Logic, Computability and Incompleteness

Recursive Functions

Introduction

- Recursive Functions constitute a very broad class, expressed explicitly in terms mathematical equations.
- Functions in this class include members of the familiar series addition, multiplication, exponentiation…
- Indeed, the class is so broad it seems intuitively plausible that **all** *effectively computable functions* are recursive.
- We will return to recursive functions again when we look at basic number theory formalized in first order logic.
- We'll first define this class of functions, and then provide further evidence in support of the Church-Turing Thesis by showing that all recursive functions are Abacus computable, i.e. $R \subseteq A$

Primitive Recursive Functions

- We begin by defining the proper subclass of Primitive Recursive Functions.
- First we specify an initial stock of basic functions belonging to the class of primitive recursive functions,

  and then define 2 types of operation which yield members of that class when applied to members of that class.
- There are 3 distinct categories of basic functions:

  1) **zero function**

  2) **successor function**

  3) **projection functions**

Basic Functions

1) zero function, for all natural numbers x,

$$\mathbf{z}(x) = 0.$$

2) successor function, for all natural numbers x,

$\mathbf{s}(x)$ = the natural number which is the successor of x

3) projection (or identity) functions, come in assorted arities:

$\mathbf{id}^{\mathbf{1}}_{\mathbf{1}}(x) = x$, $\mathbf{id}^{\mathbf{2}}_{\mathbf{1}}(x, y) = x$, $\mathbf{id}^{\mathbf{2}}_{\mathbf{2}}(x, y) = y$

In general $\mathbf{id}^{\boldsymbol{n}}_{\boldsymbol{i}}(x_1,\ldots, x_i,\ldots, x_n) = x_i$

- All such basic functions are primitive recursive.

Operations

- From the basic functions we can form new primitive recursive functions through the operations of composition and primitive recursion.
- Composition: if f is a function of m arguments and $g_1, \ldots\ g_m$ are functions of n arguments, then the composition h is the function of n arguments such that

$$h^n\,(x_1, \ldots, x_n) = f^{\,m}\,(g_1(x_1, \ldots, x_n)\ , \ldots, g_m\,(x_1, \ldots, x_n))$$

- So if f and the g's are primitive recursive, then so is the composition h, written $h = \mathbf{Cn[}\ f, g_1, \ldots, g_m\ \mathbf{]}$

Examples

- Want to define 1-place **p.r.** function h^1 such that $h^1(x) = x + 3$.

  $h^1 = \mathbf{Cn}[\mathbf{s},\ \mathbf{Cn}[\mathbf{s},\mathbf{s}]]$ where $\mathbf{Cn}[\mathbf{s},\mathbf{s}] = \mathbf{s}(\mathbf{s}(x))$ and so

  $$\mathbf{Cn}[\mathbf{s},\ \mathbf{Cn}[\mathbf{s},\mathbf{s}]] = \mathbf{s}(\mathbf{s}(\mathbf{s}(x)))$$

- Want to define 3-place **p.r.** function h^3 such that

  $$h^3(x_1, x_2, x_3) = x_2 + 3$$

  $$h^3 = \mathbf{Cn}[h^1, \mathbf{id}^3{}_2] = \mathbf{Cn}[\mathbf{Cn}[\mathbf{s},\ \mathbf{Cn}[\mathbf{s},\mathbf{s}]],\ \mathbf{id}^3{}_2]$$

  $$\mathbf{Cn}[h^1, \mathbf{id}^3{}_2](x_1, x_2, x_3) = h^1(\mathbf{id}^3{}_2(x_1, x_2, x_3))$$

  $$= h^1(x_2) = \mathbf{s}(\mathbf{s}(\mathbf{s}(x_2))) = x_2 + 3$$

Operations

- Primitive recursion: will first specify in terms of a schema for defining a 2-place function $h(x,y)$ in terms of a 1-place function f and a 3-place function g.

$$h(x, 0) = f(x)$$

$$h(x, \mathbf{s}(y)) = g(x, y, h(x,y))$$

- So, given **p.r.** functions f and g, this definition will recursively generate all values of h for a given argument x, starting with $y = 0$ and then using the previous value to define the next one.
- First yields $h(x, 0)$ then $h(x, 1)$, $h(x, 2)$, …
- So given any pair of numbers x, y this procedure will compute the value $h(x, y)$ in $y + 1$ iterations.

