tapl-isabelle

takei

February 14, 2022

Contents

theory *Term31*
imports *Main*
begin

1 definition

datatype *t*
  = *true*
  | *false*
  | *Cond t t t* (*If* - *Then* - *Else* - [*95*,*95*,*95*] *90*)

inductive *eval* :: $t \Rightarrow t \Rightarrow bool$ (- $\mapsto$ - [*50*,*50*] *40*)
  where
    E-IfTrue: *If true Then t2 Else t3* $\mapsto$ *t2*
  | *E-IfFalse*: *If false Then t2 Else t3* $\mapsto$ *t3*

| *E-If*: $t1 \mapsto t1' \Longrightarrow$ *If t1 Then t2 Else t3* $\mapsto$ *If t1′ Then t2 Else t3*

definition *is-value* :: $t \Rightarrow bool$
where *is-value t* $\longleftrightarrow$ (t = *true* $\vee$ t = *false*)

lemma *true-is-value*[*intro*]: *is-value true* **using** *is-value-def* **by** *simp*
lemma *false-is-value*[*intro*]: *is-value false* **using** *is-value-def* **by** *simp*
lemma *if-isnot-value*: *is-value* (*If t1 Then t2 Else t3*) $\Longrightarrow$ *False* **using** *is-value-def* **by** *simp*

lemma *ex-353*:
assumes *s*: s = *If true Then false Else false*
assumes *t*: t = *If s Then true Else true*
assumes *u*: u = *If false Then true Else true*
shows *If t Then false Else false* $\mapsto$ *If u Then false Else false*
apply (*simp add*: *s t u*)
apply (*rule E-If*)
apply (*rule E-If*)
apply (*rule E-IfTrue*)
done

lemma *true-cannot-eval*: $\neg(\exists\, t.$ *true* $\mapsto t)$
using *eval.cases* **by** *blast*

lemma *false-cannot-eval*: $\neg(\exists\, t.$ *false* $\mapsto t)$
using *eval.cases* **by** *blast*

lemma *eval-IfTrue-eq-then*: *If true Then t2 Else t3* $\mapsto t2' \Longrightarrow t2' = t2$
using *eval.cases true-cannot-eval* **by** *auto*

lemma *eval-IfFalse-eq-else*: *If false Then t2 Else t3* $\mapsto t3' \Longrightarrow t3' = t3$
using *eval.cases false-cannot-eval* **by** *auto*

2 3.5.4 deterministic of eval

theorem *eval-deterministic*: $[\![\ t \mapsto t';\ t \mapsto t''\ ]\!] \Longrightarrow t' = t''$
proof(*induct t arbitrary*: t' t'')
case *true* **thus** *?case* **using** *true-cannot-eval* **by** *blast*
next
case *false* **thus** *?case* **using** *false-cannot-eval* **by** *blast*
next
case *Cond*
fix *t1 t2 t3 t* t''
assume *t1-induct*: $\bigwedge t'\ t''.\ t1 \mapsto t' \Longrightarrow t1 \mapsto t'' \Longrightarrow t' = t''$
assume *et′*: *If t1 Then t2 Else t3* $\mapsto t'$
assume *et″*: *If t1 Then t2 Else t3* $\mapsto t''$
show $t' = t''$
proof (*cases t1*)
case *true*

have $t' = t2$ **using** *true eval-IfTrue-eq-then et'* **by** *simp*
also have $t'' = t2$ **using** *true eval-IfTrue-eq-then et''* **by** *simp*
finally show $t' = t''$ **by** *simp*
next
case *false*
have $t' = t3$ **using** *false eval-IfFalse-eq-else et'* **by** *simp*
also have $t'' = t3$ **using** *false eval-IfFalse-eq-else et''* **by** *simp*
finally show $t' = t''$ **by** *simp*
next
case *Cond*
obtain $t1'$ **where** $t1t1'$: $t1 \mapsto t1'$ **and** $t'if$: $t' = (\mathit{If}\ t1'\ \mathit{Then}\ t2\ \mathit{Else}\ t3)$
using *Cond et' eval.cases* **by** *blast*
obtain $t1''$ **where** $t1t1''$: $t1 \mapsto t1''$ **and** $t''if$: $t'' = (\mathit{If}\ t1''\ \mathit{Then}\ t2\ \mathit{Else}\ t3)$
using *Cond et'' eval.cases* **by** *blast*
have $t1' = t1''$ **using** *t1-induct t1t1' t1t1''* **by** *simp*
with $t'if\ t''if$ **show** $t' = t''$ **by** *simp*
qed
qed

