Burt Rosenberg, July 4, 1996

0.0.1 Context Free Languages

Definition 1 *A context free grammar is a set Σ of terminals and a set $\mathcal{N}$ of non-terminals, a start symbol $S \in \mathcal{N}$, and a finite set of grammar rules, each of the form $A \rightarrow \alpha$ where $A \in \mathcal{N}$ and $\alpha \in (\Sigma \cup \mathcal{N})^*$.*

By using rules to substitute in for a nonterminal, from one string of terminals and nonterminals, another is derived. For σ, σ' in $(\Sigma \cup \mathcal{N})^*$, σ derives σ', $\sigma \Rightarrow \sigma'$, $\sigma = \beta A \beta'$, $\sigma' = \beta\alpha\beta'$ and $A \rightarrow \alpha$ is a grammar rule. The transitive closure of $\Rightarrow$ is $\Rightarrow^*$. The context free language associated with this grammar is the set,

$$\{\, \sigma \in \Sigma^* \,|\, S \Rightarrow^* \sigma \,\}.$$

Definition 2 *A subset of Σ^* is a context free language if it is associated with some context free grammar.*

Just as the Finite Automata are the machine equivalents of all regular expressions, the Pushdown Automata are the machine equivalents of the Context Free Languages. A pushdown automata is a nondeterministic finite automata augmented with the ability to consult and manipulate the top of a stack organized storage.

Definition 3 *A Pushdown Automata is a finite set of statest Q with a distinguished start state q_o, an input alphabet Σ, a stack alphabet Γ, with distinguished stack symbol Z_o, and transition relations $\delta \subseteq Q \times \Sigma \times \Gamma \times Q \times \Gamma^*$ and $\delta^\epsilon \subseteq Q \times \Gamma \times Q \times \Gamma^*$*

The automata starts in state q_o with the single character Z_o on the stack, and the current input symbol is the start of the input work. At each step, the stack can be represented as an element of Γ^*, with leftmost symbol at the top of stack. Transition $(q, a, z, q', w) \in \delta$ means that in state q, with current input symbol a, current top of stack symbol z, the automata consumes the input, moves to state q' and replaces z with w at the top of the stack. The epsilon moves δ^ϵ indicate transitions which neither consult nor consume input. Accepting the input is an empty stack, that is, when a transition leaves the stack equal to the empty string.

Definition 4 *A sentiential form of a grammar is any $\alpha \in (\Sigma \cup \mathcal{N})^*$ such that $S \Rightarrow^* \alpha$. If in the derivation $\alpha A \alpha' \Rightarrow \alpha\beta\alpha'$ the string α' is free from nonterminals, the derivation is called rightmost. The transitive closure $\Rightarrow^*_R$ of rightmost derivation gives the right sentential forms: any $\alpha \in (\Sigma \cup \mathcal{N})^*$ such that $S \Rightarrow^*_R \alpha$. Similar definitions exist for leftmost derivations and left sentential forms.*

If a string can be derived, it can be leftmost derived. This gives a quick proof that Pushdown Automata can recognize any Context Free Languages. With an epsilon move the automata places the start symbol for the grammar on the top of stack. Coded into the machine's transition relation are the grammar rules. If a nonterminal X is at the top of the stack, in epsilon move can replace it with a string α if $X \rightarrow \alpha$ is a grammar rule. If a terminal is at the top of the stack, the machine attempts to match it against the current input symbol. The stack symbol Z_0 must match against the end of input. This procedure amounts to guessing and verifying the leftmost derivation of the input string from the start symbol.

Theorem 1 *A language can be described by a Context Free Grammar if and only if it can be accepted by a Pushdown Automata.*

Half the theorem was proved above. To show the converse, we begin with a Pushdown Automata and construct a grammar. The nonterminal symbols of the grammar are of the form $[p, D, q]$ where p and q are machines states and D is a stack symbol. This nonterminal should reduce according to our grammar to any sting of terminals α if the machine accepts α traveling from state p to state q, while consuming D from an otherwise empty stack. The grammar rules are:

1. Transition $(p, a, D, q, D_1 D_2 \cdots D_m) \in \delta$ writes a family of rules:

$$[p, D, q_m] \rightarrow a[q, D_1, q_1][q_1, D_2, q_2] \cdots [q_{m-1}, D_m, q_m]$$

   where $q_1, \ldots, p_m$ is any sequence of machine states, for any $m > 0$; If $m = 0$, the rule is:

$$[p, D, q] \rightarrow a$$

2. Epsilon transition $(p, D, q, D_1 D_2 \cdots D_m) \in \delta^\epsilon$ writes a family of rules:

$$[p, D, q_m] \rightarrow [q, D_1, q_1][q_1, D_2, q_2] \cdots [q_{m-1}, D_m, q_m]$$

   where $q_1, \ldots, p_m$ is any sequence of machine states, for any $m > 0$; If $m = 0$, the rule is:

$$[p, D, q] \rightarrow \epsilon$$

3. The start state q_0 and start stack symbol Z_0 writes a family of start state rules:

$$S \rightarrow [q_0, Z_0, p]$$

   for all machines states q.

The remainder of the proof is left to the reader.

Since Finite Automata are Pushdown Automata which never use the stack, all regular expressions give languages which are context free. However, the language $\{a^i b^i \mid i \text{ any positive integer } \}$ is context free but not given by any regular expression. Also, with Pushdown Automata it makes a big difference if the underlying Finite Automata is deterministic or nondeterministic. For instance, the language $\{\, \alpha\alpha^R \mid \alpha \in \Sigma^* \,\}$, where α^R denotes α written backwards, can be represented by a pushdown automata only if the underlying FA is nondeterministic.

0.0.2 $SLR(1)$ Grammars

If a string can be derived from the start symbol, it can be rightmost derived from the start symbol. Rightmost derivations are important for us because they are used by shift-reduce parsers, such as Yacc, to analyze the context-free structure of a programming language. A shift-reduce parser uses a deterministic pushdown automata to scan the string left to right. At each step, the automata either pushes the next input symbol onto its stack, called a shift move, or replaces symbols on the top of its stack with a single nonterminal, according to a grammar rule. This is called a reduce move. A shift-reduce parser tries to find the rightmost derivation of the given string, working the derivations backwards until only the start symbol is left on the stack. Shift-reduce parsers do not work for all grammars, and more importantly, some context free languages have no grammars for which a sift-reduce parser will work. The language $\{\, \alpha\alpha^R \,\}$ is an example: it requires the underlying FA to be nondeterministic. The grammars for which shift-reduce parsing does work are called LR grammars.

We describe the shift-reduce parser in more detail. Both the current state and the current symbol are pushed onto the stack during a shift, so that former state can be recovered after a reduction. The entire process is driven by two tables, the action table and the goto table. The action table $\mathcal{A}$ takes a state and an input symbol, should indicates whether the machine should shift or reduce, and if reduce, by which rule. The goto table takes a state and a grammar symbol and gives the next state. For a shift move, the grammar symbol is the terminal which is shifted. For a reduce move, the grammar symbol is the nonterminal on the lefthand side of the reduction.

Which leaves us with the job of creating the action and goto tables for the parser. Various methods are employed, of increasing strength yielding tables of increasing size. The methods which work for a given grammar give subclassifications to LR grammars. We describe in detail the $SLR(1)$ method, and the grammars for which this method works are called $SLR(1)$ grammars.

Definition 5 *An item is a grammar rule with a dot inserted somewhere in the lefthand side. That is, for any grammar rule $A \rightarrow \alpha\beta$ with α and β possibly empty strings of terminals and non-terminals, then $A \rightarrow \alpha \cdot \beta$ is an item.*

Definition 6 *The closure of a set of items E, denoted $Cl(E)$, is the smallest set of items containing E and closed by the following rule:*

$$\text{if } (A \rightarrow \alpha \cdot B\beta) \in Cl(E) \text{ and } B \rightarrow \gamma \text{ is a grammar rule, then } (B \rightarrow \cdot\gamma) \in Cl(E)$$

Definition 7 *The canonical $LR(0)$ collection of items for a grammar, augmented with start rule $S' \rightarrow S$, is the smallest collection of sets of items containing $Cl(S' \rightarrow \cdot S)$ and containing $g(I, X)$ for every $I \in LR(0)$ and grammar symbol X. The "goto function" $g(I, X)$ is defined as:*

$$g(I, X) = Cl\big(\{\, (A \rightarrow \alpha X \cdot \beta) \,|\, (A \rightarrow \alpha \cdot X\beta) \in I \,\}\big).$$

Definition 8 *The "follow function" f from nonterminals to sets of terminals is defined as:*

$$f(X) = \{\, a \in \Sigma \,|\, \exists \alpha, \beta \in (\Sigma \cup \mathcal{N})^* : S \Rightarrow_R^* \alpha X a \beta \,\}.$$