Primitive Recursion

- Notation: $h = \mathbf{Pr}[\, f, g \,]$
- Example: *informal* recursive definition of '+' in terms of **s**

$$x + 0 = x$$

$$x + \mathbf{s}(y) = \mathbf{s}(x + y)$$

Need to put in *official* format

$$\mathbf{sum}(x, 0) = f(x)$$

$$\mathbf{sum}(x, \mathbf{s}(y)) = g(x, y, \mathbf{sum}(x, y))$$

So let $f = \mathbf{id}^1{}_1$ and $g = \mathbf{Cn}[\mathbf{s}, \mathbf{id}^3{}_3]$

Then $\mathbf{sum}(x, 0) = \mathbf{id}^1{}_1(x)$

$$\mathbf{sum}(x, \mathrm{s}(y)) = \mathbf{Cn}[\mathbf{s}, \mathbf{id}^3{}_3](x, y, \mathbf{sum}(x, y))$$

Examples

- Given our formal recursive specification of **sum** as

$$\text{(i)}\quad \mathbf{sum}\,(x, 0) = \mathbf{id}^1{}_1\,(x)$$

$$\text{(ii)}\quad \mathbf{sum}\,(x, \mathbf{s}(y)) = \mathbf{Cn[s, id^3{}_3]}\,(x, y, \mathbf{sum}\,(x, y))$$

We can see that

$$\text{(i)}\quad \mathbf{sum}\,(x, 0) = \mathbf{id}^1{}_1\,(x) = x \qquad \text{and}$$

$$\begin{aligned} \text{(ii)}\quad \mathbf{sum}\,(x, \mathbf{s}(y)) &= \mathbf{Cn[s, id^3{}_3]}\,(x, y, \mathbf{sum}\,(x, y)) \\ &= \mathbf{s(id^3{}_3}\,(x, y, \mathbf{sum}\,(x, y))) \\ &= \mathbf{s(sum}\,(x, y)) \end{aligned}$$

so that $\mathbf{sum}\,(x, 0) = x$ and $\mathbf{sum}\,(x, \mathbf{s}(y)) = \mathbf{s(sum}\,(x, y))$

Officially: $\mathbf{sum} = \mathbf{Pr}[\mathbf{id}^1{}_1, \mathbf{Cn}[\mathbf{s}, \mathbf{id}^3{}_3]]$

Sample (informal) Computation with **sum**

- Recursively compute the value 2+3, *i.e* **sum** (2, 3):

$$\mathbf{sum}\,(2,0) = \mathbf{id}^1_1\,(2) = 2$$

$$\mathbf{sum}\,(2, \mathbf{s}(0)) = \mathbf{s}(\mathbf{sum}(2,0)) = \mathbf{s}(2) = 3$$

$$\mathbf{sum}\,(2, \mathbf{s}(\mathbf{s}(0))) = \mathbf{s}(\mathbf{sum}(2, \mathbf{s}(0))) = \mathbf{s}(3) = 4$$

$$\mathbf{sum}\,(2, \mathbf{s}(\mathbf{s}(\mathbf{s}(0)))) = \mathbf{s}(\mathbf{sum}(2, \mathbf{s}(\mathbf{s}(0)))) = \mathbf{s}(4) = 5$$

- It’s mechanical!

Examples

- Product expressed recursively in terms of **sum**.
- Informally:

$$x \cdot 0 = 0$$

$$x \cdot \mathbf{s}(y) = x + (x \cdot y)$$

Need to put in *official* format using **p.r.** functions f and g

$$\mathbf{prod}(x, 0) = f\,(x)$$

$$\mathbf{prod}(x, \mathrm{s}(y)) = g\,(x, y, \mathbf{prod}(x, y))$$

So let $f = \mathbf{z}$ and $g = \mathbf{Cn[sum, id^3_1, id^3_3]}$

Then $\mathbf{prod}\,(x, 0) = \mathbf{z}(x)$

$$\mathbf{prod}\,(x, \mathrm{s}(y)) = \mathbf{Cn[sum, id^3_1, id^3_3]}\,(x, y, \mathbf{prod}\,(x, y))$$

Product Defined Recursively in Terms of **sum**

- Given our formal recursive specification of **prod** as

  (i) $\mathbf{prod}\,(x, 0) = \mathbf{z}(x)$

  (ii) $\mathbf{prod}\,(x, \mathbf{s}(y)) = \mathbf{Cn[sum, id^3_1, id^3_3]}\,(x, y, \mathbf{prod}\,(x, y))$

