Types and Programming Languages
Isabelle Formalization of Chapter 3

Jeremy Siek

February 2, 2008

Contents

theory *Booleans*
imports *Main*
begin

1 Boolean Expression

datatype *trm* =
  Tru
| *Fls*
| *IF trm trm trm*

inductive *eval-Booleans* :: *trm* ⇒ *trm* ⇒ *bool* (**infix** ⟶ *51*)
where
  E-IfTrue[*intro*!]: *IF Tru t2 t3* ⟶ *t2* |
  E-IfFalse[*intro*!]: *IF Fls t2 t3* ⟶ *t3* |
  E-If[*intro*!]: *t1* ⟶ *t1'* ⟹ *IF t1 t2 t3* ⟶ *IF t1' t2 t3*

lemma *IF Tru Tru* (*IF Fls Fls Fls*) ⟶ *Tru* **by** *auto*

Inversion rules are generated by Isabelle.

inductive-cases *inversion-Tru*: *Tru* ⟶ *t*

Tru ⟶ *t* ⟹ *P*

inductive-cases *inversion-Fls*: *Fls* ⟶ *t*

Fls ⟶ *t* ⟹ *P*

inductive-cases *inversion-if-Tru*: *IF Tru t2 t3* ⟶ *t''*

⟦*IF Tru t2 t3* ⟶ *t''*; *t2* = *t''* ⟹ *P*;
 ⋀*t1'*. ⟦*t''* = *IF t1' t2 t3*; *Tru* ⟶ *t1'*⟧ ⟹ *P*⟧
⟹ *P*

inductive-cases *inversion-if-Fls*: *IF Fls t2 t3* ⟶ *t''*

⟦*IF Fls t2 t3* ⟶ *t''*; *t3* = *t''* ⟹ *P*;
 ⋀*t1'*. ⟦*t''* = *IF t1' t2 t3*; *Fls* ⟶ *t1'*⟧ ⟹ *P*⟧
⟹ *P*

inductive-cases *inversion-if*: *IF t1 t2 t3* ⟶ *t''*

⟦*IF t1 t2 t3* ⟶ *t''*; ⟦*t1* = *Tru*; *t2* = *t''*⟧ ⟹ *P*; ⟦*t1* = *Fls*; *t3* = *t''*⟧ ⟹ *P*;
 ⋀*t1'*. ⟦*t''* = *IF t1' t2 t3*; *t1* ⟶ *t1'*⟧ ⟹ *P*⟧
⟹ *P*

theorem *thm-3-5-4-determinacy*:
  (*t*::*trm*) ⟶ *t'* ⟹ (⋀ *t''*. *t* ⟶ *t''* ⟹ *t'* = *t''*)
proof (*induct rule*: *eval-Booleans.induct*)
  fix *t2 t3 t''* **assume** *IF Tru t2 t3* ⟶ *t''*

thus *t2* = *t''* **by** (*blast elim*: *inversion-if-Tru inversion-Tru*)
next
fix *t2 t3 t''* **assume** *IF Fls t2 t3* ⟶ *t''*
thus *t3* = *t''* **by** (*blast elim*: *inversion-if-Fls inversion-Fls*)
next
fix *t1*::*trm* **and** *t1' t2 t3 t''*
assume *t1t1*: *t1* ⟶ *t1'* **and** *IH*: ⋀*t''*. *t1* ⟶ *t''* ⟹ *t1'* = *t''*
and *iftp*: *IF t1 t2 t3* ⟶ *t''*

from *iftp* **show** *IF t1' t2 t3* = *t''*
proof (*rule inversion-if*)
assume *t1* = *Tru* **with** *t1t1* **have** *False* **using** *inversion-Tru* **by** *simp* **thus** *?thesis* **by** *simp*
next
assume *t1* = *Fls* **with** *t1t1* **have** *False* **using** *inversion-Fls* **by** *simp* **thus** *?thesis* **by** *simp*
next
fix *t1''* **assume** *tpp*: *t''* = *IF t1'' t2 t3* **and** *t1t1p*: *t1* ⟶ *t1''*
from *t1t1p IH* **have** *t1'* = *t1''* **by** *simp*
with *tpp* **show** *IF t1' t2 t3* = *t''* **by** *simp*
qed
qed

