## 1 A small imperative Language

theory Imp imports Main begin

typedef ident
  — an unspecified (arbitrary) type of variable names

datatype val = T | F | num int

types state = "ident ⇒ val"

datatype
  exp = Num int
  | Var ident
  | Plus exp exp
  | Eq exp exp

datatype
  stmt = Assign ident exp ("_ := _ " 60)
  | Semi stmt stmt ("_; _" [60, 60] 10)
  | Cond exp stmt stmt ("IF _ THEN _ ELSE _" 60)
  | While exp stmt ("WHILE '(_') _" 60)

consts
  eval_exp :: "(state × exp × val) set"

syntax
  eval_exp' :: "(state × exp) ⇒ val ⇒ bool" ("_⇓e _" [61,61] 60)

translations
  "(σ, e)⇓ e v" == "(σ, e, v) ∈ eval_exp"

inductive eval_exp intros
  Num: "(σ, Num n)⇓ e num n"
  Var: "(σ, Var x)⇓ e σ x"
  Plus: "[[ (σ, e1)⇓ e num n1; (σ, e2)⇓ e num n2; v = num (n1 + n2)]" 
       "⇒ (σ, Plus e1 e2)⇓ e v"
  Eq1: "[[ (σ, e1)⇓ e v1; (σ, e2)⇓ e v2; v1 = v2 ]" 
       "⇒ (σ, Eq e1 e2)⇓ e T"
  Eq2: "[[ (σ, e1)⇓ e v1; (σ, e2)⇓ e v2; v1 ≠ v2 ]" 
       "⇒ (σ, Eq e1 e2)⇓ e F"

consts
  eval_stmt :: "(state × stmt × state) set"

syntax
  eval_stmt' :: "(state × stmt) ⇒ state ⇒ bool" ("_⇓s _" [61,61] 60)

translations
  "(σ, e)⇓ s σ'" == "(σ, e, σ') ∈ eval_stmt"

inductive eval_stmt intros
  Ass: "[[ (σ, e)⇓ e v ]" 
       "⇒ (σ, x := e)⇓ s σ[x ← v]"
  Semi:
  "[[ (σ, s1)⇓ s σ'; (σ', s2)⇓ s σ'' ]" 
       "⇒ (σ, s1;; s2)⇓ s σ'']"
  IfT:
lemma eval1: "(σ, e) \downarrow_σ" shows "∀v. (σ, e) \downarrow_σ v → v_1 = v_2"

using eval1 proof induct
  case (Num σ n)
  show "∀v_2. (σ, Num n) \downarrow_σ v_2 → num n = v_2"
  proof clarify
    fix v_2
    assume "(σ, Num n) \downarrow_σ v_2"
    thus "num n = v_2" by cases auto
  qed
  next
  case (Var σ x)
  show "∀v_2. (σ, Var x) \downarrow_σ v_2 → σ x = v_2"
  proof clarify
    fix v_2
    assume "(σ, Var x) \downarrow_σ v_2"
    thus "σ x = v_2" by cases auto
  qed
  next
  case (Plus σ e_1 e_2 n_1 n_2 v)
  have v: "v = num (n_1 + n_2)"
  have IH_1: "∀v_2. (σ, e_1) \downarrow_σ v_2 → num n_1 = v_2".
  have IH_2: "∀v_2. (σ, e_2) \downarrow_σ v_2 → num n_2 = v_2".
  show "∀v_2. (σ, Plus e_1 e_2) \downarrow_σ v_2 → v = v_2"
  proof clarify
    fix v_2
    assume "(σ, Plus e_1 e_2) \downarrow_σ v_2"
    then obtain n_1' n_2' where
      e_1: "(σ, e_1) \downarrow_σ num n_1'" and
      e_2: "(σ, e_2) \downarrow_σ num n_2'" and
      v_2: "v_2 = num (n_1' + n_2')"
    by cases auto
    from e_1 IH_1 have "n_1' = n_1" by blast
    moreover
    from e_2 IH_2 have "n_2' = n_2" by blast
    ultimately
    have "v_2 = num (n_1 + n_2)" using v_2 by simp
    thus "v = v_2" using v by simp
  qed
  next
  case (Eq1 σ e_1 e_2 n_1 n_2)
  have IH: "∀v_2. (σ, e_1) \downarrow_σ v_2 → n_1 = v_2" "∀v_2. (σ, e_2) \downarrow_σ v_2 → n_2 = v_2".
  have eq: "n_1 = n_2".
  show "∀v_2. (σ, Eq e_1 e_2) \downarrow_σ v_2 → T = v_2"
proof clarify
  fix \(v_2\)
  assume "\((\sigma, Eq e_1 e_2) \Downarrow c v_2\)"
  then obtain \(n_1'\) \(n_2'\) where
  \(e_1: "(\sigma, e_1) \Downarrow c n_1'" \) and
  \(e_2: "(\sigma, e_2) \Downarrow c n_2'" \) and
  \(T: "n_1' = n_2' \longrightarrow v_2 = T" \)
  by cases auto
  from \(e_1\) \(e_2\) IH
  have "\(n_1 = n_1'\)" and "\(n_2 = n_2'\)" by auto
  with \(T eq\)
  show "\(T = v_2\)" by simp
qed

