(Pumping lemma for CFLs) *For every CFL A, there exists $k \geq 0$ such that every $z \in A$ of length at least k can be broken up into five substrings $z = uvwxy$ such that $vx \neq \epsilon$, $|vwx| \leq k$, and for all $i \geq 0$, $uv^iwx^iy \in A$.*

For all $k \geq 0$, there exists $z \in A$ of length at least k such that for all ways of breaking z up into substrings $z = uvwxy$ with $vx \neq \epsilon$ and $|vwx| \leq k$, there exists an $i \geq 0$ such that $uv^iwx^iy \notin A$. (not CFL condition)

Disprove for $a^nb^na^n$ not CFL, pick $z = a^kb^kc^k$ given k.

$A = \{w.w\}$ - intx with $a^*b^*a^*b^* \to a^nb^ma^nb^m$. Pick $z = a^kb^ka^kb^k$.

Surprisingly, the complement of A, namely

$$\{a,b\}^* - \{ww \mid w \in \{a,b\}^*\},$$

is a CFL. Here is a CFG for it:

$$S \to AB \mid BA \mid A \mid B,$$
$$A \to CAC \mid a,$$
$$B \to CBC \mid b,$$
$$C \to a \mid b.$$

Acceptance: empty stack or $q \in F$

CFG → NPDA

$$M = (Q, \Sigma, \Gamma, \delta, s, \bot, F),$$

where

- Q is a finite set (the *states*),
- Σ is a finite set (the *input alphabet*),
- Γ is a finite set (the *stack alphabet*),
- $\delta \subseteq (Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma) \times (Q \times \Gamma^*)$, δ finite (the *transition relation*),
- $s \in Q$ (the *start state*),
- $\bot \in \Gamma$ (the *initial stack symbol*), and
- $F \subseteq Q$ (the *final* or *accept states*).

The transition relation δ is defined as follows. For each production

$$A \to cB_1B_2\cdots B_k$$

in P, let δ contain the transition

$$((q,c,A),\ (q,B_1B_2\cdots B_k)).$$

$$((p,a,A),\ (q,B_1B_2\cdots B_k)) \in \delta,$$

this means intuitively that whenever the machine is in state p reading input symbol a on the input tape and A on the top of the stack, it can pop A off the stack, push $B_1B_2\cdots B_k$ onto the stack (B_k first and B_1 last), move its read head right one cell past the a, and enter state q. If

pf: *For any $z, y \in \Sigma^*$, $\gamma \in N^*$, and $A \in N$, $A \xrightarrow[G]{n} z\gamma$ via a leftmost derivation if and only if $(q, zy, A) \xrightarrow[M]{n} (q, y, \gamma)$.*

(induct on n)

$$\begin{aligned} & x \in L(G) \\ \iff & S \xrightarrow[G]{*} x \text{ by a leftmost derivation} && \text{definition of } L(G) \\ \iff & (q,x,S) \xrightarrow[M]{*} (q,\epsilon,\epsilon) && \text{Lemma 24.1} \\ \iff & x \in L(M) && \text{definition of } L(M). \end{aligned}$$ □

for NPDA → CFG, we have similar construction, proof.

1-state

n NPDA → 1 NPDA

$$\Gamma' \stackrel{\text{def}}{=} Q \times \Gamma \times Q.$$
$$M' = (\{*\}, \Sigma, \Gamma', \delta', *, \langle s \bot t\rangle, \varnothing)$$

Let M' be the NPDA constructed from M as above. Then

$$(p, x, B_1B_2\cdots B_k) \xrightarrow[M]{n} (q,\epsilon,\epsilon)$$

if and only if there exist $q_0, q_1, \ldots, q_k$ such that $p = q_0$, $q = q_k$, and

$$(*, x, \langle q_0\ B_1\ q_1\rangle\langle q_1\ B_2\ q_2\rangle\cdots\langle q_{k-1}\ B_k\ q_k\rangle) \xrightarrow[M']{n} (*,\epsilon,\epsilon).$$

