3.6 The Cocke-Younger-Kasami parsing algorithm

Recursive descent parsing is a clear and effective method for parsing LL(1)-grammars: a string of n symbols can be parsed in $O(n)$ time. However, LL(1)-grammars are a fairly limited class; solving the general parsing problem is not as easy.

In principle the problem could be solved by applying the recursive descent parsing method, but in practice the large number of alternatives to be tested becomes a problem (typically $O(c^n)$ for some $c \geq 2$).

The *Cocke-Younger-Kasami algorithm* is a method for parsing strings in any context-free grammar G. It is based on the general technique of dynamic programming (tabulating partial solutions). The method requires $O(n^3)$ time, where n is the length of the string to be parsed.

First some grammar transformations are defined.

1. Removing ε-productions

Let $G = (V, \Sigma, P, S)$ be a context-free grammar. The nonterminal $A \in V - \Sigma$ is *nullable* if $A \underset{G}{\Rightarrow}^* \varepsilon$.

Lemma 3.5. Any context free grammar G can be transformed into an equivalent grammar G' where at most the initial symbol is nullable.

Proof. Let $G = (V, \Sigma, P, S)$. First determine the nullable nonterminals of G as follows:

(i) first set

$$\text{NULL} := \{A \in V - \Sigma \mid A \to \varepsilon \text{ is a production of } G\};$$

(ii) repeat the following expansion of the set NULL until it no longer grows:

$$\begin{aligned} \text{NULL} \quad := \quad & \text{NULL} \cup \\ & \{A \in V - \Sigma \mid A \to B_1 \dots B_k \text{ is a production of } G, \\ & B_i \in \text{NULL for all } i = 1, \dots, k\}. \end{aligned}$$

Now replace each production $A \to X_1 \dots X_k$ by the set of productions of the form

$$A \to \alpha_1 \dots \alpha_k, \quad \text{where}$$

$$\alpha_i = \begin{cases} X_i, & \text{if } X_i \notin \text{NULL}; \\ X_i \text{ or } \varepsilon, & \text{if } X_i \in \text{NULL}. \end{cases}$$

Finally remove all productions of the form $A \to \varepsilon$. If the production $S \to \varepsilon$ would be to remove, we introduce the new initial symbol S' and the productions $S' \to S$ ja $S' \to \varepsilon$. □

Example. Remove the ε-productions from the grammar:

$$\begin{array}{lcll} S & \to & A \mid B & \\ A & \to & aBa \mid \varepsilon & \Rightarrow \quad (\text{NULL} = \{A, B, S\}) \\ B & \to & bAb \mid \varepsilon & \end{array}$$

$$\begin{array}{lcll} S & \to & A \mid B \mid \varepsilon & \\ A & \to & aBa \mid aa \mid \varepsilon & \Rightarrow \\ B & \to & bAb \mid bb \mid \varepsilon & \end{array}$$

$$\begin{array}{lcl} S' & \to & S \mid \varepsilon \\ S & \to & A \mid B \\ A & \to & aBa \mid aa \\ B & \to & bAb \mid bb. \end{array}$$

2. Removing unit productions

A production of the form $A \to B$, where A and B are nonterminals, is a *unit production*.

Lemma 3.6. Any context-free grammar G can be transformed into an equivalent grammar G' without unit productions.

Proof. Let $G = (V, \Sigma, P, S)$. Find first the "unit successors" of each nonterminal in G as follows:

(i) at start for each $A \in V - \Sigma$ let

$$F(A) := \{B \in V - \Sigma \mid A \to B \text{ is a production of } G\};$$

(ii) repeat the following expansion operation of the F-sets until they no longer grow:

$$F(A) := F(A) \cup \bigcup \{F(B) \mid A \to B \text{ is a production of } G\}.$$

After this remove all unit productions from G and replace them with all possible productions of the form $A \to \omega$, where $B \to \omega$ is a non-unit production of G for some $B \in F(A)$. □

Example. Remove the unit productions from the previous grammar:

$$\begin{aligned} S' &\to S \mid \varepsilon \\ S &\to A \mid B \\ A &\to aBa \mid aa \\ B &\to bAb \mid bb. \end{aligned}$$

The unit successors of the nonterminals are: $F(S') = \{S, A, B\}$, $F(S) = \{A, B\}$, $F(A) = F(B) = \emptyset$. By replacing the unit productions as described we obtain the grammar:

$$\begin{aligned} S' &\to aBa \mid aa \mid bAb \mid bb \mid \varepsilon \\ S &\to aBa \mid aa \mid bAb \mid bb \\ A &\to aBa \mid aa \\ B &\to bAb \mid bb. \end{aligned}$$

(We may note that S is now "unnecessary", i.e., it cannot occur in the derivation of any sentence in the grammar. Unnecessary nonterminals can be removed from the grammar by a similar method. (exercise).)

