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

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

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

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

Theorem: $K$, HALT, and $A_{0,17}$ are r.e. complete.
Def: We say that \( S \) is reducible to \( T \), \( S \leq T \), iff \( f \in F(L) \),

\[
(\forall w \in N) \quad (w \in S) \iff (f(w) \in T)
\]

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

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

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 $\mathbb{N}$ 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 $\ell$ 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 $M_\ell$ 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 $M_\ell$ on the original input. This machine simulates $M_\ell$ 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.)

$\spadesuit$
On HW#2 we defined the programming language Bloop, with integer variables and bounded loops. It turns out that the class of functions from $\mathbb{N}$ to $\mathbb{N}$ 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:**

\[ \zeta() = 0 \]

\[ \sigma(x) = x + 1 \]

\[ \pi^n_i(x_1, \ldots, x_n) = x_i, \quad n = 1, 2, \ldots, \quad 1 \leq i \leq n \]

**Composition:** \[ g_i : \mathbb{N}^k \to \mathbb{N}, 1 \leq i \leq m; \quad ; h : \mathbb{N}^m \to \mathbb{N} : \]

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

**Primitive Recursion:** \[ g : \mathbb{N}^k \to \mathbb{N}; \quad h : \mathbb{N}^{k+2} \to \mathbb{N} : \]

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

\[ f(0, y_1, \ldots y_k) = g(y_1, \ldots, y_k) \]

\[ f(n + 1, y_1, \ldots y_k) = h(f(n, y_1, \ldots, y_k), n, y_1, \ldots, y_k) \]

**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. $M_1(x) = \text{if } (x > 0) \text{ then } (x - 1) \text{ else } 0$

2. $x \Theta y = \text{if } (y \leq x) \text{ then } (x - y) \text{ else } 0$

3. +

4. *

5. $\exp(x, y) = y^x$

6. $\exp^*(x) = \text{if } (x = 0) \text{ then } 1 \text{ else } 2^{\exp^*(x-1)}$

$$\exp^*(x) = 2^2 \overbrace{}^{x}$$
Gödel discovered that you can \textit{code sequences} in \texttt{PrimRecFcn}s, which he did using number theory:

\textbf{Proposition 8.4} Prime, PrimeF $\in$ \texttt{PrimRecFcn}s, where,

$$\text{Prime}(x) = \textbf{if} \ ("x \ is \ prime") \ \textbf{then} \ 1 \ \textbf{else} \ 0$$

PrimeF(n) = \textit{prime number} n, \textit{i.e}, PrimeF(0) = 2, PrimeF(1) = 3, PrimeF(2) = 5, PrimeF(3) = 7, PrimeF(4) = 11.

\textbf{Proof:}

$$x \mid y = (\exists z \leq y)(xz = y)$$

$$\text{Prime}(x) = x > 1 \land (\forall y < x)(y \mid x \rightarrow y = 1)$$

NextPrime(x) = \mu\{t \leq (x + 1)^{x+1} \mid t > x \land \text{Prime}(t)\}

PrimeF(0) = 2

PrimeF(x + 1) = \text{NextPrime}(\text{PrimeF}(x))$$

\hfill \spadesuit
Proposition 8.5

IsSeq, Length, Item ∈ PrimRecFcns, where,

\[ \text{Seq}(a_0, a_1, \ldots, a_n) = 2^{a_0+1}3^{a_1+1} \cdots \text{PrimeF}(n)^{a_n+1} \]

\[ \text{IsSeq}(S') = \begin{cases} 1 & \text{if } (S \text{ is a Sequence number}) \\ 0 & \text{else} \end{cases} \]

\[ \text{Length}(\text{Seq}(a_0, a_1, \ldots, a_n)) = n + 1 \]

\[ \text{Item}(\text{Seq}(a_0, a_1, \ldots, a_n), i) = a_i \]

Proof:

\[ \text{Good}(x, S) = (\forall y < S)((y < x \land \text{PrimeF}(y)|S) \lor (y \geq x \land \text{PrimeF}(y) \not|S)) \]

\[ \text{IsSeq}(S') = (\exists x < S)\text{Good}(x, S) \]

\[ \text{Length}(S') = \mu \{ x < S \mid \text{Good}(x, S) \} \]

\[ \text{Item}(S, i) = \mu \{ y < S \mid \text{IsSeq}(S') \land \text{PrimeF}(i)^{y+1}|S \land \text{PrimeF}(i)^{y+2} \not|S \} \]

\[ \hfill \blacklozenge \]
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 \( \text{COMP}(n, x, c, y) \) mean \( M_n(x) = y \), and that \( c \) is \( M_n \)'s complete computation on input \( x \).

Then \( \text{COMP} \) 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:

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

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

\[
\text{START}(\text{Item}(c, 0), x) \land \text{END}(\text{Item}(c, \text{Length}(c) - 1), y) \land
(\forall i < \text{Length}(c)) \text{NEXT}(n, \text{Item}(c, i), \text{Item}(c, i + 1))
\]
Theorem 8.6 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 \quad HW4

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

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

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

HW \#3
MemberCFL = \{ \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:** $N \rightarrow AB$, $N \rightarrow a$.

2. **Input:** $w = w_1w_2\ldots w_n$; $G$ with nonterminals $S, A, B, \ldots$

3. $N_{ij} \equiv \begin{cases} 1 \text{ if } N \ \overset{*}{\rightarrow} \ w_i\ldots w_j \\ 0 \text{ otherwise} \end{cases}$

4. **return**($S_{1n}$)

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

\[ N_{i,j} = \bigvee_{\text{"N \rightarrow AB"} \in R} (\exists k)(i \leq k < j \wedge A_{i,k} \wedge B_{k+1,j}) \]