TDDD14/TDDD85 Lecture 8: Rewriting CFGs and normal forms

Jonas Wallgren

Abstract

This lecture presents methods for transforming CFGs.

1 Introduction

You saw in lecture 5 that e.g. for efficiency reasons you may want to minimize automata. Some similar thoughts arise with grammars.

If you want to reason about a grammar, e.g. prove some properties of it, or if you want to use it to implement a parser, it can benefit your work if the grammar has another form, a form more fit for your application, but still, of course, defines the same language. Some of the rewritings in this lecture reduce the size of a grammar while other ones instead restructures a grammar for some other purpose and may increase its size. All rewriting methods in this lecture will be presented via various examples.
In all example grammars in this lecture the start symbol is S.

A reminder from last lecture:

- A, B, C, P, Q, S, T represent nonterminals.
- a, b, c, p, q, r represent terminals.
- α,β, γ represent strings of terminals and nonterminals.

A-B is used to express the (set) difference between the two sets A and B.

2 Simplification — unnecessary symbols

Let's start with a very simple grammar over Σ={a}:

Grammar 1
$S \rightarrow AB|a$
$A \rightarrow a$

Even this small, this grammar contains unnecessary parts and can be simplified. The simplication is performed in two steps. The first step looks like this:

- From A a string in Σ^* can be derived.
- From S a string in Σ^* can be derived.
- From B no string in Σ^* can be derived.

Beginning with rules having only terminals in their right-hand sides the grammar is gone trough and information about from which nonterminals strings can be derived is collected. You can, e.g., not say anything about a rule S→PQ until you,ve got that information for both P and Q.

Now, since no string can be derived from B it can be removed:

Grammar 2
$S \rightarrow a$
$A \rightarrow a$

The first part of the simplification looked at the rules from right to left. The second part looks at what can be reached from the start symbol:

- S can be reached from S.
- A can't be reched from S.

Beginning from the start symbol, see which nonterminals it uses. Then see what nonterminals they use, etc. until all reachable nonterminals are found.

Now, since A can't be reached it could be removed:

Grammar 3
$S \rightarrow a$

These two parts of the rewriting produce a grammar without various unnecessary parts still defining the same language. So L(**Grammar 3**)=L(**Grammar 1**).

3 Simplification — ε productions and unit productions

It can be convenient to use rules like A→ ε (called epsilon productions) and A→ B (called unit productions) when writing a grammar, but such rules may interfere with e.g. implementation or proving properties of the grammar by making them more time-consuming or more difficult. This section handles removal of such rules, but still, of course, keeping the language of the grammar the same.

This section is based on the following grammar:

Grammar 4
$S \rightarrow aSb \mid T$
$T \rightarrow pTq \mid \varepsilon$

L(**Grammar 4**)=$\{a^m p^n q^n b^m | m \geqslant 0 \wedge n \geqslant 0\}$.

P is the set of rules of **Grammar 4**. We will construct an extended set $\hat{P}$ of rules. Begin by setting $\hat{P}$=P. Then continue using these two steps until no more extension is posible:

- If A→ αBγ and B→ ε are in $\hat{P}$ then put A→ $\alpha\gamma$ in $\hat{P}$.
- If A→B and B→ β are in $\hat{P}$ then put A→ β in $\hat{P}$.

The first step assures that the possible ε for B is taken care of already in the rule for A. Later that ε production can be removed. The second step removes the "unnecessary" step through B and uses its right-hand side immediately. If β is just a nonteminal that unit production can later be removed.

To perform this procedure for grammar 4 we thus start with $\hat{P}$={S→aSb, S→T, T→pTq, T→ ε}. Then these additions to $\hat{P}$ will follow:

- Since S→T and T→ ε in $\hat{P}$, put S→ ε in $\hat{P}$.
- Since T→pTq and T→ ε in $\hat{P}$, put T→pq in $\hat{P}$.
- Since S→aSb and S→ ε in $\hat{P}$, put S→ab in $\hat{P}$.
- Since S→T and T→pTq in $\hat{P}$, put S→ pTq in $\hat{P}$.
- Since S→pTq and T→ ε in $\hat{P}$, put S→pq in $\hat{P}$.