The Chomsky normal form

A context free grammar $G = (V, \Sigma, P, S)$ is in *Chomsky normal form*, if no nonterminal other than S is nullable, and apart from the possible production $S \to \varepsilon$ the remaining productions are of the form

$$A \to BC \qquad \text{or} \qquad A \to a,$$

where A, B and C are nonterminals and a is a terminal.

For simplicity it is additionally required that the initial symbol S does not appear on the right hand side of any production.

Theorem 3.7. Any context free grammar G can be transformed into an equivalent grammar G' that is in Chomsky normal form.

Proof. Let $G = (V, \Sigma, P, S)$. First remove the ε-productions and unit productions from G by the constructions in Lemmata 3.5 and 3.6. Now all productions in G are of the form $A \to a$ or $A \to X_1 \dots X_k$, $k \geq 2$ (or $S \to \varepsilon$).

For each terminal a add a new nonterminal C_a and a production $C_a \to a$. Then first replace all terminals in productions of the form $A \to X_1 \dots X_k$, $k \geq 2$, with the new nonterminals, and then replace the whole production by the set of productions

$$\begin{aligned} A &\to X_1 A_1 \\ A_1 &\to X_2 A_2 \\ &\vdots \\ A_{k-2} &\to X_{k-1} X_k, \end{aligned}$$

where $A_1, \dots, A_{k-2}$ are again new nonterminals. □

Example. A grammar:

$$\begin{aligned} S &\to aBCd \mid bbb \\ B &\to b \\ C &\to c \end{aligned}$$

The Chomsky normal form obtained by the previous construction:

$$\begin{aligned} S &\to C_a S_1^1 \\ S_1^1 &\to B S_2^1 \\ S_2^1 &\to C C_d \\ S &\to C_b S_1^2 \\ S_1^2 &\to C_b C_b \\ B &\to b \\ C &\to c \\ C_a &\to a \\ C_b &\to b \\ C_c &\to c \\ C_d &\to d. \end{aligned}$$

The CYK algorithm

Let $G = (V, \Sigma, P, S)$ be a context-free grammar. By Theorem 3.7 we may assume that G is in Chomsky normal form. The question whether the string x is in the language $L(G)$ can be solved as follows:

If $x = \varepsilon$, then $x \in L(G)$ iff $S \to \varepsilon$ is a production of G.

Otherwise denote $x = a_1 \dots a_n$ and investigate producing various substrings of x.

Let N_{ik} denote the set of those nonterminals A from which one can derive the substring of x that starts at position i and whose length is k:

$$\begin{aligned} N_{ik} &= \{A \in V - \Sigma \mid A \underset{G}{\Rightarrow}^* a_i \dots a_{i+k-1}\}, \\ &\quad 1 \le i \le i + k - 1 \le n. \end{aligned}$$

The sets N_{ik} can be computed by tabulating them from shorter to longer strings as presented in the following. Clearly $x \in L(G)$ iff $S \in N_{1n}$.

Computing the sets N_{ik}:

(i) first let for all $i = 1, \dots, n$:

$$N_{i1} := \{A \in V - \Sigma \mid A \to a_i \text{ is a production of } G\};$$

(ii) then compute for all $k = 2, \dots, n$ and each k for all $i = 1, \dots, n - k + 1$:

$$N_{ik} := \bigcup_{j=1}^{k-1} \{A \in V - \Sigma \mid A \to BC \text{ is a production of } G, \text{ where } B \in N_{ij} \text{ and } C \in N_{i+j,k-j}\}. \quad \square$$

Example. A grammar G in Chomsky normal form:

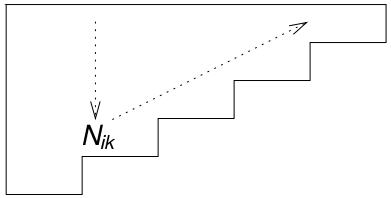
$$\begin{aligned} S &\to AB \mid BC \\ A &\to BA \mid a \\ B &\to CC \mid b \\ C &\to AB \mid a \end{aligned}$$

The computation of the CYK algorithm with grammar G and input $x = baaba$:

		$i \to$				
	N_{ik}	$1 : b$	$2 : a$	$3 : a$	$4 : b$	$5 : a$
	1	B	A, C	A, C	B	A, C
	2	S, A	B	S, C	S, A	–
$k \downarrow$	3	$\emptyset$	B	B	–	
	4	$\emptyset$	S, A, C	–		
	5	S, A, C	–			

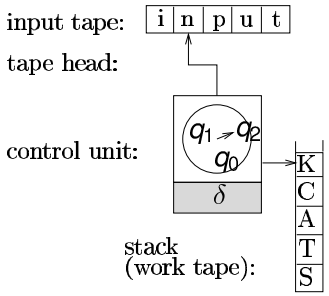
Since the initial symbol S is in N_{15}, we deduce that x is in $L(G)$.

