Recall From Last Time

Def: The **primitive recursive functions**, **PrimRecFcns**, is the smallest class of functions containing the Initial functions and closed under Composition and Primitive Recursion.

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:

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

Facts and Exercises:

1. A function is primitive recursive iff it is computable in Bloop.
2. Every primitive recursive function is total recursive.
3. There is a total recursive function that is not primitive recursive.
4. There are primitive recursive functions that encode and decode sequences of integers by single integers.

Using sequences, primitive recursive functions are powerful enough to talk about Turing machines:

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:

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

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

♠

Definition 12.1 The **general recursive functions** are the set of partial functions obtained by closing the initial functions under composition and the μ**-operator**. We define $\mu\{x : f(x, y) = 0\}$ for an input y to be the least x such that $f(x, y) = 0$, if one exists, or otherwise undefined. ♠

Facts and Exercises:

- The general recursive functions are the closure of the p.r. functions under the μ-operator.
- A partial function is general recursive iff it is computable in Floop, the language obtained from Bloop by adding a `while` statement.
- A partial function is general recursive iff it is partial recursive (computable by some TM).
- A total function is partial recursive iff it is in **DTIME**$[f]$ for some primitive recursive function f.

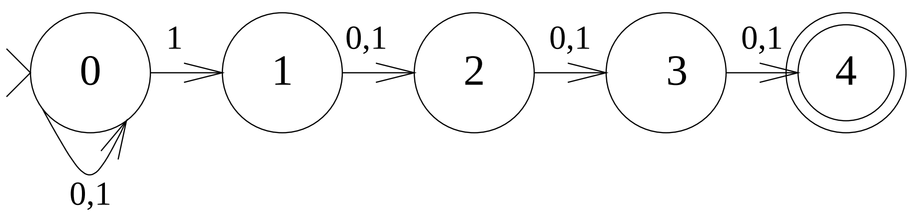
Theorem 12.2 *The following problems are decidable in polynomial time.*

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

$$\begin{aligned}
\text{EmptyNFA} &= \{N : \text{No start-final path in graph of } N\} \\
\Sigma^\star\text{DFA} &= \{D \mid D \text{ is a DFA; } \mathcal{L}(D) = \Sigma^\star\} \\
D \in \Sigma^\star\text{DFA} &\Leftrightarrow \overline{D} \in \text{EmptyNFA}
\end{aligned}$$

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

Convert to another reachability problem:



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

$$
\text{EmptyCFL} \qquad HW\#4
$$

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

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})

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

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

Theorem 12.3 *The following problem is co-***r.e.***-complete:*

$$\Sigma^\star\mathrm{CFL} \quad = \quad \{G \mid G \text{ is a CFG; } \mathcal{L}(G) = \Sigma_G^\star\}$$

Proof: [J. Hartmanis, Neil's advisor]

$\overline{\Sigma^\star\mathrm{CFL}} \in$ **r.e.**:

Input: G

Define: $\Sigma_G^\star = \{w_0, w_1, w_2, \ldots\}$

1. **for** $i := 0$ to ∞ $\{$
2. **if** $w_i \notin \mathcal{L}(G)$, **then return**$(1)\}$

(We use the the CYK algorithm for each MemberCFL check.)

Clearly this returns 1 iff $G \in \overline{\Sigma^\star\mathrm{CFL}}$.

Proposition 12.4 *EMPTY is co-r.e. complete, where,*

$$\textit{EMPTY} \quad = \quad \{n \mid W_n = \emptyset\}$$

Proof: Show NON-EMPTY to be r.e.-complete: show it r.e. and reduce K to it. (Good practice!) ♠

Claim 12.5 *EMPTY* $\leq \Sigma^\star$CFL.

Corollary 12.6 $\Sigma^\star$CFL *is co-r.e. complete and thus not recursive.*

How can we prove the Claim?

