CDM
Context-Free Grammars

Klaus Sutner
Carnegie Mellon Universality

60-cont-free 2017/12/15 23:17

Generation vs. Recognition 3

Turing machines can be used to check membership in decidable sets. They can also be used to enumerate semidecidable sets, whence the classical notion of recursively enumerable sets.

For languages $L \subseteq \Sigma^\star$ there is a similar notion of generation.

The idea is to set up a system of simple rules that can be used to derive all words in a particular formal language. These systems are typically highly nondeterministic and it is not clear how to find (efficient) recognition algorithms for the corresponding languages.

Noam Chomsky 4

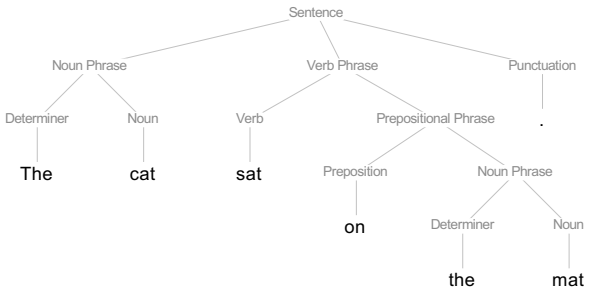
Historically, these ideas go back to work by Chomsky in the 1950s. Chomsky was mostly interested natural languages: the goal is to develop grammars that differentiate between grammatical and and ungrammatical sentences.

1. The cat sat on the mat.
2. The mat on the sat cat.

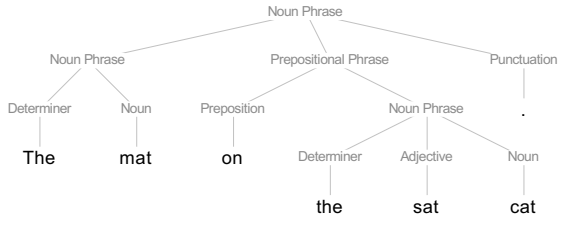
Alas, this turns out to be inordinately difficult, syntax and semantics of natural languages are closely connected and very complicated.

But for artificial languages such as programming languages, Chomsky's approach turned out be perfectly suited.

Cat-Mat Example 5



Mat-Cat Example 6



Killer App: Programming Languages

Many programming languages have a block structure like so:

```
begin
    begin
    end
    begin
        begin
        end
        begin
        end
    end
end
```

Clearly, this is not a regular language and cannot be checked by a finite state machine. We need more compute power.

Generalizing

We have two rather different ways of describing regular languages:

- finite state machine acceptors
- regular expressions

We could try to generalize either one of these.

Let's start with the algebra angle and handle the machine model later.

Grammars

Definition

A (formal) grammar is a quadruple

$$G = \langle V, \Sigma, \mathbb{P}, S \rangle$$

where V and Σ are disjoint alphabets, $S \in V$, and $\mathbb{P}$ is a finite set of productions or rules.

- the symbols of V are (syntactic) variables,
- the symbols of Σ are terminals,
- S is called the start symbol (or axiom).

We often write $\Gamma = V \cup \Sigma$ for the complete alphabet of G.

Context Free Grammars

Definition (CFG)

A context free grammar is a grammar where the productions have the form

$$\mathbb{P} \subseteq V \times \Gamma^\star$$

It is convenient to write productions in the form

$$\pi : A \to \alpha$$

where $A \in V$ and $\alpha \in \Gamma^\star$.

The idea is that we may replace A by α.

Naming Conventions

- $A, B, C \ldots$ represent elements of V,
- $S \in V$ is the start symbol,
- $a, b, c \ldots$ represent elements of Σ,
- $X, Y, Z \ldots$ represent elements of Γ,
- $w, x, y \ldots$ represent elements of $\Sigma^\star$,
- $\alpha, \beta, \gamma \ldots$ represent elements of $\Gamma^\star$.