In computing a set N_{ik} the CYK algorithm proceeds simultaneously towards the set N_{ik} in column N_{ij} and away from N_{ik} along the diagonal $N_{i+j,k-j}$:



3.7 Pushdown automata

Context free languages can be given an automaton characterization by *pushdown automata*:



A pushdown automaton is like a finite state automaton, to which a *stack* of unbounded size has been added. Using the stack is fairly limited: the automaton may read, write, remove or add symbols only at the top of the stack.

Definition 3.2 A *pushdown automaton* is a 6-tuple

$$M = (Q, \Sigma, \Gamma, \delta, q_0, F),$$

where

- Q is the finite set of *states*;
- Σ is the *input alphabet*;
- Γ is the *stack alphabet*;
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\}) \to \mathcal{P}(Q \times (\Gamma \cup \{\varepsilon\}))$ is the (set-valued) *transition function*;
- $q_0 \in Q$ is the *initial state*;
- $F \subseteq Q$ is the set of *(accepting) final states*.

The interpretation of the value

$$\delta(q, \sigma, \gamma) = \{(q_1, \gamma_1), \ldots, (q_k, \gamma_k)\}$$

of the transition function is that in state q upon reading the input symbol σ and the stack symbol γ the automaton may move to one of the states $q_1, \ldots, q_k$ and replace the top element of the stack by one of the symbols $\gamma_1, \ldots, \gamma_k$ respectively. In the general case, pushdown automata are therefore *nondeterministic*.

If $\sigma = \varepsilon$, the automaton makes a transition without reading an input symbol. If $\gamma = \varepsilon$, the automaton does not read a stack symbol and the new written symbol is put on the top of the stack (a *push* operation). If the symbol read from the stack is $\gamma \neq \varepsilon$ and the symbol to be written is $\gamma_i = \varepsilon$, the element on the top of the stack is removed (a *pop* operation).

The *configuration* of the automaton is the triple $(q, w, \alpha) \in Q \times \Sigma^* \times \Gamma^*$; in particular, the *initial configuration with input x* is the triple (q_0, x, ε).

Intuition: in configuration (q, w, α) the automaton is in state q, the remaining part of the input string is w the stack, read from top to bottom, contains the string α.

The configuration (q, w, α) *directly leads to* configuration (q', w', α'), denoted by

$$(q, w, \alpha) \vdash_M (q', w', \alpha'),$$

if we may write $w = \sigma w'$, $\alpha = \gamma\beta$, $\alpha' = \gamma'\beta$ ($|\sigma|, |\gamma|, |\gamma'| \leq 1$), such that

$$(q', \gamma') \in \delta(q, \sigma, \gamma).$$

The configuration (q, w, α) *leads to* configuration (q', w', α'), denoted by

$$(q, w, \alpha) \vdash_M^* (q', w', \alpha'),$$

if there is a sequence of configurations (q_0, w_0, α_0), (q_1, w_1, α_1), ..., (q_n, w_n, α_n), $n \geq 0$ such that

$$(q, w, \alpha) = (q_0, w_0, \alpha_0) \vdash_M (q_1, w_1, \alpha_1) \vdash_M \cdots \vdash_M (q_n, w_n, \alpha_n) = (q', w', \alpha').$$

A pushdown automaton M *accepts* the string $x \in \Sigma^*$, if

$$(q_0, x, \varepsilon) \vdash_M^* (q_f, \varepsilon, \alpha) \quad \text{for some } q_f \in F \text{ and } \alpha \in \Gamma^*,$$

that is if it is in an accepting final state at the end of the input; otherwise M *rejects* x:n.

The language *recognized* by M is:

$$L(M) = \{x \in \Sigma^* \mid (q_0, x, \varepsilon) \vdash_M^* (q_f, \varepsilon, \alpha) \text{ for some } q_f \in F \text{ and } \alpha \in \Gamma^*\}.$$

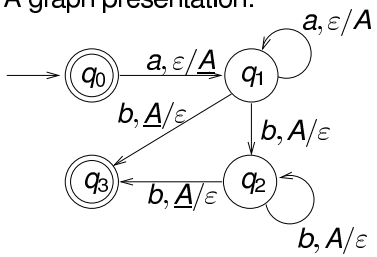
Example. A pushdown automaton for the language $\{a^k b^k \mid k \geq 0\}$:

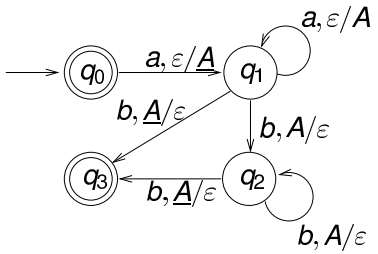
$M = (\{q_0, q_1, q_2, q_3\}, \{a, b\}, \{A, \underline{A}\}, \delta, q_0, \{q_0, q_3\})$,

missä

$$\begin{aligned}
\delta(q_0, a, \varepsilon) &= \{(q_1, \underline{A})\},\\
\delta(q_1, a, \varepsilon) &= \{(q_1, A)\},\\
\delta(q_1, b, A) &= \{(q_2, \varepsilon)\},\\
\delta(q_1, b, \underline{A}) &= \{(q_3, \varepsilon)\},\\
\delta(q_2, b, A) &= \{(q_2, \varepsilon)\},\\
\delta(q_2, b, \underline{A}) &= \{(q_3, \varepsilon)\},\\
\delta(q, \sigma, \gamma) &= \emptyset \quad \text{for other } (q, \sigma, \gamma).
\end{aligned}$$

A graph presentation:





The computation of the automaton with input *aabb*:

$$\begin{aligned}
(q_0, aabb, \varepsilon) &\vdash (q_1, abb, \underline{A}) \vdash (q_1, bb, A\underline{A})\\
&\vdash (q_2, b, \underline{A}) \vdash (q_3, \varepsilon, \varepsilon).
\end{aligned}$$

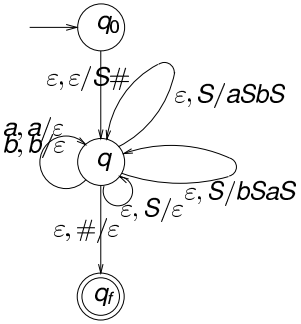
Since $q_3 \in F = \{q_0, q_3\}$, we have $aabb \in L(M)$.

Pushdown automata and context-free languages

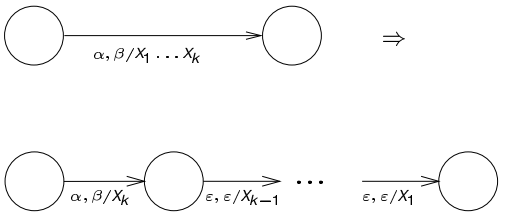
Theorem 3.8 A language is context-free, if and only if it can be recognized by some (nondeterministic) pushdown automaton. □

The proof is omitted here, but the principle in constructing the pushdown automaton M_G that corresponds to the given grammar G is that the stack of the automaton is used to simulate the left derivation $S \underset{\text{lm}}{\Rightarrow}^* x$ of x in G: if the top element of the stack is a nonterminal, some production of G is applied and the corresponding symbols pushed onto the stack; if the topmost element is a terminal, it is matched with the next input symbol.

Example. The pushdown automaton that corresponds to the grammar $\{S \to aSbS \mid bSaS \mid \varepsilon\}$:



The natural abbreviated notation is used in describing the automaton:



For example there is an accepting computation for the input *abab*:

$$\begin{array}{lllll} (q_0, abab, \varepsilon) & \vdash & (q, abab, S\#) & \vdash^* & (q, abab, aSbS\#) \\ & \vdash & (q, bab, SbS\#) & \vdash^* & (q, bab, bSaSbS\#) \\ & \vdash & (q, ab, SaSbS\#) & \vdash & (q, ab, aSbS\#) \\ & \vdash & (q, b, SbS\#) & \vdash & (q, b, bS\#) \\ & \vdash & (q, \varepsilon, S\#) & \vdash & (q, \varepsilon, \#) \\ & \vdash & (q_f, \varepsilon, \varepsilon). & & \end{array}$$

This corresponds to the left derivation of *abab* in the grammar:

$$\begin{array}{lllllll} \underline{S} & \Rightarrow & a\underline{S}bS & \Rightarrow & ab\underline{S}aSbS & \Rightarrow & aba\underline{S}bS \\ & \Rightarrow & abab\underline{S} & \Rightarrow & abab. & & \end{array}$$

A pushdown automaton M is *deterministic*, if every configuration (q, w, α) has at most one possible immediate successor (q', w', α'), for which

$$(q, w, \alpha) \underset{M}{\vdash} (q', w', \alpha')$$

Unlike for finite state automata, *nondeterministic pushdown automata are strictly more powerful than deterministic ones*. For example the language $\{ww^R \mid w \in \{a, b\}^*\}$ can be recognized by a nondeterministic, but not by a deterministic, pushdown automaton (proof omitted).

A context-free language is *deterministic*, if it can be recognized by a deterministic pushdown automaton. Deterministic languages can be parsed in $O(n)$ time; general context-free language require nearly $O(n^3)$ time with known methods.