Notions of Computability

- Many notions of computability have been proposed, e.g.
  - *(Type 0 a.k.a. Unrestricted or context sensitive)* Grammars
  - Partial Recursive Functions
  - Lambda calculus
  - *Markov Algorithms*
  - Post Algorithms
  - Post Canonical Systems,
- • All have been shown equivalent to Turing machines by simulation proofs

Other Systems we'll study

- Primitive Recursive Functions
- Partial recursive functions

- Neither of these are covered in detail in Sipser's text, but are both interesting and worthy of study

Computation using Numerical Functions

- We're used to thinking about computation as something we do with **numbers (e.g.** on the natural numbers)

- What kinds of functions from numbers to numbers can we actually compute?

- To study this, we make a very careful selection of building blocks

Primitive Recursive Functions

- The primitive recursive functions from $\mathbb{N}$ x $\mathbb{N}$ x ... x $\mathbb{N} \rightarrow \mathbb{N}$ are those built from these primitives:
  - zero(x_1,…, x_n) = 0
  - succ(x) = x+1
  - $\Pi_{k,j}$ (x_1,x_2,...,x_k) = x_j for 0 < j ≤ k

- using these mechanisms:
  - Function composition, and
  - Primitive recursion

Function Composition

- Define a new function f in terms of functions h and g_1, g_2, ..., g_m as follows:

  $f(x_1,...x_n) = h(g_1(x_1,...,x_n), ... ,g_m(x_1,...,x_n))$

  Note that f and g_i have arity n but that h has arity m

Example: f(x) = x + 3 can be expressed using two compositions as f (x) = succ(succ(succ(x)))

Primitive Recursion

- Primitive recursion defines a new function f in terms of functions h and g as follows:

- f(0,x1, ..., xk) = h(x1,...,xk)
- f(Succ(n),x1, ..., xk)= g(n, f(n,x1,...,xk**),** x1,...,xk**)**

Many ordinary functions can be defined using primitive recursion, e.g.

add(0,x) = π1,1(x)

add(Succ(y),x) = succ(π3,3(y, add(y,x),x))

More P.R. Functions

- For simplicity, we omit projection functions and write 0 for zero(_) and 1 for succ(0)

```
add(x,0) = x
add(x,succ(y)) = succ(add(x,y))

mult(x,0) = 0
mult(x,succ(y)) = add(x,mult(x,y))

factorial(0) = 1
factorial(succ(n)) =  mult(succ(n),factorial(n))

exp(n,0) = 1
exp(n, succ(n)) = mult(n,exp(n,m))

pred(0) = 0
pred(succ(n)) = n
```

- Essentially all practically **useful arithmetic** functions are primitive recursive, but...

Ackermann's Function is not Primitive Recursive

- A famous example of a function that is clearly well-defined but not primitive recursive

```
A(m, n)=
   if m=0 then n+1
   else if n=0 then A(m–1, 1)
   else A(m–1, A(m,n–1))
```

This function grows extremely fast!

Values of $A(m, n)$

$m \backslash n$	0	1	2	3	4	n
0	1	2	3	4	5	$n + 1$
1	2	3	4	5	6	$n + 2 = 2 + (n + 3) - 3$
2	3	5	7	9	11	$2n + 3 = 2 \cdot (n + 3) - 3$
3	5	13	29	61	125	$2^{(n+3)} - 3$
4	13	65533	$2^{65536} - 3$	$2^{2^{65536}} - 3$	$A(3, A(4, 3))$	$\underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{n+3 \text{ twos}} - 3$
5	65533	$\underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{65536 \text{ twos}} - 3$	$A(4, A(5, 1))$	$A(4, A(5, 2))$	$A(4, A(5, 3))$	$A(4, A(5, n\text{-}1))$
6	$A(5, 1)$	$A(5, A(6, 0))$	$A(5, A(6, 1))$	$A(5, A(6, 2))$	$A(5, A(6, 3))$	$A(5, A(6, n\text{-}1))$

A is not primitive recursive

- Ackermann's function grows faster than any primitive recursive function, that is:
- for *any primitive recursive function f, there is* an *n such that*
- $A(n, x) > f\ x$
- So *A can't be primitive recursive*

Formalizing the Prim Rec Functions

- In the next section we formalize when a function is primitive recursive.
- We define a grammar that describes the syntactically correct form of the PR-functions
- We explain how complex functions can be made from combining simpler ones.