constdefs
normalform :: *trm* ⇒ *bool*
normalform t ≡ ¬ (∃ *t'*::*trm*. *t* ⟶ *t'*)

isvalue :: *trm* ⇒ *bool*
isvalue t ≡ (*t* = *Tru* ∨ *t* = *Fls*)

theorem *thm-3-5-7*: *isvalue t* ⟹ *normalform t*
proof (*induct t*)
show *normalform Tru* **using** *normalform-def* **by** (*blast elim*: *inversion-Tru*)
next
show *normalform Fls* **using** *normalform-def* **by** (*blast elim*: *inversion-Fls*)
next
fix *t1 t2 t3* **assume** *isvalue* (*IF t1 t2 t3*)
hence *False* **using** *isvalue-def* **by** *simp*
thus *normalform* (*IF t1 t2 t3*) **by** *simp*
qed

lemma *no-val-not-normal*: ¬ *isvalue t* ⟹ ¬ *normalform t*
proof (*induct t*)
assume ¬ *isvalue Tru* **thus** ¬ *normalform Tru*
using *isvalue-def* **by** *blast*
next
assume ¬ *isvalue Fls* **thus** ¬ *normalform Fls*
using *isvalue-def* **by** *blast*
next
fix *t1 t2 t3* **assume** *IH1*: ¬ *isvalue t1* ⟹ ¬ *normalform t1*
and *IH2*: ¬ *isvalue t2* ⟹ ¬ *normalform t2*
and *IH3*: ¬ *isvalue t3* ⟹ ¬ *normalform t3*
show ¬ *normalform* (*IF t1 t2 t3*)
proof (*cases t1*)
assume *t1* = *Tru* **thus** ¬ *normalform* (*IF t1 t2 t3*)
using *normalform-def* **by** *blast*
next

```
    assume t1 = Fls thus ¬ normalform (IF t1 t2 t3)
      using normalform-def by blast
  next
    fix trm1 trm2 trm3 assume t1 = IF trm1 trm2 trm3
    hence ¬ isvalue t1 using isvalue-def by simp
    with IH1 have X: ∃ t'. t1 ⟶ t' using normalform-def by simp
    from X obtain t' where t1 ⟶ t' by blast
    hence IF t1 t2 t3 ⟶ IF t' t2 t3 by blast
    thus ¬ normalform (IF t1 t2 t3) using normalform-def by blast
  qed
qed

theorem thm-3-5-8: normalform t ⟹ isvalue t
  using no-val-not-normal by blast

— The following works, but is different from Peirce's def.

inductive evals-B :: trm ⇒ trm ⇒ bool (infix ⟶* 51)
  where
  ebrefl: t ⟶* t and
  ebstep: ⟦ t ⟶ t'; t' ⟶* t'' ⟧ ⟹ t ⟶* t''

inductive-cases evals-B-inv: t ⟶* t'


⟦t ⟶* t'; t' = t ⟹ P; ⋀t'a. ⟦t ⟶ t'a; t'a ⟶* t'⟧ ⟹ P⟧ ⟹ P

lemma evals-normal-id: ⟦ t ⟶* t'; normalform t ⟧ ⟹ t = t'
proof (induct rule: evals-B.induct)
  fix t show t = t by simp
next
  fix t::trm and t' t''
  assume t ⟶ t' and normalform t
  hence False using normalform-def by simp
  thus t = t'' by simp
qed

theorem thm-3-5-11:
  t ⟶* u ⟹ normalform u ⟶ (∀ u'. t ⟶* u' ∧ normalform u' ⟶ u = u')
proof (induct rule: evals-B.induct)
  fix t show normalform t ⟶ (∀ u'. t ⟶* u' ∧ normalform u' ⟶ t = u')
    using evals-normal-id by simp
next
  fix t::trm and t' t'' u'
  assume ttp: t ⟶ t' and tptpp: t' ⟶* t''
    and IH: normalform t'' ⟶ (∀ u'. t' ⟶* u' ∧ normalform u' ⟶ t'' = u')
  show normalform t'' ⟶ (∀ u'. t ⟶* u' ∧ normalform u' ⟶ t'' = u')
  proof clarify
    fix u'
    assume nftpp: normalform t'' and tup: t ⟶* u' and nfup: normalform u'
    from IH nftpp have IH2: ∀ u'. t' ⟶* u' ∧ normalform u' ⟶ t'' = u' by simp
    from tup show t'' = u'
    proof (rule evals-B-inv)
      assume u' = t
      with nfup have normalform t by simp
      with ttp have False by (simp add: normalform-def)
```

```
    thus t'' = u' by simp
  next
    fix w assume tw: t ⟶ w and wup: w ⟶* u'
    from ttp tw have t' = w by (rule thm-3-5-4-determinacy)
    with wup have t' ⟶* u' by simp
    with IH2 nfup show t'' = u' by simp
  qed
 qed
qed

theorem progress:
 isvalue t ∨ (∃ t'. t ⟶ t')
proof (induct t)
 have isvalue Tru using isvalue-def by simp
 thus isvalue Tru ∨ (∃ t'. Tru ⟶ t') by simp
next
 have isvalue Fls using isvalue-def by simp
 thus isvalue Fls ∨ (∃ t'. Fls ⟶ t') by simp
next
 fix t1 t2 t3
 assume IH1: isvalue t1 ∨ (∃ t'. t1 ⟶ t')
   and IH2: isvalue t2 ∨ (∃ t'. t2 ⟶ t')
   and IH3: isvalue t3 ∨ (∃ t'. t3 ⟶ t')
 from IH1 have ∃ t'. IF t1 t2 t3 ⟶ t'
 proof
   assume isvalue t1
   hence t1 = Tru ∨ t1 = Fls using isvalue-def by simp
   thus ?thesis
   proof
     assume t1 = Tru
     hence IF t1 t2 t3 ⟶ t2 by blast
     thus ?thesis by blast
   next assume t1 = Fls
     hence IF t1 t2 t3 ⟶ t3 by blast
     thus ?thesis by blast
   qed
 next
   assume X: ∃ t'. t1 ⟶ t'
   from X obtain t1' where t1t1p: t1 ⟶ t1' by blast
   hence IF t1 t2 t3 ⟶ IF t1' t2 t3 by blast
   thus ∃ t'. IF t1 t2 t3 ⟶ t' by blast
 qed
 thus isvalue (IF t1 t2 t3) ∨ (∃ t'. IF t1 t2 t3 ⟶ t') by simp
qed

end


theory Naturals
imports Main
begin
```

