Other Models of Computation

CS 81: Computability and Logic

April 25, 2010

Two-Stack Machine

- PDA with 2 stacks
  - Transition can push or pop one stack
  - Can transition based on the top symbol of one stack.

Theorem:
A Two-Stack Machine can simulate a TM

- Proof idea:
  - one stack to hold the symbols to the left of the head
  - one stack to hold the symbols below and to the right.
  - Moving the head corresponds to copying a symbol from one stack to the other.

Counter Machines (CM)

- A counter machine is like a TM or PDA, but instead of having a stack or tape, it has one or more integer counters (unbounded capacity)
  - Transitions can increment or decrement one counter
  - Transitions can test whether a counter is zero/nonzero
- Equivalent: PDA with one or more stacks, each having only one symbol (except $ at the bottom)
  - All that matters is the depth of each stack
  - Can only see the top of stack, so can only test empty/nonempty
- NB: Any language accepted by a CM is recognizable.

Theorem:
A 3-counter machine can simulate a TM.

- **Idea**: A 3-counter machine can simulate a 2-stack machine.
- Suppose our 2-stack machine uses r different stack symbols.
  WLOG, call these symbols 0 .. (r-1).
- Represent the stack $X_1X_2 \cdots X_n$ (with X_1 on top) by the number

$$X_n r^{n-1} + X_{n-1} r^{n-2} + \cdots + X_2 r + X_1$$

  - Pop = divide by r; top symbol is the remainder.
  - Push k = multiply by r and add k.
- Two of the counters hold two (encoded) stacks.
- Arithmetic operations use the third counter as scratch space

Theorem:
A 2-counter machine can simulate a TM.

- **Idea**: A 2-counter machine can simulate a 3-counter machine.
- Represent the 3 counters i, j, and k by the single number

$$2^i 3^j 5^k$$

- All necessary arithmetic can be done using the second counter as scratch space!

Partial Recursive Functions

- Purely mathematical formulations of functions of natural numbers.
  - Small set: Primitive Recursive functions (total)
  - Larger set: Partial Recursive functions (possibly partial)

- Theorem:
  - A TM can compute exactly the partial recursive functions (partial because TM might not terminate on all inputs)
  - Partial Recursive functions can (by encoding tapes as integers) simulate any TM.

Primitive Recursive Functions (an inductive definition!)

- The constantly-zero functions
  (take $k \geq 0$ arguments; return 0)

- The projection functions
  (take $k \geq 0$ arguments; return the i^{th})

- The successor function
  (take one argument n; return n+1)

...continued...

Primitive Recursive Functions (continued)

- The composition of prim. rec. functions is prim rec.
  e.g., h(x,y) = f(g1(x,y), g2(x,y), g3(x,y))

Corollary: the constantly-7 functions are PRF, for any n, because they are compositions of the constantly-zero function and successors.

Note: we often write explicit definitions like

h(x,y) = f(g1(y), g2(x), 2)

rather than

h(x,y) = f(g1(proj2(x,y)), g2(proj1(x,y)), succ(succ(zero())))

...continued...

Primitive Recursive Functions (continued)

- If b and r are prim rec. (with appropriate # of arguments) then so is the function f defined by

$$f(\mathbf{0}, x_1, \ldots, x_n) = b(x_1, \ldots, x_n)$$

$$f(\mathbf{n+1}, x_1, \ldots, x_n) = r(x_1, \ldots, x_n, n, f(\mathbf{n}, x_1, \ldots, x_n))$$

Note: this "primitive recursion" is a very stylized (restricted) template for coding recursively.

Examples of Primitive Recursive Functions

- plus(x, y)
- times(x, y)
- pred(x)
- monus(x, y)
- mod(x, y)
- div(x, y)
- sqrt(x)
- ?:
- Predicates like even, equality, etc. (return 0/1)

```
plus(0, y) = y
plus(n+1, y) = succ(plus(n,y))
```

```
mult(0, y) = 0
mult(n+1, y) = plus(y, times(n, y))
```

```
fact(0) = 1
fact(n) = mult(n, fact(n-1))
```

Primitive Recursion

- **Theorem**: Primitive Recursive functions are total.
- **Proof**: By induction.