An Algebra of PR functions

- A grammar for well formed term PR terms

Term $\rightarrow$ Z

| S

| P n — nth projection

| C Term [$Term_1$, … ,$Term_n$] — composition

| PR Term Term — primitive recursion

- n $\rightarrow$ 1 | 2 | 3 | 4 | …

Example 1

- Equations
- `f(x) = succ(succ(succ(x)))`

- Algebra
- `F = C S [C S [S]]`

Notice that the argument x is implicit

Term $\rightarrow$ Z
| S
| P N — nth projection
| C Term [$Term_1$, … ,$Term_n$] — composition
| PR Term Term — primitive recursion

Example 2

f(0,x1, ..., xk) = h(x1,...,xk)
f(Succ(n),x1, ..., xk)= g(n, f(n,x1,...,xk**),** x1,...,xk**)**

- Equations

```
add(0,x) = x
add(succ(y),x) = succ(add(y,x))
```

- Algebra

```
add = PR (P 1) (C S [P 2])
```

Term → Z
| S
| P N — nth projection
| C Term [$Term_1$, … ,$Term_n$] — composition
| PR Term Term — primitive recursion

Example 3

f(0,x1, ..., xk) = h(x1,...,xk)
f(Succ(n),x1, ..., xk)= g(n, f(n,x1,...,xk**),** x1,...,xk**)**

- Equations

```
pred Zero = Zero
pred (Succ n) = n
```

- Algebra

```
pred = PR Z (P 1)
```

Term $\rightarrow$ Z
| S
| P N — nth projection
| C Term [$Term_1$, … ,$Term_n$] — composition
| PR Term Term — primitive recursion

Example 4

f(0,x1, ..., xk) = h(x1,...,xk)
f(Succ(n),x1, ..., xk)= g(n, f(n,x1,...,xk**),** x1,...,xk**)**

- Equations

```
monus Zero x = x
monus (Succ n) x = pred(monus n x)
minus x y = monus y x
```

- Algebra

```
minus = C (PR (P 1)
              (C pred [P 2]))
          [P 2, P 1]
```

Term $\rightarrow$ Z
| S
| P N
| C Term [$Term_1$, … ,$Term_n$]
| PR Term Term

Summary

- The algebra denotes functions by combining other functions.
- The simplest functions: Z, S, P n are trivial
- Yet by using
  - Composition - C Term [$Term_1$, … ,$Term_n$]
  - Primitive recursion - PR Term Term

