Recursive functions

Madhavan Mukund, **S P Suresh**

Programming Language Concepts

Lecture 19, 26 March 2024

Church numerals and arithmetic functions

- $\ll n \gg = \lambda f x \cdot f^n x$

Church numerals and arithmetic functions

- $\ll n \gg = \lambda f x \cdot f^n x$
  - $f^n x = f(f(\cdots(fx)\cdots))$, where f is applied repeatedly n times

Church numerals and arithmetic functions

- $\ll n \gg = \lambda f x \cdot f^n x$
  - $f^n x = f(f(\cdots(fx)\cdots))$, where f is applied repeatedly n times
  - $\ll n \gg g\, y = (\lambda f x \cdot f(\cdots(fx)\cdots))gy \quad \xrightarrow{*}_{\beta} \quad g(\cdots(gy)\cdots) = g^n y$

Church numerals and arithmetic functions

- $\ll n \gg = \lambda f x \cdot f^n x$
  - $f^n x = f(f(\cdots(fx)\cdots))$, where f is applied repeatedly n times
  - $\ll n \gg g\, y = (\lambda f x \cdot f(\cdots(fx)\cdots))gy \quad \xrightarrow{*}_\beta \quad g(\cdots(gy)\cdots) = g^n y$
- $\mathbf{succ} = \lambda p f x \cdot f(p\ f x)$ $\qquad \mathbf{succ} \ll m \gg \xrightarrow{*}_\beta \ll m+1 \gg$

  $\mathbf{plus} = \lambda p q\ f x \cdot p f(q\ f x)$ $\qquad \mathbf{plus} \ll m \gg \ll n \gg \xrightarrow{*}_\beta \ll m+n \gg$

  $\mathbf{mult} = \lambda p q f \cdot p(q\ f)$ $\qquad \mathbf{mult} \ll m \gg \ll n \gg \xrightarrow{*}_\beta \ll mn \gg$

  $\mathbf{exp} = \lambda p q \cdot pq$ $\qquad \mathbf{exp} \ll m \gg \ll n \gg \xrightarrow{*}_\beta \ll n^m \gg$

Computability

- Church numerals encode $n \in \mathbb{N}$

Computability

- Church numerals encode $n \in \mathbb{N}$
- Can we encode computable functions $f : \mathbb{N}^k \longrightarrow \mathbb{N}$?

Computability

- Church numerals encode $n \in \mathbb{N}$
- Can we encode computable functions $f : \mathbb{N}^k \longrightarrow \mathbb{N}$?
- Let **f** be the encoding of f

Computability

- Church numerals encode $n \in \mathbb{N}$
- Can we encode computable functions $f : \mathbb{N}^k \to \mathbb{N}$?
- Let **f** be the encoding of f
- We want $\mathbf{f}\ \text{«}n_1\text{»} \cdots \text{«}n_k\text{»} \xrightarrow{*}_\beta \text{«}m\text{»}$ iff $f(n_1, \cdots, n_k) = m$

Computability

- Church numerals encode $n \in \mathbb{N}$
- Can we encode computable functions $f : \mathbb{N}^k \to \mathbb{N}$?
- Let **f** be the encoding of f
- We want $\mathbf{f}\,\text{«}n_1\text{»} \cdots \text{«}n_k\text{»} \xrightarrow{*}_\beta \text{«}m\text{»}$ iff $f(n_1, \cdots, n_k) = m$
  1. If $f(n_1, \cdots, n_k)$ is defined, then $\mathbf{f}\,\text{«}n_1\text{»} \cdots \text{«}n\text{»}_k \xrightarrow{*}_\beta \text{«}f(n_1, \cdots, n_k)\text{»}$

Computability

- Church numerals encode $n \in \mathbb{N}$
- Can we encode computable functions $f : \mathbb{N}^k \to \mathbb{N}$?
- Let **f** be the encoding of f
- We want $\mathbf{f}\,\text{«}n_1\text{»} \cdots \text{«}n_k\text{»} \xrightarrow{*}_\beta \text{«}m\text{»}$ iff $f(n_1, \cdots, n_k) = m$
  1. If $f(n_1, \cdots, n_k)$ is defined, then $\mathbf{f}\,\text{«}n_1\text{»} \cdots \text{«}n\text{»}_k \xrightarrow{*}_\beta \text{«}f(n_1, \cdots, n_k)\text{»}$
  2. If $\mathbf{f}\,\text{«}n_1\text{»} \cdots \text{«}n\text{»}_k \xrightarrow{*}_\beta \text{«}m\text{»}$ and $\mathbf{f}\,\text{«}n_1\text{»} \cdots \text{«}n\text{»}_k \xrightarrow{*}_\beta \text{«}p\text{»}$, then $m = p$