We can see that

(i) $\mathbf{prod}\,(x, 0) = \mathbf{z}(x) = 0$ and

(ii) $\mathbf{prod}\,(x, \mathbf{s}(y)) = \mathbf{Cn[sum, id^3_1, id^3_3]}\,(x, y, \mathbf{prod}\,(x, y))$

$= \mathbf{sum}(\mathbf{id^3_1}\,(x, y, \mathbf{prod}\,(x, y)),\ \mathbf{id^3_3}(x, y, \mathbf{prod}\,(x, y)))$

$= \mathbf{sum}\,(x, \mathbf{prod}(x, y))$

so that $\mathbf{prod}(x, 0) = 0$ and $\mathbf{prod}\,(x, \mathbf{s}(y)) = \mathbf{sum}(x, \mathbf{prod}(x, y))$.

- **Officially:** $\mathbf{prod} = \mathbf{Pr}[\mathbf{z}, \mathbf{Cn}[\mathbf{sum}, \mathbf{id^3_1}, \mathbf{id^3_3}]]$

Different Arities

- The **p.r.** schema has been given in terms of defining a 2-place function, but we can generalize to cover functions of any arity.
- For example, a 3-place function $h(x_1,x_2, y)$ can be defined in terms of a 2-place function f and a 4-place function g such that

$$h(x_1,x_2, 0) = f(x_1,x_2)$$

$$h(x_1,x_2, \mathbf{s}(y)) = g(x_1,x_2, y, h(x_1,x_2, y))$$

- And a 1-place function $h(y)$ can be defined in terms of

  a constant **c** (i.e. a 0-place function) and

  a 2-place function $g(y,x)$ such that

$$h(0) = \mathbf{c}$$

$$h(\mathbf{s}(y)) = g(y,h(y))$$

Different Arities

- **So in the general case**:

  an n-place function h^n is defined in terms of

  an n–1 place function f^{n-1} and

  an n+1 place function g^{n+1},

  such that f^{n-1} and g^{n+1} are both primitive recursive and

$$h^n(x_1,\ldots, x_{n-1}, 0) = f^{n-1}(x_1,\ldots, x_{n-1})$$

$$h^n(x_1,\ldots, x_{n-1}, \mathbf{s}(y)) = g^{n+1}(x_1,\ldots, x_{n-1}, y, h^n(x_1,\ldots, x_{n-1}, y))$$

written: $h = \mathbf{Pr}[f, g]$

Recursive Functions

- Now we will expand to the wider class of recursive functions:

  retain the same set of base functions, and all functions obtainable through finite applications of composition and primitive recursion

  plus the new operation of minimization.
- Minimization, when applied to a function f of n+1 arguments, yields the n-place function **Mn**[f] such that:

  Mn[f]$(x_1, \ldots, x_n)$ = {the least y for which $f(x_1, \ldots, x_n, y) = 0$

  = {undefined if $f(x_1, \ldots, x_n, y) = 0$ for no y

Minimization

All **p.r.** functions are total, but **Mn** can yield partial functions.

Mn[**sum**] is a partial function:

Mn[**sum**](x) = {0, if $x = 0$

= {undefined otherwise

In effect, **Mn** allows unbounded search – can't necessarily tell in a finite number of steps whether or not **Mn**[f] is defined on a given input.

If it is, then value will be computed in a finite number of steps.

If it is not, then computation won't halt.

Hence <u>bounded</u> Minimization is a **p.r.** operation (as we'll see a bit later).

More Primitive Recursive Functions

- We need recursive functions as defined through the operation of minimization in order to characterize the entire class of computable functions.
- However, the proper subclass of **p.r. functions** is quite vast and we will now continue investigating its members.
- Basic strategy is to use previously defined **p.r. functions** as ingredients for constructing progressively more complex **p.r. functions**.
- We've seen **sum** defined as iterated **s**uccessor and **prod** defined as iterated **sum**. Can in turn define exponentiation, **exp**, as iterated **prod**:

