Assignment 3
Nontermination
Sample Solution

15-814: Types and Programming Languages
Farzaneh Derakhshan
Due Tuesday, October 1, 2019

Task 1 (L6.2, 15 points) Consider adding a new expression ⊥ to our call-by-value language (with functions and Booleans) with the following evaluation and typing rules:

\[
\begin{align*}
\bot & \mapsto \bot \\
& \text{step/bot} \\
\Gamma & \vdash \bot : \tau
\end{align*}
\]

We do not change our notion of value, that is, ⊥ is not a value.

1. Does preservation (Theorem L6.2) still hold? If not, provide a counterexample. If yes, show how the proof has to be modified to account for the new form of expression.

Yes, we need to add a case for step/bot:

\[
\bot \mapsto \bot
\]

Where, \( e = e' = \bot \) and Since \( \bot \) reduces to itself, \( \cdot \vdash e' : \tau \) is trivially true by assumption.

2. Does the canonical forms theorem (L6.4) still hold? If not, provide a counterexample. If yes, show how the proof has to be modified to account for the new form of expression.

Yes, since \( \bot \) is not a value, there is no extra case to consider, and the proof is not changed.

3. Does progress (Theorem L6.3) still hold? If not, provide a counterexample. If yes, show how the proof has to be modified to account for the new form of expression.
Yes, we need to consider a new case:

\[ \vdash \bot : \tau \]

where \( e = \bot \). By step/bot rule we know that \( \bot \) can always step to itself. So, for \( e' = \bot \), we have \( \bot \rightarrow \bot \).

Once we have nonterminating computation, we sometimes compare expressions using Kleene equality: \( e_1 \) and \( e_2 \) are Kleene equal (\( e_1 \simeq e_2 \)) if they evaluate to the same value, or they both diverge (do not compute to a value). Since we assume we cannot observe functions, we can further restrict this definition: For \( \vdash e_1 : \text{bool} \) and \( \vdash e_2 : \text{bool} \) we write \( e_1 \simeq e_2 \) iff \( e_1 \rightarrow^* v \) and \( e_2 \rightarrow^* v \) for some value \( v \).

4. Give an example of two closed terms \( e_1 \) and \( e_2 \) of type bool such that \( e_1 \simeq e_2 \) but not \( e_1 =_\beta e_2 \), or indicate that no such example exists (no proof needed in either case).

\[
\begin{array}{ccc}
\vdash \bot : \text{bool} & \vdash \text{false} : \text{bool} \\
\vdash \bot \rightarrow \text{false} & \vdash \text{true} : \text{bool} & \vdash \text{false} : \text{bool} \\
\end{array}
\]

Similarly, we have \( \vdash \bot \rightarrow \text{true} : \text{bool} \). Neither \( \bot \rightarrow \text{false} \) nor \( \bot \rightarrow \text{true} \) will eventually reduce to a value, since \( \bot \) is not a value, and can never reduces to anything other than itself. So, \( \bot \rightarrow \text{false} \simeq \bot \rightarrow \text{true} \). But \( \bot \rightarrow \text{false} \neq_\beta \bot \rightarrow \text{true} \).

Task 2 (L6.3, 15 points) In our call-by-value language with functions, Booleans, and \( \bot \) (see Task 1) consider the following specification of \texttt{or}, sometimes called “short-circuit or”:

\[
\begin{align*}
\text{or } \text{true } e & \simeq \text{true} \\
\text{or } \text{false } e & \simeq e
\end{align*}
\]

where \( e_1 \simeq e_2 \) is Kleene equality from Task 1.

- We cannot define a function \texttt{or} : \texttt{bool} \rightarrow (\texttt{bool} \rightarrow \texttt{bool}) with this behavior. Prove that it is indeed impossible.

Assume for the sake of contradiction, that we can define such function \( \vdash \text{or} : \text{bool} \rightarrow (\text{bool} \rightarrow \text{bool}) \). Then \( \text{or } \text{true } \bot \rightarrow^* \text{true} \), since they are Kleene equal and \text{true} \val. By induction on the structure of the derivation of \( \text{or } \text{true } \bot \rightarrow^* \text{true} \), we have \( \text{or } \rightarrow^* \lambda x. e_1 \) for some \( x \) and \( e_1 \). And by step/beta/val rule, \( (\lambda x. e_1) \text{true } \bot \rightarrow^* (\text{true } \lambda y. e_2) \bot \). Again, by Kleene equality, we know that \( (\text{true } \lambda y. e_2) \bot \rightarrow^* \text{true} \). By induction on the structure of the derivation, \( (\text{true } \lambda y. e_2) \rightarrow^* \lambda y. e_2 \) for some \( y \) and \( e_2 \). But \( (\lambda y. e_2) \bot \) cannot take any step other than \( (\lambda y. e_2) \bot \rightarrow^* (\lambda y. e_2) \bot \), since \( \bot \) is not a value. This forms a contradiction.
- Show how to translate an expression \( or \ e_1 \ e_2 \) into our language so that it satisfies the specification, and verify the given equalities by calculation.

\[
\begin{align*}
or\ e_1\ e_2 & \triangleq \text{if } e_1 \text{ true } e_2. \text{ So } \Gamma \vdash or\ e_1\ e_2 : \text{ bool } \text{ iff } \Gamma \vdash e_1 : \text{ bool } \text{ and } \Gamma \vdash e_2 : \text{ bool }.
\end{align*}
\]

Also

\[
\begin{align*}
\text{if true true } e & \mapsto \text{ true by step/if/true} \\
\text{if false true } e & \mapsto e \text{ by step/if/false}
\end{align*}
\]

**Task 3 (L6.4, 30 points)** In our call-by-value language with functions, Booleans, and \( \perp \) (see Task 1) consider the following specification of \( por \), sometimes called “parallel or”:

\[
\begin{align*}
por \text{ true } e & \simeq \text{ true} \\
por e \text{ true} & \simeq \text{ true} \\
por \text{ false false} & \simeq \text{ false}
\end{align*}
\]

where \( e_1 \simeq e_2 \) is Kleene equality as in Tasks 1 and 2.