definition *is-normal-form* :: $t \Rightarrow bool$
where *is-normal-form* $t \longleftrightarrow \neg(\exists t'.\ t \mapsto t')$

lemma *is-normal-formE*[*intro*]: $\neg(\exists t'.\ t \mapsto t') \Longrightarrow$ *is-normal-form* t **using** *is-normal-form-def* **by** *simp*
lemma *is-normal-formI*[*elim*]: *is-normal-form* $t \Longrightarrow \neg(\exists t'.\ t \mapsto t')$ **using** *is-normal-form-def* **by** *simp*

lemma *true-is-normal-form*[*intro*]: *is-normal-form true* **using** *true-cannot-eval is-normal-form-def* **by** *simp*
lemma *false-is-normal-form*[*intro*]: *is-normal-form false* **using** *false-cannot-eval is-normal-form-def* **by** *simp*
lemma *normal-form-cannot-eval*: $\llbracket$ *is-normal-form* t; $t \mapsto t'$ $\rrbracket \Longrightarrow$ *False* **using** *is-normal-form-def* **by** *blast*
lemma *if-isnot-normal-form*: $\neg$(*is-normal-form* (*If t1 Then t2 Else t3*))
apply (*induct t1 arbitrary*: *t2 t3*)
using *E-IfTrue is-normal-form-def* **apply** *blast*
using *E-IfFalse is-normal-form-def* **apply** *blast*
using *E-If is-normal-form-def* **by** *metis*

3 3.5.7 value is normal form

theorem *value-is-normal-form*: *is-value* $t \Longrightarrow$ *is-normal-form* t
using *is-value-def* **by** *fastforce*

4 3.5.8 normal form is value

theorem *normal-form-is-value*: *is-normal-form* $t \Longrightarrow$ *is-value* t

apply (*rule t.exhaust*[*of t*], *blast*, *blast*)
apply (*simp add*: *if-isnot-normal-form*)
done

5 3.5.9 definition of eval*

inductive *eval-star* :: $t \Rightarrow t \Rightarrow bool$ (- $\mapsto*$ - [*50*,*50*] *40*) **where**
EsRefl: $t \mapsto* t$
| *EsStep*: $[\![t \mapsto t']\!] \Longrightarrow t \mapsto* t'$
| *EsTrans*: $[\![t \mapsto* t'; t' \mapsto* t'']\!] \Longrightarrow t \mapsto* t''$

lemma *eval-star-refl*[*intro*]: $t = t' \Longrightarrow t \mapsto* t'$ **using** *EsRefl* **by** *simp*
lemma *eval-star-step*[*intro*]: $t \mapsto t' \Longrightarrow t \mapsto* t'$ **by** (*rule EsStep*)
lemma *eval-star-trans1*[*intro*]: $[\![t \mapsto* t'; t' \mapsto* t'']\!] \Longrightarrow t \mapsto* t''$ **by** (*rule EsTrans*)
lemma *eval-star-trans2*[*intro*]: $[\![t \mapsto t'; t' \mapsto* t'']\!] \Longrightarrow t \mapsto* t''$ **using** *EsStep* *EsTrans* **by** *blast*
lemma *eval-star-trans3*[*intro*]: $[\![t \mapsto* t'; t' \mapsto t'']\!] \Longrightarrow t \mapsto* t''$ **using** *EsStep* *EsTrans* **by** *blast*

lemma *eval-starI1*: $[\![t = t' \lor t \mapsto t' \lor (\exists t''. t \mapsto* t'' \land t'' \mapsto* t')]\!] \Longrightarrow t \mapsto* t'$
using *EsRefl EsStep EsTrans* **by** *blast*
lemma *eval-starI2*: $[\![t = t' \lor t \mapsto t' \lor (\exists t''. t \mapsto* t'' \land t'' \mapsto t')]\!] \Longrightarrow t \mapsto* t'$
using *EsRefl EsStep EsTrans* **by** *blast*
lemma *eval-starI3*: $[\![t = t' \lor t \mapsto t' \lor (\exists t''. t \mapsto t'' \land t'' \mapsto* t')]\!] \Longrightarrow t \mapsto* t'$
using *EsRefl EsStep EsTrans* **by** *blast*