Computability

- Church numerals encode $n \in \mathbb{N}$
- Can we encode computable functions $f : \mathbb{N}^k \to \mathbb{N}$?
- Let $\mathbf{f}$ be the encoding of f
- We want $\mathbf{f}\,\text{«}n_1\text{»} \cdots \text{«}n_k\text{»} \xrightarrow{*}_\beta \text{«}m\text{»}$ iff $f(n_1, \cdots, n_k) = m$
  1. If $f(n_1, \cdots, n_k)$ is defined, then $\mathbf{f}\,\text{«}n_1\text{»} \cdots \text{«}n\text{»}_k \xrightarrow{*}_\beta \text{«}f(n_1, \cdots, n_k)\text{»}$
  2. If $\mathbf{f}\,\text{«}n_1\text{»} \cdots \text{«}n\text{»}_k \xrightarrow{*}_\beta \text{«}m\text{»}$ and $\mathbf{f}\,\text{«}n_1\text{»} \cdots \text{«}n\text{»}_k \xrightarrow{*}_\beta \text{«}p\text{»}$, then $m = p$
  3. If $f(n_1, \ldots, n_k)$ is undefined, then $\neg\left(\mathbf{f}\,\text{«}n\text{»}_1 \cdots \text{«}n\text{»}_k \xrightarrow{*}_\beta \text{«}m\text{»}\right)$ for any m

Computability

- Church numerals encode $n \in \mathbb{N}$
- Can we encode computable functions $f : \mathbb{N}^k \to \mathbb{N}$?
- Let $\mathbf{f}$ be the encoding of f
- We want $\mathbf{f}\,\text{«}n_1\text{»} \cdots \text{«}n_k\text{»} \xrightarrow{*}_\beta \text{«}m\text{»}$ iff $f(n_1, \cdots, n_k) = m$
  1. If $f(n_1, \cdots, n_k)$ is defined, then $\mathbf{f}\,\text{«}n_1\text{»} \cdots \text{«}n\text{»}_k \xrightarrow{*}_\beta \text{«}f(n_1, \cdots, n_k)\text{»}$
  2. If $\mathbf{f}\,\text{«}n_1\text{»} \cdots \text{«}n\text{»}_k \xrightarrow{*}_\beta \text{«}m\text{»}$ and $\mathbf{f}\,\text{«}n_1\text{»} \cdots \text{«}n\text{»}_k \xrightarrow{*}_\beta \text{«}p\text{»}$, then $m = p$
  3. If $f(n_1, \ldots, n_k)$ is undefined, then $\neg\left(\mathbf{f}\,\text{«}n\text{»}_1 \cdots \text{«}n\text{»}_k \xrightarrow{*}_\beta \text{«}m\text{»}\right)$ for any m
- We need a syntax for computable functions!

Recursive functions

- Recursive functions [**Dedekind**, **Skolem**, **Gödel**, **Kleene**]

Recursive functions

- Recursive functions [**Dedekind**, **Skolem**, **Gödel**, **Kleene**]
  - Equivalent to Turing machines

Recursive functions

- Recursive functions [**Dedekind**, **Skolem**, **Gödel**, **Kleene**]
  - Equivalent to Turing machines
- $f : \mathbb{N}^k \longrightarrow \mathbb{N}$ is obtained by **composition** from $g : \mathbb{N}^l \longrightarrow \mathbb{N}$ and $h_1, \ldots, h_l : \mathbb{N}^k \longrightarrow \mathbb{N}$ if

$$f(\vec{n}) = g(h_1(\vec{n}), \ldots, h_l(\vec{n}))$$

Recursive functions

- Recursive functions [**Dedekind**, **Skolem**, **Gödel**, **Kleene**]
  - Equivalent to Turing machines
- $f : \mathbb{N}^k \to \mathbb{N}$ is obtained by **composition** from $g : \mathbb{N}^l \to \mathbb{N}$ and $h_1, \ldots, h_l : \mathbb{N}^k \to \mathbb{N}$ if

$$f(\vec{n}) = g(h_1(\vec{n}), \ldots, h_l(\vec{n}))$$

- **Notation**: $f = g \circ (h_1, h_2, \ldots, h_l)$

Recursive functions

- $f : \mathbb{N}^{k+1} \longrightarrow \mathbb{N}$ is obtained by **primitive recursion** from $g : \mathbb{N}^k \longrightarrow \mathbb{N}$ and $h : \mathbb{N}^{k+2} \longrightarrow \mathbb{N}$ if

