Lists and Pairs

Examples 8

CSE 607: E-7

Title: Lists and Pairs
SML files: none
Objective: Practice with Lists and Pairs

1 Pairs

Pairs are terms of the form \((x, y)\). N-tuples are created from \(n - 1\) pairs, e.g. a 3-tuple \((x, y, z)\) is the nested pair \((x, (y, z))\). The following theorems characterize the behavior of the pair operator, “,”. Note: \(\alpha\) and \(\beta\) denote terms of any type. \(\alpha\) and \(\beta\) are type variables. Variables which are any type are polymorphic.

\[
\forall(x : \alpha \times \beta). \exists(x 1 : \alpha)(x 2 : \beta).[x = (x 1, x 2)] \quad (1)
\]

\[
\forall(x 1 : \alpha)(x 2 : \beta)(y 1 : \alpha)(y 2 : \beta).((x 1, x 2) = (y 1, y 2)) \cup (x 1 = y 1) \land (x 2 = y 2) \quad (2)
\]

The accessor functions \(\text{FST}\) and \(\text{SND}\) for pairs are described below. \(\text{FST}\) when applied to a pair returns the first element of the pair. \(\text{SND}\) when applied to a pair, returns the second element (possibly another pair) of the pair.

\[
\forall x y. \text{FST}(x, y) = x \quad (3)
\]

\[
\forall x y. \text{SND}(x, y) = y \quad (4)
\]

Given the above definitions, the theory of pairs is presented as shown below.

\[A 0. x = x\]

\[A 1. \exists(x 1 : \alpha)(x 2 : \beta).[x : (\alpha \times \beta)] = (x 1, x 2)\]

\[A 2. ((x 1, x 2) = (y 1, y 2)) \cup (x 1 = y 1) \land (x 2 = y 2)\]

\[A 3. \text{FST}(x, y) = x\]

\[A 4. \text{SND}(x, y) = y\]

Given the above theory we can prove \(\forall x.((\text{FST} x), (\text{SND} x)) = x\).
A0. $x = x$
A1. $\exists(x_1 : \alpha)(x_2 : \beta).[(x : (\alpha \times \beta)) = (x_1, x_2)]$
A2. $((x_1, x_2) = (y_1, y_2)) \supset (x_1 = y_1) \land (x_2 = y_2)$
A3. $FST(x, y) = x$
A4. $SND(x, y) = y$
  G1. $\forall x.(((FST \ x), (SND \ x)) = x)$
  G2. $((FST \ a), (SND \ a)) = a$
A5. $\theta = \{x \leftarrow a\}$
  $\exists x, x_2, a = (x_1, x_2)$
A6. $a = (a_1, a_2)$
  $G3. (FST(a_1, a_2), SND(a_1, a_2)) = (a_1, a_2)$
  $G4. (a_1, a_2) = (a_1, a_2)$
  $G5. \text{true}$

2 Lists

Lists are often used to model stacks, queues, and other structures. Lists are either empty – [], or are non-empty in which case they have the form $h :: t$ where $h$ is the first element of the list and $t$ is the remainder of the list (possibly empty).

$$(\alpha)\text{list} ::= [[] | \alpha :: (\alpha)\text{list} \quad (5)$$

The notation $\alpha$ and $(\alpha)\text{list}$ mean any type $\alpha$ and list of elements of type $\alpha$, respectively. The uniqueness and induction properties are given below.

$$\forall h, t. \neg([[ = h :: t])$$

$$\forall h_1, t_1, h_2, t_2. (h_1 :: t_1 = h_2 :: t_2) = (h_1 = h_2) \land (t_1 = t_2)$$

$$\forall P.P [[] \land \forall t.P \supset (\forall h.P(h :: t)) \supset (\forall l.P l)$$

Based on the above higher-order induction theorem, we add another rule to the tableau rule set as shown below in Figure 1 where $r : (\alpha)\text{list}$ is some arbitrary list and $a : \alpha$ is some arbitrary element of type $\alpha$ which extends $r$ by one element.

<table>
<thead>
<tr>
<th>Assertions</th>
<th>Goals</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\forall(x : (\alpha)\text{list}).F[x]$</td>
<td>$F([[]] \land (F[r] \supset F[a :: r])$</td>
</tr>
</tbody>
</table>

Figure 1: Induction Rule

From the above, the theorem asserting that all lists are either empty or of the form $h :: t$ can be proved.

$$\forall l. (l = [[]]) \lor (\exists t. h. l = (h :: t))$$

The proof of (9) is similar to the proof of the cases theorem for numbers. It is proved by induction on $l$. 