We need to define: $g : \mathbf{N} \to \{0,1\}^\star$,

$$\begin{aligned} n \in \text{EMPTY} \quad &\Leftrightarrow \quad g(n) \in \Sigma^\star\text{CFL} \\ (\forall x) M_n(x) \neq 1 \quad &\Leftrightarrow \quad \mathcal{L}(g(n)) = \Sigma_n^\star \\ M_n \text{ has no accepting computations} \quad &\Leftrightarrow \quad \mathcal{L}(g(n)) = \Sigma_n^\star \end{aligned}$$

We need to represent *entire computations* of TM's by strings. Assume that M_n is a one-tape machine.

We first defined a string called an **Instantaneous Description** or **ID** of a computation of M_n:

M_n has alphabet $\{0, 1\}$, states $\{\hat{0}, \hat{1}, \ldots, \hat{q}\}$ where $\hat{0}$ is the halting state and $\hat{1}$ is the start state.

$$\mathrm{ID}_0 \quad = \quad \hat{1} \ \triangleright \ w_1 \, w_2 \ \cdots \ w_r \sqcup$$

Suppose M_n in state $\hat{1}$ looking at a "$\triangleright$" writes a "$\triangleright$" changes to state $\hat{3}$, and moves to the right.

$$\mathrm{ID}_1 \quad = \quad \triangleright \, \hat{3} \, w_1 \, w_2 \ \cdots \ w_r \sqcup$$

In general the ID shows the tape up to and including the first blank after the last non-blank, with a character for the state inserted just left of the head position. It is easy to tell whether a string is a valid ID.

YesComp(n) =

$$\left\{ \mathrm{ID}_0 \# \mathrm{ID}_1^R \# \mathrm{ID}_2 \# \mathrm{ID}_3^R \# \cdots \# \mathrm{ID}_t \;\middle|\; \mathrm{ID}_0 \cdots \mathrm{ID}_t \begin{array}{c} \text{accepting} \\ \text{comp of } M_n \end{array} \right\}$$

Note that ID_0 can have any string in $\{0,1\}^*$ as the input string. We write every other ID *backwards* to allow easy checking by a CFL.

Lemma 12.7 *For each* n, $\overline{\mathrm{YesComp}(n)}$ *is a CFL.*

Furthermore, there is a function $g \in F(\mathbf{L})$*, for all* n, $g(n)$ *codes a context-free grammar and*

$$\mathcal{L}(g(n)) \quad = \quad \overline{\mathrm{YesComp}(n)}$$

$\Sigma_n \;=\; \{0, 1, \triangleright, \sqcup, \#, \hat{0}, \hat{1}, \ldots, \hat{q_n}\}$ where M_n has q_n states.

$n \in \mathrm{EMPTY} \quad \Leftrightarrow \quad \overline{\mathrm{YesComp}(n)} = \Sigma_n^\star \quad \Leftrightarrow \quad g(n) \in \Sigma^\star\mathrm{CFL}$

The grammar must generate every string that does *not* code an accepting computation of M_n.

Proof:

$$\overline{\text{YesComp}(n)} \;=\; U(n) \;\cup\; A(n) \;\cup\; D(n) \;\cup\; Z(n)$$

$$U(n) \;=\; \{w \in \Sigma^\star \mid w \text{ not in form } \mathrm{ID}_0\#\cdots\#\mathrm{ID}_t\}$$

$$A(n) \;=\; \{w \in \Sigma^\star \mid w \text{ doesn't start with an initial ID of } M_n\}$$

$$D(n) \;=\; \{w \in \Sigma^\star \mid (\exists i)(\mathrm{ID}_{i+1} \text{ doesn't follow from } \mathrm{ID}_i\}$$