lemma *E-If-star*: $t1 \mapsto* t1' \Longrightarrow$ *If t1 Then t2 Else t3* $\mapsto*$ *If t1' Then t2 Else t3*
proof (*induct rule*: *eval-star.induct*)
case (*EsRefl t*)
then show *?case* **by** (*rule eval-star-refl*, *rule refl*)
next
case (*EsStep t t'*)
show *?case* **by** (*rule eval-star-step*, *rule E-If*, *rule EsStep*)
next
case (*EsTrans t t' t''*)
show *?case* **by** (*rule eval-star-trans1*, *rule EsTrans*(*2*), *rule EsTrans*(*4*))
qed

lemma *eval-star-IfTrue*: $[\![t1 \mapsto* true]\!] \Longrightarrow$ *If t1 Then t2 Else t3* $\mapsto* t2$
using *E-If-star*[*of t1*, *of true*] *E-IfTrue eval-star-trans2 eval-star-trans3* **by** *blast*

lemma *eval-star-IfFalse*: $[\![t1 \mapsto* false]\!] \Longrightarrow$ *If t1 Then t2 Else t3* $\mapsto* t3$
using *E-If-star*[*of t1*, *of false*] *E-IfFalse eval-star-trans2 eval-star-trans3* **by** *blast*

lemma *normal-form-eval-star-refl*:
assumes *es*: $t \mapsto* t'$
assumes *vt*: *is-normal-form t*
shows $t = t'$
using *es vt*

apply (*induct rule*: *eval-star.induct*)
apply *simp*
using *normal-form-cannot-eval* **apply** *blast*
using *normal-form-cannot-eval* **by** *blast*

6 3.5.11 deterministic of eval*

lemma *eval-star-deterministic*: $[\![t \mapsto* t'; \textit{is-normal-form } t'; t \mapsto t''; \textit{is-normal-form } t'']\!] \Longrightarrow t' = t''$
proof (*induct rule*: *eval-star.induct*)
case (*EsRefl t*)
then show *?case*
using *normal-form-cannot-eval* **by** *blast*
next
case (*EsStep t t'*)
then show *?case*
using *eval-deterministic* **by** *blast*
next
case (*EsTrans t tmid' t'*)
then show *?case*
oops

7 3.5.11' definition of eval n

inductive *eval-n* :: $t \Rightarrow nat \Rightarrow t \Rightarrow bool$ (- $\mapsto$^- - [*50*,*90*,*50*] *40*) **where**
En0: $t \mapsto$^*0* t
| *EnN*: $[\![t \mapsto \hat{} n\ t'; t' \mapsto t'']\!] \Longrightarrow t \mapsto \hat{}(Suc\ n)\ t''$

lemma *exist-first-step*: $[\![t \mapsto \hat{}(Suc\ n)\ t']\!] \Longrightarrow \exists t''.\ t \mapsto \hat{} n\ t'' \wedge t'' \mapsto t'$
by (*metis eval-n.simps nat.inject old.nat.distinct*(*2*))
lemma *exist-mid-step*: $[\![t \mapsto \hat{}(n + m)\ t']\!] \Longrightarrow \exists t''.\ t \mapsto \hat{} n\ t'' \wedge t'' \mapsto \hat{} m\ t'$
proof (*induct m arbitrary*: *n t t'*)
case *0*
then show *?case*
using *En0* **by** *auto*
next
case (*Suc m*)
then show *?case*
by (*metis EnN exist-first-step nat-arith.suc1*)
qed
lemma *exist-mid-step2*: $[\![t \mapsto \hat{} n\ t'; m < n]\!] \Longrightarrow \exists t''.\ t \mapsto \hat{} m\ t'' \wedge t'' \mapsto \hat{}(n - m)\ t'$
using *exist-mid-step* **by** *force*

lemma *eval-0-refl*[*elim*]: $t \mapsto \hat{} 0\ t' \Longrightarrow t = t'$
by (*metis Zero-neq-Suc eval-n.simps*)
lemma *eval-1-step*[*elim*]: $t \mapsto \hat{} 1\ t' \Longrightarrow t \mapsto t'$
by (*metis eval-n.simps lessI less-one nat.simps*(*3*) *one-neq-zero*)

lemma *eval-n-star*[*elim*]: $t \mapsto\widehat{\ } n\ t' \Longrightarrow t \mapsto\!* \ t'$
proof (*induct rule*: *eval-n.induct*)
  case (*En0 t*)
  then show *?case* **by** (*rule EsRefl*)
next
  case (*EnN t t′ n t′′*)
  then show *?case* **by** *blast*
qed