CSE 607: E-7  List Examples: 2
AO. \(x = x\)
A1. \(\neg([]) = (h :: t)\)
A2. \(((h1 :: t1) = (h2 :: t2)) \supset (h1 = h2) \land (t1 = t2)\)
G1. \(\forall t. (l = []) \lor (\exists h. t. l = (h :: t))\)

**Induction on \(l\)**

**Case \(m = \[]\)**

G2.1 \(\[] = [] \lor (\exists h. t. \[] = (h :: t))\)
G3.1 \(true\)

**Induction Step**

G2.2 \([l'] = [] \lor (\exists h. t. l' = (h :: t)) \supset \)
G3.2 \(([h :: t'] = [] \lor (\exists h. t. (h :: t') = (h :: t))\]

A3.2 \(t' = [] \lor (\exists h. t. t' = (h :: t))\)
G3.2 \(\exists h. t. (h :: t') = (h :: t)\)
G4.2 \(\exists h. t. (h :: t') = (h :: t)\)
G5.2. \(\exists h. (h :: t') = (h :: t)\)
G6.2 \(\exists h. (h :: t) \leftarrow t'\)

**true**

A0, G5 Res

Once the domain of lists is defined, the normal operations on lists and their properties are defined and proved. Basic operations on lists include:

- **NULL** which tests if a list is empty.
- **HD** and **TL** which return the head element and tail (all but the first element) of a non-empty list.
- **APPEND** concatenates two lists.
- **LENGTH** returns the number of elements in a list.
- **EL \(n\)** returns the \(n^{th}\) element of a list.

\[
\null \[\] = true \quad (10)
\forall h. t. \null (h :: t) = false \quad (11)
\forall h. t. \hd (h :: t) = h \quad (12)
\forall h. t. \tl (h :: t) = t \quad (13)
\forall l. \append \[\] l = l \quad (14)

\forall l1 l2 h. \append (h :: l1) l2 = h :: (\append l1 l2) \quad (15)
\length \[\] = 0 \quad (16)
\forall h. t. \length (h :: t) = (\length t) + 1 \quad (17)
\forall l. \el 0 l = \hd l \quad (18)
\forall l n. \el (n + 1) l = \el n (\tl l) \quad (19)

To illustrate the use and some of the properties of the above definitions, we prove the following.

\[
\forall l1 l2. \length (\append l1 l2) = \length l1 + \length l2 \quad (20)
\]

What (20) says is given two lists, say [1; 2] and [3; 4; 5], the following is true.
**LENGTH(APPEND[1; 2][3; 4; 5]) → LENGTH[1; 2; 3; 4; 5] → 5**

The proof of (20) is done by induction on l1, and by using the definitions of APPEND and LENGTH. The entire proof is shown below.

A0. \( x = x \)

A1. \( -[[i] = [h :: t] \)

A2. \((h1 :: t1) = (h2 :: t2) ) \cup (h1 = h2) \cup (t1 = t2) \)

A3. NULL \([i] = T \)

A4. NULL \((h :: t) = F \)

A5. HD(h :: t) = h

A6. TL(h :: t) = t

A7. APPEND([i] l = l

A8. APPEND (h :: [i]) l2 = h :: (APPEND [i] l2)

A9. LENGTH ([i] = 0

A10. LENGTH (h :: t) = (LENGTH t) + 1

A11. \( \forall l1.2. \text{LENGTH}(\text{APPEND}[l1])2 = \text{LENGTH}[l1 + \text{LENGTH}[2] \)

**Induction on l1**

**Base Case**

G2.1 \( \forall [2.2. \text{LENGTH}(\text{APPEND}[l1])2 = \text{LENGTH}[l1] + \text{LENGTH}[2] \)

G3.1 \( \text{LENGTH}(\text{APPEND}[l1])2 = \text{LENGTH}[l1] + \text{LENGTH}[2] \)

G4.1 \( \text{LENGTH}[l1] = \text{LENGTH}[l1'] + \text{LENGTH}[2] \)

G5.1 \( \text{LENGTH}[l1] = \text{LENGTH}[l1'] + \text{LENGTH}[2] \)

G6.1 \( \text{true} \)