Derivations

Given a CFG G define a one-step relation $\overset{1}{\Longrightarrow} \subseteq \Gamma^\star \times \Gamma^\star$ as follows:

$$\alpha A \beta \overset{1}{\Longrightarrow} \alpha\gamma\beta \quad \text{if} \quad A \to \gamma \in \mathbb{P}$$

As usual, by induction define

$$\alpha \overset{k+1}{\Longrightarrow} \beta \quad \text{if} \quad \exists \gamma \, (\alpha \overset{k}{\Longrightarrow} \gamma \wedge \gamma \overset{1}{\Longrightarrow} \beta)$$

and

$$\alpha \overset{*}{\Longrightarrow} \beta \quad \text{if} \quad \exists k \; \alpha \overset{k}{\Longrightarrow} \beta$$

in which case one says that α derives or yields β. α is a sentential form if it can be derived from the start symbol S.

To keep notation simple we'll often just write $\alpha \Longrightarrow \beta$.

Context Free Languages

Definition

The language of a context free grammar G is defined to be

$$\mathcal{L}(G) = \{\, x \in \Sigma^\star \mid S \stackrel{*}{\Longrightarrow} x \,\}$$

Thus $\mathcal{L}(G)$ is the set of all sentential forms in $\Sigma^\star$. We also say that G generates $\mathcal{L}(G)$.

A language is context free (CFL) if there exists a context free grammar that generates it.

Note that in a CFG one can replace a single syntactic variable A by strings over Γ independently of were A occurs; whence the name "context free." Later on we will generalize to replacement rules that operate on a whole block of symbols (context sensitive grammars).

Example: Regular

Let $G = \langle\, \{S, A, B\}, \{a, b\}, \mathbb{P}, S \,\rangle$ where the set $\mathbb{P}$ of productions is defined by:

$$S \to aA \mid aB$$
$$A \to aA \mid aB$$
$$B \to bB \mid b.$$

A typical derivation is:

$$S \Rightarrow aA \Rightarrow aaA \Rightarrow aaaB \Rightarrow aaabB \Rightarrow aaabb$$

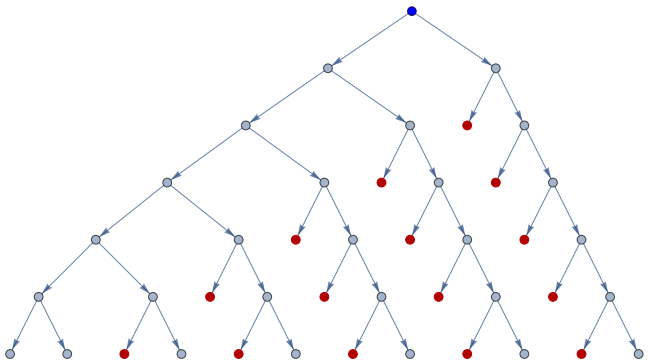
It is not hard to see that

$$\mathcal{L}(G) = a^+b^+$$

Not too interesting, we already know how to deal with regular languages.

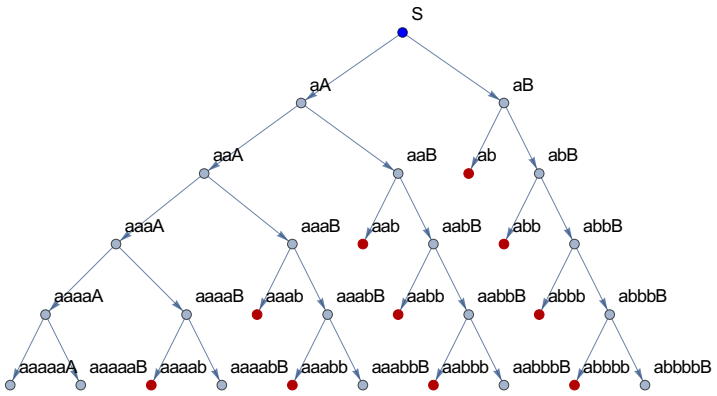
Can you see the finite state machine hiding in the grammar? Is it minimal?

Derivation Graph



Derivations of length at most 6 in this grammar.