1. We cannot define a function \( por : \text{ bool } \rightarrow (\text{ bool } \rightarrow \text{ bool}) \) in our language with this behavior. Prove that it is indeed impossible.

For the same reason as in Task 3, neither \( por \text{ true } \perp \) nor \( por \perp \text{ true} \) reduces to a value.

2. We also cannot translate expressions \( por \ e_1 \ e_2 \) into our language so that the result satisfies the given properties (which you do not need to prove). Instead consider adding a new primitive form of expression \( por \ e_1 \ e_2 \) to our language.

(a) Give one or more typing rules \( por \ e_1 \ e_2 \).

\[
\frac{\Gamma \vdash e_1 : \text{ bool } \quad \Gamma \vdash e_2 : \text{ bool}}{\Gamma \vdash por\ e_1\ e_2 : \text{ bool}} \text{ tp/por}
\]

(b) Provide one or more evaluation rules for \( por \ e_1 \ e_2 \) so that it satisfies the given specification and, furthermore, such that preservation, canonical forms, and progress continue to hold.

\[
\begin{align*}
\frac{e_1 \mapsto e'_1}{por\ e_1\ e_2 \mapsto por\ e'_1\ e_2} \text{ step/por1} \\
\frac{e_2 \mapsto e'_2}{por\ e_1\ e_2 \mapsto por\ e_1\ e'_2} \text{ step/por2} \\
\frac{\text{por true } e \mapsto \text{ true}}{\text{step/por/t1}} \\
\frac{\text{por } e \text{ true} \mapsto \text{ true}}{\text{step/por/t2}} \\
\frac{\text{por false false} \mapsto \text{ false}}{\text{step/por/f2}}
\end{align*}
\]
(c) Show the new case(s) in the preservation theorem. 

We need to prove five new cases for evaluation of \( \text{por} \) term:

\[
\begin{align*}
\text{Case:} & \quad e_1 \mapsto e'_1 \\
\text{por} e_1 e_2 & \mapsto \text{por} e'_1 e_2 \quad \text{step/por1}
\end{align*}
\]

where \( e = \text{por} e_1 e_2 \) and \( e' = \text{por} e'_1 e_2 \).

\[
\begin{align*}
\cdot \vdash \text{por} e_1 e_2 &: \tau \\
\tau &= \text{bool}, \cdot \vdash e_1 : \text{bool}, \cdot \vdash e_2 : \text{bool} \\
\cdot \vdash e'_1 &: \text{bool} \\
\cdot \vdash \text{por} e'_1 e_2 &: \text{bool}
\end{align*}
\]

Assumption \quad By inversion \quad By ind.hyp. \quad By rule \text{tp/por}

\[
\begin{align*}
\text{Case:} & \quad e_2 \mapsto e'_2 \\
\text{por} e_1 e_2 & \mapsto \text{por} e_1, e'_2 \quad \text{step/por2}
\end{align*}
\]

where \( e = \text{por} e_1 e_2 \) and \( e' = \text{por} e_1, e'_2 \).

\[
\begin{align*}
\cdot \vdash \text{por} e_1 e_2 &: \tau \\
\tau &= \text{bool}, \cdot \vdash e_1 : \text{bool}, \cdot \vdash e_2 : \text{bool} \\
\cdot \vdash e'_2 &: \text{bool} \\
\cdot \vdash \text{por} e_1 e'_2 &: \text{bool}
\end{align*}
\]

Assumption \quad By inversion \quad By ind.hyp. \quad By rule \text{tp/por}

\[
\begin{align*}
\text{Case:} & \quad \text{por true} e_2 \mapsto \text{true} \quad \text{step/por/t1}
\end{align*}
\]

where \( e = \text{por true} e_2 \) and \( e' = \text{true} \).

\[
\begin{align*}
\cdot \vdash \text{por true} e_2 &: \tau \\
\tau &= \text{bool}, \cdot \vdash \text{true} : \text{bool}, \cdot \vdash e_2 : \text{bool}
\end{align*}
\]

Assumption \quad By inversion

The two other cases for \text{step/por/t2} and \text{step/por/f2} are similar to previous cases.

(d) Show the new case(s) in the progress theorem.

We have one case to consider:

\[
\begin{align*}
\cdot \vdash e_1 &: \text{bool} \\
\cdot \vdash e_2 &: \text{bool} \\
\cdot \vdash \text{por} e_1 e_2 &: \text{bool}
\end{align*}
\]

\text{tp/por} 

Either \( e_1 \mapsto e'_1 \) for some \( e'_1 \) or \( e_1 \text{ val} \) \quad By ind.hyp.

\[ e_1 \mapsto e'_1 \]
\[ e = \text{por} e_1 e_2 \mapsto \text{por} e'_1 e_2 \]

Subcase \quad By rule \text{step/por1}

\[ e_1 \text{ val} \]

Either \( e_1 = \text{true} \) or \( e_1 = \text{false} \) \quad By inversion

\[ e_1 = \text{true} \] \quad Sub^2 case
(e) Do your rules satisfy single-step determinacy (see Exercise L6.1)? If not, provide a counterexample. If yes, just indicate that it is the case (you do not need to prove it).

Counterexample: $\text{por } \bot \text{ true } \mapsto \text{true}$ by step/por/t2, and $\text{por } \bot \text{ true } \mapsto \text{por } \bot \text{ true }$ by step/por/1.