**Induction Step**

G2.2 \[ \forall [2.2. \text{LENGTH}(\text{APPEND}[l1'])2 = \text{LENGTH}[l1'] + \text{LENGTH}[2] \]

[ \forall [2.2. \text{LENGTH}(\text{APPEND}[l1'])2 = \text{LENGTH}[l1'] + \text{LENGTH}[2] \]

A11.2 \( \forall [2.2. \text{LENGTH}(\text{APPEND}[l1'])2 = \text{LENGTH}[l1'] + \text{LENGTH}[2] \)

G3.2 \( \forall [2.2. \text{LENGTH}(\text{APPEND}[l1'])2 = \text{LENGTH}[l1'] + \text{LENGTH}[2] \)

G4.2 \( \text{LENGTH}(\text{APPEND}[l1'])2 = \text{LENGTH}[l1'] + \text{LENGTH}[2] \)

G5.2 \( \text{LENGTH}(\text{APPEND}[l1'])2 = \text{LENGTH}[l1'] + \text{LENGTH}[2] \)

G6.2 \( \text{true} \)

The elements from two lists lists are paired or **zipped** together to form a single list by the ZIP operator. A list of pairs is **unzipped** into two lists by UNZIP. Similarly, a list extracted from either the first or second elements of a list of pairs is gotten by UNZIP FST or UNZIP SND. The ZIP operators are useful when dealing with pairs of words or bit-vectors as arguments to hardware descriptions, e.g. adders, ALUs, etc.

\[
\text{ZIP}([], []) = [] \quad \text{(21)}\\
\forall x1 \ l1 \ x2 \ \text{ZIP}((x1 :: l1), (x2 :: l2)) = (x1, x2) :: (\text{ZIP}(l1, l2)) \quad \text{(22)}\\
\text{UNZIP} [ ] = [], [] \quad \text{(23)}\\
\forall x \ l. \ \text{UNZIP}(x :: l) = \quad \text{(24)}

((FST x) :: (FST(UNZIP l)), (SND x) :: (SND(UNZIP l)))

CSE 607: E-7 List Examples: 4
∀l. UNZIP$\_FST$ l = FST(UNZIP l)  \hspace{0.5cm} (25)
∀l. UNZIP$\_SND$ l = SND(UNZIP l)  \hspace{0.5cm} (26)

The following operators are higher-order operators in that they accept functions as parameters as well as lists. We introduce them because they simplify the definition of other list operators used on words and finite-state machines. MAP applies a function to each element of a list to create a new list, e.g. $MAP(-)[1;2;3] = [-1;-2;-3]$. MAP2 applies a function to elements from two lists, e.g. $MAP2 + [1;2;3][4;5;6] = [5;7;9]$. FOLDL operates like an accumulator, e.g. $FOLDL + 0 [1;2;3] = 0 + 1 + 2 + 3$.

\[
\forall f. MAP f [] = [] \hspace{0.5cm} (27)
\forall f, h, t. MAP f (h :: t) = (f h) :: (MAP f t) \hspace{0.5cm} (28)
\forall f. MAP2 f [[]] = [] \hspace{0.5cm} (29)
\forall f, h_1, t_1, h_2, t_2. MAP2 f (h_1 :: t_1)(h_2 :: t_2) = (f h_1 h_2) :: (MAP2 f t_1 t_2) \hspace{0.5cm} (30)
\forall f, e. FOLDL f e [] = e \hspace{0.5cm} (31)
\forall f, e, x, l. FOLDL f e (x :: l) = FOLDL f (f e x) l \hspace{0.5cm} (32)
\]

As an example, the following is a lemma using FOLDL and "\forall.

\[\forall x. FOLDL \land (in0 \lor x) l = x \lor (FOLDL \land in0) l\] \hspace{0.5cm} (33)

The proof is done by induction on l.

\begin{itemize}
    \item A0. \(x = x\)
    \item A1. \(fold f u[] = u\)
    \item A2. \(fold f u :: xs = fold f (f(u x)) xs\)
        \begin{itemize}
            \item G1. \(\forall x \in 0. fold \lor (in0 \lor x) l = x \lor (fold \lor in0) l\)
        \end{itemize}