$$f(0, \vec{n}) = g(\vec{n})$$

$$f(i+1, \vec{n}) = h(i, f(i, \vec{n}), \vec{n})$$

Recursive functions

- $f : \mathbb{N}^{k+1} \longrightarrow \mathbb{N}$ is obtained by **primitive recursion** from $g : \mathbb{N}^k \longrightarrow \mathbb{N}$ and $h : \mathbb{N}^{k+2} \longrightarrow \mathbb{N}$ if

$$f(0, \vec{n}) = g(\vec{n})$$

$$f(i+1, \vec{n}) = h(i, f(i, \vec{n}), \vec{n})$$

- **Note** If g and h are total functions, so is f

Recursive functions

- $f : \mathbb{N}^{k+1} \longrightarrow \mathbb{N}$ is obtained by **primitive recursion** from $g : \mathbb{N}^k \longrightarrow \mathbb{N}$ and $h : \mathbb{N}^{k+2} \longrightarrow \mathbb{N}$ if

$$f(0, \vec{n}) = g(\vec{n})$$

$$f(i+1, \vec{n}) = h(i, f(i, \vec{n}), \vec{n})$$

- **Note** If g and h are total functions, so is f
- Equivalent to a `for` loop:

```
result = g(n1, ..., nk);          // f(0, n1, ..., nk)
for (i = 0; i < n; i++) {         // computing f(i+1, n1, ..., nk)
   result = h(i, result, n1, ..., nk);
}
return result;
```

Recursive functions

- $f : \mathbb{N}^k \to \mathbb{N}$ is obtained by **μ-recursion** or **minimization** from $g : \mathbb{N}^{k+1} \to \mathbb{N}$ if

$$f(\vec{n}) = \begin{cases} i & \text{if } g(i,\vec{n}) = 0 \text{ and } \forall j < i : g(j,\vec{n}) > 0 \\ \text{undefined} & \text{otherwise} \end{cases}$$

Recursive functions

- $f : \mathbb{N}^k \longrightarrow \mathbb{N}$ is obtained by **μ-recursion** or **minimization** from $g : \mathbb{N}^{k+1} \longrightarrow \mathbb{N}$ if

$$f(\vec{n}) = \begin{cases} i & \text{if } g(i,\vec{n}) = 0 \text{ and } \forall j < i : g(j,\vec{n}) > 0 \\ \text{undefined} & \text{otherwise} \end{cases}$$

- **Notation**: $f(\vec{n}) = \mu i\,(\,g(i,\vec{n}) = 0)$

Recursive functions

- $f : \mathbb{N}^k \longrightarrow \mathbb{N}$ is obtained by **μ-recursion** or **minimization** from $g : \mathbb{N}^{k+1} \longrightarrow \mathbb{N}$ if

$$f(\vec{n}) = \begin{cases} i & \text{if } g(i,\vec{n}) = 0 \text{ and } \forall j < i : g(j,\vec{n}) > 0 \\ \text{undefined} & \text{otherwise} \end{cases}$$

- **Notation**: $f(\vec{n}) = \mu i\,(\,g(i,\vec{n}) = 0)$
- f need not be total even if g is

Recursive functions

- $f : \mathbb{N}^k \to \mathbb{N}$ is obtained by **μ-recursion** or **minimization** from $g : \mathbb{N}^{k+1} \to \mathbb{N}$ if

$$f(\vec{n}) = \begin{cases} i & \text{if } g(i,\vec{n}) = 0 \text{ and } \forall j < i : g(j,\vec{n}) > 0 \\ \text{undefined} & \text{otherwise} \end{cases}$$

- **Notation**: $f(\vec{n}) = \mu i\,(\,g(i,\vec{n}) = 0)$
- f need not be total even if g is
- If $f(\vec{n}) = i$, then $g(j,\vec{n})$ is defined for all $j \leqslant i$

Recursive functions

- $f : \mathbb{N}^k \to \mathbb{N}$ is obtained by **μ-recursion** or **minimization** from $g : \mathbb{N}^{k+1} \to \mathbb{N}$ if

$$f(\vec{n}) = \begin{cases} i & \text{if } g(i,\vec{n}) = 0 \text{ and } \forall j < i : g(j,\vec{n}) > 0 \\ \text{undefined} & \text{otherwise} \end{cases}$$

