IMP in HOLCF

Tobias Nipkow and Robert Sandner

October 1, 2005

Contents

1 Denotational Semantics of Commands in HOLCF

theory *Denotational* **imports** *HOLCF Natural* **begin**

1.1 Definition

```
constdefs
  dlift :: "(('a::type) discr -> 'b::pcpo) => ('a lift -> 'b)"
  "dlift f == (LAM x. case x of UU => UU | Def y => f·(Discr y))"

consts D :: "com => state discr -> state lift"

primrec
  "D(skip) = (LAM s. Def(undiscr s))"
  "D(X :== a) = (LAM s. Def((undiscr s)[X ↦ a(undiscr s)]))"
  "D(c0 ; c1) = (dlift(D c1) oo (D c0))"
  "D(if b then c1 else c2) =
        (LAM s. if b (undiscr s) then (D c1)·s else (D c2)·s)"
  "D(while b do c) =
        fix·(LAM w s. if b (undiscr s) then (dlift w)·((D c)·s)
                      else Def(undiscr s))"
```

1.2 Equivalence of Denotational Semantics in HOLCF and Evaluation Semantics in HOL

```
lemma dlift_Def [simp]: "dlift f·(Def x) = f·(Discr x)"
  by (simp add: dlift_def)
```

```
lemma cont_dlift [iff]: "cont (%f. dlift f)"
  by (simp add: dlift_def)

lemma dlift_is_Def [simp]:
    "(dlift f·l = Def y) = (∃x. l = Def x ∧ f·(Discr x) = Def y)"
  by (simp add: dlift_def split: lift.split)

lemma eval_implies_D: "⟨c,s⟩ ⟶c t ==> D c·(Discr s) = (Def t)"
  apply (induct set: evalc)
        apply simp_all
   apply (subst fix_eq)
   apply simp
  apply (subst fix_eq)
  apply simp
  done

lemma D_implies_eval: "!s t. D c·(Discr s) = (Def t) --> ⟨c,s⟩ ⟶c t"
  apply (induct c)
      apply simp
     apply simp
    apply force
   apply (simp (no_asm))
   apply force
  apply (simp (no_asm))
  apply (rule fix_ind)
    apply (fast intro!: adm_lemmas adm_chfindom ax_flat)
   apply (simp (no_asm))
  apply (simp (no_asm))
  apply safe
  apply fast
  done

theorem D_is_eval: "(D c·(Discr s) = (Def t)) = (⟨c,s⟩ ⟶c t)"
  by (fast elim!: D_implies_eval [rule_format] eval_implies_D)

end
```

2 Correctness of Hoare by Fixpoint Reasoning

theory *HoareEx* **imports** *Denotational* **begin**

An example from the HOLCF paper by Müller, Nipkow, Oheimb, Slotosch [1]. It demonstrates fixpoint reasoning by showing the correctness of the Hoare rule for while-loops.

```
types assn = "state => bool"

constdefs
  hoare_valid :: "[assn, com, assn] => bool"    ("|= {(1_)}/ (_)/ {(1_)}" 50)
```

```
  "|= {A} c {B} == ∀s t. A s ∧ D c $(Discr s) = Def t --> B t"

lemma WHILE_rule_sound:
    "|= {A} c {A} ==> |= {A} while b do c {λs. A s ∧ ¬ b s}"
  apply (unfold hoare_valid_def)
  apply (simp (no_asm))
  apply (rule fix_ind)
    apply (simp (no_asm)) — simplifier with enhanced adm-tactic
   apply (simp (no_asm))
  apply (simp (no_asm))
  apply blast
  done

end
```

References

[1] O. Müller, T. Nipkow, D. v. Oheimb, and O. Slotosch. HOLCF = HOL + LCF. *J. Functional Programming*, 9:191–223, 1999.