07—The Language rePL

CS 4215: Programming Language Implementation

Martin Henz

March 2, 2012

Generated on Friday 2 March, 2012, 06:32
1 Data Structures
   - Data Structures in simPL
   - Motivation
   - Data Structures in rePL
   - Syntactic Sugar

2 Exception Handling

3 Denotational Semantics of rePL
Pairs as Functions

Construsting a pair using a conditional

let p = fun i ->
  if i=1 then 10 else 20 end
end

in ...
end
Accessing Pairs

Accessing pairs by application

let p = fun i ->
    if i=1 then 10 else 20 end
end
in ... (p 1) ... (p 2) ... end
Constructing Pairs

```
let pair =
  fun x y ->
    fun i ->
      if i=1 then x else y end
  end
end

in ...
let p = (pair 10 20)
in ...
end
end
```
Motivation

Disadvantages of data structures as functions:

- difficult to distinguish functions from data structures
- definition of data structures with many components gives rise to large nested conditionals
- the only values that we can use to access data structures are integers
- inefficient, due to the function closures created, and linear execution of nested conditionals.
Motivation

With data structures, we will introduce another partial function. Thus we will need to take another look at error handling, giving rise to exception handling.
rePL Inherits From simPL

\[
\begin{array}{cccc}
& x & \quad & n \\
E_1 & E_2 & \quad & \text{true} \\
\hline
\end{array}
\]

where \( p \in \{|, \& , + , - , *\} \)

\[
p[E_1, E_2]
\]

\[
\begin{array}{c}
E \\
\hline
\end{array}
\]

where \( p \in \{/\} \)

\[
p[E]
\]
rePL Inherits From simPL

\[
\begin{align*}
E & \quad E_1 & \quad E_2 & \quad E & \quad E_1 & \cdots & \quad E_n \\
\text{if } E \text{ then } E_1 \text{ else } E_2 \text{ end} & \quad (E \ E_1 \cdots E_n) \\
E & \quad x_1, \ldots, x_n \text{ pairwise distinct} & \quad \text{fun } x_1 \cdots x_n \rightarrow E \text{ end}
\end{align*}
\]
rePL Inherits From simPL

\[
\begin{array}{c}
E \\
\hline
\text{recfun } f \ x_1 \cdots x_n \rightarrow E \ \text{end}
\end{array}
\]

\[
\begin{array}{c}
E_1 \quad \cdots \quad E_n \\
\hline
\text{let } x_1 = E_1 \cdots x_n = E_n \text{ in } E \ \text{end}
\end{array}
\]

$f, x_1, \cdots, x_n$ pairwise distinct
Records in rePL are bracket-enclosed sequences of property-value associations.

In rePL we can represent a pair containing 10 and 20 as a record of the form

[First:10, Second:20]
Accessing Records Using “Dot”

let p = [First: 10, Second: 20]
in p.First + p.Second
end
Nested Records

Records can appear inside of other records, as in the following record representing a color point on the screen.

[X:100, Y:200, Color:[Red:255, Green:127, Blue:0]]
Syntax of Records in rePL

\[ E_1 \quad E_n \]
\[ \begin{array}{c}
\[ q_1 : E_1, \ldots, q_n : E_n \]
\end{array} \]
\[ E \]
\[ E \cdot q \]

where \( q, q_1, \ldots, q_n \) denote properties
More Record Operators

The operator

\[ E \text{ hasproperty } q \]

returns \textit{true}, if the record resulting from \( E \ E \) has property \( q \), and \textit{false} otherwise.

Examples:

\[
\begin{align*}
[\text{Red:0, Blue:127, Green:255}] \text{ hasproperty } \text{Green} \\
[\text{Red:0, Blue:127, Green:255}] \text{ hasproperty } \text{Yellow}
\end{align*}
\]
The Empty Record

The empty record [] does not have any properties.

The operator `empty` checks whether its argument is the empty record.

Examples:

```plaintext
empty []
empty [SomeProperty: 1]
```
Syntax

\[ E \]

where \( p \in \{ \emptyset, \text{empty} \} \)

\[ p[E] \]
Syntactic Sugar for rePL

We write $E_1 :: E_2$ as abbreviation for $[First: E_1, Second: E_2]$.

The operator $::$ is right-associative.

We can write $10 :: 20 :: 30$ instead of $10 :: (20 :: 30)$. 
Lists

A list is either empty—in which case it is represented by the empty record `[]`—or a pair, whose second component is also a list.

The elements of the list are the first components of the pairs that make up the list.