- Are all total functions TM-computable?
- Are all TM-computable total functions primitive recursive?

Computable but not Primitive Recursive: Diagonalization

- We (hence a TM) can enumerate all the primitive recursive functions p_1, p_2, p_3, ...

- Then $q(n) = p_n(n) + 1$ is total and computable.

Computable but not Primitive Recursive: Ackermann Hierarchy

$$A_0(m) = S(m)$$

$$A_{n+1}(0) = A_n(1)$$
$$A_{n+1}(m+1) = A_n(A_{n+1}(m))$$

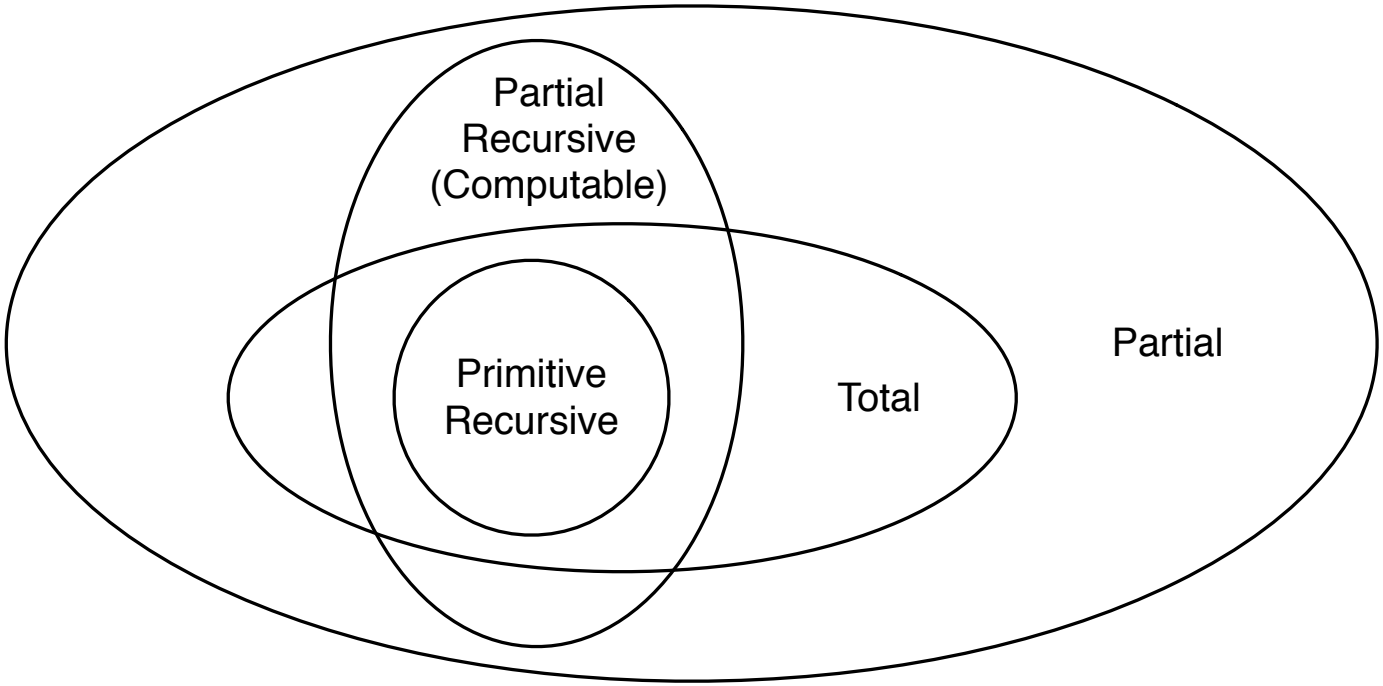
$$A(m) = A_m(m)$$

- **Theorem**:

  A(m) is a total function that grows faster than any primitive recursive function of one argument.

$$A(1) = 3$$
$$A(2) = 7$$
$$A(3) = 61$$
$$A(4) = 2^{2^{2^{65536}}} - 3$$

Primitive Recursive vs. Partial Recursive Functions



(not to scale)

Partial Recursive Functions

- Prim. Rec. + a partial "minimization operator"

If $h(x_1, ..., x_n)$ is partial recursive, then

$$\mu k\,[h(k, x2, ..., x_n) = 0]$$

is the function that (given $x_2, ..., x_n$) computes the least k (if any) that makes $h(k, x2, ..., x_n)$ zero.