2 Arithmetic Expressions

```
datatype trm =
```

```
    Tru
  | Fls
  | IF trm trm trm
  | Zero
  | Succ trm
  | Pred trm
  | IsZero trm

inductive num-val :: trm ⇒ bool
  where
  NumZero[intro!]: num-val Zero |
  NumSucc[intro!]: num-val t ⟹ num-val (Succ t)

inductive-cases num-if[elim!]: num-val (IF t1 t2 t3)
inductive-cases num-succ[elim!]: num-val (Succ t)
inductive-cases num-pred[elim!]: num-val (Pred t)
inductive-cases num-iszero[elim!]: num-val (IsZero t)

constdefs val :: trm ⇒ bool
  val t ≡ (t = Tru ∨ t = Fls ∨ num-val t)
declare val-def[simp]

inductive eval-Naturals :: trm ⇒ trm ⇒ bool (infix ↪ 51)
where
  E-IfTrue[intro!]: IF Tru t2 t3 ↪ t2 |
  E-IfFalse[intro!]: IF Fls t2 t3 ↪ t3 |
  E-If[intro!]: t1 ↪ t1' ⟹ IF t1 t2 t3 ↪ IF t1' t2 t3 |
  E-Succ[intro!]: t ↪ t' ⟹ Succ t ↪ Succ t' |
  E-PredZero[intro!]: Pred Zero ↪ Zero |
  E-PredSucc[intro!]: num-val t ⟹ Pred (Succ t) ↪ t |
  E-Pred[intro!]: t ↪ t' ⟹ Pred t ↪ Pred t' |
  E-IsZeroZero[intro!]: IsZero Zero ↪ Tru |
  E-IsZeroSucc[intro!]: num-val t ⟹ IsZero (Succ t) ↪ Fls |
  E-IsZero[intro!]: t ↪ t' ⟹ IsZero t ↪ IsZero t'

inductive-cases inversion-Tru[elim!]: Tru ↪ t
inductive-cases inversion-SuccTru[elim!]: Succ Tru ↪ t

inductive multi-evals :: trm ⇒ trm ⇒ bool (infix ↪* 51)
  where
  me-refl[intro!]: t ↪* t and
  me-step[intro!]: ⟦ t ↪ t'; t' ↪* t'' ⟧ ⟹ t ↪* t''

— Counter example:
theorem ¬ (∃ t. Succ Tru ↪ t) by blast

inductive big-step :: trm ⇒ trm ⇒ bool (infix ⇓ 51)
  where
  B-Value[intro!]: val v ⟹ v ⇓ v |
  B-IfTrue[intro!]: ⟦ t1 ⇓ Tru; t2 ⇓ v2 ⟧ ⟹ IF t1 t2 t3 ⇓ v2 |
  B-IfFalse[intro!]: ⟦ t1 ⇓ Fls; t3 ⇓ v3 ⟧ ⟹ IF t1 t2 t3 ⇓ v3 |
  B-Succ[intro!]: ⟦ t ⇓ nv; num-val nv ⟧ ⟹ Succ t ⇓ Succ nv |
  B-PredZero[intro!]: t ⇓ Zero ⟹ Pred t ⇓ Zero |
  B-PredSucc[intro!]: ⟦ t ⇓ Succ nv; num-val nv ⟧ ⟹ Pred t ⇓ nv |
  B-IsZeroZero[intro!]: t ⇓ Zero ⟹ IsZero t ⇓ Tru |
```

