Structured Programming and Recursive Functions

Notes by William J. Rapaport
(based on lectures by John Case)

Department of Computer Science & Engineering,
Department of Philosophy, and Center for Cognitive Science
State University of New York at Buffalo, Buffalo, NY 14260-2000
`rapaport@cse.buffalo.edu`, `http://www.cse.buffalo.edu/∼rapaport`

Last Update: 2 February 2004
Note: NEW or UPDATED material is highlighted

---

1. Structured Programming:

   (a) Classification of structured programs:

      i. Basic programs:

         A. the empty program =def **start halt.**

         B. the 1-operation program =def **start** F **halt.**
         (where 'F' is some primitive operation, e.g., an assignment statement).

      ii. Program constructors:

         Let π, π' be programs with 1 **halt**-operation each.
         Then new programs can be constructed by:

         A. linear concatenation =def **start** π; π' **halt.**

         B. conditional branching =def

            start
              if P
                then π
                else π'
            halt.

            (where 'P' is a Boolean test, i.e., a predicate; e.g., "$x > 0$").

         C. count looping (or "for-loop", or "bounded loop"):

            start
              while $y > 0$ **do**
                start
                  π;
                  $y \leftarrow y - 1$
                halt
            halt.

         D. while-looping (or "free" loop):

            start
              while P **do** π
            halt.

(b) Categories of structured programs (based on above classifications):

i. π *is a count-program*
(or a "for-program", or a "Bounded LOOP program") =def

A. π is a basic program, OR

B. π is constructed from count-programs by:

- linear concatenation, OR
- conditional branching, OR
- count looping

C. Nothing else is a count-program.

ii. π *is a while-program*
(or a "Free LOOP program") =def

A. π is a basic program, OR

B. π is constructed from while-programs by:

- linear concatenation, OR
- conditional branching, OR
- count-looping, OR
- while-looping

C. Nothing else is a while-program.

2. Recursive Functions

   (a) Classification of functions:

      i. Basic functions:

         A. *successor:* $S(x) = x + 1$

         B. *predecessor:* $P(x) = x \dot{-} 1$

            (where $a \dot{-} b$ =def $\begin{cases} a - b, & \text{if } a \geq b \\ 0, & \text{otherwise} \end{cases}$)

         C. *projection:* $P_k^j(x_1, \ldots, x_j, \ldots, x_k) = x_j$

      ii. Function constructors:

         A. *f is defined from $g, h_1, \ldots, h_m$ by generalized composition* =def

            $f(x_1, \ldots, x_k) = g(h_1(x_1, \ldots, x_k), \ldots, h_m(x_1, \ldots, x_k))$

            - Cf. linear concatenation (e.g., first compute h; then compute g)

         B. *f is defined from g, h, i by conditional definition* =def

            $f(x_1, \ldots, x_k) = \begin{cases} g(x_1, \ldots, x_k), & \text{if } x_i = 0 \\ h(x_1, \ldots, x_k), & \text{if } x_i > 0 \end{cases}$

            - Cf. conditional branch

         C. *f is defined from $g, h_1, \ldots, h_k, i$ by while-recursion* =def

            $f(x_1, \ldots, x_k) = \begin{cases} g(x_1, \ldots, x_k), & \text{if } x_i = 0 \\ f(h_1(x_1, \ldots, x_k), \ldots, h_k(x_1, \ldots, x_k)), & \text{if } x_i > 0 \end{cases}$

            - Cf. while-loop (e.g., while $x_i > 0$, compute f)

   (b) Categories of functions:

      i. *f is a while-recursive function* =def

         A. f is a basic function, OR

         B. f is defined from while-recursive functions by:

            - generalized composition, OR
            - conditional definition, OR
            - while-recursion

         C. Nothing else is while-recursive.

      ii. A. *f is defined from g, h by primitive recursion* =def

            $f(x_1, \ldots, x_k, y) = \begin{cases} g(x_1, \ldots, x_k), & \text{if } y = 0 \\ h(x_1, \ldots, x_k, f(x_1, \ldots, x_k, y - 1)), & \text{if } y > 0 \end{cases}$

            - Cf. count-loop (e.g., while $y > 0$, decrement y & compute f)

         B. *f is a primitive-recursive function* =def

            - f is a basic function, OR
            - f is defined from primitive-recursive functions by:
               – generalized composition, OR
               – primitive recursion
            - Nothing else is primitive-recursive.

iii. A. *f is defined from h by the μ-operator* [pronounced: "mu"-operator] =def
$f(x_1,\ldots,x_k) = \mu z[h(x_1,\ldots,x_k,z) = 0]$,
where:

$$\mu z[h(x_1,\ldots,x_k,z)=0] =\text{def} \begin{cases} min\{z : \begin{cases} h(x_1,\ldots,x_k,z)=0 \\ \textbf{and} \\ (\forall y < z)[h(x_1,\ldots,x_k,y) \text{ has a value}] \end{cases} \}, & \textbf{if} \text{ such } z \text{ exists} \\ \text{undefined}, & \textbf{if} \text{ no such } z \text{ exists} \end{cases}$$

B. *f is a partial-recursive function* =def

- *f* is a basic function, OR
- *f* is defined from partial-recursive functions by:
  - generalized composition, OR
  - primitive recursion, OR
  - the *μ*-operator
- Nothing else is partial-recursive.

C. *f is a recursive function* =def

- *f* is partial-recursive, AND
- *f* is a total function
  (i.e., defined ∀ elements of its domain)

3. The Connections:

$$\begin{array}{ccc} f \text{ is primitive-recursive} & \Leftrightarrow & f \text{ is count-program–computable} \\ \Downarrow & & \Downarrow \\ f \text{ is partial-recursive} & \Leftrightarrow & f \text{ is while-program–computable} \\ \Updownarrow & & \Updownarrow \\ & f \text{ is Turing-machine–computable} & \\ & \Updownarrow & \\ & f \text{ is } \lambda\text{-definable, etc.} & \end{array}$$

  rapaport@cse.buffalo.edu