1 Pretty syntax for lattice operations

theory *Lattice_Syntax*
imports *Complete_Lattices*
begin

notation
bot (“⊥”) **and**
top (“⊤”) **and**
inf (**infixl** “⊓” *70*) **and**
sup (**infixl** “⊔” *65*) **and**
Inf (“⨅_” [*900*] *900*) **and**
Sup (“⨆_” [*900*] *900*)

syntax (*xsymbols*)
“_*INF1*” :: “*pttrns* ⇒ *'b* ⇒ *'b*” (“(*3*⨅_./ _)” [*0*, *10*] *10*)
“_*INF*” :: “*pttrn* ⇒ *'a set* ⇒ *'b* ⇒ *'b*” (“(*3*⨅_∈_./ _)” [*0*, *0*, *10*] *10*)
“_*SUP1*” :: “*pttrns* ⇒ *'b* ⇒ *'b*” (“(*3*⨆_./ _)” [*0*, *10*] *10*)
“_*SUP*” :: “*pttrn* ⇒ *'a set* ⇒ *'b* ⇒ *'b*” (“(*3*⨆_∈_./ _)” [*0*, *0*, *10*] *10*)

end

Technische Universität München **WS 2013/14**
Institut für Informatik **4. 2. 2014**
Prof. Tobias Nipkow, Ph.D.
Peter Lammich, Johannes Hölzl

Semantics of Programming Languages

Exercise Sheet 14

2 Nondeterministic Choice

To the standard *com* datatype, add a command for nondeterministic choice:

datatype *com* = ... | *Or com com*

Augment the inductive definition of big-step semantics by adding two new rules:

$$\frac{(c_1,\ s) \Rightarrow s'}{(Or\ c_1\ c_2,\ s) \Rightarrow s'}\ \text{OR-LEFT} \qquad \frac{(c_2,\ s) \Rightarrow s'}{(Or\ c_1\ c_2,\ s) \Rightarrow s'}\ \text{OR-RIGHT}$$

Also extend the inductive definition of the Hoare calculus with a new rule:

$$\frac{\vdash \{P\}\ c_1\ \{Q\} \qquad \vdash \{P\}\ c_2\ \{Q\}}{\vdash \{P\}\ Or\ c_1\ c_2\ \{Q\}}\ \text{OR}$$

Show that the Hoare calculus remains sound, i.e., that we still have

$\vdash \{P\}\ c\ \{Q\} \Longrightarrow \models \{P\}\ c\ \{Q\}$

Note: You need not reproduce parts of the proof that remain unchanged w.r.t. the proof for the original while-language without *Or*.

2.1 Solution

How an Isabelle proof might look (we explicitly mention the necessary unfoldings, inductions and case distinctions):

lemma assumes "$\vdash \{P\}\ c\ \{Q\}$" **shows** "$\models \{P\}\ c\ \{Q\}$"
  using *assms*
proof (*induction rule*: *hoare.induct*)
  fix $P\ Q\ c_1\ c_2$ **assume** *IH*: "$\models \{P\}\ c_1\ \{Q\}$" "$\models \{P\}\ c_2\ \{Q\}$"
  show "$\models \{P\}\ Or\ c_1\ c_2\ \{Q\}$"
    unfolding *hoare_valid_def*
  proof (*intro allI impI*)
    fix $s\ t$ **assume** "$(Or\ c_1\ c_2, s) \Rightarrow t$" "$P\ s$"
    then show "$Q\ t$"
    proof (*cases rule*: *big_step.cases*)
      case *Or_Left* **with** ‹$P\ s$› *IH* **show** *?thesis*
        unfolding *hoare_valid_def* **by** *auto*
    next
      case *Or_Right* **with** ‹$P\ s$› *IH* **show** *?thesis*
        unfolding *hoare_valid_def* **by** *auto*
    qed
  qed
oops — We do not show the remaining cases.