B-IsZeroSsucc[*intro*!]: ⟦ $t \Downarrow$ *Succ nv*; *num-val nv* ⟧ $\Longrightarrow$ *IsZero t* $\Downarrow$ *Fls*

inductive-cases *if-inv*: *IF t1 t2 t3* $\Downarrow$ *v*
inductive-cases *succ-inv*: *Succ t* $\Downarrow$ *v*
inductive-cases *pred-inv*: *Pred t* $\Downarrow$ *v*
inductive-cases *iszero-inv*: *IsZero t* $\Downarrow$ *v*
inductive-cases *zero-inv*: *Zero* $\Downarrow$ *v*
inductive-cases *tru-inv*: *Tru* $\Downarrow$ *v*
inductive-cases *fls-inv*: *Fls* $\Downarrow$ *v*

lemma *val-if-inv*: *val* (*IF e1 e2 e3*) $\Longrightarrow$ *P* **by** *auto*

lemma *val-inv*[*rule-format*]: $\forall$ *t*. *val v* $\longrightarrow$ *v* $\Downarrow$ *t* $\longrightarrow$ *t* = *v*
apply (*induct v*)
apply *clarify* **apply** (*erule tru-inv*) **apply** *simp*
apply *clarify* **apply** (*erule fls-inv*) **apply** *simp*
apply *clarify* **apply** (*erule val-if-inv*)
apply *clarify* **apply** (*erule zero-inv*) **apply** *simp*
apply *clarify* **apply** (*erule succ-inv*) **apply** *blast*
apply *simp* **apply** (*erule num-succ*) **apply** *blast*
apply *clarify* **apply** *simp* **apply** *blast*
apply *clarify* **apply** *simp* **apply** *blast*
done