Many other functions can be built

- Almost every function we use can be built this way

Sanity Check

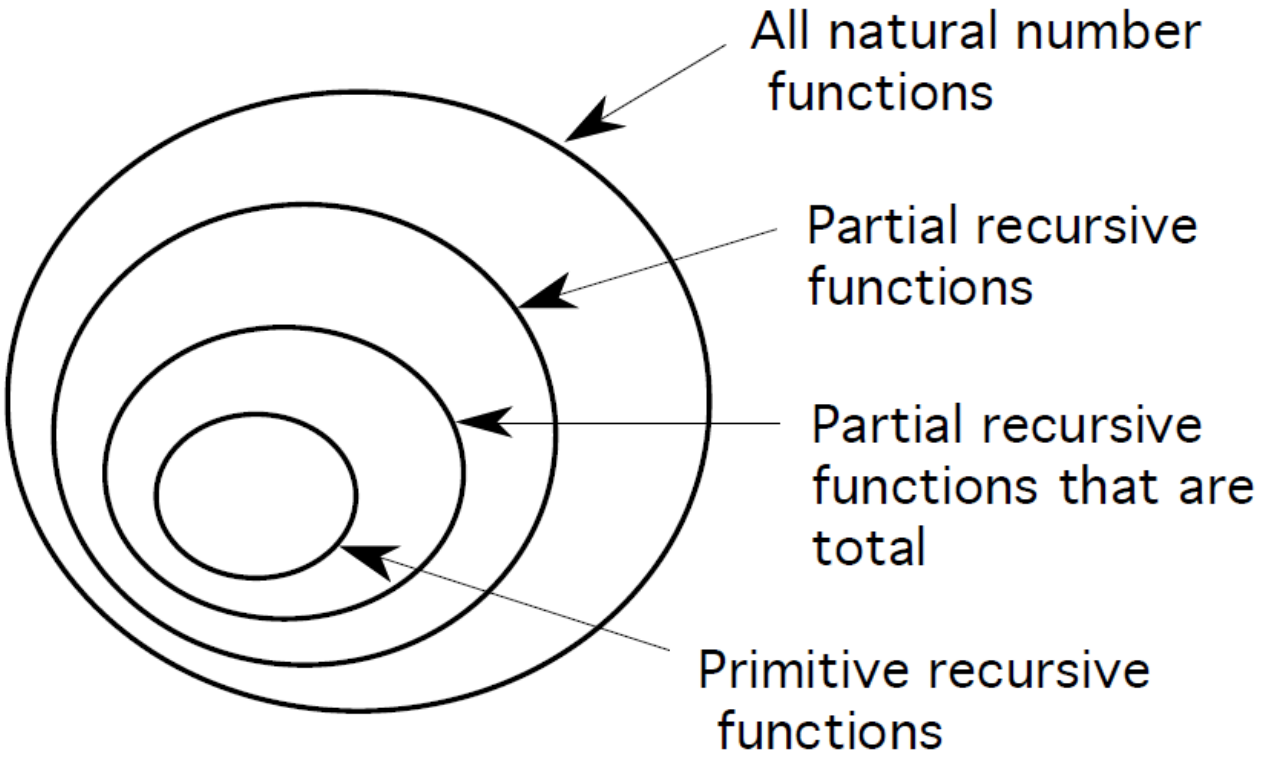
- We can check if a term in the algebra is a function of n arguments

```
check Z _ = True
check S 1 = True
check (P n) m = n <= m
check (C f gs) n =
  check f (length gs) &&
  (all (\g -> check g n) gs)
check (PR g h) n =
  check g (n-1) && check h (n+1)
```

Partial Recursive Functions

- *A belongs to class of **partial recursive*** **functions, a superset of the primitive recursive** functions.
- Can be built from primitive recursive operators & new **minimization operator**
  - Let *g be a (k+1)-argument function.*
  - Define *f(x1,...,xk) as the **smallest m such that g(x1,...,xk,m)*** = 0 (if such an m exists)
  - Otherwise, *f(x1,...,xn) is undefined*
  - We write *f(x1,...,xk) = μm.[g(x1,...,xk,m) = 0]*
  - Example: *μm.[mult(n,m) = 0] = zero(_)*

Hierarchy of Numeric Functions



Turing-computable functions

- To formalize the connection between partial recursive functions and Turing machines, mathematicians have described how to use TM's to compute functions on $\mathbb{N}$.
- We say a function f : $\mathbb{N}$ x $\mathbb{N}$ x ... x $\mathbb{N}$ → $\mathbb{N}$ is **Turing-computable** if there exists a TM that, when started in configuration $q_0 1^{n1} \sqcup 1^{n2} \sqcup \ldots \sqcup 1^{nk}$, halts with just $1^{f(n1,n2,\ldots nk)}$ on the tape.
- **Fact: f is Turing-computable iff it is partial recursive.**

A reference

- If you own the book
  - Discrete Structures, Logic and Computability
  - By James L Hein
  - The text used in CS250
  - You may look at Section 13.2.3 pages 832-835

- Note that the order of arguments for prim recursion here differs slightly from the order in the Hein book (page 832). Here we place the number being analyzed first rather than last.