1 Pumping Lemmas

1.1. $L_{a,b,c}$ is regular if and only if it is unary.

1.1.1) If at least two of a, b, c are zero, then $(0^a1^b2^c)^*$ is a regular expression recognizing $L_{a,b,c}$.

1.1.2) We show that if at least two of a, b, c are non-zero, then $L_{a,b,c}$ is not regular. Let n be the pumping length and take $w := 0^{an}1^{bn}2^{cn} \in L_{a,b,c}$. We can decompose w into xyz with $|xy| \leq n$ and $|y| \geq 1$ such that xy^iz is in the language for all $i \in \mathbb{N}$. Since $|xy| \leq n$, string y is composed of 0s only when $a \neq 0$, and 1s only otherwise (a and b cannot be both zero). We pump down and look at $wz \in L_{a,b,c}$. If y has 0s only, then $|wz|_0 < an$ but still $|wz|_1 = bn$ and $|wz|_2 = cn$. Since $a \neq 0$ and either b or c is non-zero, xz is not of the correct form. Thus xz is not in the language, a contradiction. If y has 1s only, then $|wz|_0 = 0 = an$ but $|wz|_1 < bn$ and $|wz|_2 = cn$. In this case both b and c are non-zero. Once again xz is not of the correct form, which is a contradiction.

1.2. $L_{a,b,c}$ is context-free if and only if it is binary.

1.2.1) If at least one of a, b, c is zero (say, $a = 0$), then $S \to 1^b\ S\ 2^c \mid \varepsilon$ is a context-free grammar that generates $L_{a,b,c}$.

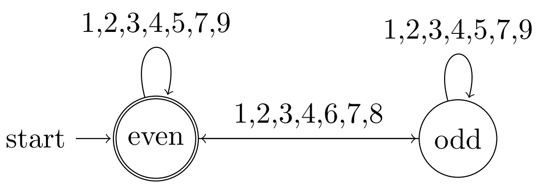
1.2.2) We show that if a, b, c are all non-zero, then $L_{a,b,c}$ is not context-free. Let n be the pumping length and take $z := 0^{an}1^{bn}2^{cn} \in L_{a,b,c}$. We can decompose z into $uvwxy$ with $|vwx| \leq n$ and $|vx| \geq 1$ such that uv^iwx^iy is in the language for all $i \in \mathbb{N}$. Since $|vwx| \leq n$, string vwx spans across at most two letters (since a, b, c are all non-zero). That is, vwx misses either 0s or 2s. We pump down and look at $uwy \in L_{a,b,c}$. Suppose vwx misses 0s. Since $|vx| \geq 1$, string vx has a 1 or a 2. But then $|uwy|_0 = an$ and either $|uwy|_1 < bn$ or $|uwy|_2 < cn$. Thus uwy is not in the language, a contradiction. Now suppose vwx misses 2s. Since $|vx| \geq 1$, string vx has a 0 or a 1. But then $|uwy|_2 = cn$ and either $|uwy|_0 < an$ or $|uwy|_1 < bn$. Again, this means uwy is not in the language.

(Alternatively use closure under the homomorphism $(0^a, 1^b, 2^c) \mapsto (0, 1, 2)$, and the facts that 0^n1^n is not regular and $0^n1^n2^n$ is not context-free.)

2 A Game of Dominoes

2.1. The language of dominoes is regular since it is $\Sigma^* \setminus \bigcup_{(i,j)\in F} \Sigma^* ij \Sigma^*$ where F is the set of forbidden pairs (u, v) where $r(u) = 1 \wedge l(v) = 2$ or $r(u) = 2 \wedge l(v) = 1$. These pairs are $F = \{2, 5, 8\} \times \{7, 8, 9\} \cup \{3, 6, 9\} \times \{4, 5, 6\}$.