\textbf{Induction on l.}

\begin{itemize}
    \item Base Case
        \begin{itemize}
            \item G2.1 \(fold \lor (in0 \lor a) [] = a \lor (fold \lor in0) []\)
            \item G2.1 \(in0 \lor a = a \lor in0\)
            \item G2.1 \(true\) \hspace{0.5cm} G2. A1 EQ – LR
        \end{itemize}
    \item Induction step
        \begin{itemize}
            \item G2.2 \(\forall x \in 0. fold \lor (in0 \lor x) \mathrm{alist} = x \lor (fold \lor \mathrm{alist}) \land (\forall x \in 0. fold \lor (in0 \lor x) \mathrm{alist} = x \lor (fold \lor \mathrm{alist}) \land (\forall x \in 0. fold \lor (in0 \lor x) \mathrm{alist} = x \lor (fold \lor \mathrm{alist}))
            \item G3.2 \(\forall x \in 0. fold \lor (in0 \lor x) \mathrm{alist} = x \lor (fold \lor \mathrm{alist}) \land (\forall x \in 0. fold \lor (in0 \lor x) \mathrm{alist} = x \lor (fold \lor \mathrm{alist}) \land (\forall x \in 0. fold \lor (in0 \lor x) \mathrm{alist} = x \lor (fold \lor \mathrm{alist})))
            \item A3.2 \(\forall x \in 0. fold \lor (in0 \lor x) \mathrm{alist} = x \lor (fold \lor \mathrm{alist}) \land (\forall x \in 0. fold \lor (in0 \lor x) \mathrm{alist} = x \lor (fold \lor \mathrm{alist}))
            \item G4.2 \(\forall x \in 0. fold \lor (in0 \lor x) \mathrm{alist} = x \lor (fold \lor \mathrm{alist}) \land (\forall x \in 0. fold \lor (in0 \lor x) \mathrm{alist} = x \lor (fold \lor \mathrm{alist}))
            \item G5.2 \(\forall x \in 0. fold \lor (in0 \lor x) \mathrm{alist} = x \lor (fold \lor \mathrm{alist}) \land (\forall x \in 0. fold \lor (in0 \lor x) \mathrm{alist} = x \lor (fold \lor \mathrm{alist}))
            \item G6.2 \(\forall x \in 0. fold \lor (in0 \lor x) \mathrm{alist} = x \lor (fold \lor \mathrm{alist}) \land (\forall x \in 0. fold \lor (in0 \lor x) \mathrm{alist} = x \lor (fold \lor \mathrm{alist}))
            \item G4.2 \(\forall x \in 0. fold \lor (in0 \lor x) \mathrm{alist} = x \lor (fold \lor \mathrm{alist}) \land (\forall x \in 0. fold \lor (in0 \lor x) \mathrm{alist} = x \lor (fold \lor \mathrm{alist}))
            \item G7.2 \(true\) \hspace{0.5cm} A0, G6 res
        \end{itemize}
\end{itemize}

The following properties of MAP, MAP2, FOLDL, LENGTH, and APPEND are true.
3 Additional Proofs