Labeled



Example: Mystery

Let $G = \langle\, \{A, B\}, \{a, b\}, \mathbb{P}, A \,\rangle$ where the set $\mathbb{P}$ of productions is defined by:

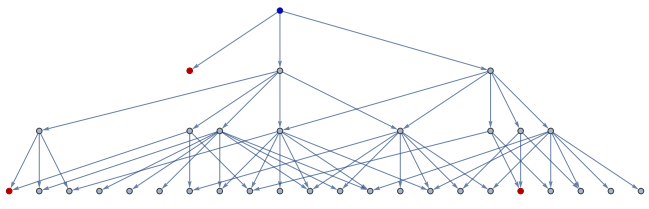
$$A \to AA \mid AB \mid a$$
$$B \to AA \mid BB \mid b.$$

A typical derivation is:

$$A \Rightarrow AA \Rightarrow AAB \Rightarrow AABB \Rightarrow AABAA \Rightarrow aabaa$$

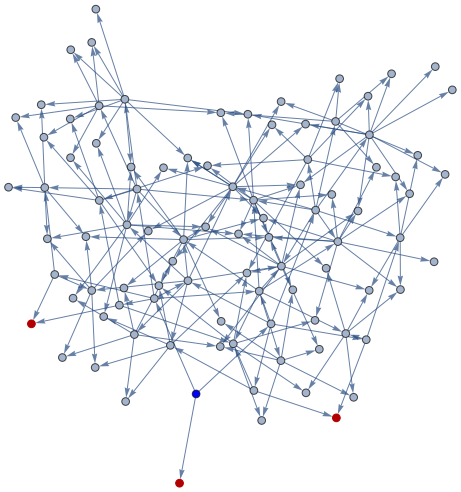
In this case it is not obvious what the language of G is (assuming it has some easy description, it does). More next time when we talk about parsing.

Derivation Graph



Derivations of length at most 3 in this grammar. Three terminal strings appear at this point.

Depth 4



Example: Counting

Let $G = \langle \{S\}, \{a, b\}, \mathbb{P}, S \rangle$ where the set $\mathbb{P}$ of productions is defined by:

$$S \to aSb \mid \varepsilon$$

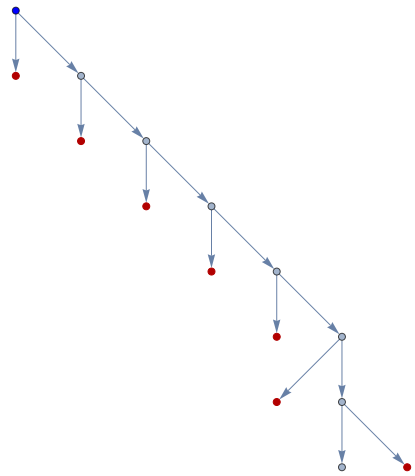
A typical derivation is:

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbb$$

Clearly, this grammar generates the language $\{ a^i b^i \mid i \geq 0 \}$

It is easy to see that this language is not regular.

Derivation Graph



Example: Palindromes

Let $G = \langle \{S\}, \{a, b\}, \mathbb{P}, S \rangle$ where the set $\mathbb{P}$ of productions is defined by:

$$S \to aSa \mid bSb \mid a \mid b \mid \varepsilon$$

A typical derivation is:

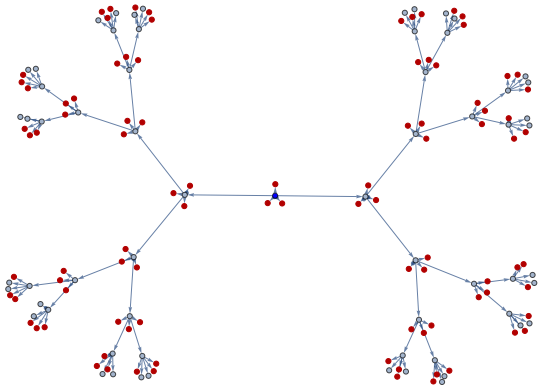
$$S \Rightarrow aSa \Rightarrow aaSaa \Rightarrow aabSbaa \Rightarrow aababaa$$

This grammar generates the language of palindromes.

Exercise

Give a careful proof of this claim.

