Guards and Computing Functions

September 21, 2001

Review

• A function whose code does no work after a recursive call is said to be tail-recursive.

• Advantages
  – Can be more space-efficient
  – Sometimes is even faster.

• Disadvantages
  – Code is often more complex, harder to understand
  – Not all tasks benefit from tail recursion
    * e.g., append, drop, ...

Converting Numbers to Binary

• Want toBinary(N) to return a list of 0’s and 1’s, the binary representation of the positive integer N.

  toBinary(37) ==> [1, 0, 0, 1, 0, 1]
  [since 37 = 1\cdot2^5+0\cdot2^4+0\cdot2^3+1\cdot2^2+0\cdot2^1+1\cdot2^0]

  \[\text{toBinary}(N) = \text{toBinary}(N, \text{[]});\]
  \[\text{toBinary}(0, \text{Acc}) \Rightarrow \text{Acc};\]
  \[\text{toBinary}(N, \text{Acc}) \Rightarrow \text{toBinary}(N/2, \text{[N\%2 | Acc]});\]

Execution Sequence

\[\text{toBinary}(37, \text{[]}) \Rightarrow\]
\[\text{toBinary}(18, \text{[1]}) \Rightarrow\]
\[\text{toBinary}(9, \text{[0,1]}) \Rightarrow\]
\[\text{toBinary}(4, \text{[1,0,1]}) \Rightarrow\]
\[\text{toBinary}(2, \text{[0,1,0,1]}) \Rightarrow\]
\[\text{toBinary}(1, \text{[0,0,1,0,1]}) \Rightarrow\]
\[\text{toBinary}(0, \text{[1,0,0,1,0,1]}) \Rightarrow\]
\[1,0,0,1,0,1\]
Conversion from Binary

- Could write \textit{iterative} pseudo-code, then construct recursive equivalent.

```plaintext
fromBinary([], Result) => Result;
fromBinary([F|R], Result) =>
  fromBinary(R, 2*Result+F);
fromBinary(L) =
  fromBinary(L, 0);
```

\[ L = \ldots\text{list to be converted}\ldots \]
\[ \text{Result} = 0; \]
\[ \text{while} \ (L \neq []) \ { \}
  \quad \text{Result} = 2*\text{Result} + \text{first} (L);
  \quad L = \text{rest} (L);
\{ \]
\[ \ldots \text{answer is in Result} \ldots \]

Guards! Guards!

- Sometimes when choosing which case of a function
to evaluate, we want more information than whether
the argument matches a pattern or not.
- Rex provides two sorts of \textit{guards}, which are extra
conditions that must be satisfied before a case will be
chosen.

"Normal" Guards

- Definitions of the form
\[ F(\text{pattern}) \Rightarrow \text{guard} \ ? \ expression; \]
  - This case applies only when the argument
    matches the given pattern \textit{and} the guard
    expression evaluates to true.
  - Example: Euclid's algorithm

```
gcd(0,Y) => Y;
gcd(X,Y) => X<=Y ? gcd(Y-X,X);
gcd(X,Y) => X>Y ? gcd(Y,X);
```

"Equational" "Guards"

- Definitions of the form
\[ F(\text{pattern}) \Rightarrow \text{definitions}, expression; \]
  - Note that there's a , instead of a ?
  - This case applies only when the argument
    matches the given pattern \textit{and} the definitions
    succeed
    - in this case, variables defined in the definitions
      can be used in computing the function's value
    - How rex defines local variables.
    - Advice: only use definitions that should succeed.
Example

- Raising a number to the fourth power:

  \[ \text{hypercube}(X) \Rightarrow Y = X^2, \quad Y^2; \]

  Compare with:

  \[ \text{hypercube}(X) \Rightarrow (X^2)(X^2); \]

Insertion Sort

// insertion_sort sorts its argument list
insertion_sort([]) \Rightarrow [];
insertion_sort([F|R]) \Rightarrow
  insert(F, insertion_sort(R));

// insert(Y,Z) inserts element Y into the // correct position in the sorted list Z.
insert(A,[]) \Rightarrow [A];
insert(A,[B|X]) \Rightarrow

Selection Sort

// selection_sort sorts its argument list
selection_sort([]) \Rightarrow [];
selection_sort(L) \Rightarrow
  [M | R] = min_to_front(L),
  [M | selection_sort(R)];

// min_to_front(L) moves a minimal element of // the non-empty list L to the front (but may // not preserve the order of the other elements)
min_to_front([]) \Rightarrow [];
min_to_front([A|L]) \Rightarrow
  [B | R] = min_to_front(L),

Functions Returning Functions

- Consider \((X) \Rightarrow 5 \times X\)
  
  - A perfectly valid, although anonymous, function
- Functions in rex are “first-class citizens”:
  
  - They can be passed as arguments.
  - They can be put into lists.
  - They can be returned as values.
- A function-value returning function:

  \[ \text{scaleBy}(Y) = (X) \Rightarrow Y \times X; \]
  
  - Here \text{scaleBy}(Y) returns a function, the function that scales its argument by \(Y\).
Use of scaleBy

- Given the definition
  \[ \text{scaleBy}(Y) = (X) \Rightarrow Y \times X; \]

  we have
  \[
  \begin{align*}
  \text{map}(\text{scaleBy}(5), [1,2,3]) & \Rightarrow [5,10,15] \\
  \text{scaleby}(4)(7) & \Rightarrow 28
  \end{align*}
  \]

- We could have defined
  \[ \text{scale}(Y, L) = \text{map}(\text{scaleBy}(Y), L); \]

Curried Functions (yum!)

- \text{scaleBy} and \text{mapster} are examples of \textit{curried} functions
  - They take their arguments via sequential applications, each returning a function until the last argument.
- Another way to write them in rex is:
  \[
  \begin{align*}
  \text{scaleBy}(F)(X) & = F \times X; \\
  \text{mapster}(F)(L) & = \text{map}(F, L);
  \end{align*}
  \]

- Even with this notation, \text{scaleBy}(F) has a meaning without the \((X)\).

Anonymous returning Anonymous

- Yet another way to write \text{scaleBy} and \text{mapster} in rex is:
  \[
  \begin{align*}
  \text{scaleBy} & = (F) \Rightarrow (X) \Rightarrow F \times X; \\
  \text{mapster} & = (F) \Rightarrow (L) \Rightarrow \text{map}(F, L);
  \end{align*}
  \]

- The names \text{scaleBy} and \text{mapster} are incidental
  - Right-hand sides of the equations can be applied directly
Curried Functions in Math, Science, and Engineering

• In a math, science, or engineering text, you might see
  \( f_k(x) \)
• Typical nomenclature is that \( x \) is the “argument” and \( k \) is a “parameter”.
• **Both** \( k \) and \( x \) may be viewed as arguments to \( f \), though typically \( k \) is “more fixed”.
• We could define this as \( f(k)(x) = \ldots \)
• Then we can use \( f(k) \) as \( f_k \) would be used.