CS 154. Winter 1996.
Handout 8, February 5, 1996.
Based on a CS254 handout of Vaughan Pratt, Winter 1994.

Context-Free Languages

1 Definitions

Definition 1. A *context-free grammar* (CFG) $G = (V, \Sigma, \Pi, S)$ consists of a finite set V of *variables* (aka nonterminals) $A_1, A_2, \ldots, A_n$, a finite alphabet Σ, a finite set $\Pi \subset V \times (V \cup \Sigma)^*$ of *productions* $A \to \alpha$ where $A \in V$ and $\alpha \in (V \cup \Sigma)^*$, and a nonterminal $S \in V$, the *start symbol.*

Definition 2. A *solution to* a CFG (V, Σ, Π, S) is a function $s : V \to 2^{\Sigma^*}$ from variables to languages (equivalently a V-tuple of languages $\langle L_1, L_2, \ldots, L_n \rangle$) such that for all productions $A \to \alpha$, $s(A) \supseteq s(\alpha)$ where $s' : (V \cup \Sigma)^* \to 2^{\Sigma^*}$ extends the domain of s by defining $s'(A) = s(A)$ for all $A \in V$, $s'(a) = \{a\}$ for all $a \in \Sigma$ and $s(\alpha_1 \cdots \alpha_k) = s'(\alpha_1) \cdots s'(\alpha_k)$. The *pointwise inclusion order* on solutions s, t is the partial order defined by $s \leq t$ just when $s(A) \subseteq t(A)$ for all $A \in V$.

Definition 3. The *context-free language* $L(G)$ *generated by* the CFG G with start symbol S is the value of L_S in the least solution to G.

We now show in either of two ways that definition 3 is a *well-definition* in that there exists such a solution and it is unique.

Theorem 1. There exists a unique solution to a CFG that is *least* (less or equal to every solution) with respect to the pointwise inclusion order on solutions.

Proof. *Uniqueness.* Let s, s' be two least solutions. Then for all A, $s(A) \subseteq s'(A) \subseteq s(A)$ whence $s(A) = s'(A)$.

Existence. There exists at least one solution, namely the maximal solution $(\Sigma^*)^V$ assigning Σ^* to all variables. We give two constructions of the least solution, top-down and bottom-up.

1. Top-down. Form the pointwise intersection m of all solutions, that is, $m(A)$ is the set of those strings that appear in $s(A)$ for every solution s. Certainly m is less than every solution, so it suffices to show that m is a solution. For if not, then for some production $A \to \alpha$ there exists a string $u \in \Sigma^*$ in $m'(\alpha)$ but not in $m(A)$. Now u must be of the form $u_1 \ldots u_k$ where $u_i \in m'(\alpha_k)$ for all α_k in α. But each of these u_i's must appear in every solution s since $m'(\alpha_k) \subseteq s'(\alpha_k)$, whence $u \in s'(\alpha)$ for every solution s. But then u must appear in $s(A)$ for every solution s, by definition of a solution, a contradiction. ∎

2. Bottom-up. We construct a sequence of functions $m_i : V \to 2^{\Sigma^*}$ each less or equal to all solutions, and show that its pointwise union, defined by $m(A) = \bigcup_i m_i(A)$ for all $A \in V$, itself necessarily less or equal to all solutions, is a solution.

Take $m_0(A) = \emptyset$. Take $m_{i+1}(A) = m_i(A) \cup \bigcup_{A \to \alpha \in \Pi} m'_i(\alpha)$, the union of $m_i(A)$ and all $m_i(\alpha)$ for productions $A \to \alpha$ having A on their left side.

Given a solution s, we argue by induction that each m_i is less than s. The base case is self-evident. By the induction hypothesis $m'_i(\alpha_j) \subseteq s'(\alpha_j)$ for every right side α_j whence $m_{i+1}(A) \subseteq s(A)$. It follows that the union m of all the solutions is also less than s, since every string in $m(A)$ must appear in $m_i(A)$ for some i.

Now suppose m is not a solution. Then for some production $A \to \alpha$ there exists a string $u \in \Sigma*$ in $m'(\alpha)$ but not in $m(A)$. By construction of m, u must appear in $m_i(\alpha)$ for some i. But then $u \in m_{i+1}(A)$, a contradiction. ∎

Notice the similarities and differences near the end of each construction. In both cases we showed that for all productions $A \to \alpha$, every $u \in m'(\alpha)$ must belong to $m(A)$. Going top-down, we used the fact that u must belong to $s(A)$ for *every* solution. Going bottom-up, we found u in *some* $m_i(A)$. The appearance of "every" in one method and "some" in the other is characteristic of such *dual* solutions to problems.