lemma *small-big-big*[*rule-format*]: *t* $\hookrightarrow$ *t′* $\Longrightarrow$ $\forall$ *v*. *t′* $\Downarrow$ *v* $\longrightarrow$ *t* $\Downarrow$ *v*
apply (*induct rule*: *eval-Naturals.induct*)
apply *clarify* **using** *val-def* **apply** *blast*
apply *clarify* **using** *val-def* **apply** *blast*
apply *clarify* **apply** (*erule if-inv*) **apply** *simp* **apply** (*erule num-if*)
apply (*rule B-IfTrue*) **apply** *blast* **apply** *simp*
apply (*rule B-IfFalse*) **apply** *blast* **apply** *simp*
apply *clarify* **apply** (*erule succ-inv*) **apply** *simp* **apply** (*erule num-succ*)
apply (*erule-tac x*=*t′* **in** *allE*) **apply** (*erule impE*)
apply (*rule B-Value*) **apply** *simp*
apply (*rule B-Succ*) **apply** *simp*
apply *simp* **apply** (*erule-tac x*=*nv* **in** *allE*) **apply** *simp* **apply** (*rule B-Succ*)
apply *simp* **apply** *simp*
apply (*rule allI*) **apply** (*rule impI*)
apply (*erule zero-inv*) **apply** *simp* **apply** (*rule B-PredZero*)
apply (*rule B-Value*) **apply** *simp*
apply *clarify* **apply** (*rule B-PredSucc*) **apply** (*rule B-Succ*) **apply** *simp*
using *val-inv* **apply** *simp* **apply** *blast*
using *val-inv* **apply** *simp* **apply** *blast*
apply (*rule allI*) **apply** (*rule impI*)
apply (*erule pred-inv*) **apply** *simp* **apply** (*erule num-pred*)
apply *simp* **apply** (*erule-tac x*=*Zero* **in** *allE*) **apply** *simp*
apply (*rule B-PredZero*) **apply** *simp*
apply (*rule B-PredSucc*) **apply** *simp* **apply** *simp*
apply (*rule allI*) **apply** (*rule impI*) **apply** (*erule tru-inv*)
apply *simp* **apply** (*rule B-IsZeroZero*) **apply** (*rule B-Value*) **apply** *simp* **apply** *blast*
apply (*rule allI*) **apply** (*rule impI*) **apply** (*erule fls-inv*)
apply *simp* **apply** (*rule B-IsZeroSucc*)
apply (*rule B-Value*) **apply** *simp* **apply** *blast* **apply** *simp*
apply (*rule allI*) **apply** (*rule impI*) **apply** (*erule iszero-inv*)
apply *simp* **apply** (*erule num-iszero*)

```
    apply simp apply blast
    apply blast
  done

lemma small-implies-big[rule-format]: t ↪* v ⟹ val v ⟶ t ⇓ v
proof (induct rule: multi-evals.induct)
  fix t show val t ⟶ t ⇓ t by blast
next
  fix t::trm and t' t''
  assume e1: t ↪ t' and e2: t' ↪* t''
    and IH: val t'' ⟶ t' ⇓ t''
  from e1 show val t'' ⟶ t ⇓ t''
  proof (rule eval-Naturals.cases)
    fix t2 t3
    assume t: t = IF Tru t2 t3 and tp: t' = t2
    show val t'' ⟶ t ⇓ t''
    proof
      assume vtpp: val t''
      with IH tp have e3: t2 ⇓ t'' by blast
      have tru: Tru ⇓ Tru apply (rule B-Value) by simp
      from t tru e3 show t ⇓ t''
        apply simp apply (rule B-IfTrue) by blast
    qed
  next
    fix t2 t3
    assume t: t = IF Fls t2 t3 and tp: t' = t3
    show val t'' ⟶ t ⇓ t''
    proof
      assume vtpp: val t''
      with IH tp have e3: t3 ⇓ t'' by blast
      have fls: Fls ⇓ Fls apply (rule B-Value) by simp
      from t fls e3 show t ⇓ t''
        apply simp apply (rule B-IfFalse) by blast
    qed
  next
    fix t1 t1' t2 t3
    assume t: t = IF t1 t2 t3 and tp: t' = IF t1' t2 t3
      and t11: t1 ↪ t1'
    show val t'' ⟶ t ⇓ t''
    proof
      assume vtpp: val t''
      with IH have tp-tpp: t' ⇓ t'' by blast
      with tp have e3: IF t1' t2 t3 ⇓ t'' by simp
      from t11 t tp have ttp: t ↪ t' by blast
      from ttp tp-tpp show t ⇓ t'' by (rule small-big-big)
    qed
  next
    fix tt t'a assume t: t = Succ tt and tp: t' = Succ t'a
      and ttp: tt ↪ t'a
    show val t'' ⟶ t ⇓ t''
    proof
      assume vtpp: val t''
      with IH have tp-tpp: t' ⇓ t'' by blast
      with tp have Succ t'a ⇓ t'' by simp
      from ttp t tp have ttp: t ↪ t' by blast
