1 STLC with Pairs

We want to add types Pairs to the Simply Typed Lambda Calculus with Booleans that we studied in the book and lectures. A Pair is the equivalent of Scala's 'Tuple2'. You can find a longer introduction to Pairs in the book, Section 11.6. For the sake of the exercise, we stay at a dry level of summarizing the syntactic forms, evaluation rules and typing rules:

$$
\begin{array}{lcll}
\mathtt{t} & ::= & & \textbf{terms}: \\
 & | & \mathtt{x} & \text{variable} \\
 & | & \lambda\mathtt{x}\mathtt{:}\mathtt{T}\mathtt{.}\mathtt{t} & \text{abstraction} \\
 & | & \mathtt{t}\ \mathtt{t} & \text{application} \\
 & | & \mathtt{true} \mid \mathtt{false} & \text{boolen values} \\
 & | & \mathtt{if}\ \mathtt{t}\ \mathtt{then}\ \mathtt{t}\ \mathtt{else}\ \mathtt{t} & \text{conditional} \\
 & | & \{\mathtt{t},\ \mathtt{t}\} & \text{pair} \\
 & | & \mathtt{t.1} \mid \mathtt{t.2} & \text{projection} \\
 & & & \\
\mathtt{v} & ::= & & \textbf{values}: \\
 & | & \lambda\mathtt{x}\mathtt{:}\mathtt{T}\mathtt{.}\mathtt{t} & \text{abstraction} \\
 & | & \mathtt{true} \mid \mathtt{false} & \text{boolen values} \\
 & | & \{\mathtt{v},\ \mathtt{v}\} & \text{pair value} \\
\mathtt{T} & ::= & & \textbf{types}: \\
 & | & \mathtt{T} \to \mathtt{T} & \text{function type} \\
 & | & \mathtt{Bool} & \text{boolean type} \\
 & | & \mathtt{T} \times \mathtt{T} & \text{pair type}
\end{array}
$$

Small-step reduction rules:

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{t}_1\ \mathtt{t}_2 \longrightarrow \mathtt{t}_1'\ \mathtt{t}_2} \quad \text{(E-App1)}$$

$$\frac{\mathtt{t}_2 \longrightarrow \mathtt{t}_2'}{\mathtt{v}_1\ \mathtt{t}_2 \longrightarrow \mathtt{v}_1\ \mathtt{t}_2'} \quad \text{(E-App2)}$$

$$(\lambda\mathtt{x}\mathtt{:}\mathtt{T}\mathtt{.}\mathtt{t}_{12})\ \mathtt{v}_2 \longrightarrow [\mathtt{x} \mapsto \mathtt{v}_2]\mathtt{t}_{12} \quad \text{(E-AppAbs)}$$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\{\mathtt{t}_1,\ \mathtt{t}_2\} \longrightarrow \{\mathtt{t}_1',\ \mathtt{t}_2\}} \quad \text{(E-Pair1)}$$

$$\frac{\mathtt{t}_2 \longrightarrow \mathtt{t}_2'}{\{\mathtt{v}_1,\ \mathtt{t}_2\} \longrightarrow \{\mathtt{v}_1,\ \mathtt{t}_2'\}} \quad \text{(E-Pair2)}$$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{t}_1\mathtt{.1} \longrightarrow \mathtt{t}_1'\mathtt{.1}} \quad \text{(E-Proj1)}$$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{t}_1\mathtt{.2} \longrightarrow \mathtt{t}_1'\mathtt{.2}} \quad \text{(E-Proj2)}$$

$$\{\mathtt{v}_1,\ \mathtt{v}_2\}\mathtt{.1} \longrightarrow \mathtt{v}_1 \quad \text{(E-PairBeta1)}$$

$$\{\mathtt{v}_1,\ \mathtt{v}_2\}\mathtt{.2} \longrightarrow \mathtt{v}_2 \quad \text{(E-PairBeta2)}$$