2 Derivations

The bottom-up construction may be associated with derivations (B&F p.342) as follows. Given a CFG G, define $n_i : V \cup \Sigma \to (V \cup \Sigma)^*$ inductively as $n_i(a) = \{a\}$ for all i and all $a \in \Sigma$, and for all $A \in V$ define $n_0(A) = \{A\}$, $n_{i+1}(A) = n_i(A) \cup \bigcup_{A \to \alpha \in \Pi} n_i(\alpha)$. (The difference from m_i in the bottom-up construction is that nonterminals can appear in $n_i(A)$; the strings of $n_i(A)$ are called *sentential forms*.)

Theorem 2. A CFG G has a derivation $S \stackrel{*}{\Rightarrow} \alpha$ if and only if $\alpha \in n(S)$.

Proof. Exercise 1.

We associate derivations with the bottom-up construction as follows.

Theorem 3. $m_i(A)$ consists of those strings of $n_i(A)$ containing no nonterminals.

Proof. Exercise 2.

3 $NSA \equiv CFL$

Theorem 4. The class NSA of languages accepted by NSA's coincides with the class CFL of context-free languages.

Proof. $CFL \subseteq NSA$. See text.

$NSA \subseteq CFL$. Define an NTA to be a control-less machine [stack,input] whose stack instructions are POP a, POP a;PUSH b;PUSH c, and NOOP, with a distinguished stack symbol S, which initializes its stack to hold S and accepts only when the input is empty and S is on the stack.

Lemma. Every NTA can be simulated by an NSA and vice versa.

Proof. It is easy to see how to simulate the pop-push-push instruction of an NTA with a three-instruction NSA subprogram, establishing one direction.

To simulate an NSA with an NTA, the idea will be to store the current control state on the top of the stack alongside the top stack symbol. The hard part is popping the stack, which erases the current control. Here we use the nondeterminism to *guess*, at the time we push, the state we want to be in later on when we pop, checking that the guess was correct when we eventually pop.

To this end, we take the NTA stack alphabet to be $Q \times \Gamma \times Q$ where Q and Γ are the control set and stack alphabet of the NSA. Simulate $(q_i \to q_j, \text{POP}A, in)$ (where in is any input operation or NOOP) by $(\text{POP}(q_i, A, q_j), in)$. Simulate each instruction $(q_i \to q_j, \text{PUSH}A, in)$ with $Q \times \Gamma \times Q$ instructions of the form (POP (q_i, B, q_k); PUSH (q_m, B, q_k); PUSH $(q_j, A, q_m), in)$, one for each NTA symbol (q_m, B, q_k). Simulate each instruction $(q_i \to q_j, NOOP, in)$ with (POP (q_i, B, q_k); PUSH (q_j, B, q_k); PUSH $(q_w, A, q_w), in)$, where q_w is a new state not in Q, associated with an additional NTA instruction $(\text{POP}(q_w, A, q_w), NOOP)$.

This works as follows. Before pushing $(q_j, A, ?)$ on top of (q_i, B, q_k), guess that when we pop it back we will want to be in state q_m. Do this by replacing it by (q_m, B, q_k) and then pushing (q_j, A, q_m). The last component creates a link between these two cells that ensures that when we later pop while jumping to state q_j, that state will miraculously appear on the stack, achieved by how we use the POP instruction. The details and rest of this construction should now be straightforward.

Exercise 3. Design an NUA [stack,input] having stack alphabet $Q \times \Gamma$ and different instruction set from the above NTA, such that the lemma follows with NUA in place of NTA. Prove the new lemma.

Given any NTA M, it is now straightforward to construct a CFG G generating the language accepted by M. Take $V = \Gamma$ and $S = S$. Define Π as follows.

For each instruction (POP A, SCAN b), a rule $A \to b$. For each instruction (POP A, NOOP), a rule $A \to \Lambda$. For each instruction (POP A;PUSH B;PUSH C, SCAN d), a rule $A \to dCB$. For each instruction (POP A;PUSH B;PUSH C, NOOP), a rule $A \to CB$. For each instruction (NOOP, SCAN a) and for each variable A, a rule $A \to aA$. The claimed connection should now be clear. ∎