```

```
    from ttp tp-tpp show t ⇓ t'' by (rule small-big-big)
  qed
next
  assume t: t = Pred Zero and tp: t' = Zero
  show val t'' ⟶ t ⇓ t''
  proof
    assume vtpp: val t''
    with IH have tp-tpp: t' ⇓ t'' by blast
    with tp have Zero ⇓ t'' by simp
    hence tpp: t'' = Zero using zero-inv by blast
    with t show t ⇓ t'' apply simp apply (rule B-PredZero) apply (rule B-Value)
      apply simp apply blast done
  qed
next
  fix tt assume t: t = Pred (Succ tt) and tp: t' = tt and nvt: num-val tt
  show val t'' ⟶ t ⇓ t''
  proof
    assume vtpp: val t''
    with IH have tp-tpp: t' ⇓ t'' by blast
    with tp have tt-tpp: tt ⇓ t'' by simp
    with nvt have t'' = tt using val-inv by simp
    with t tp vtpp tt-tpp nvt show t ⇓ t'' apply simp apply (rule B-PredSucc)
      apply (rule B-Succ) by auto
  qed
next
  fix tt t'a assume t: t = Pred tt and tp: t' = Pred t'a
    and ttp: tt ↪ t'a
  show val t'' ⟶ t ⇓ t''
  proof
    assume vtpp: val t''
    with IH have tp-tpp: t' ⇓ t'' by blast
    with tp have Pred t'a ⇓ t'' by simp
    from ttp t tp have ttp: t ↪ t' by blast
    from ttp tp-tpp show t ⇓ t'' by (rule small-big-big)
  qed
next
  assume t: t = IsZero Zero and tp: t' = Tru
  show val t'' ⟶ t ⇓ t''
  proof
    assume vtpp: val t''
    with IH have tp-tpp: t' ⇓ t'' by blast
    from tp tp-tpp have tpp: t'' = Tru using tru-inv by blast
    from t tpp tp-tpp show t ⇓ t'' apply simp apply (rule B-IsZeroZero)
      apply (rule B-Value) apply simp apply blast done
  qed
next
  fix tt assume t: t = IsZero (Succ tt) and tp: t' = Fls
    and nvt: num-val tt
  show val t'' ⟶ t ⇓ t''
  proof
    assume vtpp: val t''
    with IH have tp-tpp: t' ⇓ t'' by blast
    with tp have tt-tpp: Fls ⇓ t'' by simp
    hence tpp: t'' = Fls using fls-inv by blast
    from nvt have val tt by simp
```

```
    hence tt-tt: tt ⇓ tt by (rule B-Value)
    hence Succ tt ⇓ Succ tt by (rule B-Succ)
    with t tpp nvt show t ⇓ t'' apply simp apply (rule B-IsZeroSucc) by auto
   qed
  next
   fix tt t'a assume t: t = IsZero tt and tp: t' = IsZero t'a
    and ttp: tt ↪ t'a
   show val t'' ⟶ t ⇓ t''
   proof
    assume vtpp: val t''
    with IH have tp-tpp: t' ⇓ t'' by blast
    with tp have IsZero t'a ⇓ t'' by simp
    from ttp t tp have ttp: t ↪ t' by blast
    from ttp tp-tpp show t ⇓ t'' by (rule small-big-big)
   qed
  qed
qed


lemma A-6-if[rule-format]: t1 ↪* t1' ⟹ (∀ t2 t3. IF t1 t2 t3 ↪* IF t1' t2 t3)
proof (induct rule: multi-evals.induct)
  fix t
  show ∀ t2 t3. IF t t2 t3 ↪* IF t t2 t3 using me-refl by blast
next
  fix t t' t''
  assume ttp: t ↪ t' and t' ↪* t'' and IH: ∀ t2 t3. IF t' t2 t3 ↪* IF t'' t2 t3
  show ∀ t2 t3. IF t t2 t3 ↪* IF t'' t2 t3
  proof clarify
   fix t2 t3
   from ttp have a: IF t t2 t3 ↪ IF t' t2 t3 by (rule E-If)
   from IH have b: IF t' t2 t3 ↪* IF t'' t2 t3 by simp
   from a b show IF t t2 t3 ↪* IF t'' t2 t3 by (rule me-step)
  qed
qed

lemma A-6-succ[rule-format]: t1 ↪* t1' ⟹ Succ t1 ↪* Succ t1'
  apply(induct rule: multi-evals.induct)
  using me-refl apply blast
  using E-Succ me-step by blast

lemma A-6-pred[rule-format]: t1 ↪* t1' ⟹ Pred t1 ↪* Pred t1'
  apply(induct rule: multi-evals.induct)
  using me-refl apply blast
  using E-Pred me-step by blast

lemma A-6-iszero[rule-format]: t1 ↪* t1' ⟹ IsZero t1 ↪* IsZero t1'
  apply(induct rule: multi-evals.induct)
  using me-refl apply blast
  using E-IsZero me-step by blast

lemma multi-single-multi[rule-format]: t1 ↪* t2 ⟹ (∀ t3. t2 ↪ t3 ⟶ t1 ↪* t3)
  apply (induct rule: multi-evals.induct)
  apply clarify using me-step[of t t3 t3] apply blast
  apply clarify apply (erule-tac x=t3 in allE) apply simp using me-step by blast
```

