Counter Machines

A *counter machine* is a machine consisting of a read-only input tape, a finite-state control, and a fixed number of *counters*, each capable of storing an arbitrary natural number.

The following operations can be performed on the counters:

- increment by 1;
- decrement by 1 if nonzero; and
- comparison with 0.

**Theorem 8.14:** Every RE language is accepted by a 3-counter machine.

**Proof sketch:** Let $L$ be an RE language, and let $M$ be a 2-stack machine such that $L(M) = L$.

- Suppose the stack alphabet $\Gamma$ of $M$ contains $r - 1$ symbols.
- We can define $f : \Gamma \xrightarrow{1-1} \text{onto} \{1, \ldots, r - 1\}$. 
- We define $g : \Gamma^* \rightarrow \mathbb{N}$ such that
  \[
  g(\epsilon) = 0 \\
  g(X\alpha) = f(X) + rg(\alpha) \text{ for } X \in \Gamma
  \]
- It is easily seen by induction on $\alpha \in \Gamma^*$ that if $g(\alpha) = g(\beta)$, then $\alpha = \beta$.
- We can store the contents $\alpha$ of a stack in a counter as $g(\alpha)$.
- The third counter is used as temporary storage.

To examine the top symbol of a stack, we compute
  \[
  g(X\alpha) \mod r = f(X)
  \]
  as follows:
  - Use finite control to implement a $\mod r$ counter $c$.
  - While decrementing the counter storing $g(X\alpha)$, increment the temporary counter and $c$. 
To remove the top symbol of a stack, we compute

\[ [g(X\alpha)/r] = g(\alpha). \]

To push a symbol \( X \), we compute

\[ rg(\alpha) + f(X) = g(X\alpha) \]

Theorem 8.15: Every RE language is accepted by a 2-counter machine.

Proof sketch: Let \( L \) be an RE language, and let \( M \) be a 3-counter machine such that \( L(M) = L \).

- We define \( f : \mathbb{N} \times \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N} \) such that
  \[ f(i, j, k) = 2^i 3^j 5^k. \]
- Because 2, 3, and 5 are prime, and because every positive integer has a unique prime factorization, \( f \) is one-to-one.
- We encode the three counter values \( i, j, \) and \( k \) as \( f(i, j, k) \).
• We can increment/decrement these values by multiplying/dividing by 2, 3, or 5, using the other counter as temporary storage.

• We can compare any of these values with 0 by computing \( f(i, j, k) \mod 2, 3, \) or 5.

Reduction

**Defn:** Let \( L_1 \subseteq \Sigma^* \), \( L_2 \subseteq \Delta^* \). We say that \( L_1 \) is many-one reducible to \( L_2 \) (\( L_1 \leq L_2 \)) if there is a TM \( M = (Q, \Sigma, \Gamma, \delta, q_0, B, \{q\}) \) such that

- \( \Delta \subseteq \Gamma \);

- on every input, \( M \) halts in an ID \( qy \) for some \( y \in \Delta^* \); and

- if \( q_0x \vdash_M^* qy \), then \( x \in L_1 \) iff \( y \in L_2 \).

**Claim:** \( L_1 \leq L_2 \) iff \( \overline{L_1} \leq \overline{L_2} \).
Theorem 9.7: Let $L_1 \leq L_2$. Then

- If $L_2$ is RE, then $L_1$ is RE.
- If $L_2$ is recursive, then $L_1$ is recursive.

Proof sketch: Let $M$ reduce $L_1$ to $L_2$, and let $L(M_2) = L_2$. We construct $M_1$ that on input $x$:

1. Simulates $M$ on $x$, producing $y$.
2. Simulates $M_2$ on $y$.

$x \in L(M_1)$ iff $y \in L(M_2)$ iff $x \in L_1$, and $M_1$ halts on every input iff $M_2$ halts on every input.

Defn: Given a string $w \in \{0,1\}^*$, let $M(w)$ denote the TM with input alphabet $\{0,1\}$ that $w$ encodes in some fixed TM encoding scheme.

We can define a function $f : \{0,1\}^* \rightarrow 2^{\{0,1\}^*}$ such that

$$f(w) = L(M(w)).$$

Note that the range of $f$ is the set of RE languages over $\{0,1\}$. 
Cantor’s Theorem: Let $A$ be a set, and let $f : A \to 2^A$. Then

$$B = \{x \in A \mid x \not\in f(x)\} \not\in \text{ran}(f)$$

Proof: By contradiction.

- Assume that for some $a \in A$, $f(a) = B$.
- Then $a \in B$ iff $a \not\in B$ — a contradiction.

Theorem 9.2: Let $L_d = \{w \in \{0,1\}^* \mid w \not\in L(M(w))\}$. Then $L_d$ is not RE.

The Universal Language

Defn: Let

$$L_U = \{wx \in \{0,1\}^* \mid x \in L(M(w))\}.$$  

We call $L_U$ the universal language.

Assumption: Our TM encoding scheme is such that if $w$ is a valid TM encoding and $x$ is a proper prefix of $w$, then $x$ is not a valid TM encoding.
Claim: $\overline{L_U}$ is not RE.

Proof sketch: We will show that $L_d \leq \overline{L_U}$. Let $M$ be a TM that operates as follows on input $x \in \{0,1\}^*$:

1. If $x$ is not a valid TM encoding, output $\epsilon$.
2. Otherwise, output $xx$.

$x \in L_d$ iff $y \in \overline{L_U}$, where $y$ is the string output by $M$.

Claim: $L_U$ is RE.

Proof sketch: We define a 4-tape TM $U$ such that $L(U) = L_U$. 

- **Tape 1:** The input.
- **Tape 2:** The contents of $M(w)$'s tape encoded in binary.
- **Tape 3:** The current state of $M(w)$, encoded in binary.
- **Tape 4:** Temporary storage.
• $U$ first scans the input for a prefix $w$ that is a valid TM encoding.

• $U$ then encodes $x$, the remainder of the input, on Tape 2 in binary, with encoded symbols separated by #.

• $U$ then writes 0 on Tape 3.

• $U$ then simulates $M(w)$ on $x$.

Theorem 9.3: For any alphabet $\Sigma$, the set of recursive languages over $\Sigma$ is closed under complement.

Proof sketch:

• Let $L \subseteq \Sigma^*$ be recursive.

• Let $M = (Q, \Sigma, \delta, q_0, B, F)$ be a TM such that $L(M) = L$ and $M$ halts on all inputs.

• W.o.l.o.g., assume there are no transitions from any final state of $M$. 
We construct a TM $M' = (Q \cup \{q_f\}, \Sigma, \Gamma, \delta', q_0, B, \{q_f\})$, where

- $\delta'(q, X) = \delta(q, X)$ if $\delta(q, X)$ is defined or if $q \in F$;
- $\delta'(q, X) = (q_f, X, R)$ if $q \in Q$, $\delta(q, X)$ is undefined, and $q \not\in F$; and
- $\delta'(q_f, X)$ is undefined.

Then $x \in L(M')$ iff $x \not\in L(M)$ iff $x \in \overline{L}$, and $M'$ halts on all inputs.

**Corollary:** $L_U$ is not recursive.

**Corollary:** $\overline{L_d}$ is RE but not recursive.