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 $\mathit{pos, zero, neg, any}$, where $\mathit{pos}$ abstracts positive values, $\mathit{zero}$ abstracts zero, $\mathit{neg}$ abstracts negative values, and any abstracts any value.

\begin{verbatim}
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))"
\end{verbatim}
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 Al for Conditionals

Our current constant analysis does not regard conditionals. For example, it cannot figure out, that after executing the program 
\[ x := 2; \text{IF } x < 2 \text{ THEN } x := 2 \text{ ELSE } x := 1, \]
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: \( x \leq x \sqcup y \) and \( y \leq x \sqcup y \)
- Least: \( x \leq z \land y \leq z \rightarrow x \sqcup y \leq z \)
Is every finite ⊔-semilattice with a bottom element ⊥ 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 ⊔ 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 ≡ ∀s∈S. l ≤ s”
abbreviation “greatest S l ≡ ∀l’. (lower S l’ → l’ ≤ l)”
end

Complete lattice, as stated in the lecture:

class complete_lattice = order +
  assumes “∀S::’a set. ∃l. (lower S l ∧ greatest S l)”

Finite semilattice with ⊔ and ⊥:

class finite_semilattice_sup_bot = semilattice_sup + order_bot + finite
begin

⊔ on sets (as predicate), with initial element b.

inductive set_sup :: “’a ⇒ ’a set ⇒ ’a ⇒ bool” (“⊔/ ⊔/ := ⊔” [59, 59, 59]) for b where
  empty[intro]: “⊔/ b {} := b”
| insert[intro]: “⊔/ b A := y ⇒ ⊔/ b (insert x A) := (x ⊔ y)”

theorem sup_pres_p:
  assumes sup: “⊔/ b A := y”
  assumes pres: “∀x y. P x ⇒ P y ⇒ P (x ⊔ y)”
  shows “∀x ∈ A. P x ⇒ P b ⇒ P y”

Case proof:

theorem complete_lattice_prf: “class.complete_lattice (≤) (<)”
proof
end

Case counterexample:

Put in your type here

datatype cex_a = TODO
instantiation $cex_a :: \text{finite}\textunderscore\text{semilattice}\_\text{sup}\_\text{bot}$
begin

definition $\text{less}\_\text{eq}\_\text{cex}\_a :: \langle cex\_a \Rightarrow cex\_a \Rightarrow \text{bool} \rangle$ where

$\text{less}\_\text{eq}\_\text{cex}\_a \_\_ = \text{True}$

definition $\text{less}\_\text{cex}\_a :: \langle cex\_a \Rightarrow cex\_a \Rightarrow \text{bool} \rangle$ where

$\text{less}\_\text{cex}\_a \_\_ = \text{False}$

definition $\text{bot}\_\text{cex}\_a :: \langle cex\_a \rangle$ where

$\text{bot}\_\text{cex}\_a = \text{TODO}$

definition $\text{sup}\_\text{cex}\_a :: \langle cex\_a \Rightarrow cex\_a \Rightarrow cex\_a \rangle$ where

$\text{sup}\_\text{cex}\_a \_\_ = \text{TODO}$

instance sorry

lemma $\text{complete}\_\text{lattice}\_\text{cex} :: \lnot \text{class}.\text{complete}\_\text{lattice} (\leq) (<)$
proof -

have $\exists S. \nexists l. (\text{lower} S l \land \text{greatest} S l)$

end

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

lemmas $\text{prf}\_\text{lor}\_\text{cex} =$

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 $\infty$. The corresponding type is $\text{ereal}$. We will consider $-\infty + \infty$ and $\infty + (-\infty)$ erroneous computations. We propagate errors by using the $\text{option}$ type, i.e. we set $\text{val} = \text{ereal option}$. Your task is now to design an abstract interpreter on the domain consisting of subsets of $\{\infty^{-}, \infty^{+}, \text{NaN}, \text{Real}\}$ 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 $\text{Val}\_\text{semilattice}$. For instance, you could reformulate $\text{gamma}\_\text{plus}'$ as: $i_1 \in \gamma a_1 \implies i_2 \in \gamma a_2 \implies i = i_1 + i_2 \implies i \in \gamma(\text{plus}' a_1 a_2)$. (You will need to change the interface $\text{Val}\_\text{semilattice}$).

You can start the formalization of the AI like this:

datatype $\text{bound} = \text{NegInf} (\infty^{-}) | \text{PosInf} (\infty^{+}) | \text{NaN} | \text{Real}$


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