CS 231 : Types and Programming Languages
Homework #2

Collaboration : Avneet Oberoi & Ronak Sumbaly

October 14, 2015

Citations: Questions discussed with Sharavani Senapathy and George Fang but the solutions were written separately.

Question 1

Part a.

Computation Rules

$$\frac{}{false \ \&\& \ t2 \ \rightarrow false} \qquad \text{(E-AndFalse)}$$

$$\frac{}{true \ \&\& \ v \ \rightarrow v} \qquad \text{(E-AndTrue)}$$

Congruence Rules

$$\frac{t2 \rightarrow \text{t2'}}{true \ \&\& \ t2 \ \rightarrow t2'} \qquad \text{(E-AndTStep)}$$

$$\frac{t1 \rightarrow \text{t1'}}{t1 \ \&\& \ t2 \ \rightarrow t1' \ \&\& \ t2} \qquad \text{(E-And)}$$

Part b.

Typing Rule

$$\frac{t1 : Bool \qquad t2 : Bool}{t1 \ \&\& \ t2 : \text{Bool}} \qquad \text{(T-And)}$$

Part c.

Progress Theorem

If `t`:`T`, then either `t` is a value or there exists some term `t`' such that `t` $\rightarrow$ `t`'.

Proof

Using induction on derivation of `t`:`T`, we proceed as follows

Induction Hypothesis : If `t0`:`T0` and `t0`:`T0` is a sub-derivation of `t`:`T` then either `t0` is a value or there exists some term `t0`' such that `t0` $\rightarrow$ `t0`'.

Canonical Form Lemma: If v:Bool then v is either true or false. If v:Int then v is a number.

Performing case analysis on the last rule of derivation of t:T,

Case T-And

From the rule we know that t = t1 && t2, t1:Bool, t2:Bool and t:Bool.

By applying the *induction hypothesis* twice we get, t1 is either a value or it steps ,and t2 is either a value or it steps.

If t1 is a value, then as per *canonical form lemma*, since t1:Bool, t1 is either true or false. Performing induction on the values of t1

a. If t1=true, by using the evaluation rule E-AndTrue, t= *true* && $t2$ and t $\rightarrow$ $t2'$. Hence we have showed that t steps.

b. If t1=false, by using evaluation rule E-AndFalse, t= $false$ && $t2$ and t $\rightarrow$ $false$. Hence we have showed that t steps.

c. If t1 steps to t1' then for the term t , by using evaluation rule E-And, t = t1' && t2 , thus t $\rightarrow$ t1'. Hence we have showed that t steps.

Hence from above proof we can say that the Progress Theorem holds for T-And.

Part d.

Preservation Theorem

If t:T and t $\rightarrow$ t' then t':T.

Proof

Using induction on derivation of t:T we proceed as follows,

Induction Hypothesis : If t0:T0 and t0 $\rightarrow$ t0',then t0':T0 such that t0:T0 is a sub-derivation of t:T.

If the last rule in the derivation of t $\rightarrow$ t' is T-And then, t = t1 && t2 , t1:Bool , t2:Bool, t:Bool. Thus the following evaluation rules can be applied to t,

1. E-AndTrue: Given t = true && v and t $\rightarrow$ v, thus t1:Bool, t2:Bool, we get t' = v and t':Bool by using the T-And. Hence t' and t have the same type Bool.

2. E-AndFalse: Given t = true && false and t $\rightarrow$ false, we get t1:Bool, t2:Bool, thus t' = false and t':Bool by T-False. Hence t' and t have the same type Bool.

3. E-AndTStep: Given t = true && t2 and t $\rightarrow$ t2, we get t1:Bool, t2:Bool, thus t' = t2'. By Induction Hypothesis as t2 $\rightarrow$ t2' and t2:T, thus t2':T. Hence t' and t have the same type Bool.

5 E-And: Given t = t1 && t2 and t $\rightarrow$ t1' && t2, with t1:Bool, t2:Bool, and t' = t1' && t2. By Induction Hypothesis as t1 $\rightarrow$ t1' and t1:T, thus t1':T. Hence t' and t have the same type Bool.

Question 2

Part a.

Syntactic Sugar form of the operational semantics for `t1` && `t2`

$$\frac{}{t1 \ \&\& \ t2 \ \rightarrow \text{if } t1 \text{ then } t2 \text{ else } false} \quad \text{(E-AND-SUGAR)}$$

Part b.

Considering an eager version of &&, in which both operands are evaluated (in order from left to right) before producing the overall value of the term, this form of version cannot be represented using any of the available term present in language. Hence this version is **NOT** syntactic sugar.