More Primitive Recursive Functions

- Intuitively, exp(x, y) = x^y

  which corresponds to the informal recursive specification:

  $$x^0 = 1$$

  $$x^{y+1} = x \cdot x^y$$

  Or more officially $\mathbf{exp}(x, 0) = 1$

  $$\mathbf{exp}(x, \mathbf{s}(y)) = x \cdot \mathbf{exp}(x,y)$$

Need *fully official* format using **p.r.** functions f and g

$$\mathbf{exp}(x, 0) = f(x)$$

$$\mathbf{exp}(x, \mathbf{s}(y)) = g(x, y, \mathbf{exp}(x, y))$$

More Primitive Recursive Functions

let f = $\mathbf{Cn}[\mathbf{s}, \mathbf{z}]$ and g = $\mathbf{Cn}\,[\mathbf{prod}, \mathbf{id}^3_1, \mathbf{id}^3_3]$

Then $\mathbf{exp}(x, 0) = \mathbf{Cn}[\mathbf{s}, \mathbf{z}](x)$

$\mathbf{exp}(x, \mathbf{s}(y)) = \mathbf{Cn}[\mathbf{prod}, \mathbf{id}^3_1, \mathbf{id}^3_3]\,(x, y, \mathbf{exp}(x, y))$

Officially: $\mathbf{exp} = \mathbf{Pr}[\mathbf{Cn}[\mathbf{s}, \mathbf{z}], \mathbf{Cn}[\mathbf{prod}, \mathbf{id}^3_1, \mathbf{id}^3_3]]$

- The predecessor of x, written $\mathbf{pred}(x)$, is the number immediately preceding it (except we let $\mathbf{pred}(0) = 0$).

Informally, $\mathbf{pred}(0)= 0$, $\mathbf{pred}(\mathbf{s}(y)) = y$

So $\mathbf{pred}(0) = 0$

$\mathbf{pred}(\mathbf{s}(y)) = \mathbf{id}^2_1(y, \mathbf{pred}(y))$

Officially: $\mathbf{pred} = \mathbf{Pr}[0, \mathbf{id}^2_1]$

More Primitive Recursive Functions

- The arithmetical difference between x and y, written **dif**(x,y) (and abbreviated as $x \dot{-} y$) is defined as $x - y$ if $x \geq y$ and 0 otherwise.

  So, in abbreviated format $\quad x \dot{-} 0 = x,$

  $$x \dot{-} \mathbf{s}(y) = \mathbf{pred}(x \dot{-} y)$$

  More formally, $\mathbf{dif}(x,0) = \mathbf{id}^1_1(x)$

  $$\mathbf{dif}(x, \mathbf{s}(y)) = \mathbf{Cn}[\mathbf{pred}, \mathbf{id}^3_3]\,(x, y, \mathbf{dif}(x,y))$$

  Officially: $\mathbf{dif} = \mathbf{Pr}[\mathbf{id}^1_1, \mathbf{Cn}[\mathbf{pred}, \mathbf{id}^3_3]]$

- The 1-place function signum is such that signum(0) = 0 and signum(y) = 1 otherwise.

  Expressed informally (as a *composition*) $\mathbf{sg}(y) = 1 \dot{-} (1 \dot{-} y)$

More Primitive Recursive Functions

- The reverse signum function $\underline{\mathbf{sg}}(y) = 1 \dot{-} y$
- Definition by cases. Suppose f is defined in the form:

$$f(x,y) = \{g_1(x,y) \quad \text{if } C_1$$

$$\vdots$$

$$= \{g_n(x,y) \quad \text{if } C_n$$

where $C_1, \ldots, C_n$ are mutually exclusive, collectively exhaustive conditions on x,y and $g_1, \ldots, g_n$ are **p.r.**

- The characteristic function of a condition C_i on x,y is a function c_i which takes the value 1

for argument pairs (x,y) which satisfy the condition,

and the value 0 for all other argument pairs.

More Primitive Recursive Functions

- If the characteristic functions $c_1, \ldots, c_n$ of the conditions $C_1, \ldots, C_n$ in the foregoing definition are **p.r.** then so is the function f,

  for it can be defined by composition out of the gs and cs as follows:

  $$f(x,y) = g_1(x,y) \cdot c_1(x,y) + \ldots + g_n(x,y) \cdot c_n(x,y)$$

  cool…..