The following two proofs are from problem 11.8 in Manna and Waldinger [1].
A0. \( x = x \)
A1. \( \neg[[] = h :: t] \)
A2. \( ((h1 :: t1) = (h2 :: t2)) \supset (h1 = h2) \land (t1 = t2) \)
A3. \( \textit{NULL} [[] = \text{true} \)
A4. \( \textit{NULL} (h :: t) = \text{false} \)
A5. \( hD(h :: t) = h \)
A6. \( T(h :: t) = t \)
A7. \( \textit{APPEND} [[] = [l] = l \)
A8. \( \textit{APPEND} (h :: [l]) [2 = h :: (\textit{APPEND} [1] [2]) \)
G1. \( \forall y \textit{APPEND} [[] = [l] \)
\textbf{Induction on} \( l \)
\textbf{Base Case}
G2.1 \( \textit{APPEND} [][] = [] \)
G3.1 \( [] = [] \)
G4.1 \textit{true} \)
\textbf{Induction Step}
G2.2 \( \textit{APPEND} l' [[] = l' \supset \textit{APPEND}(h' :: l')[[] = (h' :: l') \)
A9.2 \( \textit{APPEND} l' [[] = l' \)
G3.2 \( \textit{APPEND}(h' :: l')[[] = (h' :: l') \)
G4.2 \( h' :: (\textit{APPEND} l' [[] = h' :: l' \)
G5.2 \( h' :: l' = h' :: l' \)
G6.2 \textit{true} \)
\textbf{To simplify the appearance of the proof, let} \( x + y + z \) \textbf{denote} \( \textit{APPEND} x y \).
A0. \( x = x \)
A1. \( \neg[[] = h :: t] \)
A2. \( ((h1 :: t1) = (h2 :: t2)) \supset (h1 = h2) \land (t1 = t2) \)
A3. \( \textit{NULL} [[] = \text{true} \)
A4. \( \textit{NULL} (h :: t) = \text{false} \)
A5. \( hD(h :: t) = h \)
A6. \( T(h :: t) = t \)
A7. \( \textit{APPEND} [[] = [l] = l \)
A8. \( \textit{APPEND} (h :: [l]) [2 = h :: (\textit{APPEND} [1] [2]) \)
G1. \( \forall y \textit{APPEND} [[] = [l] \)
\textbf{Induction on} \( x \)
\textbf{Base Case}
G2.1 \( [[] + y + z = [l] + (y + z) \)
G3.1 \( y + z = y + z \)
G4.1 \textit{true} \)
\textbf{Inductive Step}
G2.2 \( \forall y z. ([x' + y + z] + + z = x' + + (y + z)] \supset \forall y z. ((a :: x') + + y + + z = (a :: x') + + (y + z)] \)
A9.2 \( \forall y z. ([x' + y + z] + + z = x' + + (y + z)] \)
G3.2 \( \forall y z. ((a :: x') + + y + + z = (a :: x') + + (y + z)] \)
A10.2 \( (x' + + y + + z = x' + + (y + z)] \)
G4.2 \( (a :: x') + + y + + z = (a :: x') + + (y + z)] \)
G5.2 \( (a :: x') + + y + + z = a :: (x' + + (y + z)] \)
G6.2 \( a :: (x' + + y') + + z = a :: (x' + + (y' + z)] \)
G7.2 \( a :: (x' + + (y' + z)] = a :: (x' + + (y' + z)] \)
G8.2 \textit{true} \)
\textbf{The next proof proves that} \( \textit{ZIP} \) \textbf{and} \( \textit{UNZIP} \) \textbf{invert each other.}
A0. $x = x$
A1. $\exists (x_1 : \alpha)(x_2 : \beta). [(x : (\alpha \times \beta)) = (x_1, x_2)]$
A2. $(x_1, x_2) = (y_1, y_2) \equiv (x_1 = y_1) \land (x_2 = y_2)$
A3. $FST(x, y) = x$
A4. $SND(x, y) = y$
A5. $((FST \ x), (SND \ x)) = x$
A6. $-([] = \text{true})$
A7. $(\text{true} :: 1) = (\text{false} :: 2) \equiv (1 = 2)$
A8. $\text{APPEND}([l]) = l$
A9. $\text{APPEND}([h : 1], [l]) = [h : \text{APPEND}([1], [l])]$
A10. $\text{ZIP}([l], [l]) = [l]$
A11. $\text{ZIP}([x_1 :: 1], [x_2 :: 2]) = (x_1, x_2) :: ([\text{ZIP}([1], [l])]$
A12. $U \text{NZIP}([]) = ([], [])$
A13. $U \text{NZIP}(x :: []) = ((FST \ x) :: FST(U \text{NZIP} l), (SND \ x) :: (SND(U \text{NZIP} l)))$

Induction on $l$

Base Case

G2.1 $\text{ZIP}(U \text{NZIP} [l]) = []$
G3.1 $\text{ZIP}([l], [l]) = []$
G4.1 $[l] = []$
G5.1 true

Induction Step

G2.2 $\text{ZIP}(U \text{NZIP} r) = r \supseteq \text{ZIP}(U \text{NZIP}(a :: r)) = (a :: r)$
A14.2 $\text{ZIP}(U \text{NZIP} r) = r$
G2.2 $\text{ZIP}((FST a :: FST(U \text{NZIP} r), (SND a :: SND(U \text{NZIP} r))) = a :: r$
G5.2 $\text{ZIP}(FST (a, SND a) :: \text{ZIP}(FST(U \text{NZIP} r), SND(U \text{NZIP} r))) = a :: r$
A15.2 $\theta = \{x \leftarrow a\}$
G6.2 $a :: \text{ZIP}(FST(U \text{NZIP} r), SND(U \text{NZIP} r)) = a :: r$
A16.2 $\theta = \{x \leftarrow U \text{NZIP} r\}$
G7.2 $a :: \text{ZIP}(U \text{NZIP} r) = a :: r$
G8.2 $a :: r = a :: r$
G9.2 true

References