Example:

```
10 :: 20 :: 30 :: 40 :: []
```
Constructing Lists

```plaintext
let even = recfun even i counter done ->
  if counter=done then []
  else i :: (even i+2 counter+1 done)
end

in let evennumbers = fun n -> (even 2 0 n) end
  in ...
end

end
```

The expression `(evennumbers 3)` returns the list `2 :: 4 :: 6 :: []`. 
Length of Lists

The following function computes the length of a given list.

```plaintext
recfun length xs ->
  if empty xs then 0
  else 1+(length xs.Second)
end
end
```
Mapping Lists

`recfun map xs f -> if empty xs then []
   else (f xs.First) :: (map xs.Second f)
end`

Example:

`(map 1 :: 2 :: 3 :: [] fun x -> x * x end)`

returns `1 :: 4 :: 9 :: []`. 
Folding Lists

recfun fold xs f start -> if empty xs then start
    else (f xs.First
        (fold xs.Second f start))
end

eend

Example:

(fold 1 :: 4 :: 9 :: [] (fun x y -> x + y end) 0)

returns 14.
1. Data Structures

2. Exception Handling
   - Motivation
   - Syntax of Exception Handling
   - Built-in Exceptions
   - Programmer-defined Exceptions

3. Denotational Semantics of rePL
Motivation

Errors arise from

- Division by zero
- Invalid record access
- ...

...
Handling Exceptions

```plaintext
try (evaluate input)
catch e
with if e hasproperty DivisionByZero
  then (evaluate (readNewUserInput))
else ..
end
end
```
Syntax of Exception Handling

\[
\begin{align*}
E_1 & \quad E_2 \\
\text{try } E_1 \text{ catch } x \text{ with } E_2 \text{ end}
\end{align*}
\]
Built-in Exceptions

Division by zero leads to an exception of the form [DivisionByZero: true].
Invalid record access leads to an exception of the form [InvalidRecordAccess: true].

Examples:

4711 / 0

[].SomeProperty
Let the Programmer Throw Exceptions

\[
\begin{array}{c}
E \\
\hline
\text{throw } E \text{ end}
\end{array}
\]
Example

if percentage > 100
then throw [PercentageExceeds100: true,
             PercentageValue: percentage]
   end
else ... end

The percentageExceeds100 exception can then be caught by a surrounding expression and handled appropriately.
1. Data Structures

2. Exception Handling

3. Denotational Semantics of rePL
   - rePL0: simPL plus Records
   - rePL1: rePL0 plus Exceptions
## Semantic Domains for rePL0

<table>
<thead>
<tr>
<th>Sem. domain</th>
<th>Definition</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bool</strong></td>
<td>{true, false}</td>
<td>ring of booleans</td>
</tr>
<tr>
<td><strong>Int</strong></td>
<td>{..., −2, −1, 0, 1, 2, ...}</td>
<td>ring of integers</td>
</tr>
<tr>
<td><strong>EV</strong></td>
<td><strong>Bool</strong> + <strong>Int</strong> + {⊥} + <strong>Fun</strong> + <strong>Rec</strong></td>
<td>expressible values</td>
</tr>
<tr>
<td><strong>DV</strong></td>
<td><strong>Bool</strong> + <strong>Int</strong> + <strong>Fun</strong> + <strong>Rec</strong></td>
<td>denotable values</td>
</tr>
<tr>
<td><strong>Id</strong></td>
<td>alphanumeric string</td>
<td>identifiers</td>
</tr>
<tr>
<td><strong>Env</strong></td>
<td><strong>Id</strong> (\leadsto) <strong>DV</strong></td>
<td>environments</td>
</tr>
<tr>
<td><strong>Fun</strong></td>
<td><strong>DV</strong> (\ast) (\ast) <strong>DV</strong> (\leadsto) <strong>EV</strong></td>
<td>function values</td>
</tr>
<tr>
<td><strong>Rec</strong></td>
<td><strong>Id</strong> (\leadsto) <strong>DV</strong></td>
<td>records</td>
</tr>
</tbody>
</table>
Rules for rePL0

\[
\Delta \vdash E_1 \leftrightarrow v_1 \quad \cdots \quad \Delta \vdash E_n \leftrightarrow v_n
\]

\[
\begin{align*}
\Delta \vdash [q_1:E_1, \ldots, q_n:E_n] & \Rightarrow f \\
& \text{where } f = \emptyset[q_1 \leftarrow v_1] \\
& \quad \cdots [q_n \leftarrow v_n]
\end{align*}
\]
Rules for rePL0

\[ \Delta \vdash E \rightarrow v \]

\[ \Delta \vdash E \cdot q \rightarrow v' \]

where \( v' = v(q) \)
Rules for rePL0

\[ \Delta \vdash E \rightarrow v \]
\[ \text{if } \text{dom}(v) = \emptyset \]
\[ \Delta \vdash \text{empty } E \rightarrow \text{true} \]

\[ \Delta \vdash E \rightarrow v \]
\[ \text{if } \text{dom}(v) \neq \emptyset \]
\[ \Delta \vdash \text{empty } E \rightarrow \text{false} \]
Rules for rePL0

\[ \Delta \models E \Rightarrow v \]