Typing rules:

$$\frac{\mathtt{x}\mathtt{:}\mathtt{T} \in \Gamma}{\Gamma \vdash \mathtt{x}\mathtt{:}\mathtt{T}} \quad \text{(T-Var)}$$

$$\frac{\Gamma, \mathtt{x}\mathtt{:}\mathtt{T}_1 \vdash \mathtt{t}_2\mathtt{:}\mathtt{T}_2}{\Gamma \vdash (\lambda\mathtt{x}\mathtt{:}\mathtt{T}_1\mathtt{.}\mathtt{t}_2)\mathtt{:}\mathtt{T}_1 \to \mathtt{T}_2} \quad \text{(T-Abs)}$$

$$\frac{\Gamma \vdash \mathtt{t}_1\mathtt{:}\mathtt{T}_1 \to \mathtt{T}_2 \qquad \Gamma \vdash \mathtt{t}_2\mathtt{:}\mathtt{T}_1}{\Gamma \vdash \mathtt{t}_1\ \mathtt{t}_2\mathtt{:}\mathtt{T}_2} \quad \text{(T-App)}$$

$$\Gamma \vdash \mathtt{true}\mathtt{:}\mathtt{Bool} \quad \text{(T-True)}$$

$$\Gamma \vdash \mathtt{false}\mathtt{:}\mathtt{Bool} \quad \text{(T-False)}$$

$$\frac{\Gamma \vdash \mathtt{t}_1\mathtt{:}\mathtt{Bool} \qquad \Gamma \vdash \mathtt{t}_2\mathtt{:}\mathtt{T} \qquad \Gamma \vdash \mathtt{t}_3\mathtt{:}\mathtt{T}}{\Gamma \vdash \mathtt{if}\ \mathtt{t}_1\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3\mathtt{:}\mathtt{T}} \quad \text{(T-If)}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_1 \qquad \Gamma \vdash \mathtt{t}_2 : \mathtt{T}_2}{\Gamma \vdash \{\mathtt{t}_1,\ \mathtt{t}_2\} : \mathtt{T}_1 \times \mathtt{T}_2} \quad (\text{T-Pair}) \qquad\qquad \frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_1 \times \mathtt{T}_2}{\Gamma \vdash \mathtt{t}_1.2 : \mathtt{T}_2} \quad (\text{T-Proj2})$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_1 \times \mathtt{T}_2}{\Gamma \vdash \mathtt{t}_1.1 : \mathtt{T}_1} \quad (\text{T-Proj1})$$

1. State the Canonical Forms lemma for this new system, without proving it.
2. Extend the proof of Progress of STLC with Booleans to deal with Pairs. Only handle the cases that are specific to Pairs.
3. Likewise, extend the proof of Preservation.

1.1 Solution

1.1.1 Canonical Forms lemma

- If $\mathtt{v}$ is a value and $\Gamma \vdash \mathtt{v} : \mathtt{Bool}$, then either $\mathtt{v} = \mathtt{true}$ or $\mathtt{v} = \mathtt{false}$.
- If $\mathtt{v}$ is a value and $\Gamma \vdash \mathtt{v} : \mathtt{T}_1 \to \mathtt{T}_2$, then $\mathtt{v} = \lambda \mathtt{x} : \mathtt{T}_1 . \mathtt{t}_{12}$.
- If $\mathtt{v}$ is a value and $\Gamma \vdash \mathtt{v} : \mathtt{T}_1 \times \mathtt{T}_2$, then $\mathtt{v} = \{\mathtt{v}_1,\ \mathtt{v}_2\}$.

Alternative: replace Γ by $\emptyset$.

Additionally, we can say something about the types of the nested terms, but that is typically not necessary to prove progress.

1.1.2 Progress

If $\emptyset \vdash \mathtt{t} : \mathtt{T}$, then either $\mathtt{t}$ is a value or $\mathtt{t} \longrightarrow \mathtt{t}'$.

We extend the proof of STLC with Booleans. We add cases for the new typing rules:

Case T-Pair :

We know:

- $\mathtt{t} = \{\mathtt{t}_1,\ \mathtt{t}_2\}$
- $\emptyset \vdash \mathtt{t}_1 : \mathtt{T}_1$
- $\emptyset \vdash \mathtt{t}_2 : \mathtt{T}_2$
- $\mathtt{T} = \mathtt{T}_1 \times \mathtt{T}_2$.
- IH_1: (because we have $\emptyset \vdash \mathtt{t}_1 : \mathtt{T}_1$): either $\mathtt{t}_1$ is a value or $\mathtt{t}_1 \longrightarrow \mathtt{t}_1'$.
- IH_2: (because we have $\emptyset \vdash \mathtt{t}_2 : \mathtt{T}_2$): either $\mathtt{t}_2$ is a value or $\mathtt{t}_2 \longrightarrow \mathtt{t}_2'$.

