Recall From Last Time

Definition: We say that S is *reducible* to T, $S \leq T$, iff $\exists$ total, recursive $f : \mathbf{N} \to \mathbf{N}$,

$$(\forall w \in \mathbf{N}) \quad (w \in S) \qquad \Leftrightarrow \qquad (f(w) \in T)$$

[In the future we will insist that $f \in F(\mathbf{L})$.]

Theorem: Suppose $S \leq T$. Then,

1. If T is **r.e.**, then S is **r.e.**.
2. If T is co-**r.e.**, then S is co-**r.e.**.
3. If T is **Recursive**, then S is **Recursive**.

Definition: C is r.e.-*complete* iff

1. $C \in$ **r.e.**, and
2. $(\forall A \in \textbf{r.e.}) \quad (A \leq C)$

Theorem: K, HALT, and $A_{0,17}$ are r.e. complete.

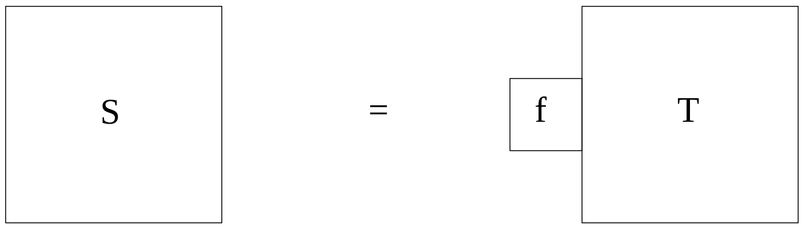
Reductions

Def: We say that S is *reducible* to T, $S \leq T$, iff $f \in F(\mathbf{L})$,

$$(\forall w \in \mathbf{N}) \quad (w \in S) \quad \Leftrightarrow \quad (f(w) \in T)$$

Intuition: $S \leq T$ iff the placement of a **very simple front end** f before a T-recognizer creates an S-recognizer.

$$\chi_S \quad = \quad \chi_T \circ f, \qquad \text{i.e.,} \quad (\forall x)(\chi_S(x) = \chi_T(f(x)))$$



The Reduction Game: To build a reduction, f, from S to T you must solve the following puzzle:

“For each input, w, what membership question, $f(w)$, can I ask T such that the answer is the membership question $w \in ?S$.”

Rice-Myhill-Shapiro Theorem:

Our proof that $A_{0,17}$ was r.e.-complete had little to do with the numbers 0 or 17. A very similar argument can be used to show that "any non-trivial property of Turing machines is undecidable". (See [P], Theorem 3.2, page 62.)

Theorem 8.1 *Let A be a set other than $\emptyset$ or $\mathbf{N}$ such that if M_i and M_j are equivalent machines, i and j are either both in A or both not in A. Then A is not recursive.*

Proof:

Suppose that the (numbers of) machines that never halt are not in A. (What if they aren't? See HW#3.) Since A is nonempty, pick a number ℓ so that $M_\ell \in A$.

We will reduce K to A, which means we must define a total recursive function f so that $n \in K$ iff $f(n) \in A$. This means that for any machine M_n, we must build a machine $M_{f(n)}$ that will have the property necessary for A iff M_n accepts n. We'll do this using our assumptions. If M_n accepts n, $M_{f(n)}$ will be equivalent to M_ℓ and thus $f(n)$ will be in A. If M_n does not accept n, $M_{f(n)}$ will never halt and thus $f(n)$ will not be in A.

This is easy. We design $M_{f(n)}$ so that it first runs M_n on n, then (if it finishes that job) runs M_ℓ on the original input. This machine simulates M_ℓ if M_n accepts n and never halts otherwise.

Since $K \leq A$, A cannot be recursive. (We cannot say that A is r.e.-complete because it might not be r.e., in fact "most" such A's are not.)

♠

Primitive Recursive Functions

On HW#2 we defined the programming language Bloop, with integer variables and bounded loops. It turns out that the class of functions from **N** to **N** that are implementable in Bloop is a very well studied class called the *primitive recursive functions*.

You may have wondered whether "recursion" as you know it from programming has anything to do with "recursive functions" in this course. The name indeed comes from defining functions recursively. We'll eventually see the definition of "general recursive functions" that is equivalent to Turing machines. But first we'll give the definition of a less powerful kind of recursion. It defines functions that are guaranteed to halt, but can't define all total recursive functions.

On HW#3 we'll prove that there are recursive functions that are not primitive recursive, and that the primitive recursive functions are exactly those implementable in Bloop.