In particular,

$$(p,x,B) \xrightarrow[M]{n} (q,\epsilon,\epsilon) \iff (*,x,\langle p\ B\ q\rangle) \xrightarrow[M']{n} (*,\epsilon,\epsilon).$$

$$((p,c,A),\ (q_0,B_1B_2\cdots B_k)) \in \delta,$$

where $c \in \Sigma \cup \{\epsilon\}$, include in δ' the transitions

$$((*,c,\langle p\ A\ q_k\rangle),\ (*,\langle q_0\ B_1\ q_1\rangle\langle q_1\ B_2\ q_2\rangle\cdots\langle q_{k-1}\ B_k\ q_k\rangle))$$

Theorem 25.2 $L(M') = L(M)$.

Proof. For all $x \in \Sigma^*$,

$$\begin{aligned} x \in L(M') &\iff (*,x,\langle s \bot t\rangle) \xrightarrow[M']{*} (*,\epsilon,\epsilon) \\ &\iff (s,x,\bot) \xrightarrow[M]{*} (t,\epsilon,\epsilon) \\ &\iff x \in L(M). \end{aligned}$$

Turing Machines

Formally, a *deterministic one-tape Turing machine* is a 9-tuple

$$M = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, s, t, r),$$

where

- Q is a finite set (the *states*);
- Σ is a finite set (the *input alphabet*);
- Γ is a finite set (the *tape alphabet*) containing Σ as a subset;
- $\sqcup \in \Gamma - \Sigma$, the *blank symbol*;
- $\vdash \in \Gamma - \Sigma$, the *left endmarker*;
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$, the *transition function*;
- $s \in Q$, the *start state*;
- $t \in Q$, the *accept state*; and
- $r \in Q$, the *reject state*, $r \neq t$.

$$\delta(p, \vdash) = (q, \vdash, R).$$

We also require that once the machine enters its accept state, it never leaves it, and similarly for its reject state; that is, for all $b \in \Gamma$ there exist $c, c' \in \Gamma$ and $d, d' \in \{L, R\}$ such that

$$\begin{aligned} \delta(t,b) &= (t,c,d), \\ \delta(r,b) &= (r,c',d'). \end{aligned} \tag{28.3}$$

We define a *configuration* to be an element of $Q \times \{y\sqcup^\omega \mid y \in \Gamma^*\} \times \mathbb{N}$, where $\mathbb{N} = \{0,1,2,\ldots\}$. A configuration is a global state giving a snapshot of all relevant information about a TM computation at some instant in time. The configuration (p,z,n) specifies a current state p of the finite control, current tape contents z, and current position of the read/write head $n \geq 0$. We usually denote configurations by α, β, γ.

The *start configuration* on input $x \in \Sigma^*$ is the configuration

$$(s, \vdash x\sqcup^\omega, 0).$$

The relation $\xrightarrow[M]{1}$ is defined by

$$(p,z,n) \xrightarrow[M]{1} \begin{cases} (q, \mathrm{s}_b^n(z), n-1) & \text{if } \delta(p,z_n) = (q,b,L), \\ (q, \mathrm{s}_b^n(z), n+1) & \text{if } \delta(p,z_n) = (q,b,R). \end{cases}$$

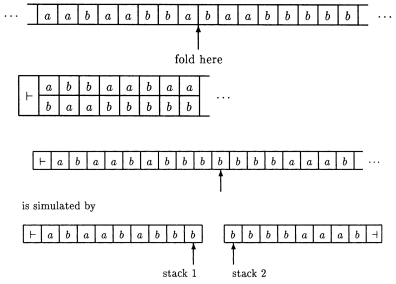
We define the reflexive transitive closure $\xrightarrow[M]{*}$ of $\xrightarrow[M]{1}$ inductively, as usual:

- $\alpha \xrightarrow[M]{0} \alpha$,
- $\alpha \xrightarrow[M]{n+1} \beta$ if $\alpha \xrightarrow[M]{n} \gamma \xrightarrow[M]{1} \beta$ for some γ, and
- $\alpha \xrightarrow[M]{*} \beta$ if $\alpha \xrightarrow[M]{n} \beta$ for some $n \geq 0$.