Now, no more such extension steps can be done. $\hat{P}$ is complete. To get the rules of the (almost) final grammar remove all epsilon and unit productions from $\hat{P}$ and the result is:

Grammar 5
$S \rightarrow aSb|ab|pTq|pq$
$T \rightarrow pTq|pq$

This grammar describes almost the same language as grammar 4. If in the original grammar S$\stackrel{*}{\Rightarrow}\varepsilon$ then that property is lost, since all epsilon productions are removed. But

L(**Grammar 5**)=L(**Grammar 4**)-{ε}.

So, since ε ∈L(**Grammar 4**) the rule S→ ε is added to **Grammar 5**.

4 Chomsky normal form

Sometimes you don't want to minimize or simplify a grammar. Instead you want it to have some other special form to e.g. ease the proof of some properties or some specific uses of it.

Definition 1 *A grammar is in Chomsky normal form if all rules have the form* $A \to a$ *or* $A \to BC$.

The starting point is a grammar without ε and unit productions, in our example

Grammar 6
$S \to aSb|ab|pTq|pq$
$T \to pTq|pq$

i.e. **Grammar 5** without the added ε rule.

The first step is to introduce rules for every nonterminal:

Grammar 7
$S \to ASB|AB|PTQ|PQ$
$T \to PTQ|PQ$
$A \to a$
$B \to b$
$P \to p$
$Q \to q$

Then for all productions of the form $\mathrm{A} \to \mathrm{B}_1\mathrm{B}_2\mathrm{B}_3 \ldots$ replace it by $\mathrm{A} \to \mathrm{B}_1\mathrm{C}$ and $\mathrm{C} \to \mathrm{B}_2\mathrm{B}_3 \ldots$. Iterate that as many steps as necessary (for long sequences of B_n.). The result in our example is

Grammar 8
$S \to AE|AB|PF|PQ$
$T \to PF|PQ$
$E \to SB$
$F \to TQ$
$A \to a$
$B \to b$
$P \to p$
$Q \to q$

This grammar is in Chomsky normal form.

L(**Grammar 8**)=L(**Grammar 6**)=L(**Grammar 4**)-$\{\varepsilon\}$.

Since these rewriting steps apply to all grammar rules all grammars can be rewritten into Chomsky normal form.

5 Greibach normal form

Definition 2 *A grammar is in Greibach normal form if all rules have the form* $A \to aB_1B_2B_3 \ldots$.[1]

[1] In strict Greibach normal form the language can't contain ε

This lecture just states this definition of Greibach normal form. It will be used later in the lectures but it is to intricate, it takes too many steps, to be discussed in details here. You don't need to be able to transform a grammar to Greibach normal form, you just need to know its name and how it looks.

6 Left recursion

The last grammar transformation has a direct practical motivation. Take this grammar rule as an example:

A→Ap|q

One possible derivation using these rules are

A⇒Ap⇒App⇒Appp⇒qppp

One way to construct a parser for this language is to have a procedure Aproc that tries to behave according to the grammar rules — first find a substring corresponding to A and then find a p or find a q. I.e it starts by calling Aproc, which starts by calling Aproc, which .... and if it begins with q it can't find the remaining ppp.

The problem with this grammar is that it is left recursive. In at least one rule for at least one nonterminal the same nonterminal occurs first in the right-hand side of that rule. If you implement a parser as sketched here it will loop.

If you could rewrite the grammar, keeping its language, so the q is read first — it *is* the first symbol of every string in the language — the recursion problem would be solved. You want a derivation like

B⇒qC⇒ $\cdots$

for some nonterminals B and C.

Now, since this could be a part of a grammar where A is used somewhere else you can't change its name. But you could introduce a new nonterminal to handle the recursion. Since there could be many recursions in a grammar I follow the tradition and use the same name with a prime instead of inventing a new name:

A→qA'
A'→pA'|ε

If there are many rules for the left-recursive nonterminal all these must be handled in one step, like:

A→Ap|Aq|Ar|a|b|c

results in

A→aA'|bA'|cA'
A'→pA'|qA'|rA'|ε

Now, every call to a parsing procedure will cause a terminal to be read and there is no infinite recursion.

A more complicated situation occurs when a rule for A begins with B and a rule for B begins with A, i.e. mutual recursion. There could also be yet more complicated cases. But all such recursivity is solvable. All grammars can be rewritten to non-leftrecursive form.

7 More to think about

1. How do the various rewriting methods affect the size, i.e. the number of rules, of a grammar. Could there e.g. be some exponential blow-up?

2. Give a grammar in Chomsky normal form for the language
   $\{a^n b^{2n} c^k | n \geqslant 1 \wedge k \geqslant 1\}$