lemma *eval-n-plus-1*: $[\![\ t \mapsto\widehat{\ } n\ t';\ t' \mapsto t''\ ]\!] \Longrightarrow t \mapsto\widehat{\ }(Suc\ n)\ t''$ **by** (*simp add*: *EnN*)
lemma *eval-n-plus-m*: $[\![\ t \mapsto\widehat{\ } n\ t';\ t' \mapsto\widehat{\ } m\ t''\ ]\!] \Longrightarrow t \mapsto\widehat{\ }(n+m)\ t''$
proof (*induct m arbitrary*: *n t t′ t′′*)
  case *0*
  then show *?case*
    using *eval-0-refl* **by** *auto*
next
  case (*Suc n*)
  then show *?case*
    by (*metis eval-n.simps nat.inject nat.simps*(*3*) *nat-arith.suc1*)
qed
lemma *eval-1-plus-n*: $[\![\ t \mapsto t';\ t' \mapsto\widehat{\ } n\ t''\ ]\!] \Longrightarrow t \mapsto\widehat{\ }(Suc\ n)\ t''$
  by (*metis One-nat-def Suc-eq-plus1-left eval-n.simps eval-n-plus-m*)

lemma *refl-eval-0*[*intro*]: $t = t' \Longrightarrow t \mapsto\widehat{\ } 0\ t'$
  using *En0* **by** *presburger*
lemma *step-eval-1*[*intro*]: $t \mapsto t' \Longrightarrow t \mapsto\widehat{\ } 1\ t'$
  using *En0 EnN* **by** *force*
lemma *eval-star-eval-n*[*intro*]: $t \mapsto\!* \ t' \Longrightarrow \exists\, n.\ t \mapsto\widehat{\ } n\ t'$
proof (*induct rule*: *eval-star.induct*)
  case (*EsRefl t*)
  then show *?case*
    using *En0* **by** *blast*
next
  case (*EsStep t t′*)
  then show *?case*
    using *step-eval-1* **by** *blast*
next
  case (*EsTrans t t′ t′′*)
  then show *?case*
    using *eval-n-plus-m* **by** *blast*
qed

lemma *can-eval-suc-inot-normal-form*: $[\![\ t \mapsto\widehat{\ } n\ t';\ n > 0\ ]\!] \Longrightarrow \neg(is\text{-}normal\text{-}form\ t)$
 by (*metis Suc-eq-plus1-left eval-1-step exist-mid-step is-normal-formI less-numeral-extra*(*3*) *not0-implies-Suc*)

8 3.5.11" deterministic of eval n

theorem *eval-n-deterministic*: $[\![\ t \mapsto\hat{}n\ t';\ t \mapsto\hat{}n\ t''\ ]\!] \Longrightarrow t' = t''$
proof (*induct n arbitrary*: $t\ t'\ t''$)
 case *0*
 then show *?case*
  using *eval-0-refl* **by** *blast*
next
 case (*Suc n*)
 assume $t \mapsto\hat{}(Suc\ n)\ t'$
 with *exist-first-step* **obtain** tp' **where** $tp'n{:}t \mapsto\hat{}n\ tp'$ **and** $tp'next{:}tp' \mapsto t'$ **by** *blast*
 assume $t \mapsto\hat{}(Suc\ n)\ t''$
 with *exist-first-step* **obtain** tp'' **where** $tp''n{:}t \mapsto\hat{}n\ tp''$ **and** $tp''next{:}tp'' \mapsto t''$ **by** *blast*
 have $eq{:}tp' = tp''$ **using** $tp'n\ tp''n\ Suc(1)$ **by** *blast*
 show $t' = t''$
  using $tp'next\ tp''next\ eq$ *eval-deterministic* **by** *simp*
qed

lemma *eval-mid-isnot-normal-form*: $[\![\ t \mapsto\hat{}n\ t';$ *is-normal-form* $t';\ t \mapsto\hat{}m\ t'';\ m < n\ ]\!] \Longrightarrow \neg($*is-normal-form* $t'')$
proof −
 assume $t \mapsto\hat{}n\ t'$ **and** $mn{:}\ m < n$
 then obtain t''' **where** $pre{:}t \mapsto\hat{}m\ t'''$ **and** $post{:}t''' \mapsto\hat{}(n-m)\ t'$ **using** *exist-mid-step2* **by** *blast*
 have *nv*: $\neg($*is-normal-form* $t''')$ **apply** (*rule can-eval-suc-inot-normal-form*[*OF post*]) **using** *mn* **by** *arith*
 assume $t \mapsto\hat{}m\ t''$ **hence** $t'' = t'''$ **using** *pre eval-n-deterministic* **by** *blast*
 with *nv*
 show $\neg($*is-normal-form* $t'')$ **by** *simp*
qed
lemma *eval-mid-isnot-normal-form2*: $[\![\ t \mapsto\hat{}n\ t';$ *is-normal-form* $t';\ t \mapsto\hat{}m\ t'';$ *is-normal-form* $t''\ ]\!] \Longrightarrow \neg(m < n)$
 using *eval-mid-isnot-normal-form* **by** *blast*