However, if both A and $\sim A$ are r.e., then A is recursive. To see this, suppose both A and $\sim A$ are r.e. Let M and M' be TMs such that $L(M) = A$ and $L(M') = \sim A$. Build a new machine N that on input x runs both M and

$$P \text{ is decidable} \iff \{x \mid P(x)\} \text{ is recursive.}$$
$$A \text{ is recursive} \iff \text{“}x \in A\text{” is decidable,}$$

$$P \text{ is semidecidable} \iff \{x \mid P(x)\} \text{ is r.e.,}$$
$$A \text{ is r.e.} \iff \text{“}x \in A\text{” is semidecidable.}$$



2 stack ≡ 4 counter ≡ 2 counters

ues i, j, k, ℓ in its counters, the two-counter automaton will have the value $2^i 3^j 5^k 7^\ell$ in its first counter. It uses its second counter to effect the counter operations of the four-counter automaton. For example, if the four-counter automaton wanted to add one to k (the value of the third counter), then the two-counter automaton would have to multiply the value in its first counter by 5. This is done in the same way as above, adding 5 to the second counter for every 1 we subtract from the first counter. To simulate a test for zero, the two-counter automaton has to determine whether the value in its first counter is divisible by 2, 3, 5, or 7, respectively, depending on which counter of the four-counter automaton is being tested.

The machine M is said to *accept* input $x \in \Sigma^*$ if

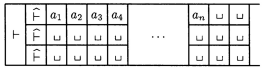
$$(s, \vdash x\sqcup^\omega, 0) \xrightarrow[M]{*} (t, y, n)$$

for some y and n, and *reject* x if

$$(s, \vdash x\sqcup^\omega, 0) \xrightarrow[M]{*} (r, y, n)$$

- *recursively enumerable* (r.e.) if it is $L(M)$ for some Turing machine M,
- *co-r.e.* if its complement is r.e., and
- *recursive* if it is $L(M)$ for some *total* Turing machine M.

Multi-tape



Each step of M is simulated by several steps of N. To simulate one step of M, N starts at the left of the tape, then scans out until it sees all three marks, remembering the marked symbols in its finite control. When it has seen all three, it determines what to do according to M's transition function δ, which it has encoded in its finite control. Based on this information, it goes back to all three marks, rewriting the symbols on each track and moving the marks appropriately. It then returns to the left end of the tape to simulate the next step of M.

Stack ≡ 2 counters

finitely many stack symbols as binary numbers of fixed length, say m; then pushing or popping one stack symbol is simulated by pushing or popping m binary digits. Then the contents of the stack can be regarded as a binary number whose least significant bit is on top of the stack. The simulation maintains this number in the first of the two counters and uses the second to effect the stack operations. To simulate pushing a 0 onto the stack, we need to double the value in the first counter. This is done by entering a loop that repeatedly subtracts one from the first counter and adds two to the second until the first counter is 0. The value in the second counter is then twice the original value in the first counter. We can then transfer that value back to the first counter, or just switch the roles of the two counters. To push 1, the operation is the same, except the value of the second counter is incremented once at the end. To simulate popping, we need to divide the counter value by two; this is done by decrementing one counter while incrementing the other counter every second step. Testing the parity of the original counter contents tells whether a simulated 1 or 0 was popped.

Once we have a suitable encoding of Turing machines, we can construct a *universal Turing machine* U such that

$$L(U) \stackrel{\text{def}}{=} \{M\#x \mid x \in L(M)\}.$$