Examples

sqrt(n) = μk. [monus(n, times(k, k)) = 0]

diverge(n) = μk. [monus(k+1, k) = 0]

strange(m, n) = μk. [not(eq(k+m, n)) = 0]

Partial Recursion vs. TMs

- Every Partial Recursive operation (including minimization) could be programmed in a TM.

TMs vs. Partial Recursion

- We can encode TM configurations as numbers.
- Primitive Recursive functions exist
  - R(x) = configuration one step beyond configuration x.
  - T(i,x) = configuration i steps beyond configuration x.
  - P(x) = whether configuration x is halting (0 or 1)
- Computation length = $\mu i.\ [P(T(i, x_0)) = 0]$
- Final configuration = $T(\mu i.\ [P(T(i, x_0)) = 0], x_0)$

Enumerating Partial Recursive Functions

- For each k, the Partial Recursive Functions with k arguments can be enumerated $f_1, f_2, \ldots$
  - There is even a k+1-argument PRF that given i, computes f_i applied to the k remaining arguments.
- Why doesn't the diagonal argument work again, giving us a computable function not in this list?
- $\{ j \mid f_j(\bullet) \text{ is total } \}$ is not recognizable/enumerable.
- $\{ j \mid f_j(j) \text{ is defined } \}$ is recognizable, not decidable.

Logic and Decidability

Recall:

Provability:

$$A_1, \ldots, A_n \vdash B$$

Validity

$$A_1, \ldots, A_n \models B$$

Decidability

$$A_1, \ldots, A_n \vdash B$$

Key question:

Given $A_1, \ldots, A_n$, is B provable using a fixed set of logical rules?

E.g., is there a decision algorithm?

Propositional Logic is Decidable

Proof: Truth tables, completeness.

Predicate Logic is not decidable

Proof: Reduce the PCP to Predicate Logic!

Given a PCP instance in binary, produce a formula that is provable iff the instance has a solution.

Setup

On the logic side, we will use

- a constant symbol e
- Two unary functions f_0 and f_1
- A binary predicate P

We will use f_0, f_1, and e to encode binary strings.

We will force P to be true when the two arguments could be the top and bottom of a sequence of dominos.

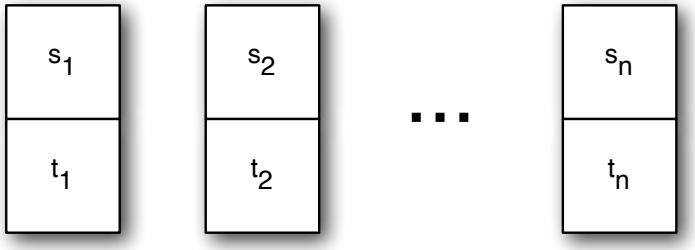
Encoding Binary Strings into Logic (by example)

$w = 000111$

$f_w(e) = f_1(f_1(f_1(f_0(f_0(f_0(e))))))$

Translation

The PCP instance



The formula

$$P(f_{s_1}(e), f_{t_1}(e)) \ \wedge \ \cdots \ \wedge \ P(f_{s_n}(e), f_{t_n}(e))$$

$$\wedge$$

$$(\forall u, v.\ P(u, v) \ \rightarrow \ P(f_{s_1}(u), f_{t_1}(v)) \wedge \cdots \wedge P(f_{s_n}(u), f_{t_n}(v)))$$

$$\rightarrow$$

$$\exists z.\ P(z, z)$$

Completeness and Incompleteness

Peano Axioms

$\forall x.\ \neg(S(x) = 0)$

$\forall x,y.\ \ S(x) = S(y) \rightarrow x=y$

$A[0/x] \wedge (\forall n.\ \ A[n/x] \rightarrow A[S(n)/x])$

$\rightarrow$

$(\forall n.\ \ A[n/x])$

For every formula A!