- if \( q \in dom(v) \)
  \[ \Delta \models E \text{ hasproperty } q \Rightarrow true \]

- if \( q \notin dom(v) \)
  \[ \Delta \models E \text{ hasproperty } q \Rightarrow false \]
## Semantic Domains for rePL1

<table>
<thead>
<tr>
<th>Sem. domain</th>
<th>Definition</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bool</strong></td>
<td>{true, false}</td>
<td>ring of booleans</td>
</tr>
<tr>
<td><strong>Int</strong></td>
<td>{..., −2, −1, 0, 1, 2, ...}</td>
<td>ring of integers</td>
</tr>
<tr>
<td><strong>EV</strong></td>
<td><strong>Bool</strong> + <strong>Int</strong> + <strong>Exc</strong> + <strong>Fun</strong> + <strong>Rec</strong></td>
<td>expressible values</td>
</tr>
<tr>
<td><strong>DV</strong></td>
<td><strong>Bool</strong> + <strong>Int</strong> + <strong>Fun</strong> + <strong>Rec</strong></td>
<td>denotable values</td>
</tr>
<tr>
<td><strong>Id</strong></td>
<td>alphanumeric string</td>
<td>identifiers</td>
</tr>
<tr>
<td><strong>Env</strong></td>
<td><strong>Id</strong> ⇝ <strong>DV</strong></td>
<td>environments</td>
</tr>
<tr>
<td><strong>Fun</strong></td>
<td><strong>DV</strong> ∗ ··· ∗ <strong>DV</strong> ⇝ <strong>EV</strong></td>
<td>function values</td>
</tr>
<tr>
<td><strong>Rec</strong></td>
<td><strong>Id</strong> ⇝ <strong>DV</strong></td>
<td>records</td>
</tr>
<tr>
<td><strong>Exc</strong></td>
<td><strong>Rec</strong></td>
<td>exceptions</td>
</tr>
</tbody>
</table>
Rules for rePL1

\[
\begin{align*}
\Delta \vdash E_1 \Rightarrow e & \quad \text{if } e \in \text{Exc} \\
\Delta \vdash E_1 + E_2 \Rightarrow e & \\
\Delta \vdash E_1 \Rightarrow v & \quad \Delta \vdash E_2 \Rightarrow e & \quad \text{if } v \notin \text{Exc} \text{ and } e \in \text{Exc} \\
\Delta \vdash E_1 + E_2 \Rightarrow e & \\
\Delta \vdash E_1 \Rightarrow v_1 & \quad \Delta \vdash E_2 \Rightarrow v_2 & \quad \text{if } v_1, v_2 \notin \text{Exc} \\
\Delta \vdash E_1 + E_2 \Rightarrow v_1 + v_2 &
\end{align*}
\]
Rules for rePL1

\[
\Delta \vdash E_1 \rightarrow e
\]

if \( e \in \text{Exc} \)

\[
\Delta \vdash E_1 / E_2 \rightarrow e
\]

\[
\Delta \vdash E_1 \rightarrow v \quad \Delta \vdash E_2 \rightarrow e
\]

if \( v \notin \text{Exc} \) and \( e \in \text{Exc} \)

\[
\Delta \vdash E_1 / E_2 \rightarrow e
\]

\[
\Delta \vdash E_1 \rightarrow v_1 \quad \Delta \vdash E_2 \rightarrow v_2
\]

if \( v_1, v_2 \notin \text{Exc} \) and \( v_2 \neq 0 \)

\[
\Delta \vdash E_1 / E_2 \rightarrow v_1 / v_2
\]
Rules for rePL1

\[ \Delta \models E_1 \rightarrow v_1 \quad \Delta \models E_2 \rightarrow 0 \]

\[ \Delta \models E_1 / E_2 \rightarrow e \]

if \( v_1 \notin \text{Exc} \) and

where \( e = \text{[DivisionByZero: true]} \), and \( e \in \text{Exc} \)
Rules for rePL1

\[ \Delta \vdash E \rightarrow e \]
\[ \Delta \vdash E \cdot q \rightarrow e \] if \( e \in \text{Exc} \)

\[ \Delta \vdash E \rightarrow v \]
\[ \Delta \vdash E \cdot q \rightarrow v' \] if \( q \in \text{dom}(v) \) and where \( v' = v(q) \)

\[ \Delta \vdash E \rightarrow v \]
\[ \Delta \vdash E \cdot q \rightarrow e \] if \( q \notin \text{dom}(v) \) and where \( e = [\text{InvalidRecordAccess: true}] \), and \( e \in \text{Exc} \)
Rules for rePL1

\[ \Delta \models E \Rightarrow v \]

\[ \text{if } v \in \text{Rec} \cup \text{Exc} \text{ and where } e = v, e \in \text{Exc} \]
Overview of Next Lecture

Friday 9/3:

- Variants: pass-by-name and pass-by-need
- Implementing rePL and its variants