lemma *multi-transitive*[*rule-format*]:
t1 $\hookrightarrow^*$ *t2* $\Longrightarrow$ ($\forall$ *t3*. *t2* $\hookrightarrow^*$ *t3* $\longrightarrow$ *t1* $\hookrightarrow^*$ *t3*)
apply (*induct rule*: *multi-evals.induct*)
apply *simp*
apply *clarify* **apply** (*erule-tac x=t3* **in** *allE*) **apply** *simp* **using** *me-step* **by** *blast*

lemma *big-implies-value*: *t* $\Downarrow$ *v* $\Longrightarrow$ *val v*
apply (*induct rule*: *big-step.induct*) **by** *auto*

lemma *big-implies-small*[*rule-format*]: *t* $\Downarrow$ *v* $\Longrightarrow$ *val v* $\longrightarrow$ *t* $\hookrightarrow^*$ *v*
proof (*induct rule*: *big-step.induct*)
fix *v* **assume** *val v*
show *val v* $\longrightarrow$ *v* $\hookrightarrow^*$ *v* **using** *me-refl* **by** *blast*
next
fix *t1 t2 v2 t3*
assume *t1* $\Downarrow$ *Tru* **and** *IH1*: *val Tru* $\longrightarrow$ *t1* $\hookrightarrow^*$ *Tru* **and** *t2* $\Downarrow$ *v2*
and *IH2*: *val v2* $\longrightarrow$ *t2* $\hookrightarrow^*$ *v2*
show *val v2* $\longrightarrow$ *IF t1 t2 t3* $\hookrightarrow^*$ *v2*
proof
assume *vv2*: *val v2*
from *IH1* **have** *t1* $\hookrightarrow^*$ *Tru* **by** *simp*
hence *a*: *IF t1 t2 t3* $\hookrightarrow^*$ *IF Tru t2 t3* **by** (*rule A-6-if*)
have *b*: *IF Tru t2 t3* $\hookrightarrow$ *t2* **by** (*rule E-IfTrue*)
from *a b* **have** *c*: *IF t1 t2 t3* $\hookrightarrow^*$ *t2* **by** (*rule multi-single-multi*)
from *vv2 IH2* **have** *t2* $\hookrightarrow^*$ *v2* **by** *simp*
with *c* **show** *IF t1 t2 t3* $\hookrightarrow^*$ *v2* **by** (*rule multi-transitive*)
qed
next
fix *t1 t3 v3 t2*
assume *t1* $\Downarrow$ *Fls* **and** *IH1*: *val Fls* $\longrightarrow$ *t1* $\hookrightarrow^*$ *Fls* **and** *t3* $\Downarrow$ *v3*
and *IH2*: *val v3* $\longrightarrow$ *t3* $\hookrightarrow^*$ *v3*
show *val v3* $\longrightarrow$ *IF t1 t2 t3* $\hookrightarrow^*$ *v3*
proof
assume *vv3*: *val v3*
from *IH1* **have** *t1* $\hookrightarrow^*$ *Fls* **by** *simp*
hence *a*: *IF t1 t2 t3* $\hookrightarrow^*$ *IF Fls t2 t3* **by** (*rule A-6-if*)
have *b*: *IF Fls t2 t3* $\hookrightarrow$ *t3* **by** (*rule E-IfFalse*)
from *a b* **have** *c*: *IF t1 t2 t3* $\hookrightarrow^*$ *t3* **by** (*rule multi-single-multi*)
from *vv3 IH2* **have** *t3* $\hookrightarrow^*$ *v3* **by** *simp*