next
case (Eq2 \(\sigma\) \(e_1\) \(e_2\) \(n_1\) \(n_2\))
  have IH: "\(\forall v_2. (\sigma, e_1) \Downarrow c v_2 \longrightarrow n_1 = v_2" \(\forall v_2. (\sigma, e_2) \Downarrow c v_2 \longrightarrow n_2 = v_2\) .
  have neq: "\(n_1 \neq n_2\)" .
  show "\(\forall v_2. (\sigma, Eq e_1 e_2) \Downarrow c v_2 \longrightarrow F = v_2\)"
  proof clarify
  fix \(v_2\)
  assume "\((\sigma, Eq e_1 e_2) \Downarrow c v_2\)"
  then obtain \(n_1'\) \(n_2'\) where
  \(e_1: "(\sigma, e_1) \Downarrow c n_1'" \) and
  \(e_2: "(\sigma, e_2) \Downarrow c n_2'" \) and
  \(F: "n_1' \neq n_2' \longrightarrow v_2 = F" \)
  by cases auto
  from \(e_1\) \(e_2\) IH
  have "\(n_1 = n_1'\)" and "\(n_2 = n_2'\)" by auto
  with \(F neq\)
  show "\(F = v_2\)" by simp
qed

corollary eval_exp_deterministic:
  assumes "\((\sigma, e) \Downarrow c v_1\)" and "\((\sigma, e) \Downarrow c v_2\)" shows "\(v_1 = v_2\)"
  using prems by (simp add: eval_exp_deterministic')

lemma eval_stmt_deterministic':
  assumes eval1: "\((\sigma, e) \Downarrow s_\sigma_1\)"
  shows "\(\forall \sigma_2. (\sigma, e) \Downarrow s_\sigma_2 \longrightarrow \sigma_1 = \sigma_2\)"
  using eval1 proof induct
  case (Ass \(\sigma\) \(e\) \(v\) \(x\))
  have v: "\((\sigma, e) \Downarrow c v" .
  show "\(\forall \sigma_2. (\sigma, x := e) \Downarrow s_\sigma_2 \longrightarrow \sigma[x := v] = \sigma_2\)"
  proof clarify
  fix \(\sigma_2\)
  assume "\((\sigma, x := e) \Downarrow s_\sigma_2\)"
  then obtain \(v'\) where
  \(v': "(\sigma, e) \Downarrow c v'" \) and
  \(\sigma_2: "\sigma_2 = \sigma[x := v']" \)
  by cases auto
  from \(v v'\)
have "v = v'" by (rule eval_exp_deterministic)
with σ₂
show "σ[x ← v] = σ₂" by simp
qed

next

case (Semi σ σ' σ'' s₁ s₂)
have IH₁: "∀σ₂. (σ, s₁) ▼ s σ₂ −→ σ' = σ₂".
have IH₂: "∀σ₂. (σ', s₂) ▼ s σ₂ −→ σ'' = σ₂".
show "∀σ₂. (σ, s₁;; s₂) ▼ s σ₂ −→ σ'' = σ₂"

proof clarify

fix σ₂
assume "(σ, s₁;; s₂) ▼ s σ₂"
then obtain σ₀', σ₀'' where
  s₁: "(σ, s₁) ▼ s σ₀'" and
  s₂: "(σ₀', s₂) ▼ s σ₂"
by cases auto
from s₁ IH₁
have "σ₀' = σ'" by simp
with s₂ IH₂
show "σ'' = σ₂" by simp
qed

next

case (IfT σ σ' e s₁ s₂)
have e: "(σ, e) ▼ s T".
have IH: "∀σ₂. (σ, s₁) ▼ s σ₂ −→ σ' = σ₂".
show "∀σ₂. (σ, IF e THEN s₁ ELSE s₂) ▼ s σ₂ −→ σ' = σ₂"

proof clarify

fix σ₂
assume "(σ, IF e THEN s₁ ELSE s₂) ▼ s σ₂"

hence "(σ, s₁) ▼ s σ₂" using e
by cases (auto dest: eval_exp_deterministic)
with IH show "σ' = σ₂" by simp
qed

next

case (IfF σ σ' e s₁ s₂)
have e: "(σ, e) ▼ s F".
have IH: "∀σ₂. (σ, s₂) ▼ s σ₂ −→ σ' = σ₂".
show "∀σ₂. (σ, IF e THEN s₁ ELSE s₂) ▼ s σ₂ −→ σ' = σ₂"

proof clarify

fix σ₂
assume "(σ, IF e THEN s₁ ELSE s₂) ▼ s σ₂"

hence "(σ, s₂) ▼ s σ₂" using e
by cases (auto dest: eval_exp_deterministic)
with IH show "σ' = σ₂" by simp
qed

next

case (WhileF σ e s)
have e: "(σ,e) ▼ s F".
show "∀σ₂. (σ, WHILE (e) s) ▼ s σ₂ −→ σ = σ₂"

proof clarify

fix σ₂
assume "(σ, WHILE (e) s) ▼ s σ₂"

thus "σ = σ₂" using e
by cases (auto dest: eval_exp_deterministic)
qed

next
case (WhileT σ σ' σ'' e s)
  have e: "(σ, e) \downarrow e T".
  have IHs: "∀σ₂. (σ, s) \downarrow s σ₂ \rightarrow σ' = σ₂".
  have IHw: "∀σ₂. (σ', WHILE (e) s) \downarrow s σ₂ \rightarrow σ'' = σ₂".
  show "∀σ₂. (σ, WHILE (e) s) \downarrow s σ₂ \rightarrow σ'' = σ₂"

proof clarify
  fix σ₂
  assume "(σ, WHILE (e) s) \downarrow s σ₂"
  then obtain σ₀' where
    s: "(σ, s) \downarrow s σ₀'" and
    w: "(σ₀', WHILE (e) s) \downarrow s σ₂"
    using e by cases (auto dest: eval_exp_deterministic)
  from s IHs
  have "σ₀' = σ''" by simp
  with w IHw
  show "σ'' = σ₂" by simp
qed

qed

corollary eval_stmt_deterministic:
  assumes "(σ, s) \downarrow s σ₁" and "(σ, s) \downarrow s σ₂" shows "σ₁ = σ₂"
  using prems by (simp add: eval_stmt_deterministic')

end