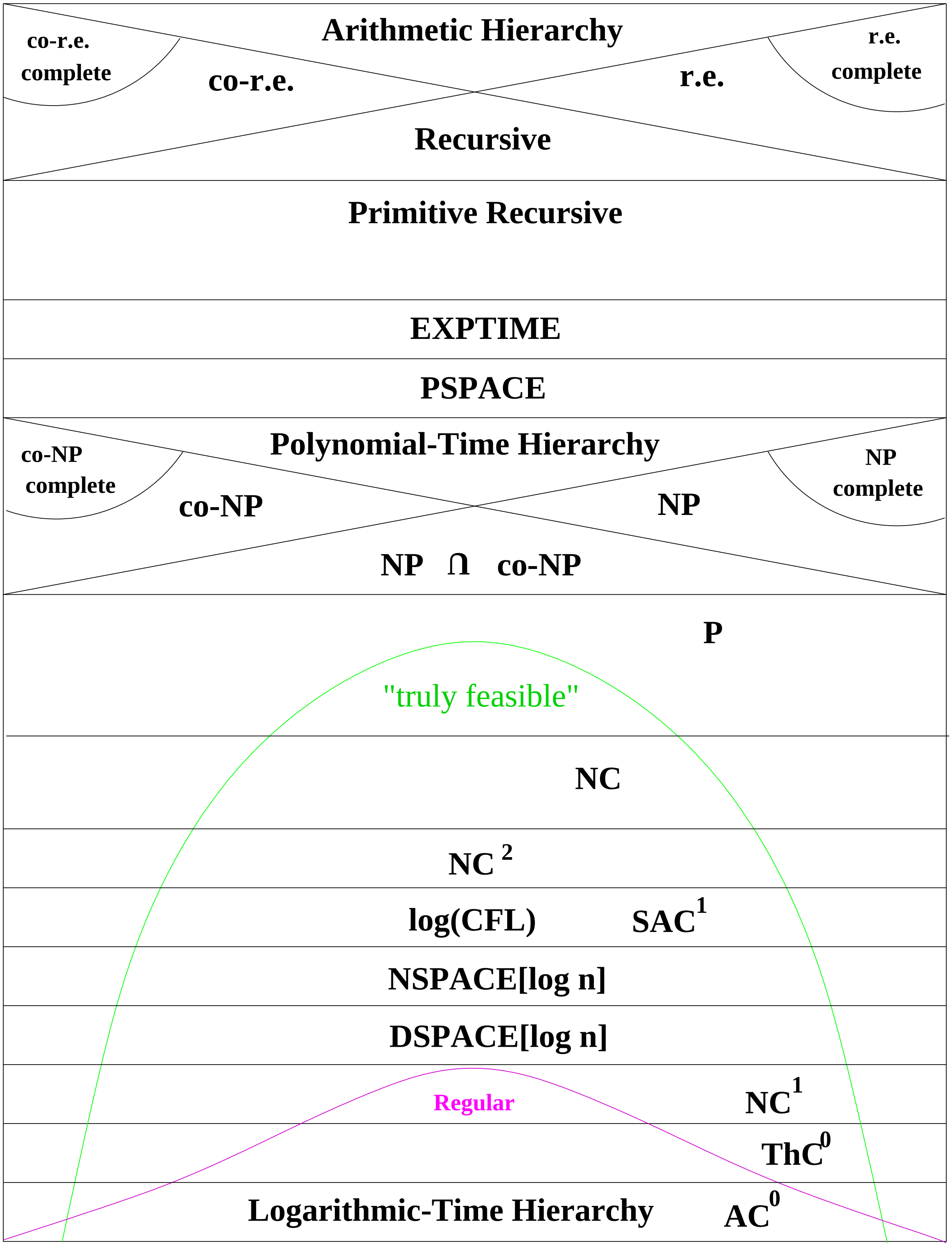
$$Z(n) \;=\; \{w \in \Sigma^\star \mid w \text{ doesn't end with } \hat{0} \;\triangleright\; 1 \sqcup\}$$

$U(n)$, $A(n)$, and $Z(n)$ are regular languages. To be in $D(n)$, a string must contain a letter in ID_{i+1} that does not follow from the corresponding place in ID_i by the rules of M_n – either the tape changes away from the head or changes the wrong way at the head. A PDA could guess and verify the point at which this happens. ♠

Thus, g : EMPTY $\leq \Sigma^\star$CFL

$$\begin{aligned} n \in \text{EMPTY} &\Leftrightarrow \overline{\text{YesComp}(n)} = \Sigma_n^\star \\ &\Leftrightarrow g(n) \in \Sigma^\star\text{CFL} \end{aligned}$$

♠


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