Derivation Graph



Example: Parens

Let $G = \langle \{S\}, \{(,)\}, \mathbb{P}, S \rangle$ where the set $\mathbb{P}$ of productions is defined by:

$$S \to SS \mid (S) \mid \varepsilon$$

A typical derivation is:

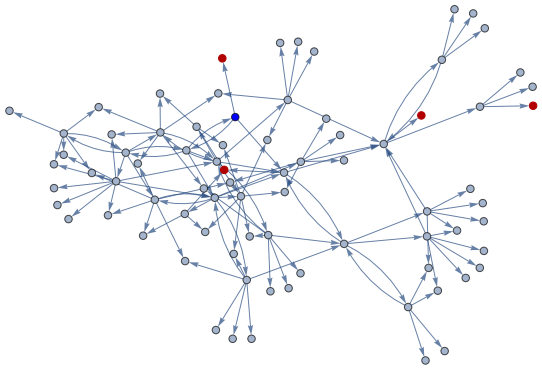
$$S \Rightarrow SS \Rightarrow (S)S \Rightarrow (S)(S) \Rightarrow (S)((S)) \Rightarrow ()(())$$

This grammar generates the language of well-formed parenthesized expressions.

Exercise

Give a careful proof of this claim.

Derivation Graph



Example: Expressions of Arithmetic

Let $G = \langle \{E\}, \{+, *, (,), v\}, \mathbb{P}, E \rangle$ where the set $\mathbb{P}$ of productions is defined by:

$$E \to E + E \mid E * E \mid (E) \mid v$$

A typical derivation is:

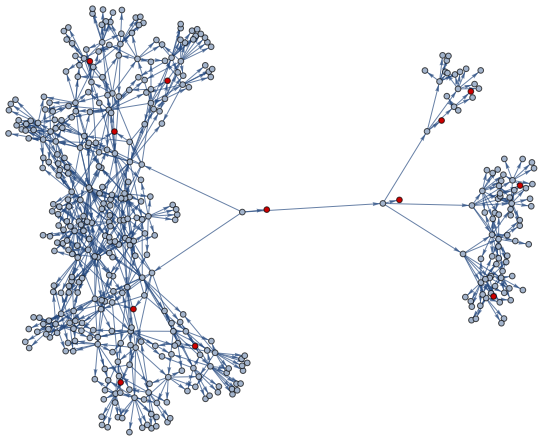
$$E \Rightarrow E * E \Rightarrow E * (E) \Rightarrow E * (E + E) \Rightarrow v * (v + v)$$

This grammar generates a language of arithmetical expressions with plus and times. Alas, there are problems: the following derivation is slightly awkward.

$$E \Rightarrow E + E \Rightarrow E + (E) \Rightarrow E + (E * E) \Rightarrow v + (v * v)$$

Our grammar is symmetric in $+$ and $*$, it knows nothing about precedence.

Derivation Graph



Ambiguity

We may not worry about awkward, but the following problem is fatal:

$$E \Rightarrow E + E \Rightarrow E + E * E \Rightarrow v + v * v$$

$$E \Rightarrow E * E \Rightarrow E + E * E \Rightarrow v + v * v$$

There are two derivations for the same word $v + v * v$.

Since derivations determine the semantics of a string this is really bad news: a compiler could interpret $v + v * v$ in two different ways, producing different results.

Parse Trees

Derivation chains are hard to read, a better representation is a tree.

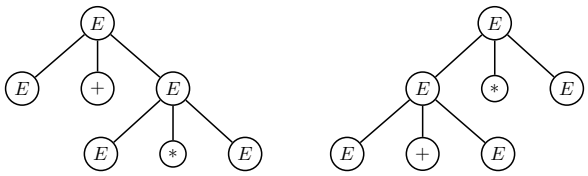
Let $G = \langle V, \Sigma, \mathbb{P}, S \rangle$ be a context free grammar.

A parse tree of G (aka grammatical tree) is an ordered tree on nodes N, together with a labeling $\lambda : N \to V \cup \Sigma$ such that

- For all interior nodes x: $\lambda(x) \in V$,
- If $x_1, \ldots, x_k$ are the children, in left-to-right order, of interior node x then $\lambda(x) \to \lambda(x_1) \ldots \lambda(x_k)$ is a production of G,
- $\lambda(x) = \varepsilon$ implies x is an only child.