Theories

Theory:

A collection of formulas derivable from a set Γ of axioms.

Implicitly or explicitly specifies constants, functions, relations.

Γ might be infinite

Any proof can only use a finite subset of Γ

We normally expect recognizable/enumerable axioms.

Semigroup Theory

$\forall x,y,z.\ \ x + (y + z) = (x + y) + z$

Monoid Theory

$\forall x,y,z.\ \ x + (y + z) = (x + y) + z$

$\forall x.\quad (x+0) = x$

Group Theory

$\forall x,y,z.\ \ x + (y + z) = (x + y) + z$

$\forall x.\quad (x+0) = x$

$\forall x.\ \exists y.\ \ x+y = 0$

Commutative Group Theory

$\forall x,y,z.\ \ x + (y + z) = (x + y) + z$

$\forall x.\quad (x+0) = x$

$\forall x.\ \exists y.\ \ x+y = 0$

$\forall x,y.\ \ x + y\ \ = y + x$

Linear Order Theory

$\forall x.\ \neg(x < x)$

$\forall x,y.\ \ (x < y) \vee (x = y) \vee (y < x)$

$\forall x,y,z.\ \ ((x < y) \wedge (y < z)) \rightarrow (x < z)$

Dense Linear Order Theory

$\forall x.\ \neg(x < x)$

$\forall x,y.\ \ (x < y) \vee (x = y) \vee (y < x)$

$\forall x,y,z.\ \ ((x < y) \wedge (y < z)) \rightarrow (x < z)$

$\exists x,y.\ \ x < y$

$\forall x,y.\ \ (x < y) \rightarrow \exists z.\ (x < z) \wedge (z < y)$

Dense Linear Order Theory without Endpoints

$\forall x.\ \neg(x < x)$

$\forall x,y.\ (x < y) \vee (x = y) \vee (y < x)$

$\forall x,y,z.\ ((x < y) \wedge (y < z)) \rightarrow (x < z)$

$\exists x,y.\ x < y$

$\forall x,y.\ (x < y) \rightarrow \exists z.\ (x < z) \wedge (z < y)$

$\forall x.\ \exists z.\ (z < x)$

$\forall x.\ \exists z.\ (x < z)$

Recall:
Predicate Logic

... is Sound.

$$\text{If } \Gamma \vdash B \text{ then } \Gamma \models B$$

... is Complete (Gödel's Completeness Theorem)

$$\text{If } \Gamma \models B \text{ then } \Gamma \vdash B$$

Validity Revisited

If B is a closed formula, then either:

$\Gamma \models B$

$\Gamma \models \neg B$

Neither.

A theory is said to be **negation-complete** (**complete** for short) if it always rules out the third case for all closed B.

That is, for every closed formula B, either $\Gamma \models B$ or $\Gamma \models \neg B$.

That is, for every closed formula B, either $\Gamma \vdash B$ or $\Gamma \vdash \neg B$.

Dense Linear Order Theory w/o Endpoints is (Negation-) Complete

$\forall x.\ \neg(x < x)$

$\forall x,y.\ (x < y) \vee (x = y) \vee (y < x)$

$\forall x,y,z.\ ((x < y) \wedge (y < z)) \rightarrow (x < z)$

$\exists x,y.\ x < y$

$\forall x,y.\ (x < y) \rightarrow \exists z.\ (x < z) \wedge (z < y)$

$\forall x.\ \exists z.\ (z < x)$

$\forall x.\ \exists z.\ (x < z)$

Presberger Arithmetic is Complete

$\forall x.\ \neg(0 = x+1)$

$\forall x,y.\ x+1 = y+1 \rightarrow x=y$

$\forall x.\ x+0 = x$

$\forall x,y.\ (x+y)+1 = x+(y+1)$

$A[0/x] \wedge (\forall n.\ A[n/x] \rightarrow A[n+1/x])$

$\rightarrow$

$(\forall n.\ A[n/x]) + 1 = y + 1 \rightarrow x = y$

Linear Order Theory isn't complete.