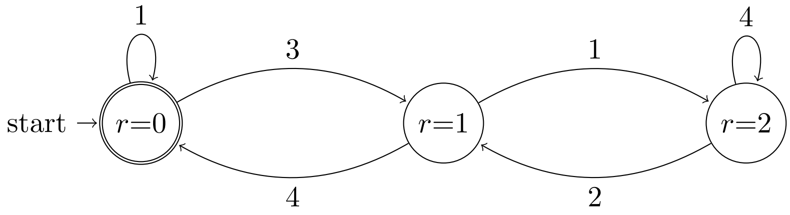
2.2. We simulate a computation of the parity using an NFA with two states (even and odd). It is an NFA because an encouter with a joker can move the automaton to both states. The automaton below accepts the empty sequence but by removing a single value the language stays regular.



2.3. We simulate the computation top $- 3 \times$ bottom. After each domino is read, we consider the reminder which is $t - 3 \times b$ where t is the top row read and b is the bottom read. Let r be the current reminder (i.e., $r = t_0 \cdots t_i - 3 \times b_0 \cdots b_i$). Then the reminder after reading $\frac{t_{i+1}}{b_{i+1}}$ is $2r + t_{i+1} - 3 \times b_{i+1}$. The effects of the domino i on the reminder r is described by $f(r, i)$ where

$$f(r,1) = 2r, \quad f(r,2) = 2r - 3, \quad f(r,3) = 2r + 1, \quad f(r,4) = 2r - 2\ .$$

Our automaton has states $r = 0$, $r = 1$, $r = 2$ and the transition function is f. The automaton starts with a reminder of zero and accepts when the reminder is zero. We only consider the states $\{0, 1, 2\}$ because once outside this set we can never return to it: for $k \in \{-3, -2, 1, 0\}$ if $r \leq -1$ then $2r + k \leq 2r + 1 \leq r$ and if $r \geq 3$ then $2r + k \geq r + (r - 3) \geq r$.



3 The Dichotomy Property

Let L be a regular language accepted by a deterministic finite automaton with states Q.

3.1. Clearly if $\exists w \in L \ : \ |w| \leq |Q|$ then L is non-empty. Now if L is non-empty then we take a word $w = w_1 \cdots w_k$ of shortest length in L. A run of the automaton for w will go through the states $q_1, \ldots, q_k$. Now if $k > |Q|$ we would have that $q_i = q_j$ for some $i < j$. But then $w_1 \cdots w_i w_{j+1} \cdots w_k$ is a shorter word that is also in L. Hence $k \leq |Q|$.

3.2. We first show that each word $w \in L$ with $|Q| < |w|$ can be reduced to a word $w' \in L$ with $|w'| < |w| \leq |w'| + |Q|$ or augmented to a word w'' with $|w| < |w''|$.

A run of the automaton on $w = w_1 \cdots w_n$ passes through states $q_1, \ldots, q_n$. Since $n > |Q|$, there are $i < j$ such that $q_i = q_j$. Consider a pair $i < j$ with minimal $j - i$. By minimality, states $q_i, \ldots, q_{j-1}$ are all distinct. Therefore $j - i \leq |Q|$ and the word $w' = w_1 \cdots w_i w_{j+1} \cdots w_n$ is in the language with $|w'| < |w|$ and $|w'| \geq |w| - |Q|$. But the word $w'' = w_1 \cdots w_i w_{i+1} \cdots w_j w_{i+1} \cdots w_j w_{j+1} \cdots w_n$ is also in the language with $|w''| > |w|$.

If L is infinite we can find a $w \in L$ such that $|Q| < |w|$. The we can repeatedly reduce w until $|Q| < |w| \leq 2|Q|$. Such a method works because at each step we reduce the length by at least by 1 and at most $|Q|$. Conversely, if $w \in L$ with $|Q| < |w|$ then by iteratively augmenting w we can create a sequence $w_0 := w$ and $w_{i+1} := w_i''$ such that $w_i \in L$ for all i. This shows L is infinite.

4 Intersection of Regular and Context-Free Languages

4.1. Let $M = (Q, \Sigma, \Gamma, \delta, q_0, Z, F)$ be a PDA recognizing L' and $A = (Q', \Sigma, \gamma, q_0', F')$ be a DFA recognizing L. Then $L' \cap L$ is recognized by $I = (Q \times Q', \Sigma, \Gamma, \rho, (q_0, q_0'), Z, F \times F')$ where $\rho((q, q'), c, p) = ((\bar{q}, \bar{q}'), \bar{p})$ with $(\bar{q}, \bar{p}) = \delta(c, q, p)$ and $\bar{q}' = \gamma(c, q')$. We also extend γ with $\gamma(\epsilon, q') = q'$.