Initial functions:

$\zeta() = 0$

$\sigma(x) = x + 1$

$\pi_i^n(x_1, \ldots, x_n) = x_i, \quad n = 1, 2, \ldots, \quad 1 \leq i \leq n$

Composition: $g_i : \mathbf{N}^k \to \mathbf{N}, 1 \leq i \leq m; \; ; h : \mathbf{N}^m \to \mathbf{N}$:

$$\mathcal{C}(h; g_1, \ldots, g_m)(x_1, \ldots, x_k) \;=\; h(g_1(\overline{x}), \ldots, g_m(\overline{x}))$$

Primitive Recursion: $g : \mathbf{N}^k \to \mathbf{N}; \; h : \mathbf{N}^{k+2} \to \mathbf{N}$: $f(n, y_1, \ldots, y_k) = \mathcal{P}(g, h)(n, y_1, \ldots y_k)$, given by:

$$\begin{aligned} f(0, y_1, \ldots y_k) &= g(y_1, \ldots, y_k) \\ f(n+1, y_1, \ldots y_k) &= h(f(n, y_1, \ldots, y_k), n, y_1, \ldots, y_k) \end{aligned}$$

Definition 8.2 The **primitive recursive functions (PrimRecFcns)** is the smallest class of functions containing the Initial functions and closed under Composition and Primitive Recursion. ♠

Proposition 8.3 *The following are in* **PrimRecFcns***:*

1. $M_1(x) \; = \; \textbf{if } (x > 0) \textbf{ then } (x - 1) \; \textbf{ else } 0$
2. $x \ominus y \; = \; \textbf{if } (y \leq x) \textbf{ then } (x - y) \; \textbf{ else } 0$
3. $+$
4. $*$
5. $\exp(x, y) \; = y^x$
6. $\exp^*(x) \; = \textbf{if } (x = 0) \textbf{ then } 1 \; \textbf{ else } 2^{\exp^*(x-1)}$

$$\exp^*(x) \; = \; 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}} \Big\} x$$

Gödel discovered that you can *code sequences* in **PrimRecFcns**, which he did using number theory:

Proposition 8.4 Prime, PrimeF $\in$ **PrimRecFcns**, *where,*

Prime$(x) =$ **if** (*"x is prime"*) **then** 1 **else** 0

PrimeF$(n) =$ *prime number n, i.e,* PrimeF$(0) = 2$, PrimeF$(1) = 3$, PrimeF$(2) = 5$, PrimeF$(3) = 7$, PrimeF$(4) = 11$.

Proof:

$$\begin{aligned} x|y &= (\exists z \leq y)(xz = y) \\ \text{Prime}(x) &= x > 1 \ \wedge \ (\forall y < x)(y|x \rightarrow y = 1) \\ \text{NextPrime}(x) &= \mu\{t \leq (x+1)^{x+1} \mid t > x \ \wedge \ \text{Prime}(t)\} \end{aligned}$$

$$\begin{aligned} \text{PrimeF}(0) &= 2 \\ \text{PrimeF}(x+1) &= \text{NextPrime}(\text{PrimeF}(x)) \end{aligned}$$

♠

Proposition 8.5

IsSeq, Length, Item $\in$ **PrimRecFcns**, *where,*

$$\begin{aligned}
\text{Seq}(a_0, a_1, \ldots, a_n) &= 2^{a_0+1}3^{a_1+1}\cdots \text{PrimeF}(n)^{a_n+1} \\
\text{IsSeq}(S) &= \textbf{if } (S \textit{ is a Sequence number }) \\
&\qquad\qquad \textbf{then } 1 \textbf{ else } 0 \\
\text{Length}(\text{Seq}(a_0, a_1, \ldots, a_n)) &= n + 1 \\
\text{Item}(\text{Seq}(a_0, a_1, \ldots, a_n), i) &= a_i
\end{aligned}$$

Proof:

$$\begin{aligned}
\text{Good}(x, S) &= (\forall y < S)((y < x \wedge \text{PrimeF}(y) | S) \\
&\qquad \vee (y \geq x \wedge \text{PrimeF}(y) \not| S))
\end{aligned}$$

$$\text{IsSeq}(S) = (\exists x < S)\text{Good}(x, S)$$

$$\text{Length}(S) = \mu\{x < S \mid \text{Good}(x, S)\}$$

$$\begin{aligned}
\text{Item}(S, i) &= \mu\{y < S \mid \text{IsSeq}(S) \wedge \text{PrimeF}(i)^{y+1} | S \\
&\qquad \wedge \text{PrimeF}(i)^{y+2} \not| S\}
\end{aligned}$$

♠

As your intuition about Bloop should begin to tell you, and your work on HW#3 should confirm, we can do almost anything with primitive recursive functions:

Primitive Recursive COMP Theorem: [Kleene]

Let COMP(n, x, c, y) mean $M_n(x) = y,$ and that c is M_n's complete computation on input x.

