Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
Fabian Huch

WS 2022/23
24.01.

Semantics of Programming Languages

Exercise Sheet 12

Exercise 12.1 Complete Lattices

Which of the following ordered sets are complete lattices?

- $\mathbb{N}$, the set of natural numbers $\{0, 1, 2, 3, \ldots\}$ with the usual order
- $\mathbb{N} \cup \{\infty\}$, the set of natural numbers plus infinity, with the usual order and $n < \infty$ for all $n \in \mathbb{N}$.
- A finite set A with a total order $\leq$ on it.

Exercise 12.2 Sign Analysis

Instantiate the abstract interpretation framework to a sign analysis over the lattice $pos, zero, neg, any$, where pos abstracts positive values, $zero$ abstracts zero, neg abstracts negative values, and any abstracts any value.

```
datatype sign = Pos | Zero | Neg | Any

instantiation sign :: order
instantiation sign :: semilattice_sup_top
fun γ_sign :: "sign ⇒ val set"
fun num_sign :: "val ⇒ sign"
fun plus_sign :: "sign ⇒ sign ⇒ sign"
global_interpretation Val_semilattice
  where γ = γ_sign and num' = num_sign and plus' = plus_sign
global_interpretation Abs_Int
  where γ = γ_sign and num' = num_sign and plus' = plus_sign
  defines aval_sign = aval' and step_sign = step' and AI_sign = AI
```

Some tests:

```
definition "test1_sign =
  ''x'' ::= N 1;;
  WHILE Less (V ''x'') (N 100) DO ''x'' ::= Plus (V ''x'') (N 2)"
value "show_acom (the(AI_sign test1_sign))"

definition "test2_sign =
  ''x'' ::= N 1;;
```

WHILE Less (V ''x'') (N 100) DO ''x'' ::= Plus (V ''x'') (N 3)”

definition *“steps c i = ((step_sign ⊤) ^^ i) (bot c)”*

value *“show_acom (steps test2_sign 0)”*

...

value *“show_acom (steps test2_sign 6)”*
value *“show_acom (the(AI_sign test2_sign))”*

Exercise 12.3 AI for Conditionals

Our current constant analysis does not regard conditionals. For example, it cannot figure out, that after executing the program *x:=2; IF x<2 THEN x:=2 ELSE x:=1*, *x* will be constant.

In this exercise, we extend our abstract interpreter with a simple analysis of boolean expressions. To this end, modify locale *Val_semilattice* in theory *Abs_Int0.thy* as follows:

- Introduce an abstract domain *'bv* for boolean values, add, analogously to *num'* and *plus'* also functions for the boolean operations and for *less*.
- Modify *Abs_Int0* to accommodate for your changes.

Homework 12.1 AI Table

Submission until Monday, Jan 30, 23:59pm. Consider the following IMP program (with extended arithmetic operations):

```
r := 1;
WHILE b DO (
  r := r * 2;
  IF b THEN
    r := r - 1
  ELSE
    r := r + 2
)
```

Run the abstract interpretation on this program, i.e., iterate the step function for parity analysis until a fixed point is reached. Again, use the format from last homework.

definition A_0 :: *“entry list”*
definition A_1 :: *“entry list”*
definition A_2 :: *“entry list”*
definition A_3 :: *“entry list”*
definition A_4 :: *“entry list”*

definition A_5 :: *"entry list"*
definition A_6 :: *"entry list"*
definition A_7 :: *"entry list"*
definition A_8 :: *"entry list"*
definition A_9 :: *"entry list"*
hide_const *None Some*

Homework 12.2 AI for the Extended Reals

Submission until Monday, Jan 30, 23:59pm. For this exercise, we will consider a modified variant of IMP that computes on real numbers extended with $-\infty$ and ∞. The corresponding type is *ereal*. We will consider "$-\infty + \infty$" and "$\infty + (-\infty)$" erroneous computations. We propagate errors by using the *option* type, i.e. we set *val* = *ereal option*. The theories up to *Collecting* for this variant are already provided. Your task is now to design an abstract interpreter on the domain consisting of subsets of $\{\infty^-, \infty^+, \mathit{NaN}, \mathit{Real}\}$ where *NaN* signals a computation error and all other values have their obvious meaning. First adopt *Abs_Int0* and *Abs_Int1* to accommodate for the changed semantics, and then instantiate the abstract interpreter with your analysis. For this step you best modify the parity analysis *Abs_Int1_parity*.

Hints: To benefit from proof automation it can be helpful to slightly change the format of the rules for addition in *Val_semilattice*. For instance, you could reformulate *gamma_plus′* as: $i1 \in \gamma\ a1 \Longrightarrow i2 \in \gamma\ a2 \Longrightarrow i = i1 + i2 \Longrightarrow i \in \gamma(\mathit{plus}'\ a1\ a2)$. (You will need to change the interface *Val_semilattice*).

You can start the formalization of the AI like this:

datatype *bound* = *NegInf* ("∞^-") | *PosInf* ("∞^+") | *NaN* | *Real*
datatype *bounds* = *S "bound set"*

instantiation *bounds* :: *order*
begin

definition *less_eq_bounds* **where**
"$x \leq y = (\mathit{case}\ (x, y)\ \mathit{of}\ (S\ x, S\ y) \Rightarrow x \subseteq y)$"

definition *less_bounds* **where**
"$x < y = (\mathit{case}\ (x, y)\ \mathit{of}\ (S\ x, S\ y) \Rightarrow x \subset y)$"

instance
end

For the AI, interpret *Abs_Int*, *Abs_Int_mono*, and *Abs_Int_measure*:

instantiation *bounds* :: *semilattice_sup_top*
begin

definition *sup_bounds*

definition *top_bounds*
instance
end

fun γ*_bounds* :: *"bounds* $\Rightarrow$ *val set"*
definition *num_bounds* :: *"ereal* $\Rightarrow$ *bounds"*
fun *plus_bounds* :: *"bounds* $\Rightarrow$ *bounds* $\Rightarrow$ *bounds"*
global_interpretation *Val_semilattice*
where $\gamma = \gamma$*_bounds* **and** *num'* = *num_bounds* **and** *plus'* = *plus_bounds*
global_interpretation *Abs_Int*
where $\gamma = \gamma$*_bounds* **and** *num'* = *num_bounds* **and** *plus'* = *plus_bounds*
defines *aval_bounds* = *aval'* **and** *step_bounds* = *step'* **and** *AI_bounds* = *AI*

global_interpretation *Abs_Int_mono*
where $\gamma = \gamma$*_bounds* **and** *num'* = *num_bounds* **and** *plus'* = *plus_bounds*
fun *m_bounds* :: *"bounds* $\Rightarrow$ *nat"*
abbreviation *h_bounds* :: *nat*

global_interpretation *Abs_Int_measure*
where $\gamma = \gamma$*_bounds* **and** *num'* = *num_bounds* **and** *plus'* = *plus_bounds*
and *m* = *m_bounds* **and** *h* = *h_bounds*