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

**Initial functions:**

- \( \zeta() = 0 \)
- \( \sigma(x) = x + 1 \)
- \( \pi^n(x_1, \ldots, x_n) = x_i, \quad n = 1, 2, \ldots, \quad 1 \leq i \leq n \)

**Composition:** \( g_i : \mathbb{N}^k \rightarrow \mathbb{N}, 1 \leq i \leq m; \quad ; h : \mathbb{N}^m \rightarrow \mathbb{N} : \)

\[
C(h; g_1, \ldots, g_m)(x_1, \ldots, x_k) = h(g_1(x), \ldots, g_m(x))
\]

**Primitive Recursion:** \( g : \mathbb{N}^k \rightarrow \mathbb{N}; \quad h : \mathbb{N}^{k+2} \rightarrow \mathbb{N} : \)

\( f(n, y_1, \ldots, y_k) = \mathcal{P}(g, h)(n, y_1, \ldots y_k), \) given by:

- \( f(0, y_1, \ldots y_k) = g(y_1, \ldots, y_k) \)
- \( f(n + 1, y_1, \ldots y_k) = h(f(n, y_1, \ldots, y_k), n, y_1, \ldots, y_k) \)
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 $\text{COMP}(n, x, c, y)$ 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 $\text{ID}_i$ is a sequence number of tape-cell contents:

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

$$\text{COMP}(n, x, c, y) \equiv$$

$$\text{START(Item(c, 0), x)} \land \text{END(Item(c, Length(c) - 1), y)} \land$$

$$(\forall i < \text{Length(c)}) \text{NEXT}(n, \text{Item}(c, i), \text{Item}(c, i + 1))$$

$\spadesuit$
Definition 12.1 The general recursive functions are the set of partial functions obtained by closing the initial functions under composition and the $\mu$-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 $\mu$-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 $\text{DTIME}[f]$ for some primitive recursive function $f$. 
Theorem 12.2 The following problems are decidable in polynomial time.

\[
\text{EmptyNFA} = \{ N \mid N \text{ is an NFA; } \mathcal{L}(N) = \emptyset \}
\]

\[
\Sigma^*\text{DFA} = \{ D \mid D \text{ is a DFA; } \mathcal{L}(D) = \Sigma^* \}
\]

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

\[
\text{EqualDFA} = \{ \langle D_1, D_2 \rangle \mid D_1, D_2 \text{ DFAs; } \mathcal{L}(D_1) = \mathcal{L}(D_2) \}
\]

\[
\text{EmptyCFL} = \{ G \mid G \text{ is a CFG; } \mathcal{L}(G) = \emptyset \}
\]

\[
\text{MemberCFL} = \{ \langle G, w \rangle \mid G \text{ is a CFG; } w \in \mathcal{L}(G) \}
\]
EmptyNFA = \{N: \text{No start-final path in graph of } N\}

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

D \in \Sigma^*\text{DFA} \iff \overline{D} \in \text{EmptyNFA}

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

Convert to another reachability problem:

![Diagram of a graph with nodes 0, 1, 2, 3, 4 and edges labeled with 0, 1 and 1, 0, 1]
\[
\text{EqualDFA} = \{ \langle D_1, D_2 \rangle \mid \mathcal{L}(D_1) = \mathcal{L}(D_2) \} \\
\langle D_1, D_2 \rangle \in \text{EqualDFA} \iff (\overline{D_1} \cap D_2) \cup (D_1 \cap \overline{D_2}) \\
\in \text{EmptyNFA}
\]

EmptyCFL \quad HW \#4
MemberCFL = \{ (G, w) \mid G \text{ is a CFG}; \ w \in \mathcal{L}(G) \} \\

**CYK Dynamic Programming Algorithm:**

1. Assume \( G \) in **Chomsky Normal Form**: \( N \rightarrow AB, N \rightarrow a \).
2. **Input**: \( w = w_1w_2\ldots w_n; G \) with nonterminals \( S, A, B, \ldots \).
3. \( N_{ij} \equiv \begin{cases} 1 & \text{if } N \rightarrow^* w_i\cdots w_j \\ 0 & \text{otherwise} \end{cases} \)
4. \textbf{return}(S_{1n})

\[ N_{i,i} = \textbf{if} \ ("N \rightarrow w_i" \in R) \ \textbf{then} \ 1 \ \textbf{else} \ 0 \]

\[ N_{i,j} = \bigvee_{\substack{\text{"N \rightarrow AB" \in R} \\ "N \rightarrow AB" \in R}} (\exists k)(i \leq k < j \land A_{i,k} \land B_{k+1,j}) \]
Theorem 12.3  The following problem is co-r.e.-complete:

\[ \Sigma^*\text{CFL} = \{ G \mid G \text{ is a CFG; } \mathcal{L}(G) = \Sigma^*_G \} \]

Proof: [J. Hartmanis, Neil’s advisor]

\[ \Sigma^*\text{CFL} \in \text{r.e.} : \]

Input: \( G \)

Define: \( \Sigma^*_G = \{ w_0, w_1, w_2, \ldots \} \)

1. for \( i := 0 \) to \( \infty \) {
2. if \( w_i \not\in \mathcal{L}(G) \), then return(1)\}

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

Clearly this returns 1 iff \( G \in \Sigma^*\text{CFL} \).
Proposition 12.4 $\text{EMPTY}$ is co-r.e. complete, where,

$$\text{EMPTY} = \{n \mid W_n = \emptyset\}$$

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

Claim 12.5 $\text{EMPTY} \leq \Sigma^*\text{CFL}$.