Then COMP is a Primitive Recursive predicate.

Proof: We will encode TM computations:

$$c = \text{Seq}(\text{ID}_0, \text{ID}_1, \ldots, \text{ID}_t)$$

Where each ID_i is a sequence number of tape-cell contents:

$$\text{ID}_i = \text{Seq}(\triangleright, a_1, \ldots, a_{i-1}, [\sigma, a_i], a_{i+1}, \ldots, a_r)$$

$$\begin{aligned}
&\text{COMP}(n, x, c, y) \;\equiv \\
&\quad \text{START}(\text{Item}(c, 0), x) \;\wedge\; \text{END}(\text{Item}(c, \text{Length}(c) - 1), y) \;\wedge \\
&\qquad (\forall i < \text{Length}(c))\text{NEXT}(n, \text{Item}(c, i), \text{Item}(c, i + 1))
\end{aligned}$$

♠

Theorem 8.6 *The following problems are decidable in polynomial time.*

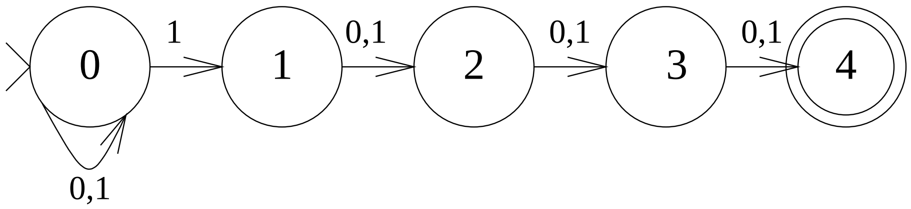
$$\begin{aligned}
\text{EmptyNFA} &= \{N \mid N \textit{ is an NFA};\ \mathcal{L}(N) = \emptyset\} \\
\Sigma^\star\text{DFA} &= \{D \mid D \textit{ is a DFA};\ \mathcal{L}(D) = \Sigma^\star\} \\
\text{MemberNFA} &= \{\langle N, w\rangle \mid N \textit{ is an NFA};\ w \in \mathcal{L}(N)\} \\
\text{EqualDFA} &= \{\langle D_1, D_2\rangle \mid D_1, D_2 \textit{ DFAs};\ \mathcal{L}(D_1) = \mathcal{L}(D_2)\} \\
\text{EmptyCFL} &= \{G \mid G \textit{ is a CFG};\ \mathcal{L}(G) = \emptyset\} \\
\text{MemberCFL} &= \{\langle G, w\rangle \mid G \textit{ is a CFG};\ w \in \mathcal{L}(G)\}
\end{aligned}$$

EmptyNFA $HW4$

$$\Sigma^\star\text{DFA} = \{D \mid D \text{ is a DFA; } \mathcal{L}(D) = \Sigma^\star\}$$

$$D \in \Sigma^\star\text{DFA} \Leftrightarrow \overline{D} \in \text{EmptyNFA}$$

$$\text{MemberNFA} = \{\langle N, w\rangle \mid N \text{ is an NFA; } w \in \mathcal{L}(N)\}$$



$$\begin{aligned}\text{EqualDFA} &= \{\langle D_1, D_2\rangle \mid \mathcal{L}(D_1) = \mathcal{L}(D_2)\} \\ \langle D_1, D_2\rangle \in \text{EqualDFA} &\Leftrightarrow (\overline{D_1} \cap D_2) \cup (D_1 \cap \overline{D_2}) \in \text{EmptyNFA}\end{aligned}$$