lemma *eval-n-normal-exist-only-one*: $[\![\ t \mapsto\hat{}n\ t';$ *is-normal-form* $t';\ t \mapsto\hat{}m\ t'';$ *is-normal-form* $t''\ ]\!] \Longrightarrow n = m$
 using *eval-mid-isnot-normal-form2*[**where** $n{=}n$] *eval-mid-isnot-normal-form2*[**where** $n{=}m$] *nat-neq-iff* **by** *blast*

lemma *eval-n-deterministic-on-normal-form*: $[\![\ t \mapsto\hat{}n\ t';$ *is-normal-form* $t';\ t \mapsto\hat{}m\ t'';$ *is-normal-form* $t''\ ]\!] \Longrightarrow t' = t''$
 using *eval-n-normal-exist-only-one eval-n-deterministic* **by** *blast*

9 3.5.11 deterministic of eval*

theorem *eval-star-deterministic*: $[\![\ t \mapsto* t';$ *is-normal-form* $t';\ t \mapsto* t'';$ *is-normal-form* $t''\ ]\!] \Longrightarrow t' = t''$

apply (*drule eval-star-eval-n*, *drule eval-star-eval-n*)
using *eval-n-deterministic-on-normal-form* **by** *blast*

10 3.5.12 eval* halt

primrec *size* :: $t \Rightarrow nat$ **where**
size-true: *size true* = *1*
| *size-false*: *size false* = *1*
| *size-if*: *size* (*If t1 Then t2 Else t3*) = *1* + *size t1* + *size t2* + *size t3*

lemma *normal-form-size-is-one*: *is-normal-form t* $\Longrightarrow$ *size t* = *1*
proof (*cases t*)
case *true*
then show *?thesis* **by** *simp*
next
case *false*
then show *?thesis* **by** *simp*
next
assume *tv*: *is-normal-form t*
case (*Cond x31 x32 x33*)
then show *?thesis* **using** *tv if-isnot-normal-form* **by** *simp*
qed

lemma *can-eval-size-two-or-more*: *size t* > *1* $\Longrightarrow \exists t'. t \mapsto t'$
by (*metis is-normal-form-def less-numeral-extra*(*4*) *normal-form-size-is-one*)

lemma *size-eval-mono*: $t \mapsto t' \Longrightarrow$ *size t* > *size t′*
proof (*induct rule*: *eval.induct*)
case (*E-IfTrue t2 t3*)
then show *?case* **using** *size-if* **by** *arith*
next
case (*E-IfFalse t2 t3*)
then show *?case* **using** *size-if* **by** *arith*
next
case (*E-If t1 t1′ t2 t3*)
then show *?case* **using** *size-if* **by** *arith*
qed

theorem *eval-star-stop*: $\exists t'. t \mapsto* t' \wedge$ *is-normal-form t′*
proof (*induct t*)
case *true*
then show *?case* **by** *blast*
next
case *false*
then show *?case* **by** *blast*
next
case (*Cond t1 t2 t3*)
obtain *t1′* **where** *t1t1′*: *t1* $\mapsto*$ *t1′* **and** *is-normal-form t1′* **using** *Cond.hyps*(*1*)

by *blast*
hence *t1-true-false*: $t1' = true \lor t1' = false$ **using** *normal-form-is-value is-value-def*
by *simp*
hence *If t1 Then t2 Else t3* $\mapsto*$ *t2* $\lor$ *If t1 Then t2 Else t3* $\mapsto*$ *t3*
using *eval-star-IfTrue eval-star-IfFalse t1t1'* **by** *blast*
with *Cond.hyps(2) Cond.hyps(3) eval-star-trans3*
show $\exists t'$. *If t1 Then t2 Else t3* $\mapsto*$ $t' \land$ *is-normal-form* t' **by** *blast*
qed

end