Prim Rec, Decidable, Undecidable, and Beyond

William Gasarch-U of MD

Primitive Recursive

An attempt to pin down the set of functions that are computable.
A function f is Primitive Recursive

1) $f(x_1, \ldots, x_n) = 0$ OR $f(x_1, \ldots, x_n) = x_i + c$, $c \in \mathbb{N}$.

2) If $g_1(x_1, \ldots, x_n)$, ..., $g_k(x_1, \ldots, x_n)$ are primrec, $h(x_1, \ldots, x_k)$ is primrec, then $f(x_1, \ldots, x_n) = h(g_1(x_1, \ldots, x_n), \ldots, g_k(x_1, \ldots, x_n))$ is primrec.

3) If $g(x_1, \ldots, x_{n-1})$ and $h(x_1, \ldots, x_{n+1})$ are primrec then the following function is primrec.

$$
\begin{aligned}
f(x_1, \ldots, x_{n-1}, 0) &= g(x_1, \ldots, x_{n-1}) \\
f(x_1, \ldots, x_{n-1}, x+1) &= h(x_1, \ldots, x_{n-1}, x, f(x_1, \ldots, x_{n-1}, x))
\end{aligned}
$$

Examples

Recursion is the KEY rule. Skipping details just show that a function is recursive and its prim rec.

$ADD(x, y+1) = ADD(x, y) + 1$. So ADD is primrec.

$MULT(x, y+1) = ADD(MULT(x, y), x)$. So MULT is primrec.

$EXP(x, y+1) = MULT(EXP(x, y), x)$. So EXP is primrec.

$TOW(x, y+1) = EXP(TWO(x, y), x)$. So TOW is primrec.

Less Obvious Examples

The following are primrec.

$MONUS(x, y) = x - y$ if $x \geq y$, 0 otherwise.

$PRIME(x) = 1$ if x is prime, 0 otherwise.

KEY: Virtually any function that comes up in normal mathematics is primitive recursive.

VOTE: Anything that is computable is primitive recursive? YES or NO?

Computable but NOT Prim Rec

Use the rules to NUMBER all of the one-variable prim rec functions $p_1, p_2, \ldots$.
Let

$$F(x) = p_x(x) + 1$$

F is NOT on the list.
This is a dumb-ass example.

Ackermann's Function

Ackermann's function is the function defined by

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

Easy: Ackermann's function is computable.

Known: Ackermann's function grows faster than any Prim Rec fn, hence Ack not Prim Rec.

Intuition: Prim Rec is a BOUNDED number of recursions.
Ackerman— the number of recursions depends on the input.

Is Any Attempt Useless

The Andy Parish's together come up with a different way to pin down the computable functions:

$$f_1, f_2, \ldots$$

PROBLEM:
$F(x) = f_x(x) + 1$ is NOT on the list

What do we really want?

Hidden Bad Assumption: We are insisting that all computable functions are computable on ALL inputs.

Definition: A partial function from A to B is a function whose domain is a subset of A. We might not know what that subset is.

Definition: f is a partial computable function from N to N if there exists a program M such that
If x is in the domain of f then $M(x)$ halts and outputs $f(x)$
If x is NOT in the domain of f then $M(x)$ does not halt.

What do we really want?

We want to model the set of partial computable functions. We need a model of computation where it is possible to diverge- not halt.
Turing Machines!

VOTE: True or False: Everything that can be partially computed can be partially computed by a Turing Machine.

What do we really want?

We want to model the set of partial computable functions. We need a model of computation where it is possible to diverge- not halt.
Turing Machines!

VOTE: True or False: Everything that can be partially computed can be partially computed by a Turing Machine.

YES- sort of. can be partially computed is not a rigorous notion; however TM's have been shown to do everything JAVA can do, so we'll say YES. Called Church-Turing Thesis.

Notation: If $M(x)$ halts we write $M(x) \downarrow$ (converges). If $M(x)$ does not halt we write $M(x) \uparrow$ (diverges).

Example

Example of a partial function where we don't know the domain.

Let $M_1, M_2, \ldots$ be a standard list of Turing Machines.

$f(e) =$ the number of steps $M_e(0)$ takes to halt, if it does. Diverges otherwise.

(SEEMS like we can't determine the domain, and we can't, but have not proven that yet.)

Are there any noncomputable sets?

Let $M_1, M_2, \ldots$ be a standard list of Turing Machines.

Let

$$HALT = \{(x, y) \mid M_x(y) \downarrow\}$$

Claim: HALT is NOT decidable.

Proof

Assume that $HALT$ is decidable. We build a machine that causes a contradiction.
Let $HALT$ be decidable by machine M. Note that $M(x, y)$ always converges- could say YES or NO. Build the following:

1. Input(x)
2. Run $M(x, x)$. If says YES (so $M_x(x) \downarrow$) then DIVERGE. If says NO then CONVERGE.

Call THIS machine M_e. Is $(e, e) \in HALT$?
DO REST ON BOARD.