If a word w is recognized by I then decompose a run of the automaton into $((q_i, q_i'), p_i, c_i)_{i \in 1..n}$ where (q_i, q_i') is the state after the i-th transition, p_i is stack state and $c_i \in \Sigma \cup \{\epsilon\}$ is the transition letter. Then $(q_i, p_i, c_i)_{i \in 1..n}$ corresponds to a run of M and $(q_i', c_i)_{i \in N | c_i \neq \epsilon}$ corresponds to a run of A both of which accept w. Therefore $w \in L \cap L'$.

Conversely, if $w \in L \cap L'$ we can find a run $(q_i, p_i, c_i)_{i \in 1..n}$ of M and a run $(q_i', c_i)_{i \in 1..n | i \neq \epsilon}$ of A both accepting w. Then $(q_i, q_i'), p_i, c_i$ is a valid run of I accepting w.

5 Boolean Expressions

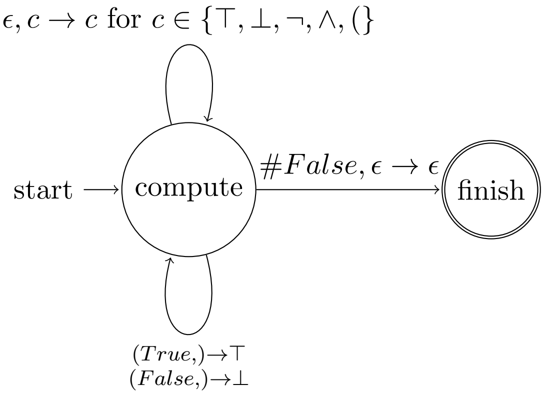
$G := (\{Be, St\}, \Sigma, R, Be)$, where $\Sigma = \{\wedge, \neg, \top, \bot, \ (, \ )\}$ and the production rules R are

$$Be \rightarrow St \wedge St \mid \neg St \mid St \quad \text{and} \quad St \rightarrow \top \mid \bot \mid (Be) \ .$$

5.1. We duplicate each term one for true and one for false (i.e. $Be^\top, Be^\perp, St^\top, St^\perp$) and adapt the rules in consequence ($Be^\top$ is the start symbol):

$$\begin{array}{rcl} St^\top & \to & \top \mid (Be^\top) \\ Be^\top & \to & St^\top \wedge St^\top \mid \neg St^\perp \mid St^\top \end{array} \qquad \begin{array}{rcl} St^\perp & \to & \perp \mid (Be^\perp) \\ Be^\perp & \to & St^\top \wedge St^\perp \mid St^\perp \wedge St^\perp \mid St^\perp \wedge St^\top \mid \neg St^\top \mid St^\perp \end{array}$$

5.2. We use one state for the end of computation and one state for the actual computation. The computing state reduces elements of St to $\top$ or $\perp$ as soon as they are completely read so the stack never contains ')' but can contain all other symbols.



The rules above exist for each $True \in \{\ \top \wedge \top\ ,\ \ \neg\perp\ ,\ \ \top\ \}$ and for each $False \in \{\ \top \wedge \perp\ ,\ \ \perp \wedge \perp\ ,\ \ \perp \wedge \top\ ,\ \ \neg\top\ ,\ \ \perp\ \}$.

5.3. A word u is well-parenthesized when all prefixes of u contain more (s than)s and in total u contain an equal number of them. This well-parenthesized property can be shown by induction on the length of derivation for terms generated by St and Be. For length 1 this is clear. For the induction step we see that all rules preserve this criterion and thus the well-parenthesizing of the terms generated.