Recursive functions

- $f : \mathbb{N}^k \to \mathbb{N}$ is obtained by **μ-recursion** or **minimization** from $g : \mathbb{N}^{k+1} \to \mathbb{N}$ if

$$f(\vec{n}) = \begin{cases} i & \text{if } g(i,\vec{n}) = 0 \text{ and } \forall j < i : g(j,\vec{n}) > 0 \\ \text{undefined} & \text{otherwise} \end{cases}$$

- Equivalent to a `while` loop:

```
i = 0;
while (g(i, n1, ..., nk) > 0) {
    i = i + 1;
}
return i;
```

Recursive functions

- The class of **primitive recursive functions** is the smallest class of functions

Recursive functions

- The class of **primitive recursive functions** is the smallest class of functions
  1. containing the **initial functions**

Recursive functions

- The class of **primitive recursive functions** is the smallest class of functions
  1. containing the **initial functions**

     Zero $Z(n) = 0$

Recursive functions

- The class of **primitive recursive functions** is the smallest class of functions
  1. containing the **initial functions**

$$\text{Zero} \quad Z(n) = 0$$

$$\text{Successor} \quad S(n) = n + 1$$

Recursive functions

- The class of **primitive recursive functions** is the smallest class of functions
  1. containing the **initial functions**

$$\begin{aligned} \text{Zero} \quad & Z(n) = 0 \\ \text{Successor} \quad & S(n) = n + 1 \\ \text{Projection} \quad & \Pi^k_i(n_1, \ldots, n_k) = n_i \end{aligned}$$

Recursive functions

- The class of **primitive recursive functions** is the smallest class of functions
  1. containing the **initial functions**

$$\begin{aligned} \text{Zero} \quad & Z(n) = 0 \\ \text{Successor} \quad & S(n) = n + 1 \\ \text{Projection} \quad & \Pi_i^k(n_1, \ldots, n_k) = n_i \end{aligned}$$

  2. closed under composition and primitive recursion

Recursive functions

- The class of **primitive recursive functions** is the smallest class of functions
  1. containing the **initial functions**

     Zero $Z(n) = 0$

     Successor $S(n) = n + 1$

     Projection $\Pi_i^k(n_1, \ldots, n_k) = n_i$

  2. closed under composition and primitive recursion
- The class of **(partial) recursive functions** is the smallest class of functions

Recursive functions

- The class of **primitive recursive functions** is the smallest class of functions
  1. containing the **initial functions**

     Zero $Z(n) = 0$

     Successor $S(n) = n + 1$

     Projection $\Pi_i^k(n_1, \ldots, n_k) = n_i$

  2. closed under composition and primitive recursion
- The class of **(partial) recursive functions** is the smallest class of functions
  1. containing the initial functions

Recursive functions

- The class of **primitive recursive functions** is the smallest class of functions
  1. containing the **initial functions**

     Zero $Z(n) = 0$

     Successor $S(n) = n + 1$

     Projection $\Pi_i^k(n_1, \ldots, n_k) = n_i$

  2. closed under composition and primitive recursion
- The class of **(partial) recursive functions** is the smallest class of functions
  1. containing the initial functions
  2. closed under composition, primitive recursion and minimization

Recursive functions: Examples

- $f(n) = n + 2$ is $S \circ S$

Recursive functions: Examples

- $f(n) = n + 2$ is $S \circ S$
- $plus(n, m) = n + m$ is got by primitive recursion from $g = \Pi^1_1$ and $h = S \circ \Pi^3_2$

$$
\begin{array}{lll}
plus(0, m) & = g(m) & = \Pi^1_1(m) \\
 & & = m \\
plus(n+1, m) & = h(n, plus(n, m), m) & \\
 & = (S \circ \Pi^3_2)(n, plus(n, m), m) & = S(plus(n, m)) \\
 & & = (n + m) + 1 \\
 & & = (n + 1) + m
\end{array}
$$

Recursive functions: Examples

- $mult(n, m) = nm$ is got by primitive recursion from $g = Z$ and $h = plus \circ (\Pi^3_2, \Pi^3_3)$

$$
\begin{aligned}
mult(0, m) &= g(m) &&= Z(m) \\
&&&= 0 \\
mult(n+1, m) &= h(n, mult(n, m), m) \\
&= (plus \circ (\Pi^3_2, \Pi^3_3))(n, mult(n, m), m) \\
&&&= nm + m \\
&&&= (n+1)m
\end{aligned}
$$

Recursive functions: Examples

- $exp(n, m) = m^n$ is got by primitive recursion from $g = S \circ Z$ and $h = mult \circ (\Pi^3_2, \Pi^3_3)$

$$
\begin{aligned}
exp(0, m) &= g(m) &&= (S \circ Z)(m) \\
&&&= 1 \\
exp(n+1, m) &= h(n, exp(n, m), m) \\
&= (mult \circ (\Pi^3_2, \Pi^3_3))(n, exp(n, m), m) \\
&&&= m^n \cdot m \\
&&&= m^{n+1}
\end{aligned}
$$