$$\text{EmptyCFL} \qquad HW\#3$$

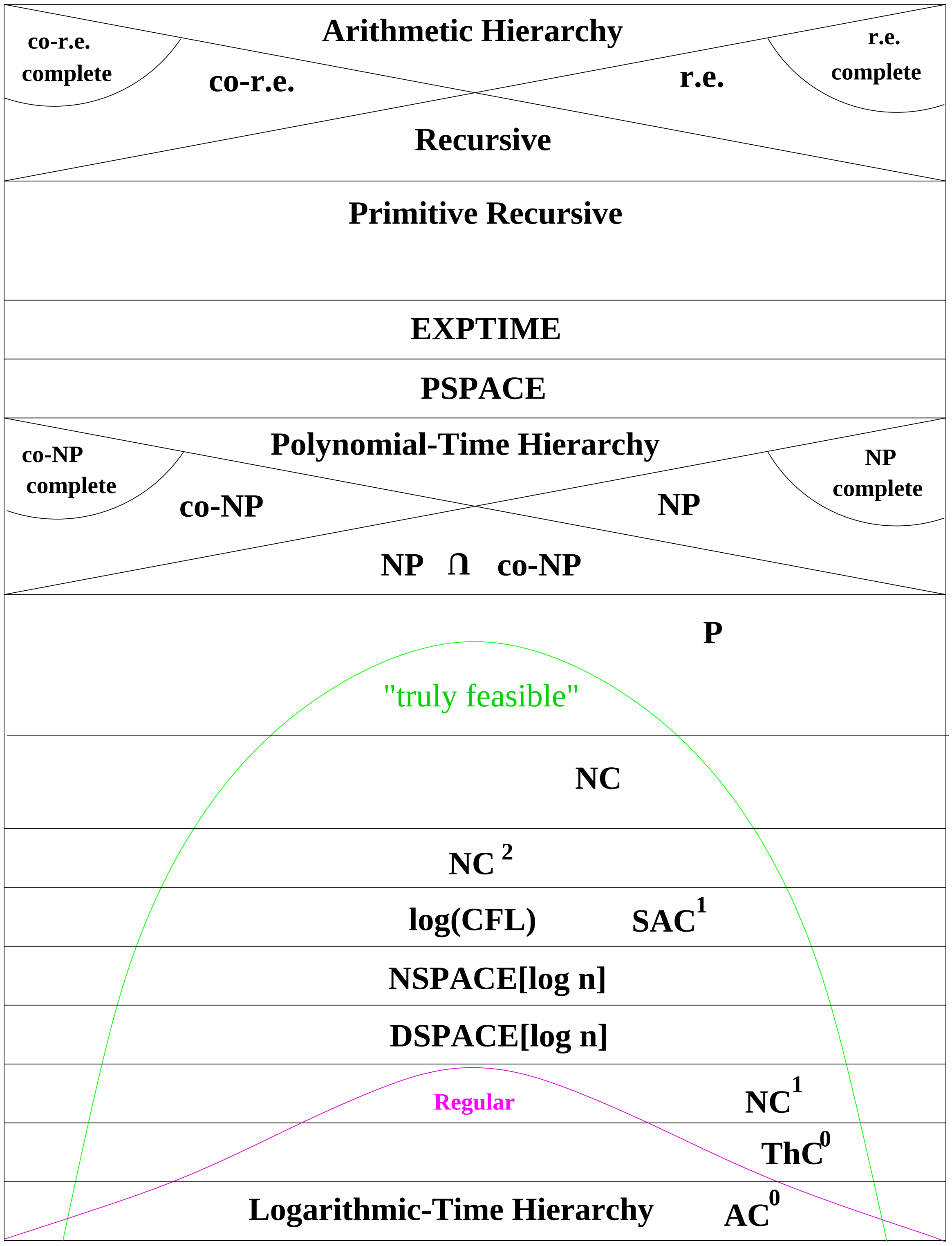
$$\text{MemberCFL} \quad = \quad \{\langle G, w\rangle \mid G \text{ is a CFG; } w \in \mathcal{L}(G)\}$$

CYK Dynamic Programming Algorithm:

1. Assume G in **Chomsky Normal Form:** $N \to AB$, $N \to a$.
2. **Input:** $w = w_1 w_2 \ldots w_n$; G with nonterminals $S, A, B, \ldots$
3. $N_{ij} \equiv \begin{cases} 1 & \text{if } N \overset{\star}{\to} w_i \cdots w_j \\ 0 & \text{otherwise} \end{cases}$
4. **return**(S_{1n})

$$N_{i,i} = \textbf{if } (\text{“}N \to w_i\text{”} \in R) \textbf{ then } 1 \textbf{ else } 0$$

$$N_{i,j} = \bigvee_{\text{“}N \to AB\text{”} \in R} (\exists k)(i \leq k < j \wedge A_{i,k} \wedge B_{k+1,j})$$


Arithmetic Hierarchy
co-r.e. complete
co-r.e.
r.e.
r.e. complete
Recursive
Primitive Recursive
EXPTIME
PSPACE
Polynomial-Time Hierarchy
co-NP complete
co-NP
NP
NP complete
NP ∩ co-NP
P
"truly feasible"
NC
NC 2
log(CFL)
SAC1
NSPACE[log n]
DSPACE[log n]
Regular
NC1
ThC0
Logarithmic-Time Hierarchy
AC0