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

WS 2020/21
25.01.2021

Semantics of Programming Languages

Exercise Sheet 11

Exercise 11.1 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 11.2 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 11.3 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 11.1 Lattice Theory

Submission until Sunday, Jan 31, 23:59.

General Submission Instructions

Note that due to the use of instantiations, submissions for this homework will fail on the submission system (with an "illegal keyword" error).

Please make sure that your submission runs locally in a reasonable amount of time, and ignore the error message.

A type $'a$ is a $\sqcup$-semilattice if it is a partial order and there is a supremum operation $\sqcup$ of type $'a \Rightarrow 'a \Rightarrow 'a$ that returns the least upper bound of its arguments:

- Upper bound: $x \le x \sqcup y$ and $y \le x \sqcup y$
- Least: $x \le z \wedge y \le z \longrightarrow x \sqcup y \le z$

Is every finite $\sqcup$-semilattice with a bottom element $\bot$ also a complete lattice? Proof or counterexample!

You might be asked to do a proof like this in the exam, on pen and paper. Do a pen and paper version first, then formalize it in Isabelle. If you get stuck, write down the rest of your informal version as comment.

Hints:

- to apply the $\sqcup$ operation to a set, you can use the *set_sup* relation
- you may use (and then need to prove) the *sup_pres_p* lemma
- for finite sets, there is also the *finite_induct* induction scheme

context *order*
begin
abbreviation "*lower S l* $\equiv \forall s \in S.\ l \le s$"
abbreviation "*greatest S l* $\equiv \forall l'.$ (*lower S l'* $\longrightarrow l' \le l$)"
end

Complete lattice, as stated in the lecture:

class *complete_lattice* = *order* +
  assumes "$\bigwedge S$::'*a set*. $\exists l.$ (*lower S l* $\wedge$ *greatest S l*)"

Finite semilattice with $\sqcup$ and $\bot$:

class *finite_semilattice_sup_bot* = *semilattice_sup* + *order_bot* + *finite*
begin

$\sqcup$ on sets (as predicate), with initial element b.

inductive *set_sup* :: "'*a* $\Rightarrow$ '*a set* $\Rightarrow$ '*a* $\Rightarrow$ *bool*" ("$\bigsqcup$ _/ _/ := _/" [*59*, *59*, *59*]) **for** *b* **where**
  empty[*intro*]: "$\bigsqcup$ *b* {} := *b*"
| *insert*[*intro*]: "$\bigsqcup$ *b A* := *y* $\Longrightarrow \bigsqcup$ *b* (*insert x A*) := ($x \sqcup y$)"

theorem *sup_pres_p*:
  assumes *sup*: "$\bigsqcup$ *b A* := *y*"
  assumes *pres*: "$\bigwedge$ *x y*. *P x* $\Longrightarrow$ *P y* $\Longrightarrow$ *P* ($x \sqcup y$)"
  shows "$\forall x \in A$. *P x* $\Longrightarrow$ *P b* $\Longrightarrow$ *P y*"

Case proof:

theorem *complete_lattice_prf*: "*class.complete_lattice* ($\le$) ($<$)"
proof
end

Case counterexample:

Put in your type here

datatype *cex_a* = *TODO*

instantiation *cex_a* :: *finite_semilattice_sup_bot*
begin

definition *less_eq_cex_a* :: *"cex_a ⇒ cex_a ⇒ bool"* **where**
"less_eq_cex_a _ _ = True"

definition *less_cex_a* :: *"cex_a ⇒ cex_a ⇒ bool"* **where**
"less_cex_a _ _ = False"

definition *bot_cex_a* :: *"cex_a"* **where**
"bot_cex_a = TODO"

definition *sup_cex_a* :: *"cex_a ⇒ cex_a ⇒ cex_a"* **where**
"sup_cex_a _ _ = TODO"

instance sorry

lemma *complete_lattice_cex*: *"¬class.complete_lattice (≤) (<)"*
proof −
have *"∃ S. ∄ l. (lower S l ∧ greatest S l)"*

end

Finally, add the name of the lemma you proved below:

lemmas *prf_or_cex* =

Homework 11.2 AI for the Extended Reals

Submission until Sunday, Jan 31, 23:59. 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*. Your task is now to design an abstract interpreter on the domain consisting of subsets of $\{\infty^-, \infty^+, NaN, Real\}$ where NaN signals a computation error and all other values have their obvious meaning. The definitions (up to abstract interpretation) have been already adapted in the template Defs.

First adopt the abstract interpretation to accommodate for the changed semantics, and then instantiate the abstract interpreter with your analysis.

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: $i1 \in \gamma\ a1 \Longrightarrow i2 \in \gamma\ a2 \Longrightarrow i = i1 + i2 \Longrightarrow i \in \gamma(plus'\ a1\ a2)$. (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 ≤ y = (case (x, y) of (S x, S y) ⇒ x ⊆ y)"

definition *less_bounds* **where**
"x < y = (case (x, y) of (S x, S y) ⇒ x ⊂ 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 ⇒ val set"*
definition *num_bounds* :: *"ereal ⇒ bounds"*
fun *plus_bounds* :: *"bounds ⇒ bounds ⇒ bounds"*

global_interpretation *Val_semilattice*
where *γ* = *γ_bounds* **and** *num'* = *num_bounds* **and** *plus'* = *plus_bounds*
global_interpretation *Abs_Int*
where *γ* = *γ_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 *γ* = *γ_bounds* **and** *num'* = *num_bounds* **and** *plus'* = *plus_bounds*
fun *m_bounds* :: *"bounds ⇒ nat"*
abbreviation *h_bounds* :: *nat*

global_interpretation *Abs_Int_measure*
where *γ* = *γ_bounds* **and** *num'* = *num_bounds* **and** *plus'* = *plus_bounds*
and *m* = *m_bounds* **and** *h* = *h_bounds*