Derivation Trees

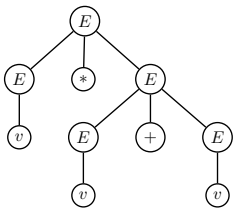
Here are the parse trees of the "expressions grammar" from above.



Note that the trees provide a method to evaluate arithmetic expressions, so the existence of two trees becomes a nightmare.

Information Hiding

A parse tree typically represents several derivations:



represents for example

$\theta_1 : E \Rightarrow E*E \Rightarrow E*E+E \Rightarrow v*E+E \Rightarrow v*v+E \Rightarrow v*v+v$
$\theta_2 : E \Rightarrow E*E \Rightarrow E*E+E \Rightarrow E*E+v \Rightarrow E*v+v \Rightarrow v*v+v$
$\theta_3 : E \Rightarrow E*E \Rightarrow v*E \Rightarrow v*E+E \Rightarrow v*v+E \Rightarrow v*v+v$
but not
$\theta_4 : E \Rightarrow E+E \Rightarrow E*E+E \Rightarrow v*E+E \Rightarrow v*v+E \Rightarrow v*v+v$

Leftmost Derivations

Let G be a grammar and assume $\alpha \stackrel{1}{\Longrightarrow} \beta$.

We call this derivation step leftmost if

$$\alpha = xA\alpha' \qquad \beta = x\gamma\alpha' \qquad x \in \Sigma^\star$$

A whole derivation is leftmost if it only uses leftmost steps. Thus, each replacement is made in the first possible position.

Proposition

Parse trees correspond exactly to leftmost derivations.

Ambiguity

Definition

A CFG G is ambiguous if there is a word in the language of G that has two different parse trees.

Alternatively, there are two different leftmost derivations.

As the arithmetic example demonstrates, trees are connected to semantics, so ambiguity is a serious problem in a programming language.

Unambiguous Arithmetic

For a "reasonable" context free language it is usually possible to remove ambiguity by rewriting the grammar.

For example, here is an unambiguous grammar for our arithmetic expressions.

$$\begin{array}{lcl} E & \to & E+T \mid T \\ T & \to & T*F \mid F \\ F & \to & (E) \mid v \end{array}$$

In this grammar, $v+v*v$ has only one parse tree.

Here $\{E, T, F\}$ are syntactic variables that correspond to expressions, terms and factors. Note that it is far from clear how to come up with these syntactic categories.

Inherently Ambiguous Languages

Alas, there are CFLs where this trick will not work: every CFG for the language is already ambiguous. Here is a well-known example:

$$L = \{\, a^i b^j c^k \mid i = j \vee j = k;\ i,j,k \geq 1 \,\}$$

L consists of two parts and each part is easily unambiguous.

But strings of the form $a^i b^i c^i$ belong to both parts and introduce a kind of ambiguity that cannot be removed.

BTW, $\{\, a^i b^i c^i \mid i \geq 0 \,\}$ is not context free.

Exercise

Show that L really is inherently ambiguous.

Regular Implies Context Free

Lemma

Every regular language is context free.

Proof. Suppose $M = \langle Q, \Sigma, \delta; q_0, F \rangle$ is a DFA for L. Consider a CFG with $V = Q$ and productions

$$\begin{array}{lcll} p & \to & a\,q & \text{if } \delta(p,a) = q \\ p & \to & \varepsilon & \text{if } p \in F \end{array}$$

Let q_0 be the start symbol.

□

Substitutions

Definition

A substitution is a map $\sigma : \Sigma \to \mathfrak{P}(\Gamma^\star)$.

The idea is that for any word $x \in \Sigma^\star$ we can define its image under σ to be language

$$\sigma(x_1) \cdot \sigma(x_2) \cdot \ldots \cdot \sigma(x_n)$$

Likewise, $\sigma(L) = \bigcup_{x \in L} \sigma(x)$.

If $\sigma(a) = \{w\}$ then we have essentially a homomorphism.