Question 3

Check whether Progress and Preservation theorem holds for the following changes in language of booleans and integers

a. Remove the rule E-IfFalse

   i. Progress: Invalidates. Counterexample - `if false then true else false` doesn't satisfy the progress theorem because even though it is a well typed expression of the form T-If but it is neither a value nor does it step to `t`' as there is no E-IfFalse rule.

   ii. Preservation Validates. Since the assumption required to prove preservation theorem is that the term can step, removing this rule will not invalidate the theorem.

b. Add axiom `0` of type `Bool`

   i. Progress: Invalidates. Counterexample - `if 0 then true else false` doesn't satisfy the progress theorem because it breaks the canonical forms lemma which requires this expression to take a step with either E-IfTrue or E-IfFalse.

   ii. Preservation Invalidates. Counterexample - 20 + (-20) $\rightarrow$ 0. The example steps to a type `Bool` but it violates the type checking rule of T-Plus. Hence we conclude that it will invalidate the theorem.

c. Add axiom `if` t_1 `then` t_2 `else` $t_3 \rightarrow t_2$

   i. Progress: Validates. Since the term is well typed and steps, the progress theorem holds.

   ii. Preservation Validates. Since the term is well typed and it steps to t_2 which is of the same type by T-If, the preservation theorem holds.

d. Add rules for addition of booleans.

   i. Progress: Invalidates. Counterexample - `false + true` doesn't satisfy the progress theorem because even though it is a well typed expression of the new form but it is neither a value nor does it step to `t`' as there is no stepping rule.

   ii. Preservation Invalidates. Counterexample - `true + true` $\rightarrow$ `true`. The example steps to a term of type `Bool` which contradicts the new rule which states that it should be of type `Int`. Hence the preservation theorem doesn't hold.

e. Add rule for `if` guard being of type `integer`

i. Progress: Invalidates. Counterexample - `if 1 then true else false` doesn't satisfy the progress theorem because even though it is a well typed expression of the new form but it is neither a value nor does it step to `t`' as there are no stepping rules for other integer values.

ii. Preservation Validates. Since the assumption required to prove preservation theorem is that the term of type `T` can step, for $\mathtt{t}_1 = 0$ the term steps to $\mathtt{t}_2$ which is of type `T` but for rest of the `integer` values the term does not step and hence the assumption is contradicted, hence the preservation theorem isn't affected.

Question 4

Part a.

Reverse Progress Theorem

Theorem If `t`':`T`, then there exists some term `t` such that `t` $\rightarrow$ `t`'.

Analysis

The theorem states that for a term `t0`', if `t`' is a well typed term there exists some term `t` such that `t` steps to `t`'.

Induction Hypothesis : If t0':T0, then there exists some term `t0` $\rightarrow$ `t0`'.

Performing induction on the derivation, we proceed as follows.

Case 1. T-True : Since this rule doesn't step as it is a value. Hence it is vacuous for the theorem.

Case 2. T-False : Since this rule doesn't step as it is a value. Hence it is vacuous for the theorem.

Case 3. T-Num : Since this rule doesn't step as it is a value. Hence it is vacuous for the theorem.

Case 4. T-If : Then `t'` has the form `if t1 then t2 else t3`, `t1` has type `Bool`, and both `t2` and `t3` have the type `T`. Applying case analysis on the last rule of derivation T-If:.

Case a. E-IfTrue - Here t = if true then t2 else t3. From this rule we get t1 = true and t' = t2 and since t2:T hence t':T. Hence there there exists a t such that t steps to t'.

Case b. E-IfFalse - Here t = if false then t2 else t3. From this rule we get t1 = false and t' = t3 and since t3:T hence t':T. Hence there there exists a t such that t steps to t'.

Case c. E-If - Here t = if t1 then t2 else t3. Where t1 $\rightarrow$ t1' , t1:Bool and t' = if t1' then t2 and t3 matches the condition for T-If. $\therefore$ t1':Bool. By using Induction Hypothesis since t1':Bool is a sub-derivation of t1:Bool we can conclude that the original t steps to t'. Hence there there exists a t such that t steps to t'.

Case 5. T-Plus : Then `t'` has the form `t1 + t2` and both `t1` and `t2` have type `Int`. Applying case analysis on the last rule of derivation T-Plus

Case a. E-Plus1 Here t = t1 + t2 and t' = t1' + t2 where t1:Int and t2:Int. Since t' matches the T-Plus rule we can say that t1':Int and t2:Int. Since we know that t1':Int is a subderivation of t1:Int by our induction hypothesis we can conclude that there exist a t such that t steps to t'

Case b. E-Plus2 Similar to above case with respect to t2.

Case c. E-PlusRed: Here t = n1 + n2 where n1:Int and n2:Int and t:Int. and t' = n. Using T-Num we get that t':Int. Now since t':Int is a subderivation of t:Int we conclude that there exists a t such that t steps to t'.