If $\mathtt{t}_1 \longrightarrow \mathtt{t}_1'$ is a value, then E-Pair1 applies. Otherwise by IH_1, $\mathtt{t}_1$ is a value $\mathtt{v}_1$.
Then if $\mathtt{t}_2 \longrightarrow \mathtt{t}_2'$ is a value, then E-Pair2 applies. Otherwise by IH_2, $\mathtt{t}_2$ is a value $\mathtt{v}_2$.
Then $\mathtt{t} = \{\mathtt{v}_1,\ \mathtt{v}_2\}$ is a value.

Case T-Proj1

We know:

- $\mathtt{t} = \mathtt{t}_1\mathtt{.1}$
- $\emptyset \vdash \mathtt{t}_1 \mathtt{:T} \times \mathtt{T}_2$
- IH$_1$: (because we have $\emptyset \vdash \mathtt{t}_1 \mathtt{:T} \times \mathtt{T}_2$): either $\mathtt{t}_1$ is a value or $\mathtt{t}_1 \longrightarrow \mathtt{t}_1'$.

If $\mathtt{t}_1 \longrightarrow \mathtt{t}_1'$ is a value, then E-Proj1 applies. Otherwise by IH$_1$, $\mathtt{t}_1$ is a value $\mathtt{v}_1$.

By the canonical forms lemma, since $\emptyset \vdash \mathtt{v}_1 \mathtt{:T} \times \mathtt{T}_2$, we have $\mathtt{v}_1 = \{\mathtt{v}_{11}\mathtt{,}\ \mathtt{v}_{12}\}$. Then, E-PairBeta1 applies.

Case T-Proj2

Similar (with E-Proj2 and E-PairBeta2).

1.1.3 Preservation

We use the book's version of preservation, with an arbitrary Γ.

If $\Gamma \vdash \mathtt{t:T}$ and $\mathtt{t} \longrightarrow \mathtt{t}'$, then $\Gamma \vdash \mathtt{t:T}$.

We extend the proof of STLC with Booleans. We add cases for the new typing rules:

Case T-Pair :

We know:

- $\mathtt{t} = \{\mathtt{t}_1\mathtt{,}\ \mathtt{t}_2\}$
- $\Gamma \vdash \mathtt{t}_1 \mathtt{:T}_1$
- $\Gamma \vdash \mathtt{t}_2 \mathtt{:T}_2$
- $\mathtt{T} = \mathtt{T}_1 \times \mathtt{T}_2$.
- IH$_1$: (because we have $\Gamma \vdash \mathtt{t}_1 \mathtt{:T}_1$): if $\mathtt{t}_1 \longrightarrow \mathtt{t}_1'$, then $\Gamma \vdash \mathtt{t}_1' \mathtt{:T}_1$.
- IH$_2$: (because we have $\Gamma \vdash \mathtt{t}_2 \mathtt{:T}_2$): if $\mathtt{t}_2 \longrightarrow \mathtt{t}_2'$, then $\Gamma \vdash \mathtt{t}_2' \mathtt{:T}_2$.

We proceed by case analysis on the last rule used to derive $\mathtt{t} \longrightarrow \mathtt{t}'$.

Subcase E-Pair1 :

We know:

- $\mathtt{t}_1 \longrightarrow \mathtt{t}_1'$
- $\mathtt{t}' = \{\mathtt{t}_1'\mathtt{,}\ \mathtt{t}_2\}$

By IH$_1$, we have that $\Gamma \vdash \mathtt{t}_1' \mathtt{:T}_1$. Then by T-Pair (applied "downwards"), we get $\Gamma \vdash \{\mathtt{t}_1'\mathtt{,}\ \mathtt{t}_2\}\mathtt{:T}_1 \times \mathtt{T}_2$, as required.