The Substitution Lemma

Lemma

Let $L \subseteq \Sigma^\star$ be a CFL and suppose $\sigma : \Sigma \to \mathfrak{P}(\Gamma^\star)$ is a substitution such that $\sigma(a)$ is context free for every $a \in \Sigma$. Then the language $\sigma(L)$ is also context free.

Proof.

Let $G = \langle V, \Sigma, \mathbb{P}, S \rangle$ and $G_a = \langle V_a, \Gamma, \mathbb{P}_a, S_a \rangle$ be CFGs for the languages L and $L_a = \sigma(a)$ respectively. We may safely assume that the corresponding sets of syntactic variables are pairwise disjoint.

Define G' as follows. Replace all terminals a on the right hand side of a production in G by the corresponding variable S_a.

It is obvious that $f(\mathcal{L}(G')) = L$ where f is the homomorphism defined by $f(S_a) = a$.

Proof, cont'd

Now define a new grammar H as follows.

The variables of H are $V \cup \bigcup_{a \in \Sigma} V_a$, the terminals are Σ, the start symbol is S and the productions are given by

$$\mathbb{P}' \cup \bigcup_{a \in \Sigma} \mathbb{P}_a$$

Then the language generated by H is $\sigma(L)$.

It is clear that H derives every word in $\sigma(L)$.

For the opposite direction consider the parse trees in H.

□

Closure Properties

Corollary

Suppose $L, L_1, L_2 \subseteq \Sigma^\star$ are CFLs. Then the following languages are also context free: $L_1 \cup L_2$, $L_1 \cdot L_2$ and L^*: context free languages are closed under union, concatenation and Kleene star.

Proof.

This follows immediately from the substitution lemma and the fact that the languages $\{a, b\}$, $\{ab\}$ and $\{a\}^*$ are trivially context free.

□

Non Closure

Proposition

CFLs are not closed under intersection and complement.

Consider

$$L_1 = \{\, a^i b^i c^j \mid i, j \geq 0 \,\} \qquad L_2 = \{\, a^i b^j c^j \mid i, j \geq 0 \,\}$$

We will see in a moment that $L_1 \cap L_2 = \{\, a^i b^i c^i \mid i \geq 0 \,\}$ fails to be context free.

More Closure

Lemma

Suppose L is a CFL and R is regular. Then $L \cap R$ is also context free.

Proof.

This will be easy once we have a machine model for CFLs (push-down automata), more later.

□

Dyck Languages

One can generalize strings of balanced parentheses to strings involving multiple types of parens.

To this end one uses special alphabets with paired symbols:

$$\Gamma = \Sigma \cup \{\, \overline{a} \mid a \in \Sigma \,\}$$

The Dyck language D_k is generated by the grammar

$$S \to SS \mid a\, S\, \overline{a} \mid \varepsilon$$

A typical derivation looks like so:

$$S \Rightarrow SS \Rightarrow aS\overline{a}S \Rightarrow aaS\overline{a}\,\overline{a}S \Rightarrow aaS\overline{a}\,\overline{a}aS\overline{a} \Rightarrow aa\overline{a}\,\overline{a}a\overline{a}$$

Exercise

Find an alternative definition of a Dyck language.

A Characterization

Let us write D_k for the Dyck language with $k = |\Sigma|$ kinds of parens.

For D_1 there is a nice characterization via a simple counting function. Define $\#_a x$ to be the number of letters a in word x.

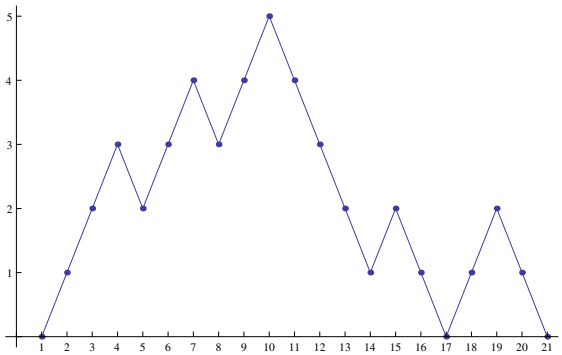
$$f_a(x) = \#_a x - \#_{\overline{a}} x$$

