Non-context-free Languages

CMPU 240 • Language Theory and Computation • Fall 2018

Previously:
   Equivalence of CFGs and PDAs

Today:
   Brief discussion of deterministic PDAs
   Cleaning grammars: Chomsky normal form

Later:
   A brand-new Pumping Lemma

Assignments
   Assignment 5 out on Tuesday
   Optional programming assignments out now

Deterministic PDAs

Intuitively: never a choice of move

\( \delta(q, a, Z) \) has at most one member for any \( q, a, Z \)
(including \( a = \varepsilon \)).

If \( \delta(q, \varepsilon, Z) \) is nonempty, then \( \delta(q, a, Z) \) must be empty
for all input symbols \( a \).
Why care about DPDAs when they’re less powerful?

Parsers, as in Yacc, are really DPDAs

Thus, the question of what languages a DPDA can accept is really the question of what programming language syntax can be parsed conveniently.

What’s Yacc?
Yet Another Compiler-Compiler

It’s a standard Unix tool (from the 1970s) that generates the parser for a programming language given a description of the grammar.

Today you’re more likely to see Bison, a replacement for Yacc.

Some language relationships

Acceptance by empty stack is hard for a DPDA

Once it enters an accept state, it cannot accept any continuation. Thus, \( N(P) \) has the prefix property: if \( w \) is in \( N(P) \), then \( wx \) is not in \( N(P) \) for any \( x \neq \epsilon \).

\( N \) stands for “null stack”; \( N(P) \) is the set of inputs that \( P \) can consume and at the same time empty its stack.

However, parsers do accept by emptying their stack

*Trick:* they really process strings followed by a unique endmarker (typically \( $ \)), e.g., if they accept \( w\$ \), they consider \( w \) to be a correct program.

If \( L \) is a regular language, then \( L \) is a DPDA language

A DPDA can simulate a DFA without using its stack (acceptance by final state)

If \( L \) is a DPDA language, then \( L \) is a CFL that is not inherently ambiguous

A DPDA yields an unambiguous grammar in the standard construction

Interesting fact: The class of languages accepted by NPDAs is larger than those accepted by DPDAs!
PDAs are more powerful than FAs

Languages accepted by nondeterministic PDA
Languages accepted by FA or NFA
Languages accepted by deterministic PDA

Cleaning grammars: Chomsky normal form

Cleaning up grammars

We can “simplify” grammars, e.g.,

Get rid of e-productions
Variables of the form variable → ε
But lose the ability to generate ε as a string in the language

Get rid of useless symbols
Variables that do not participate in any derivation of a terminal string

Get rid of unit productions
Variables of the form variable → variable

Any CFG can be converted via these and other methods to Chomsky normal form (CNF), where the only production forms are

A → BC
A → a

Getting rid of the empty string

The empty string is a nuisance with grammars and languages in general.

We will look at languages that do not contain ε.

There’s no loss of generality:

For language L, let G = (V, T, S, P) be a CFG that generates L − {ε}.
Modify the grammar by adding a new start variable S₀ and add productions S₀ → S | ε.
This grammar generates L.
Therefore any non-trivial conclusion we make for L − {ε} should transfer to L.
Eliminating ε-productions

A variable $A$ is **nullable** if $A \Rightarrow^* \varepsilon$

Find them by a recursive algorithm:

*Basis:* If $A \Rightarrow \varepsilon$ is a production, then $A$ is nullable

*Induction:* If $A$ is the head of a production whose body consists of only nullable symbols, then $A$ is nullable

Once we have the nullable symbols, we can add additional productions and then throw away the productions of the form $A \Rightarrow \varepsilon$ for any $A$

Example

If $A \Rightarrow X_1X_2\ldots X_k$ is a production, add all productions that can be formed by eliminating some or all of those $X_i$s that are nullable

* ✓ But don’t eliminate all $k$ if they are all nullable!

Example

If $A \Rightarrow BC$ is a production, and both $B$ and $C$ are nullable, add $A \Rightarrow B \mid C$

Useless symbols

For a symbol $X$ to be useful, it must:

1. Derive some terminal string (possibly $X$ is a terminal)
2. Be reachable from the start symbol, i.e., $S \Rightarrow^* \alpha X \beta$

Note:

$X$ wouldn’t really be useful if $\alpha$ or $\beta$ included a symbol that didn’t satisfy (1).