Recursive functions: Examples

- Define $pred(n) = \begin{cases} 0 & \text{if } n = 0 \\ n-1 & \text{otherwise} \end{cases}$

Recursive functions: Examples

- Define $pred(n) = \begin{cases} 0 & \text{if } n = 0 \\ n-1 & \text{otherwise} \end{cases}$
- $pred(n) = f(n,n)$ where f is got by primitive recursion from $g = Z$ and $h = \Pi^3_1$

$$\begin{array}{llll} f(0,m) & = g(m) & = Z(m) & = 0 \\ f(n+1,m) & = h(n,f(n,m),m) & = \Pi^3_1(n,f(n,m),m) & = n \end{array}$$

$$\begin{array}{lll} pred(0) & = f(0,0) & = 0 \\ pred(n+1) & = f(n+1,n+1) & = n \end{array}$$

Recursive functions: Examples

- Define $m \dot{-} n = \begin{cases} 0 & \text{if } m \leqslant n \\ m - n & \text{otherwise} \end{cases}$

Recursive functions: Examples

- Define $m \dot{-} n = \begin{cases} 0 & \text{if } m \leqslant n \\ m - n & \text{otherwise} \end{cases}$
- $m \dot{-} n = f(n, m)$ where f is got by primitive recursion from $g = \Pi^1_1$ and $h = pred \circ \Pi^3_2$

$$
\begin{array}{lllll}
f(0, m) & = g(m) & = \Pi^1_1(m) & & \\
 & & = m & = m \dot{-} 0 & \\
f(n+1, m) & = h(n, f(n, m), m) & = pred(\Pi^3_2(n, f(n, m), m)) & & \\
 & & = pred(m \dot{-} n) & = m \dot{-} (n+1) &
\end{array}
$$

Recursive functions: Examples

- Define $m \dot{-} n = \begin{cases} 0 & \text{if } m \leqslant n \\ m - n & \text{otherwise} \end{cases}$
- $m \dot{-} n = f(n, m)$ where f is got by primitive recursion from $g = \Pi^1_1$ and $h = pred \circ \Pi^3_2$

$$\begin{array}{lllll} f(0,m) & = g(m) & = \Pi^1_1(m) & & \\ & & = m & = m \dot{-} 0 & \\ f(n+1,m) & = h(n, f(n,m), m) & = pred(\Pi^3_2(n, f(n,m), m)) & & \\ & & = pred(m \dot{-} n) & = m \dot{-} (n+1) & \end{array}$$

- **Note the recursion on the second argument!**

Recursive functions: Examples

- Define $m \dot{-} n = \begin{cases} 0 & \text{if } m \leqslant n \\ m - n & \text{otherwise} \end{cases}$
- $m \dot{-} n = f(n, m)$ where f is got by primitive recursion from $g = \Pi^1_1$ and $h = pred \circ \Pi^3_2$

$$\begin{aligned} f(0, m) &= g(m) &&= \Pi^1_1(m) \\ &&&= m &&= m \dot{-} 0 \\ f(n+1, m) &= h(n, f(n, m), m) &&= pred(\Pi^3_2(n, f(n, m), m)) \\ &&&= pred(m \dot{-} n) &&= m \dot{-} (n+1) \end{aligned}$$

- **Note the recursion on the second argument!**
- $f(m) = \log_2 m$ is defined by minimization from $g(n, m) = m \dot{-} 2^n$

Recursive functions: Examples

- Define $m \dot{-} n = \begin{cases} 0 & \text{if } m \leqslant n \\ m - n & \text{otherwise} \end{cases}$
- $m \dot{-} n = f(n, m)$ where f is got by primitive recursion from $g = \Pi^1_1$ and $h = pred \circ \Pi^3_2$

$$\begin{aligned} f(0, m) &= g(m) &&= \Pi^1_1(m) \\ &&&= m &&= m \dot{-} 0 \\ f(n+1, m) &= h(n, f(n, m), m) &&= pred(\Pi^3_2(n, f(n, m), m)) \\ &&&= pred(m \dot{-} n) &&= m \dot{-} (n+1) \end{aligned}$$

- **Note the recursion on the second argument!**
- $f(m) = \log_2 m$ is defined by minimization from $g(n, m) = m \dot{-} 2^n$
  - First n such that $m \leqslant 2^n$ is $\lceil \log_2 m \rceil$