Lemma

A string x belongs to the Dyck language $D_1 \subseteq \{a, \overline{a}\}^\star$ iff

- *$f_a(x) = 0$ and*
- *for any prefix z of x: $f_a(z) \geq 0$.*

A Paren Mountain



Note that one can read off a proof for the correctness of the grammar $S \to SS \mid a\, S\, \overline{a} \mid \varepsilon$ from the picture.

k-Parens

For D_k we can still count but this time we need values in $\mathbb{N}^k$

$$f(x) = (f_{a_1}(x), f_{a_2}(x), \ldots, f_{a_k}(x))$$

Then we need $f(x) = \mathbf{0}$ and $f(z) \geq \mathbf{0}$ for all prefixes z of x, just like for D_1.

Alas, this is not enough: we also have make sure that proper nesting occurs between different types of parens.

The critical problem is that we do not want $ab\overline{a}\overline{b}$.

Matching Pairs

Let $x = x_1 x_2 \ldots x_n$.

Note that if $x \in D_k$ and $x_i = a$ then there is a unique minimal $j > i$ such that $f_a(x[i]) = f_a(x[j])$ (why?).

Intuitively, $x_j = \overline{a}$ is the matching right paren for $a = x_i$.

Hence we obtain an interval $[i, j]$ associated with the a in position i. Call the collection of all such intervals I_a, $a \in \Sigma$.

The critical additional condition for a balanced string is that none of the intervals in $\bigcup I_a$ overlap, they are all nested or disjoint.

Exercise

Show that these conditions really describe the language D_k.

Dyck vs. CF

In a strong sense, Dyck languages are the "most general" context free languages: all context free languages are built around the notion of matching parens, though this may not at all be obvious from their definitions (and, actually, not even from their grammars).

Theorem (Chomsky-Schützenberger 1963)

Every context free language $L \subseteq \Sigma^\star$ has the form $L = h(D \cap R)$ where D is a Dyck language, R is regular and h is a homomorphism.

The proof also relies on a machine model, more later.

Parikh Vectors

Suppose $\Sigma = \{a_1, a_2, \ldots, a_k\}$. For $x \in \Sigma^\star$, the Parikh vector of x is defined by

$$\#x = (\#_{a_1}x, \#_{a_2}x, \ldots, \#_{a_k}x) \in \mathbb{N}^k$$

Lift to languages via

$$\#L = \{\, \#x \mid x \in L \,\} \subseteq \mathbb{N}^k$$

In a sense, the Parikh vector gives the commutative version of a word: we just count all the letters, but ignore order entirely.

For example, for the Dyck language D_1 over $\{a, \overline{a}\}$ we have $\#D_1 = \{\, (i,i) \mid i \geq 0 \,\}$.

Semi-Linear Sets

A set $A \subseteq \mathbb{N}^k$ is semi-linear if it is the finite union of sets of the form

$$\{\, \boldsymbol{a}_0 + \sum_i \boldsymbol{a}_i x_i \mid x_i \geq 0 \,\}$$

and $\boldsymbol{a}_i \in \mathbb{N}^k$ fixed.

In the special case $k = 1$, semi-linear sets are often called ultimately periodic:

$$A = A_t + (a + m\mathbb{N} + A_p)$$

where $A_t \subseteq \{0, \ldots, a-1\}$ and $A_p \subseteq \{0, \ldots, m-1\}$ are the transient and periodic part, respectively.

Observe that for any language $L \subseteq \{a\}^\star$: L is regular iff $\#L \subseteq \mathbb{N}$ is semi-linear.

Parikh's Theorem

Theorem

For any context free language L, the Parikh set $\#L$ is semi-linear.

Instead of a proof, consider the example of the Dyck language D_1

$$S \to SS \mid aS\overline{a} \mid \varepsilon$$

Let $A = \#D_1$, then A is the least set $X \subseteq \mathbb{N}^2$ such that

- $S \to SS$: X is closed under addition
- $S \to aS\overline{a}$: X is closed under $\boldsymbol{x} \mapsto \boldsymbol{x} + (1,1)$
- $S \to \varepsilon$: X contains $(0,0)$

Clearly, $A = \{\, (i,i) \mid i \geq 0 \,\}$.