Subcase E-Pair2 :

Similar.

Case T-Proj1

We know:

- $\mathtt{t} = \mathtt{t}_1\mathtt{.1}$

- $\Gamma \vdash \mathtt{t}_1 : \mathtt{T} \times \mathtt{T}_2$
- IH_1: (because we have $\Gamma \vdash \mathtt{t}_1 : \mathtt{T} \times \mathtt{T}_2$): if $\mathtt{t}_1 \longrightarrow \mathtt{t}_1'$, then $\Gamma \vdash \mathtt{t}_1' : \mathtt{T} \times \mathtt{T}_2$.

We proceed by case analysis on the last rule used to derive $\mathtt{t} \longrightarrow \mathtt{t}'$.

Subcase E-Proj1 :

We know:

- $\mathtt{t}_1 \longrightarrow \mathtt{t}_1'$
- $\mathtt{t}' = \mathtt{t}_1'\mathtt{.1}$

By IH_1, we have that $\Gamma \vdash \mathtt{t}_1' : \mathtt{T} \times \mathtt{T}_2$. Then by T-Proj1 (applied "downwards"), we get $\Gamma \vdash \mathtt{t}_1'\mathtt{.1} : \mathtt{T}$, as required.

Subcase E-PairBeta1 :

We know:

- $\mathtt{t}_1 = \{\mathtt{v}_1\mathtt{,}\ \mathtt{v}_2\}$
- $\mathtt{t}' = \mathtt{v}_1$

By inversion of typing on $\Gamma \vdash \{\mathtt{v}_1\mathtt{,}\ \mathtt{v}_2\} : \mathtt{T} \times \mathtt{T}_2$, we have $\Gamma \vdash \mathtt{v}_1 : \mathtt{T}$, as required.

Case T-Proj2

Similar (with subcases E-Proj2 and E-PairBeta2).

2 Hacking with the untyped call-by-value lambda calculus

In this exercise, you have to implement some operations for Church encoding of lists. There are several ways to Church encode a list, among which Church encoding based on its right fold function is more popular. As an example, an empty list (`nil`) and the `cons` construct are represented as follows in this encoding:

```
nil = λc.  λn.  n
cons = λh.  λt.  λc.  λn.  c h (t c n)
```

As another example, a list of 3 elements x, y, z is encoded as:

```
λc.  λn.  c x (c y (c z n))
```

The complete list of predefined operations can be found in the appendix, and only these operations can be used in the exercise. Define the following operations on a list:
(For explanations of the solutions, see the subsection below the questions)

1. (2 points) The *map* function which applies the given function to each element of the given list.

   ```
   map = λ f. λ l. l (λ h. λ r. cons (f h) r) nil
   ```

2. (2 points) The *length* function which returns the size of the given list. The result should be in Church encoding.

   `length = λl. l (λa. λb. scc b)` c_0

3. (2 points) The *sum* function which returns the sum of all elements of the given list. Assume all elements and the result are Church encoded numbers.

   `sum = λl. l plus` c_0

4. (2 points) The *concat* function which concatenates two input lists.

   ```
   concat = λl1. λl2. l1 cons l2
   ```

5. (2 points) The *exists* function which checks if there is any element satisfying the given predicate. The given predicate and the result should be both in Church encoding.

   ```
   exists = λl. λp. l (λa. λb. p a tru b) fls
   ```

2.1 Explanations

This task is much easier if you understand how the encoding works. We say it is the "right fold" not without accident. Basically, lists in this encoding work like partial application of the `foldRight` method – `cons 1 (cons 2 nil)` is like `List(1,2).foldRight`.

In Scala, function calls like `List(1,2).foldRight(0)(_ + _)` are equivalent to `1 + (2 + 0)` – notice how `foldRight` inserts the folding function between every element of the list, puts 0 at the end, and associates operations to the right. The same thing goes for this encoding. If we have `l = cons 1 (cons 2 nil)`, then `l f z = f 1 (f 2 z)` – observe how we basically replaced `cons` with `f` and `nil` with `z`.

Get a feeling for how this encoding works! You might see something similar during the exam.