

$\langle x := x-2$; **while** $\neg(x=0)$ **do** $x := x-2$, $ps[x \mapsto ANY]\rangle$ $\qquad$ $\langle$ skip, $ps[x \mapsto ANY]\rangle$

$\Downarrow$

$ps[x \mapsto ANY]$

The derivations are omitted. What can we deduce from this abstract interpretation of the program about its real execution(s)? That executing the program from a state such that $s(x) > 0$ may terminate in a state where we cannot say anything about the value of x (but no other values will change!), but may also diverge. Not very useful... considering that the graph would have been the same if the loop body was $x := x-1$, excluding the possibility for divergence.

Since the abstract domains are finite in this Detection-of-signs analysis, the configuration graphs are guaranteed to be finite: configuration-space exploration is algorithmically possible!

How shall we formulate safety of DSOS?

(a) $\langle S, s\rangle \Rightarrow_{SOS} \langle S', s'\rangle$ implies $\exists ps'.\ (\langle S, abs(s)\rangle \Rightarrow_{DSOS} \langle S', ps'\rangle$ and $abs(s') \sqsubseteq_{ps} ps')$

$$\begin{array}{ccc} \langle S, ps\rangle & \Longrightarrow_{DSOS} & \langle S', ps'\rangle \\ & & \sqcup \\ & & ps'' \\ abs \uparrow & & \uparrow abs \\ \langle S, s\rangle & \Longrightarrow_{SOS} & \langle S', s'\rangle \end{array}$$

Every concrete execution is covered by some abstract one!
"Simulation"

(b) $\langle S, s\rangle \Rightarrow s'$ ... (similar)

- Acceleration

There are variations on the theme. To define an "analysis" means to fix a particular exploration strategy. One approach is to explore in a breadth-first manner and merge unexplored configurations having the same statement:

$\langle S, ps_1\rangle$ and $\langle S, ps_2\rangle$ are merged into $\langle S, ps_1 \sqcup_{ps} ps_2\rangle$

This technique is known as <u>acceleration</u> and can be crucial in order to obtain termination of the exploration when the abstract domain is infinite – see Assignment 5.

- Prime values

Another variation of the analysis, which makes it more precise, is to split the exploration on the prime values. A configuration $\langle S, ps\rangle$ where $ps(x) = \text{NON-POS}$ and $ps(y) = \text{NON-NEG}$ splits to four:

$\langle S, ps[x \mapsto \text{NEG}, y \mapsto \text{ZERO}]\rangle$, $\langle S, ps[x \mapsto \text{NEG}, y \mapsto \text{POS}]\rangle$,

$\langle S, ps[x \mapsto \text{ZERO}, y \mapsto \text{ZERO}]\rangle$, $\langle S, ps[x \mapsto \text{ZERO}, y \mapsto \text{POS}]\rangle$.

We shall see examples later. It requires a slight modification of the rules of the DSOS. Which one?

$$\langle x := a, ps \rangle \Rightarrow ps[x \mapsto v] \quad \text{if } v\text{-prime and } v \sqsubseteq_s DA[\![a]\!](ps)$$

which is a non-deterministic rule. We may need also to split the initial configuration into a set.

PROGRAM TRANSFORMATION

The book presents program transformation by means of rules of the shape:

$$ps_c \vdash C[S] \rightsquigarrow C[S'] \quad \langle \text{condition on } ps_c \rangle$$

C is called a "context" (see paper in Course Literature).
In our case ps_c is the "aggregated" property state at S, resulting from an abstract interpretation.