Application Parikh

It follows immediately that every context free language over $\Sigma = \{a\}$ is already regular.

As a consequence, $\{\, a^p \mid p \text{ prime} \,\}$ is not context free.

This type of argument also works for a slightly messier language like

$$L = \{\, a^k b^\ell \mid k > \ell \vee (k \leq \ell \wedge k \text{ prime}) \,\}$$

Note that in this case L and $\#L$ are essentially the same, so it all comes down to the set of primes not being semi-linear.

Markings

Another powerful method to show that a language fails to be context free is a generalization of the infamous pumping lemma for regular languages. Alas, this time we need to build up a bit of machinery.

Definition

Let $w \in \Sigma^\star$, say, $n = |w|$. A position in w is a number p, $1 \leq p \leq n$.

A set $K \subseteq [n]$ of positions is called a marking of w.

A 5-factorization of w consists of 5 words x_1, x_2, x_3, x_4, x_5 such that $x_1x_2x_3x_4x_5 = w$.

Given a factorization and a marking of w let
$K(x_i) = \{\, p \mid |x_1 \ldots x_{i-1}| < p \leq |x_1 \ldots x_i| \,\} \subseteq K$

Thus $K(x_i)$ simply consists of all the marked positions in block x_i.

The Iteration Theorem

Theorem

Let $G = \langle V, \Sigma, \mathbb{P}, S \rangle$ be a CFG. Then there exists a number $N = N(G)$ such that for all $x \in \mathcal{L}(G)$, $K \subseteq [|x|]$ a marking of x of cardinality at least N:

there exists a 5-factorization $x_1, \ldots, x_5$ of x such that, letting $K_i = K(x_i)$, we have:

- $K_1, K_2, K_3 \neq \emptyset$ *or* $K_3, K_4, K_5 \neq \emptyset$
- $|K_2 \cup K_3 \cup K_4| \leq N$.
- $\forall\, t \geq 0 \ \big(x_1 x_2^t x_3 x_4^t x_5 \in \mathcal{L}(G)\big)$.

Proof. Stare at parse trees. □

Non-Closure

Lemma

$\{\, a^i b^i c^i \mid i \geq 0 \,\}$ *is not context free.*

Proof.

Recall that this shows non-closure under complements and intersections.

So a CFG can count and compare two letters, as in

$$L_1 = \{\, a^i b^i c^j \mid i, j \geq 0 \,\}$$
$$L_2 = \{\, a^i b^j c^j \mid i, j \geq 0 \,\}$$

but three letters are not manageable.

The intuition behind this will become clear next time when we introduce a machine model.

Proof

Let N be as in the iteration theorem and set

$$w = a^N b^N c^N,$$

$K = [N+1, 2N]$ (so the b's in the middle are marked).

Then there is a factorization $x_1, \ldots, x_5$ of w such that, letting $K_i = K(x_i)$, we have:

Case 1: $K_1, K_2, K_3 \neq \emptyset$

Then $x_1 = a^N b^i$, $x_2 = b^j$, $x_3 = b^k y$ where $j > 0$.

But then $x_1 x_3 x_5 \notin L$, contradiction.

Case 2: $K_3, K_4, K_5 \neq \emptyset$

Then $x_3 = y b^i$, $x_4 = b^j$, $x_5 = b^k c^N$ where $j > 0$.

Again $x_1 x_3 x_5 \notin L$, contradiction.

More Non-Closure

It follows that

$$\{\, x \in \{a, b, c\}^* \mid |x|_a = |x|_b = |x|_c \,\}$$

is not context free: otherwise the intersection with $a^\star b^\star c^\star$ would also be context free.

Exercise

Show that the copy language

$$L_{copy} = \{\, x\, x \mid x \in \Sigma^\star \,\}$$

fails to be context free. Compare this to the palindrome language $\{\, x\, x^{\mathrm{op}} \mid x \in \Sigma^\star \,\}$