CS 410/610, MTH 410/610
Theoretical Foundations of Computing

Fall Quarter 2010

Slides 6

Pascal Hitzler
Kno.e.sis Center
Wright State University, Dayton, OH
http://www.knoesis.org/pascal/

TOC: μ-Recursive Functions

Chapter 13 of [Sudkamp 2006].

1. **Primitive Recursive Functions**
2. **μ-Recursive Functions**

Basic functions

- successor function s: $s(x) = x+1$
- zero function z: $z(x) = 0$
- projection functions $p_i^{(n)}$: $p_i^{(n)}(x_1, \ldots, x_n) = x_i$ $(1 \leq I \leq n)$

These functions are all Turing computable.

Primitive recursion

Let g,h be total functions of arities n, respectively n+2.

Define the (n+1)-ary function f recursively as follows:

1. $f(x_1,\ldots,x_n,0) \quad = g(x_1,\ldots,x_n)$
2. $f(x_1,\ldots,x_n,y+1) \quad = h(\, x_1,\ldots,x_n,y,f(x_1,\ldots,x_n,y)\,)$

We say that f is obtained from g and h by ***primitive recursion***.

Note that the definition directly gives us an algorithm for computing f provided g and h can be computed.

Primitive recursive functions

A function is called *primitive recursive*, if if can be obtained from the basic functions (successor, zero, projections) by a finite number of applications of composition and primitive recursion.

Obviously, these are all computable.

They are also all total.

How far does this definition carry?

Examples

Addition:

add(x,0) = g(x) = x [g(x) = x]

add(x,y+1) = h(x,y,add(x,y)) = add(x,y)+1 [h(x,y,z) = z+1]

Multiplication:

mult(x,0) = 0 [g(x) = ?]

mult(x,y+1) = mult(x,y) + x [h(x,y,z) = ?]

[$h = add \circ (p_3^{(3)}, p_1^{(3)})$]

Examples

Factorial

fact(0) = 1
fact(y+1) = fact(y) · (y+1)

Predecessor,
sub(x,y) = max(0,x-y),

Exponentiation,
sign: s(x) = sub(x,sub(x,1))

characteristic functions of relations:
less than, equal to, greater than, not equal to

logical expressions (on 0,1):
not, and, or

Examples

Function definition by cases

$$f(x) = eq(x,0) \cdot 2 + eq(x,1) \cdot 5 + eq(x,2) \cdot 4 + gt(x,2) \cdot x$$

Hence:

If a primitive recursive function is altered on only a finite number of input values, then the resulting new function is also primitive recursive.

Minimization

A n-ary *predicate* p is the characteristic function of an n-ary relation.

Define $\mu z[p(x_1,\ldots,x_n,z)]$ to be the smallest non-negative integer z such that $p(x_1,\ldots,x_n,z) = 1$.

Note: functions defined using minimization are not necessarily primitive recursive.

E.g., $f(x) = \mu z[eq(x,z\cdot z)]$ is not even total.

Bounded minimization

Define

$$\mu^y z\,[p(x_1,\ldots,x_n,z)] = \begin{cases} z & \text{if } z \leq y \text{ and } z \text{ is the least non-negative integer with } p(x_1,\ldots,x_n,z) = 1 \\ y+1 & \text{otherwise} \end{cases}$$

Theorem

If $p(x_1,\ldots,x_n,y)$ is a primitive recursive predicate, then

f(x_1, …, x_n, y) = $\mu^y z\,[p(x_1,\ldots,x_n,z)]$

is primitive recursive.

Examples

Quotient

quo(x,y) = sg(y) · μ^xz[gt((z+1)·y, x)]

Remainder

rem(x,y) = sub(x,y·quo(x,y))

Divides

divides(x,y) = eq(rem(x,y),0)·sg(x)

Number of divisors

ndivisors(x,y) = divides(x,0)+…+divides(x,y)

Prime

prime(x) = eq(ndivisors(x,x),2)

More general bounded minimization

Let p be an (n+1)-ary primitive recursive predicate and let u be an n-ary primitive recursive function.

Then the function

$$f(x_1,\ldots,x_n) = \mu^{u(x_1,\ldots,x_n)\}}z[p(x_1,\ldots,x_n,z)]$$

is primitive recursive.

Proof?

Examples

xth prime:

$$pn(0) = 2$$

$$pn(x+1) = \mu^{fact(pn(x))+1} z[prime(z) \cdot gt(z, pn(x))]$$

TOC: μ-Recursive Functions

Chapter 13 of [Sudkamp 2006].

1. **Primitive Recursive Functions**
2. **μ-Recursive Functions**

Computable total functions

Theorem

There are computable total functions which are not primitive recursive.

Proof?

Proof

The set of one-variable primitive recursive functions can be enumerated (e.g., create all symbol strings and check each whether it is the definition of a primitive recursive function):

f1, f2, f2, …

The function g(i) = fi(i)+1 is total and computable.

However, there is no k with g = fk (diagonalization argument).

Hence g is not primitive recursive.

Something's wrong here – what is it?

Each total computable function can be represented by a TM.
Hence, we obtain a list of all total computable functions:

f1, f2, f3

The function g(i) = fi(i)+1 is total and computable.

However, there is no k with g = fk (diagonalization argument).

Hence, the set of total computable functions is uncoubable.

Hence, the set of all TMs is uncountable, which is impossible!

Hence

The set of all total computable functions cannot be enumerated algorithmically.

Ackermann's function

$A(0,y) = y+1$

$A(x+1,0) = A(x,1)$

$A(x+1,y+1) = A(x,A(x+1,y))$

This function is effectively computable. [why?]

This function is not primitive recursive:

It can be shown that for each primitive recursive function f there is some non-negative integer x such that $f(x) < A(x,x)$.

μ-Recursive Functions

A function is μ-recursive if and only if it can be defined using a finite number of the following:

- any primitive recursive function
- function composition
- primitive recursion
- (unbounded) minimization using a total μ-recursive predicate

Theorem

Every μ-recursive function is Turing computable

Theorem

Every Turing computable function is μ-recursive.

Proof?

Proof idea

- We simulate the computations of a given TM by means of a number-theoretic function.
- Machine computations become functions (this is called *arithmetization* of TMs).

- each configuration (state, tape head position, tape content) is uniquely encoded by a number
- A function tr maps configurations to configurations, according to the transition function of M.
  - The number encoding needs to be “smart” such that this is relatively easy to define. The key idea here is *Gödel numbering*. Furthermore, it must be done such that tr is μ-recursive (in fact, it is primitive recursive).
- Using minimization, one can combine tr iterations to the overall input-output function sim of M.

Church-Turing Thesis Revisited

A number-theoretic function is computable if, and only if, it is μ-recursive.