Definition 9 *The $SLR(1)$ parsing table is constructed from the $LR(0)$ collection of items, the goto function $g(\cdot,\cdot)$ and the follow function $f(\cdot)$ according to the rules:*

1. *The $LR(0)$ items are each a state. The initial state is the one containing the item $(S' \rightarrow \cdot S)$.*
2. *The state transitions are given by the goto function g.*
3. *For $a \in \Sigma, (A \rightarrow \alpha \cdot a\beta) \in I$ and $J = g(I, a)$, then $\mathcal{A}(I, a)$ is shift.*
4. *For $(A \rightarrow \alpha\cdot) \in I$ and $a \in f(A)$, then $\mathcal{A}(I, a)$ is reduce according to $A \rightarrow \alpha$.*
5. *If $(S' \rightarrow S\cdot) \in I$, then $\mathcal{A}(I, \$)$ is to accept.*

6. All other cases are an error.

If the rules cannot be applied because of conflicts, then the procedure fails, and the grammar is not of type $SLR(1)$.

Give a state and grammar symbol, it might be that both a shift and some reduction are indicated, or that more than one reduction is indicated. The first case causes a shift-reduce conflict, the second case causes a reduce-reduce conflict.

LR grammars might cause such conflicts when using the $SLR(1)$ procedure to build the parsing tables, yet have no conflicts when using other procedures to build the parsing tables. A stronger method begins with $LR(1)$ items and builds the $LR(1)$ parsing table. Any $SLR(1)$ will work as a $LR(1)$, but the $LR(1)$ might resolve shift-reduce and reduce-reduce conflicts which cause failure for an $SLR(1)$, but at the cost of much larger parsing tables.

0.0.3 $LR(1)$ and $LALR(1)$ Grammars

Definition 10 *An $LR(1)$ item is a pair $(A \rightarrow \alpha \cdot \beta, a)$ where $A \rightarrow \alpha\beta$ is a grammar rule and $a \in \Sigma$ or the special endmarker \$.*

Definition 11 *The "first function" from words in $(\Sigma \cup \mathcal{N})^*$ to subsets of Σ is defined as:*

$$r(\gamma) = \{\, a \in \Sigma \,|\, \exists \gamma' \in \Sigma^* : \gamma \Rightarrow_R^* a\gamma' \,\}.$$

Definition 12 *The closure of a collection of $LR(1)$ items E, $Cl(E)$, is the smallest set of $LR(1)$ items containing E, and close by the following rule,*

$$\textit{if } (A \rightarrow \alpha \cdot B\beta, a) \in Cl(E) \textit{ and } B \rightarrow \gamma \textit{ is a grammar rule and } b \in r(\beta a), \textit{ then } (B \rightarrow \cdot\gamma, b) \in Cl(E).$$

Definition 13 *The canonical $LR(1)$ collection of items for a grammar, augmented with start rule $S' \rightarrow R$, is the smallest collect of sets of items containing $Cl(S' \rightarrow \cdot S, \$)$ and containing $g(I, X)$ for every $I \in LR(1)$ and grammar symbol X. The "goto function" $g(I, X)$ is defined as:*

$$g(I, X) = Cl\big(\{(A \rightarrow \alpha X \cdot \beta, a) \,|\, ((A \rightarrow \alpha \cdot X\beta, a) \in I\,\}$$

An intermediate method is the $LALR(1)$ method, which reintroduces certain reduce-reduce conflicts that the $LR(1)$ might have resolved, but never reintroduces any resolved shift-reduce conflicts. On the positive, $LALR(1)$ grammar are as small and as fast as $SLR(1)$ grammars. Yacc uses parses $LALR(1)$ grammars plus some additional rules passed on *operator precedence.*

A complete study of LR grammars is D. E. Knuth's "On the Translation of Languages from Left to Right," Information and Control Vol. 8, 1965. All deterministic grammars are $LR(k)$ for some k, and for a fixed k, it can be systemactically determined if a given context free grammar is $LR(k)$. An $LR(k)$ grammar can be rewritten as an $LR(1)$ grammar giving the same language. However, it is provably impossible to give an algorithm which, given a grammar, provides a k for which the grammar is $LR(k)$ or reports that no such k exists. This hinges on the fact that an $LR(k)$ grammar is a deterministic context free language, and it is formally undecidable whether a given context free language is determinisitic.