A text proof might look like:

Show $\vdash \{P\}\ c\ \{Q\} \Longrightarrow \models \{P\}\ c\ \{Q\}$:
Proof. *Rule induction on* $\vdash \{P\}\ c\ \{Q\}$ (only OR-case, the rest doesn't change):
Case OR: Induction hypothesis: $\models \{P\}\ c_1\ \{Q\}$ and $\models \{P\}\ c_2\ \{Q\}$.
Show $\models \{P\}\ Or\ c_1\ c_2\ \{Q\}$ by $\models$-definition:
  Fix two states s and t, assume $P\ s$ and $(Or\ c_1\ c_2, s) \Rightarrow t$.
  Show $Q\ t$ by *case distinction on* $(Or\ c_1\ c_2, s) \Rightarrow t$:
  Subcase OR-LEFT:
    Assume $(c_1, s) \Rightarrow t$: Show $Q\ t$, by IH $\models \{P\}\ c_1\ \{Q\}$, $P\ s$, and $\models$-def.
  Subcase OR-RIGHT:
    Analog to OR-LEFT, replace c_1 with c_2.

3 Recursive Functions and Structural Induction

Consider this simple datatype of boolean formulae:

datatype *form* = *Var vname* | *Impl form form*

Write a recursive definition of a function *subst* that does *simultaneous substitution*. The application *subst* σ *t* simultaneously replaces *every* variable in *t* with a new sub-formula: Each occurrence of *Var x* is replaced with $\sigma(x)$.

subst :: (*vname* ⇒ *form*) ⇒ *form* ⇒ *form*

If we do simultaneous substitution with ($\lambda x.$ *Var x*), we should get the identity function on formulae:

subst ($\lambda x.$ *Var x*) *t* = *t*

Prove this fact by structural induction on *t*.

primrec *subst* :: "(*vname* ⇒ *form*) ⇒ *form* ⇒ *form*"
where
  "*subst* σ (*Var x*) = σ *x*" |
  "*subst* σ (*Impl* t_1 t_2) = *Impl* (*subst* σ t_1) (*subst* σ t_2)"

lemma *p1*: "*subst* ($\lambda x.$ *Var x*) *t* = *t*"
proof (*induct t*)

case (Var y)

  fix *y* :: *vname*
  show "*subst* ($\lambda x.$ *Var x*) (*Var y*) = *Var y*" **by** *simp*
next

case (*Impl* t_1 t_2)

  fix t_1 t_2 :: *form*
  assume *IH1*: "*subst* ($\lambda x.$ *Var x*) t_1 = t_1"
  assume *IH2*: "*subst* ($\lambda x.$ *Var x*) t_2 = t_2"
  show "*subst* ($\lambda x.$ *Var x*) (*Impl* t_1 t_2) = *Impl* t_1 t_2"
    by (*simp add*: *IH1 IH2*)
qed

4 Simplified Sign-Analysis

Design a simplified sign analysis, that only distinguishes between positive values and any values, i.e., the lattice has the elements $L = \{Pos, Any\}$.

Define the ordering ($\leq$), supremum $\sqcup$, and indicate the $\top$-element. Prove that your definitions yield a join semilattice (class *semilattice_sup_top* from the lecture), i.e., that $\leq$ is a preorder (reflexive, transitive) with greatest element $\top$, and that $\sqcup$ is the least upper bound.

Define the concretization function $\gamma_s :: L \Rightarrow int\ set$, and the abstract operations $num_s :: int \Rightarrow L$ and $plus_s :: L \Rightarrow L \Rightarrow L$. Show that they are sound abstractions, i.e.,

$n \in \gamma_s\ (num_s\ n)$

$n_1 \in \gamma_s\ a_1 \wedge n_2 \in \gamma_s\ a_2 \Longrightarrow n_1{+}n_2 \in \gamma_s\ (plus_s\ a_1\ a_2)$

Iterate the *step′* function for the following program until a fixed point is reached, and tabulate the annotations after each iteration:

```
x := 1 {A1};
y := 2 {A2};
IF z<1 THEN (
  {A3} x := x + y {A4}
) ELSE (
  {A5} x := x + (-1) {A6}
) {A7}
```

	0	1	2	3	4	5	6
A1	*None*	$<P, A, A>$					
A2	*None*		$<P, P, A>$				
A3	*None*			$<P, P, A>$			
A4	*None*				$<P, P, A>$		
A5	*None*			$<P, P, A>$			
A6	*None*				$<A, P, A>$		
A7	*None*					$<A, P, A>$	

Notes:

- The column numbered n shall contain the annotation after applying $step'\ \top$ n times. Hence, column 0 that we filled for you contains the initial annotation.
- Remember that the annotations produced by the *step′*-function have type $(vname \Rightarrow L)\ option$.