We use the following lemma: u cannot be well-parenthesized and a strict prefix of (v) where v is well-parenthesized. All strict, non-empty prefixes of (v) are prefixes of v with a '(' at the beginning. As prefixes of v contain more '(' then ')' the additional '(' imposes that they are not well-parenthesized.

Let w be a minimal word with two distinct derivations $Be \xrightarrow{*} w$. We have the following.

- w cannot be a constant.
- If one of the first two productions of w is $Be \to St \to (Be)$ then $w = (w')$. Then in the other derivation the first production cannot be $St \wedge St$. Otherwise $w = w_1 \wedge w_2$ where w_1 is a strict prefix of (w') which is impossible by our lemma.

  The other first production also cannot be $Be \to \neg St$ (as w starts with '(') and thus all the first productions are $Be \to St \to (Be)$ and $w = (w')$ where w' should be a smaller counterexample.
- Combining the two facts above, the first production of w cannot be $Be \to St$.
- If one the derivations of w starts with $Be \to \neg St$ since St is a parenthesized word then we cannot have another derivation of the form $Be \to St \wedge St$ otherwise $w = \neg w' = w_1 \wedge w_2$ with w_1 that is either a constant or of the form (u) and thus cannot start with $\neg$. If all first productions of w are $Be \to \neg St$ then we have a smaller counterexample w'.
- Therefore all first derivations of w are $Be \to St \wedge St$. Let us consider two: $w = w_1 \wedge w_2 = w_1' \wedge w_2'$ where w_1, w_1', w_2 and w_2' are all constants ($\top$ or $\perp$) or well-parenthesized words of the form (u) (with u also well-parenthesized). If one is a constant so is the other and one cannot be the strict prefix of the other (by our lemma) thus $w_1 = w_1'$ and so $w_2 = w_2'$ which gives us a smaller counterexample.

All in all such a minimal counterexample cannot exist thus the grammar is unambiguous.

6 Finite Context-Free Languages

6.1. Suppose L is infinite and let $z_0 \in L$. Suppose we have constructed $z_0, \ldots, z_i$. Since L is infinite, there is a finite number of words of length smaller than $2|z_i|$. Therefore we can find a word z_{i+1} such that $|z_{i+1}| \geq 2|z_i|$. Let S be the subset of L recursively constructed in this manner. By assumption S is context-free. Let n be its pumping length. Since S is infinite, there is a $z \in S$ with $|w| > n$ such that we can write $z = uvwxy$ with $uv^2wx^2y \in S$ and $1 \leq |vx| \leq n$. But then $|z| < |uv^2wx^2y| \leq |z| + n < 2|z|$, which contradicts the construction of S. Therefore L is finite.

(Alternatively: there are countably many context-free languages whereas an infinite set has an uncountable number of subsets! Thanks to Florent Noisette for this hack!)

7 Universal Automata

7.1. Suppose $L_U := \{f(D)\#w : w \in L(D)\}$ is regular and let n be its pumping length. The language $L = \{0^{2n}\}$ is also regular. Let $L = L(D)$. Thus $w = f(D)\#0^{2n} \in L_U$. By pumping at the *end* of w we have that $w = xyz$ such that $|y| \geq 1$, $|yz| \leq n$ and $xz \in L_U$. But since $|xyz| > 2n+1$ and $|yz| \leq n$, there exists u such that $x = f(D)\#u$ and $uyz = 0^{2n}$. We have $xz \in L_U \Leftrightarrow f(D)\#uz \in L_U \Leftrightarrow uz \in L$ but $|uz| < |uyz|$ thus uz cannot be in L and thus L_U cannot be regular.

7.1. Two automata D_1 and D_2 accept the same language if and only if $f(D_1)\# \sim_{L_U} f(D_2)\#$. Since there are infinitely many regular languages, there are infinitely many equivalence classes for $\sim_{L_U}$ and thus L_U cannot be regular by the Myhill–Nerode theorem.

