DD2552 Seminar 5: From primitive to general recursion

Karl Palmskog

KTH

Wednesday September 13, 2023

Course material

- PFPL chapter 15, (co)inductive data types
- PFPL chapter 19, recursive functions
- PFPL chapter 20, recursive types

Revisiting sum types: Booleans

$$
\begin{array}{lcl}
T & ::= & \\
 & \mid & \mathsf{unit} \\
 & \mid & T + T'
\end{array}
$$

$$
\begin{array}{lcl}
e & ::= & \\
 & \mid & \langle\rangle \\
 & \mid & l \cdot e \\
 & \mid & r \cdot e
\end{array}
$$

$$\mathsf{bool} \stackrel{\mathsf{def}}{=\!=} \mathsf{unit} + \mathsf{unit}$$

$$\mathsf{true} \stackrel{\mathsf{def}}{=\!=} l \cdot \langle\rangle$$

$$\mathsf{false} \stackrel{\mathsf{def}}{=\!=} r \cdot \langle\rangle$$

Revisiting sum types: options of Booleans

$$\begin{array}{lcl} T & ::= & \\ & \mid & \mathsf{unit} \\ & \mid & T + T' \end{array}$$

$$\begin{array}{lcl} e & ::= & \\ & \mid & \langle\rangle \\ & \mid & l \cdot e \\ & \mid & r \cdot e \end{array}$$

$$\mathsf{option} \stackrel{\mathsf{def}}{=\!=} \mathsf{unit} + \mathsf{bool}$$

$$\mathsf{none} \stackrel{\mathsf{def}}{=\!=} l \cdot \langle\rangle$$

$$\mathsf{some}(e) \stackrel{\mathsf{def}}{=\!=} r \cdot e$$

$$\mathsf{case}\, e\, \{l \cdot \_ \rightsquigarrow e_1 \mid r \cdot x_2 \rightsquigarrow e_2\}$$

Inductive types and natural numbers

$$
\begin{array}{lcl}
\tau & ::= & \\
 & \mid & t \\
 & \mid & \mu\,(\,t\,.\tau) \\
 & \mid & \nu\,(\,t\,.\tau)
\end{array}
$$

$$\mathsf{nat} \stackrel{\text{def}}{=\!=} \mu(t.\mathsf{unit} + t)$$

$$\mathsf{z} \stackrel{\text{def}}{=\!=} \mathsf{fold}(l \cdot \langle\rangle)$$

$$\mathsf{s}(e) \stackrel{\text{def}}{=\!=} \mathsf{fold}(r \cdot e)$$

$$\mu(t.\tau) \cong [\mu(t.\tau)/t]\tau$$

$$2 = \mathsf{fold}(r \cdot \mathsf{fold}(r \cdot \mathsf{fold}(l \cdot \langle\rangle)))$$

Natural number recursor/iterator

$$
\begin{array}{lcl}
\tau & ::= & \\
 & \mid & t \\
 & \mid & \mu\,(\,t\,.\tau) \\
 & \mid & \nu\,(\,t\,.\tau)
\end{array}
$$

$$
\begin{array}{lcll}
e & ::= & & \\
 & \mid & \mathsf{fold}\,(e) & \\
 & \mid & \mathsf{rec}\,(x.e_1; e_2) & \\
 & \mid & [e_1/x]e_2 & \mathsf{M}
\end{array}
$$

$$
\mathsf{rec}(x.e_1; \mathsf{fold}(e_2)) \longrightarrow
$$
$$
[\mathsf{case}\, e_2\, \{l \cdot \_ \leadsto l \cdot \langle\rangle \mid r \cdot y \leadsto r \cdot \mathsf{rec}(x.e_1; y)\}/x]e_1
$$

Structural recursion and termination

- rec-fold expressions are guaranteed to terminate
- the recursion is **structural**: bounded structure means bounded number of recursive calls
- this prevents expressing some interesting functions directly

General recursion and functionals

Consider a mathematically defined function:

- $f(0) = 1$
- $f(n+1) = (n+1) \times f(n)$

Define the functional F by $F(f) = f'$ where

- $f'(n) = 1$ if $n = 0$
- $f'(n) = n \times f(n')$ if $n = n' + 1$

We want a fixpoint (fixed point) g of F, such that $g = F(g)$.

Totality and partiality

- "all" systems of equations have fixpoints
- no guarantee that the fixpoint function is total (may diverge)
- we can *prove* termination on all of specific inputs

Classes of functions

- primitive recursive functions
- partial recursive functions (PCF) are strictly larger
- classic example: Ackermann function

$$
\begin{aligned}
A(0, n) &= n + 1 \\
A(m + 1, 0) &= A(m, 1) \\
A(m + 1, n + 1) &= A(m, A(m + 1, n))
\end{aligned}
$$

Example function on pairs of nat

```
gcd (a, b) :=
 | a = 0 => 0
 | b = 0 => 0
 | a = b => a
 | a < b => gcd (a, b-a)
 | a > b => gcd (a-b, b)
```

How do we know it terminates?

Measures

```
gcd (a, b) :=
 | a = 0 => 0
 | b = 0 => 0
 | a = b => a
 | a < b => gcd (a, b-a)
 | a > b => gcd (a-b, b)
```

- we associate a **measure** with each step (recursive call)
- proposed measure: `fst ab + snd ab`

Measure induces relation

```
gcd (a, b) :=
 | a = 0 => 0
 | b = 0 => 0
 | a = b => a
 | a < b => gcd (a, b-a)
 | a > b => gcd (a-b, b)
```

- we compare measure for input and recursive calls
- measure must decrease with every call