  Use some shortcut notations to represent annotations, e.g.,
  $\langle a_x,a_y,a_z \rangle$ for $Some\ \langle\ x := a_x,\ y := a_y,\ z := a_z\ \rangle$,
  where $a_x,a_y,a_z \in L$.
- Use the simplest version of *step′*, i.e., the one without analysis of boolean expressions.

4.1 Solution

4.1.1 Order $\leq$

Definitions: $x \leq y \longleftrightarrow x = Pos \vee x = y$,
$x \sqcup y = (if\ x = Pos \wedge y = Pos\ then\ Pos\ else\ Any)$, $\top = Any$
$\leq$ is preorder with top and lub:

refl $x \leq x$ by def of $\leq$

trans $x \leq y \longrightarrow y \leq z \longrightarrow x \leq z$ by def of $\leq$

greatest $x \leq \top$ by def of $\leq$ and $\top$, case distinction on x

lub (1) $x \leq x \sqcup y$, (2) $y \leq x \sqcup y$, and (3) $x \leq z \longrightarrow y \leq z \longrightarrow x \sqcup y \leq z$: (1) by case dist. on x and defs, (2) by case dist. on x and defs, (3) by case dist. on $x \times y \times z$, and defs

4.1.2 Abstract Interpretation

Definitions: $\gamma_s\ Pos = \{x.\ 0 < x\}$, $\gamma_s\ Any = UNIV$, $num_s\ x = (if\ 0 < x\ then\ Pos\ else\ Any)$,
$plus_s\ x\ y = (if\ x = Pos \wedge y = Pos\ then\ Pos\ else\ Any)$
Soundness:

- $n \in \gamma_s\ (num_s\ n)$: case distinction on $0 < n$, defs
- $n_1 \in \gamma_s\ a_1 \wedge n_2 \in \gamma_s\ a_2 \Longrightarrow n_1{+}n_2 \in \gamma_s\ (plus_s\ a_1\ a_2)$:
  case distinction on $a_1 = Pos \wedge a_2 = Pos$ and arithmetic with $0 < n_1 \longrightarrow 0 < n_2 \longrightarrow 0 < n_1 + n_2$.

5 Post-fixed points

Recall that a complete lattice is a type $'a$ with a partial order $\leq$ such that every set $X :: 'a\ set$ has a greatest lower bound, denoted $\bigsqcap X$. This means that $\forall\ x \in X.\ \bigsqcap X \leq x$ and $\forall\ y.\ ((\forall\ x \in X.\ y \leq x) \longrightarrow y \leq \bigsqcap X)$.
Prove that, for a complete lattice and a monotone function $f :: 'a \Rightarrow 'a$ on it, the set of post-fixed points of f is closed under $\bigsqcap$:

$$\forall\ X :: 'a\ set.\ ((\forall\ x \in X.\ f\,x \leq x) \longrightarrow f\ (\bigsqcap X) \leq \bigsqcap X)$$

Solution

lemma
  fixes X :: "('a::*complete_lattice*) *set*"
  assumes *mono*: "*mono f*"
  and *pfp*: "$\forall X.\ \forall x{\in}X.\ f\,x \leq x$"
  shows "$f\ (\bigsqcap X) \leq \bigsqcap X$"
proof(*rule* ‹$(\bigwedge x.\ x \in X \Longrightarrow f\ (\bigsqcap X) \leq x) \Longrightarrow f\ (\bigsqcap X) \leq \bigsqcap X$›)
  fix x **assume** x: "$x \in X$"
  then have "$\bigsqcap X \leq x$" **by** (*rule* ‹$x \in X \Longrightarrow \bigsqcap X \leq x$›)
  hence "$f\ (\bigsqcap X) \leq f\,x$" **using** *mono* **unfolding** *mono_def* **by** *auto*
  also have "... $\leq x$" **using** *pfp* x **by** *auto*
  finally show "$f\ (\bigsqcap X) \leq x$" **.** — by transitivity
qed

Above, spelling out monotonicity instead of writing *mono* is also acceptable.