7.1. Let us suppose that L_U is regular and accepted by $(Q, \Sigma, \delta, q_0, F)$. We can find $|Q| + 1$ distinct languages $L_1, \ldots, L_{|Q|+1}$ accepted by DFAs $D_1, \ldots, D_{|Q|+1}$. By pigeonhole we can find $i \neq j$ such that $\tilde{\delta}(q_0, f(D_i)) = \tilde{\delta}(q_0, f(D_j))$. But this implies that for all w, $\tilde{\delta}(q_0, f(D_i)\#w) = \tilde{\delta}(q_0, f(D_j)\#w)$ and thus D_i accepts the same language as D_j.

7.1. Suppose that L_U is regular and DFA $U = (Q, \Sigma, \delta, q_0, F)$ accepts L_U. Let $U_a^b := \{w \mid \delta(a, w) = b\}$. Language U_a^b is regular as it is accepted by $(Q, \Sigma, \delta, a, \{b\})$. Consider now the language $L_R := \{w \mid w\#w \notin L_U\}$. We have $\bar{L}_R = \bigcup_{f \in Q} \bigcup_{q \in Q} U_i^q \cap U_{\delta(q,\#)}^f$. Hence $\bar{L}_R$ is regular which implies L_R is also regular. Let D_R be a DFA recognizing L_R. We have now obtained a contradiction: $f(D_R) \in L_R \Leftrightarrow f(D_R)\#f(D_R) \notin L_U \Leftrightarrow f(D_R) \notin L_R$. (Thanks to Maxime Ramzy and Nicolas Fabiano for this solution.)

8 Unary Languages

8.1. Let L be a regular unary language and D a DFA recognizing L whose states are Q and final states are F. Let q_i be the state of the automaton after reading 0^i. We have $L = \{0^i \mid i \in \mathbb{N} \text{ such that } q_i \in F\}$. Since Q is finite we have two numbers $j < k$ such that $q_k = q_j$, and since D is deterministic $q_{j+\ell} = q_{k+\ell} = q_{j+\ell \pmod{k-j}}$. Set $c := k - j$. We have:

$$L = \bigcup_{\substack{i<j \\ q_i \in F}} \{0^i\} \bigcup_{\substack{j \leq i < k \\ q_i \in F}} \{0^{i+cn} \mid n \geq 0\}$$

★★ 8.2 Let L be a context-free unary language and let P be its pumping length. For each $m \in L$ with $P \leq |m|$ the pumping lemma gives us a decomposition of m into $uvwxy$ such that $uv^iwx^iy \in L$ for all i. Since $m = 0^{|m|}$ we have $\{0^{|m|}\} \subseteq \{0^{|m|+l \cdot |xv|} \mid l \in \mathbb{N}\} \subseteq L$. For each $0^m \in L$ with $m > P$ we might have several such decompositions but for each m we choose a decomposition and fix a $k(m)$ such that

$0 < k(m) \leq P$ and $\{0^{m+l \cdot k(m)} \mid l \in \mathbb{N}\} \subseteq L$ then we have $L = \{0^{|m|} \mid 0^{|m|} \in L\} = \{w \in L \mid |w| \leq P\} \bigcup_{P<m} \{0^{m+k(m)\times n} \mid n \in \mathbb{N}\}$.

This union is infinite and we would like to rewrite it as a finite union. We notice that given m and m' we have $\{0^{|m|+n\times k(m)} \mid n \in \mathbb{N}\} \subseteq \{0^{|m'|+n\times k(m')} \mid n \in \mathbb{N}\}$ when $m \geq m'$, $k(m) = k(m')$ and $m \equiv m'[k(m)]$. Therefore we can have a finite union by looking for each pair (i, j) at the smallest m such that $k(m) = i$ and $m \equiv j[i]$ (notice that $0 \leq j < i \leq P$).

Let $c_{i,j} := \min_m\{m > P \mid (k(m) = i) \wedge (j = m \bmod i)\}$ with the convention that $c_{i,j} := \infty$ when this set is empty. We now can define L as the following finite union:

$$L = \{w \in L \mid |w| \leq P\} \bigcup_{\substack{0 \leq j < i \leq P \\ c_{i,j} < \infty}} \{0^{c_{i,j}+i\times n} \mid n \in \mathbb{N}\} \, .$$

Each language on the right hand side is regular, and hence so is their finite union L.