Case 6. T-GT : By a similar argument replacing occurrence of "Plus" with "GT" and + with >.

Part b.

Reverse Preservation Theorem

Theorem If `t'`:`T` and `t` $\rightarrow$ `t'`, then `t`:`T`.

Analysis
The theorem states that for a term `t'`, if `t'` is a well typed term and `t` $\rightarrow$ `t'` then the term `t` should also have the same type as that of `t'`.

The above theorem *doesn't hold* for the following counterexample :

$$if\ \texttt{true}\ then\ \texttt{integer}\ else\ \texttt{false} \rightarrow integer$$

For the above example we know that `t'`:`Int` and `t` $\rightarrow$ `t'` where `t` $=$ *if* `true` *then* `integer` *else* `false`. By the T-IF rule the type of $\texttt{t}_2$ and $\texttt{t}_3$ should be same, which isn't satisfied by the above term `t` making it a ill typed term. Hence we can conclude that if `t'`:`T` and `t` $\rightarrow$ `t'`, then `t` may not be of the type `T`, $\therefore$ the theorem doesn't hold.

Question 7

Prove
$((function\ \mathrm{x}\ \rightarrow\ \mathrm{x}\ \ \mathrm{x}\ )(function\ \mathrm{x}\ \rightarrow\ \mathrm{x}\ \ \mathrm{x}\ )) \rightarrow^* t$ then,
$t = ((function\ \mathrm{x}\ \rightarrow\ \mathrm{x}\ \ \mathrm{x}\ )(function\ \mathrm{x}\ \rightarrow\ \mathrm{x}\ \ \mathrm{x}\ ))$.

Proof
Using **induction on derivation** on $((function\ \mathrm{x}\ \rightarrow\ \mathrm{x}\ \ \mathrm{x}\ )(function\ \mathrm{x}\ \rightarrow\ \mathrm{x}\ \ \mathrm{x}\ )) \rightarrow^* t$, we proceed as follows,

Let $((function\ \mathrm{x}\ \rightarrow\ \mathrm{x}\ \ \mathrm{x}\ )(function\ \mathrm{x}\ \rightarrow\ \mathrm{x}\ \ \mathrm{x}\ )) \rightarrow^* t = ((\lambda x.x\ x)(\lambda x.x\ x)) \rightarrow^* t$

Induction Hypothesis
For derivation of the form $((\lambda x.x\ x)(\lambda x.x\ x)) \rightarrow^* t_0$, and $((\lambda x.x\ x)(\lambda x.x\ x)) \rightarrow^* t_0$ is a sub-derivation of $((\lambda x.x\ x)(\lambda x.x\ x)) \rightarrow^* t$ then $t_0 = ((\lambda x.x\ x)(\lambda x.x\ x))$.

Performing analysis on the last rule in derivation of $((\lambda x.x\ x)(\lambda x.x\ x)) \rightarrow^* t$ we have 3 cases,

Case 1. E-REFL
Since the rule states that $t \rightarrow^* t$ then the proof is trivially true.

Case 2. E-STEP
Applying analysis on the last rule in the derivation of $t \rightarrow^* t'$ we again have 3 cases,

a. E-APP1 : From this rule we know that `t` is of the form ($\texttt{t}_1$ $\texttt{t}_2$) and $\texttt{t}_1 \rightarrow \texttt{t'}_1$, which is a contradiction to the assumption.

b. E-APP2 : From this rule we know that `t` is of the form ($\texttt{v}_1$ $\texttt{t}_2$) and $\texttt{t}_2 \rightarrow \texttt{t'}_2$, which is again a contradiction to the assumption.

c. E-APPBETA : From this rule we know that `t` is of the shorthand form $(\lambda x.t_0)v$ where t_0 is $x\ x$ and v is $(\lambda x.x\ x)$. Substituting the value of v in all occurrences of t_0 in the function x we get $((\lambda x.x\ x)(\lambda x.x\ x))$, which is the original desired value of `t`.

Case 3. E-TRANS
From this rule we know that $t \rightarrow^* t''$ and $t'' \rightarrow^* t'$. By the induction hypothesis we can say that since the derivation $t \rightarrow^* t''$ is a sub derivation of $t \rightarrow^* t_0$ we can conclude that `t"` $=$ `t` $= ((\lambda x.x\ x)(\lambda x.x\ x))$. Again applying the induction hypothesis we can say that since the derivation $t'' \rightarrow^* t'$ is a sub derivation of $t \rightarrow^* t''$ we can conclude that `t'` $=$ `t"` $= ((\lambda x.x\ x)(\lambda x.x\ x))$. Since the rule steps to `t'`, we can conclude that `t` $=$ `t'` which is the desired result.