$\forall x.\ \neg(x < x)$

$\forall x,y.\ \ (x < y) \vee (x = y) \vee (y < x)$

$\forall x,y,z.\ \ ((x < y) \wedge (y < z)) \rightarrow (x < z)$

Why Completeness Matters

Most theories aren't complete.

But completeness is a nice property.

- If the axioms can be enumerated, so can the theorems.
- "Is B provable" becomes decidable.
  - Enumerate theorems and wait for B or ¬B.
- (Of course, this assumes the theory is consistent.)

Gödel's First Incompleteness Theorem, 1931
(Refined by Rosser, 1936)

No consistent theory
with recognizable axioms
extending number theory
is (negation-)complete.

Consequence:

If you want to do math, you generally need at least
addition, multiplication, and induction on natural numbers.

Any such theory will be (negation-) incomplete.

Number Theory?

Like Presburger Arithmetic + Multiplication

$\forall x.\ x \times 0 = 0$

$\forall x,y.\ \ (x \times (y+1)) = ((x \times y) + x)$

Gödel's Proof Setup

Main idea: "Gödel Numbering"

encode logical formulas (strings) as numbers.

encode proofs (lists of formulas) as numbers.

Gödel's Proof Setup (continued)

He showed how to define a (big and complicated) formula $\Pi(x,p,a)$ true exactly when

- p encodes a unary predicate P()
- x encodes a proof of P(a) in the theory

By the way:

- Π is built using partial recursive functions!
- PRFs turn out to be the functions naturally definable in number theory.

The Proof Strategy(1)

- $\Pi(x,p,a)$ means that x is a proof of P(a), where p encodes P.
- Define $\Delta(p) := \forall x.\ \neg\Pi(x,p,p)$ [i.e., there is no proof of P(p)]
- Let d be the Gödel number of Δ.

- Is $\Delta(d)$ provable?
  - $\Delta(d) = \forall x.\ \neg\Pi(x,d,d)$ [i.e., there is no proof of $\Delta(d)$]
  - If it's provable then it's true, but then there's no proof.
  - So, no.

The Proof Strategy (2)

- $\Pi(x,p,a)$ means that x is a proof of P(a), where p encodes P.
- Define $\Delta(p) := \forall x.\ \neg\Pi(x,p,p)$ [i.e., there is no proof of P(p)]
- Let d be the Gödel number of Δ.

- Well, then, is $\neg\Delta(d)$ provable?
  - $\neg\Delta(d) = \neg\forall x.\ \neg\Pi(x,d,d)$ [i.e., there is a proof of $\Delta(d)$]
  - But we just showed there isn't a proof $\Delta(d)$.
  - So, no.

Gödel's Second Incompleteness Theorem

Within any consistent extension of number theory,

- there is a logical formula Con that expresses the consistency of the theory,
- but Con is not provable in the theory.

Corollary: In any extension of number theory,
Con is provable if and only if ...

Note: number theory can be proved consistent...if you're willing to believe set theory is consistent.

Decidability of Number Theory

Number Theory isn't negation-complete.

We can't decide whether A is a theorem by simply listing theorems and looking for A or ¬A.

But might theorem-hood still be decidable using some other algorithm?

Recall:

- We can encode TM configurations as numbers.
- Primitive Recursive functions exist
  - R(x) = configuration one step beyond configuration x.
  - T(i,x) = configuration i steps beyond configuration x.
  - P(x) = whether configuration x is halting (0 or 1)
- Computation length = $\mu i.\ [P(T(i, x_0)) = 0]$
- Final configuration = $T(\mu i.\ [P(T(i, x_0)) = 0], x_0)$

Number Theory is Undecidable

Number Theory is strong enough to define all primitive recursive functions.

Halting is then the logical formula

$$\exists i.\ P(T(i, x_0)) = 0$$