with *c* **show** *IF t1 t2 t3* $\hookrightarrow^*$ *v3* **by** (*rule multi-transitive*)
qed
next
fix *t v*
assume *t* $\Downarrow$ *v* **and** *IH*: *val v* $\longrightarrow$ *t* $\hookrightarrow^*$ *v*
show *val* (*Succ v*) $\longrightarrow$ *Succ t* $\hookrightarrow^*$ *Succ v*
proof
assume *val* (*Succ v*)
hence *val v* **by** *auto*
with *IH* **have** *t* $\hookrightarrow^*$ *v* **by** *clarify*
thus *Succ t* $\hookrightarrow^*$ *Succ v* **by** (*rule A-6-succ*)
qed
next
fix *t* **assume** *t* $\Downarrow$ *Zero* **and** *IH*: *val Zero* $\longrightarrow$ *t* $\hookrightarrow^*$ *Zero*
show *val Zero* $\longrightarrow$ *Pred t* $\hookrightarrow^*$ *Zero*
proof

```
    assume val Zero
    with IH have t ↪* Zero by simp
    hence a: Pred t ↪* Pred Zero by (rule A-6-pred)
    have Pred Zero ↪ Zero by (rule E-PredZero)
    with a show Pred t ↪* Zero by (rule multi-single-multi)
  qed
next
  fix t v
  assume tsv: t ⇓ Succ v and IH: val (Succ v) ⟶ t ↪* Succ v
  show val v ⟶ Pred t ↪* v
  proof
    assume vv: val v
    from tsv have val (Succ v) by (rule big-implies-value)
    hence nv: num-val v by auto
    with IH have ts: t ↪* Succ v apply simp apply (erule impE) by auto
    from ts have pp: Pred t ↪* Pred (Succ v) by (rule A-6-pred)
    from nv have pv: Pred (Succ v) ↪ v by (rule E-PredSucc)
    from pp pv show Pred t ↪* v by (rule multi-single-multi)
  qed
next
  fix t
  assume t ⇓ Zero and IH: val Zero ⟶ t ↪* Zero
  from IH have t ↪* Zero apply simp apply (erule impE) apply blast by simp
  hence zz: IsZero t ↪* IsZero Zero by (rule A-6-iszero)
  have zt: IsZero Zero ↪ Tru by (rule E-IsZeroZero)
  from zz zt show val Tru ⟶ IsZero t ↪* Tru
    apply clarify apply (rule multi-single-multi) by auto
next
  fix t nv
  assume t ⇓ Succ nv and IH: val (Succ nv) ⟶ t ↪* Succ nv
    and nv: num-val nv
  from nv have val (Succ nv) by auto
  with IH have t ↪* Succ nv by simp
  hence zz: IsZero t ↪* IsZero (Succ nv) by (rule A-6-iszero)
  have zf: IsZero (Succ nv) ↪ Fls by (rule E-IsZeroSucc)
  from zz zf show val Fls ⟶ IsZero t ↪* Fls
    apply clarify apply (rule multi-single-multi) by auto
qed

theorem big-eq-small: val v ⟹ t ⇓ v = t ↪* v
  apply (rule iffI)
  apply (rule big-implies-small) apply simp apply simp
  apply (rule small-implies-big) apply auto done


end
```