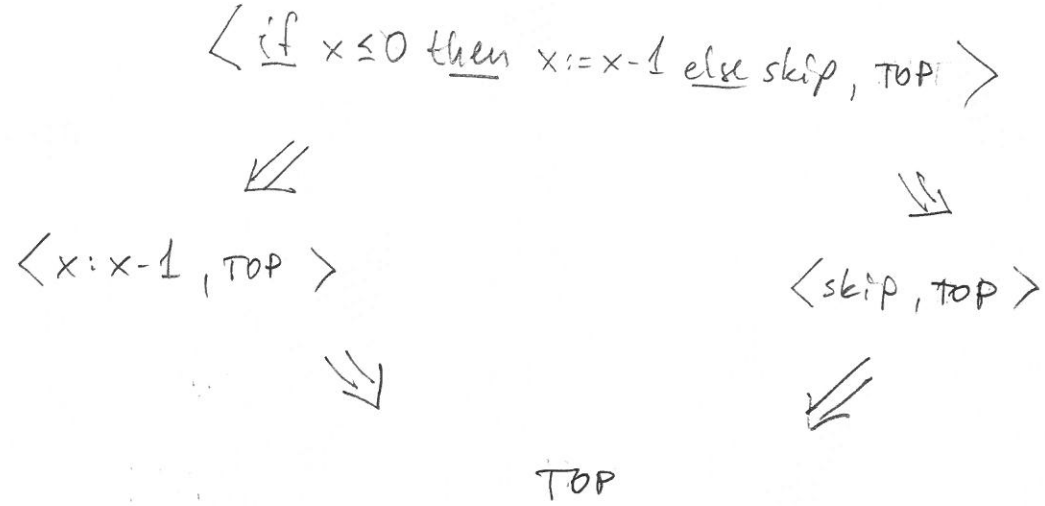
Example: $ps_c \vdash C[\underline{\text{if}}\ b\ \underline{\text{then}}\ S_1\ \underline{\text{else}}\ S_2] \rightsquigarrow C[S_1]$ if $DB[\![b]\!](ps_c) = TT$

Let's see an application of this transformation to the program:

$P \stackrel{\text{def}}{=} (\underline{\text{if}}\ x \leq 0\ \underline{\text{then}}\ x := x-1\ \underline{\text{else}}\ \text{skip}); (\underline{\text{if}}\ \neg(x=0)\ \underline{\text{then}}\ x := x+1\ \underline{\text{else}}\ \text{skip}); x := x+1$

seen as $C[\underline{\text{if}}\ \neg(x=0)\ \underline{\text{then}}\ x := x+1\ \underline{\text{else}}\ \text{skip}]$.

We need to explore $\langle P, \text{TOP} \rangle$ where $\text{TOP}(x) \stackrel{\text{def}}{=} \text{ANY}$ for all $x \in \text{Var}$.



Which doesn't allow us to apply the transformation.

But we can have a <u>finer analysis</u>, based on a finer DSOS:

All reasoning can be reduced to the "prime" values.

a) $\langle$ if $x \le 0$ then $x := x-1$ else skip , $ps_1 \rangle$ where $ps_1(x) = NEG$

$\Downarrow$

$\langle x := x-1 , ps \rangle$

$\Downarrow$

ps_1

b) $\langle$ if $x \le 0$ then $x := x-1$ else skip , $ps_2 \rangle$ where $ps_2(x) = ZERO$

$\Downarrow$

$\langle x := x-1 , ps_2 \rangle$

$\Downarrow$

ps_1

c) $\langle$ if $x \le 0$ then $x := x-1$ else skip , $ps_3 \rangle$ where $ps_3(x) = POS$

$\Downarrow$

$\langle skip , ps_3 \rangle$

$\Downarrow$

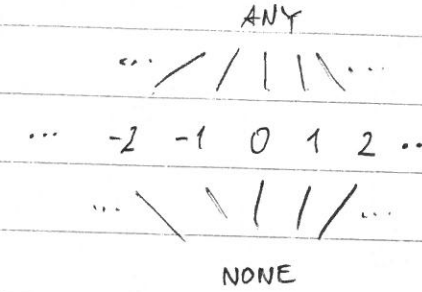
ps_2

The resulting property state is $ps_c = ps_1 \sqcup ps_3 = ps[x \mapsto NON\text{-}ZERO]$

Now, $DB[\![\neg(x=0)]\!](ps_c) = TT$ and we can apply the transformation!

Assignment 5: Develop a Constant Propagation Analysis in a SOS style and apply it to program transformation.

$(Const, \sqsubseteq_c)$:



It is an infinite domain, so we have to be smart how we set up the analysis to guarantee its termination!

PROGRAM ANALYSES WITH ABSTRACT INTERPRETATION

- Type Safety: In a typed programming language (with type casting) you may want to check statically that no type violations can occur at run-time. There are type systems for this, but you can also employ AI: simply set up an abstract domain with the types as values!

- Security: Assume Var is partitioned into VPri and VPub, the "private" variables whose value should not "leak" to the public ones, which the environment can observe. We can set up an AI with just two values, HIGH and LOW. In this way we can compute "direct flow" from HIGH to LOW ("taint analysis"). Needs to be refined, though, as the following program leaks:

  if $h \leq 0$ then $l := 0$ else $l := 1$.

- Property Simulation: has been used for C-driver verification.
- Symbolic Execution: we'll come back to it.