- Example of definition by cases: **max**(x,y) = the larger of x,y

  So $\mathbf{max}(x,y) = \{x \text{ if } x \geq y$

  $= \{y \text{ if } x < y$

  In this case $g_1(x,y) = \mathbf{id}^2{}_1$,

  $g_2(x,y) = \mathbf{id}^2{}_2$,

More Primitive Recursive Functions

$c_1(x,y) = \underline{\mathbf{sg}}(y \dot{-} x)$ **=** 1 if x ≥ y and 0 otherwise

$c_2(x,y) = \mathbf{sg}(y \dot{-} x)$ **=** 1 if x < y and 0 otherwise

Putting these ingredients together:

$\mathbf{max}(x,y) = \mathbf{id}^2_1(x,y) \cdot \underline{\mathbf{sg}}(y \dot{-} x) + \mathbf{id}^2_2(x,y) \cdot \mathbf{sg}(y \dot{-} x)$

- Example: $\mathbf{max}(1,2) =$

$\mathbf{id}^2_1(1,2) \cdot \underline{\mathbf{sg}}(2 \dot{-} 1) + \mathbf{id}^2_2(1,2) \cdot \mathbf{sg}(2 \dot{-} 1)$

$= \quad 1 \cdot \underline{\mathbf{sg}}(1) \quad + \quad 2 \cdot \mathbf{sg}(1)$

$= \quad 1 \cdot 0 \quad + \quad 2 \cdot 1$

$= \quad 0 \quad + \quad 2$

$= \quad 2$

More Primitive Recursive Functions

- General sum: $g(x_1, \ldots, x_n, y) = {}_{i=0}{}^{y}\Sigma\ f(x_1, \ldots, x_n, i)$

  Recursive definition (with $f(x_1, \ldots, x_n, y)$ **p.r.**)

  $g(x_1, \ldots, x_n, 0) = f(x_1, \ldots, x_n, 0)$

  $g(x_1, \ldots, x_n, \mathbf{s}(y)) = g(x_1, \ldots, x_n, y) + f(x_1, \ldots, x_n, \mathbf{s}(y))$
- General product: $g(x_1, \ldots, x_n, y) = {}_{i=0}{}^{y}\Pi\ f(x_1, \ldots, x_n, i)$

  Recursive definition (with $f(x_1, \ldots, x_n, y)$ **p.r.**)

  $g(x_1, \ldots, x_n, 0) = f(x_1, \ldots, x_n, 0)$

  $g(x_1, \ldots, x_n, \mathbf{s}(y)) = g(x_1, \ldots, x_n, y) \cdot f(x_1, \ldots, x_n, \mathbf{s}(y))$

More Primitive Recursive Functions

- Logical composition of conditions:
- Negation

  If $c(x,y)$ is the characteristic function for condition C

  then $\underline{c}(x,y) = \underline{\mathbf{sg}}(c(x,y))$ is the characteristic function for $\neg\, C$
- Conjunction

  The characteristic function for $C_1 \wedge \ldots \wedge C_n$ is

  $c_1(x,y) \cdot \ldots \cdot c_n(x,y)$ [= 0 if any of the terms are 0]

  Since $\{\neg\, , \wedge\}$ is a truth-functionally adequate set of logical connectives,

  the above is sufficient to express **all** truth functional combinations of conditions.

More Primitive Recursive Functions

- For example, Disjunction

$C_1 \vee C_2 \equiv \neg(\neg C_1 \wedge \neg C_2)$

So the characteristic function of the disjunction of two conditions is

$c_d(x,y) = \underline{sg}(\underline{sg}(c_1(x,y)) \cdot \underline{sg}(c_2(x,y)))$

- Bounded quantification:
- Universal $\forall_i\,(i \leq y \rightarrow c(x, i))$

Characteristic function: $u(x,y) = {}_{i=0}{}^{y}\Pi\, c(x,i)$

- Existential $\exists_i\,(i \leq y \wedge c(x, i))$

Characteristic function: $e(x,y) = \mathbf{sg}({}_{i=0}{}^{y}\Sigma\, c(x,i))$

More Primitive Recursive Functions

- Bounded minimization: with $f(x_1, \ldots, x_n, y)$ **p.r.**

  $\mathbf{Mn}_w[f]$ = least y such that $0 \le y \le w$ and

  $f(x_1, \ldots, x_n, y) = 0$

  Definition by cases:

  $\mathbf{Mn}_w[f](x_1, \ldots, x_n)$

  $$= \{0 \text{ if } \forall y\,(y \le w \rightarrow f(x_1, \ldots, x_n, y) \ne 0$$

  $$= \{\sum_{i=0}^{w} \mathbf{sg}(\prod_{k=0}^{i} f(x_1, \ldots, x_n, k)) \text{ otherwise.}$$