So, it’s important to test (1) first, and eliminate symbols that don’t derive terminal strings before testing (2).
Finding symbols that don’t derive any terminal string

Recursive construction:

**Basis:** A terminal clearly derives a terminal string.

**Induction:** If $A$ is the head of a production whose body is $X_1X_2\ldots X_k$, and each $X_i$ is known to derive a terminal string, then $A$ derives a terminal string.

Keep going until no more symbols that derive terminal strings are discovered.

Example

$$S \rightarrow AB \mid C$$
$$A \rightarrow 0B \mid C$$
$$B \rightarrow 1 \mid Ao$$
$$C \rightarrow AC \mid C1$$

✓ **Round 1:** $0$ and $1$ are “in”
✓ **Round 2:** $B \rightarrow 1$ says $B$ is in
✓ **Round 3:** $A \rightarrow 0B$ says $A$ is in
✓ **Round 4:** $S \rightarrow AB$ says $S$ is in
✓ **Round 5:** Nothing more can be added

So $C$ can be eliminated, along with any production that mentions it.

Finding symbols that can’t be derived from the start symbol

Another recursive algorithm:

**Basis:** $S$ is “in”

**Induction:** If variable $A$ is in, then so is every symbol in the production bodies for $A$.

Keep going until no more symbols derivable from $S$ can be found.

The symbols that can’t be derived are those not found by this algorithm.

Example

$$S \rightarrow AB$$
$$A \rightarrow 0B$$
$$B \rightarrow 1 \mid Ao$$

✓ **Round 1:** $S$ is in
✓ **Round 2:** $A$ and $B$ are in
✓ **Round 3:** $0$ and $1$ are in
✓ **Round 4:** Nothing can be added

→ In this case, all symbols are derivable from $S$, so no change to grammar

The book has an example where not only are there symbols not derivable from $S$, but you must eliminate first the symbols that don’t derive terminal strings, or you get the wrong grammar.
Eliminating unit productions

1. Eliminate useless symbols and $\varepsilon$-productions
2. Discover those pairs of variables $(A, B)$ such that $A \Rightarrow B$
   
   Because there are no $\varepsilon$-productions, this derivation can only use unit productions
3. Replace each combination where $A \Rightarrow B \Rightarrow \alpha$ and $\alpha$ is other than a single variable by $A \rightarrow \alpha$
   
   I.e., “short circuit” sequences of unit productions, which must eventually be followed by some other kind of production
4. Remove all unit productions

Chomsky normal form

1. Get rid of useless symbols, $\varepsilon$-productions, and unit productions (already done)
2. Get rid of productions whose bodies are mixes of terminals and variables or consist of more than one terminal
3. Break up production bodies longer than 2

Result

All productions are of the form $A \rightarrow BC$ or $A \rightarrow a$

No mixed bodies

1. For each terminal $a$, introduce a new variable $A_a$, with one production $A_a \rightarrow a$
2. Replace $a$ in any body where it is not the entire body by $A_a$

Now, every body is either a single terminal or it consists only of variables

Example

$A \rightarrow aB1$ becomes $A_o \rightarrow o; A_r \rightarrow \tau; A \rightarrow A_oBA_r$

Example: Earlier grammar

Grammar from which $\varepsilon$-productions were removed

Contained no unit productions or useless symbols

\[
\begin{align*}
S & \rightarrow aA \\
A & \rightarrow aABC | bB | a | aAB | A_A | aAC | b \\
B & \rightarrow b \\
C & \rightarrow c
\end{align*}
\]

\[
\begin{align*}
S & \rightarrow aA \\
A & \rightarrow aABC | bB | a | aAB | A_A | aAC | b \\
B & \rightarrow b \\
C & \rightarrow c \\
A_o & \rightarrow a
\end{align*}
\]

Already have variables for $b$ and $c$
Making bodies short

If we have a production like $A \rightarrow BCDE$, we can introduce some new variables that allow the variables of the body to be introduced one at a time.

A body of length $k$ requires $k - 2$ new variables.

Example

Introduce $F$ and $G$; replace $A \rightarrow BCDE$ by $A \rightarrow BF; F \rightarrow CG; G \rightarrow DE$

Full procedure

Do in this order:

1. Eliminate $\varepsilon$-productions
2. Eliminate useless symbols
3. Eliminate unit productions
4. Eliminate mixed bodies
5. Make all bodies short

Example: Earlier grammar

Summary theorem

If $L$ is any CFL, there is a grammar $G$ that generates $L - \{\varepsilon\}$, for which each production is of the form $A \rightarrow BC$ or $A \rightarrow a$, and there are no useless symbols.