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

How can we prove the Claim?

We need to define: $g : \mathbb{N} \to \{0, 1\}^*$,

$$n \in \text{EMPTY} \iff g(n) \in \Sigma^*\text{CFL}$$

$$(\forall x) M_n(x) \neq 1 \iff \mathcal{L}(g(n)) = \Sigma_n$$

$M_n$ has no accepting computations $\iff \mathcal{L}(g(n)) = \Sigma_n$
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.

\[
\text{ID}_0 = \hat{1} \searrow w_1 w_2 \cdots w_r \searrow
\]

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

\[
\text{ID}_1 = \searrow \hat{3} w_1 w_2 \cdots w_r \searrow
\]

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) =
\{ ID_0 \# ID_1^R \# ID_2 \# ID_3^R \# \cdots \# ID_t \mid ID_0 \cdots ID_t \text{ accepting comp of } M_n \}

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{\text{YesComp}(n)} is a CFL.*

Furthermore, there is a function g ∈ F(L), for all n, g(n) codes a context-free grammar and
\[ L(g(n)) = \overline{\text{YesComp}(n)} \]

\[ \Sigma_n = \{0, 1, \triangleright, \sqcup, \#, \hat{0}, \hat{1}, \ldots, \hat{q_n} \} \text{ where } M_n \text{ has } q_n \text{ states.} \]

\[ n \in \text{EMPTY} \iff \overline{\text{YesComp}(n)} = \Sigma_n^* \iff g(n) \in \Sigma^*\text{CFL} \]

The grammar must generate every string that does not code an accepting computation of \( M_n \).
Proof:

\[ \text{YesComp}(n) = U(n) \cup A(n) \cup D(n) \cup Z(n) \]

\[ U(n) = \{ w \in \Sigma^* \mid w \text{ not in form ID}_0\# \cdots \#\text{ID}_t \} \]

\[ A(n) = \{ w \in \Sigma^* \mid w \text{ doesn’t start with an initial ID of } M_n \} \]

\[ D(n) = \{ w \in \Sigma^* \mid (\exists i)(\text{ID}_{i+1} \text{ doesn’t follow from ID}_i) \} \]

\[ Z(n) = \{ w \in \Sigma^* \mid w \text{ doesn’t end with } \hat{0} \triangleright 1 \square \} \]

\( U(n), A(n), \) and \( Z(n) \) are regular languages. To be in \( D(n) \), a string must contain a letter in \( \text{ID}_{i+1} \) that does not follow from the corresponding place in \( \text{ID}_i \) 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.

\[ \spadesuit \]
Thus, $g : \text{EMPTY} \leq \Sigma^*\text{CFL}$

$$n \in \text{EMPTY} \iff \overline{\text{YesComp}(n)} = \Sigma^*_n$$

$$\iff g(n) \in \Sigma^*\text{CFL}$$
<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic Hierarchy</td>
<td>Recursive</td>
</tr>
<tr>
<td></td>
<td>co-r.e. complete</td>
</tr>
<tr>
<td></td>
<td>r.e. complete</td>
</tr>
<tr>
<td></td>
<td>co-r.e.</td>
</tr>
<tr>
<td></td>
<td>r.e.</td>
</tr>
<tr>
<td>Polynomial-Time Hierarchy</td>
<td>NP</td>
</tr>
<tr>
<td></td>
<td>co-NP</td>
</tr>
<tr>
<td></td>
<td>NP ∩ co-NP</td>
</tr>
<tr>
<td></td>
<td>P</td>
</tr>
<tr>
<td></td>
<td>&quot;truly feasible&quot;</td>
</tr>
<tr>
<td></td>
<td>NC</td>
</tr>
<tr>
<td></td>
<td>NC²</td>
</tr>
<tr>
<td></td>
<td>log(CFL)</td>
</tr>
<tr>
<td></td>
<td>SAC¹</td>
</tr>
<tr>
<td></td>
<td>NSPACE[log n]</td>
</tr>
<tr>
<td></td>
<td>DSPACE[log n]</td>
</tr>
<tr>
<td></td>
<td>Regular</td>
</tr>
<tr>
<td></td>
<td>NC¹</td>
</tr>
<tr>
<td></td>
<td>ThC⁰</td>
</tr>
<tr>
<td></td>
<td>Logarithmic-Time Hierarchy</td>
</tr>
<tr>
<td></td>
<td>AC⁰</td>
</tr>
</tbody>
</table>