TDDD14/TDDD85 Lecture 10: Equivalence between CFG and PDA

Jonas Wallgren

Abstract

In this lecture will be shown that context free grammars and push down automata have same power.

1 Introduction

In lecture 7 we introduced CFGs to express languages where there are some dependencies inside the strings that can't be decribed using regular expressions, e.g. that brackets match. In lecture 9 we introduced PDAs for the same puropse. Like we did for regular languages in lecture 4 we will now prove that the automata and the notation for this new class of languages (CFLs) are equivalent. This is done in two steps: first the equivalence is proved as an implication in one direction, then in the other direction. All proofs are sketchy to just give a hint of the details. The importance doesn't lie in the proofs but in the existence of the conversions and their consequences.

Finally there is a comment on DPDAs. We will return to PDAs later when we compare different classes of languages.

2 CFG→PDA

In this section we prove that given a CFG we can construct a PDA that accepts the language of the CFG.

The starting point is a grammar $G = \langle N, \Sigma, P, S\rangle$ on Greibach normal form[1], e.g. every rule in P has the form

$A \to cB_1B_2 \dots B_n$(where $c \in \Sigma, B_i \in N, n \geqslant 0$).

Now, create a PDA $M = (\{q\}, \Sigma, N, \delta, q, S, \emptyset\rangle$, where

- There is just one state.

[1] All CFGs can be transformed into Greibach normal form.

- The stack symbols are the nonterminals with the grammar start symbol as the stack start symbol.
- There are no final states, so acceptance will be by empty stack.

The idea is that when there is a nonterminal on the stack top you choose one of the grammar rules for that nonterminal, read the a first in its right-hand side, and "queue" all the B:s on the stack. So, δ will for each grammar rule contain

$\langle\langle q, c, A\rangle, \langle q, B_1, B_2 \dots B_n\rangle\rangle$.

Example 1. *Take this grammar, with start symbol E,*

$E \rightarrow aX|bX|aY|bY|aYX|bYX|a|b$
$T \rightarrow aY|bY|a|b$
$X \rightarrow +E$
$Y \rightarrow *T$

which is a grammar on Greibach normal form for arithmetic expressions. If we construct a PDA as above 5 of the 14 elements in δ will be:

$\langle\langle q, a, E\rangle, \langle q, YX\rangle\rangle,$
$\langle\langle q, *, Y\rangle, \langle q, T\rangle\rangle,$
$\langle\langle q, +, X\rangle, \langle q, E\rangle\rangle,$
$\langle\langle q, b, T\rangle, \langle q, \varepsilon\rangle\rangle,$
$\langle\langle q, a, E\rangle, \langle q, \varepsilon\rangle\rangle.$

If you, using the grammar, perform the following derivation:

$E \Rightarrow aYX \Rightarrow a * TX \Rightarrow a * bX \Rightarrow a * b + E \Rightarrow a * b + a$

that corresponds to the following next-configuration steps using the PDA:

$\langle q, a * b + a, E\rangle \rightarrow$
$\langle q, *b + a, YX\rangle \rightarrow$
$\langle q, b + a, TX\rangle \rightarrow$
$\langle q, +a, X\rangle \rightarrow$
$\langle q, a, E\rangle \rightarrow$
$\langle q, \varepsilon, \varepsilon\rangle$ *and the string is accepted with empty stack.*

Theorem 1. *If a language is defined by a CFG then it is accepted by a PDA.*

Proof. Let $z, y \in \Sigma^*, \gamma \in N^*, A \in N$. Construct the PDA as above. Then[2] $A \overset{n}{\Rightarrow}_{lm} z\gamma \Leftrightarrow \langle q, zy, A\rangle \overset{n}{\rightarrow} \langle q, y, \gamma\rangle$. The grammar is on Greibach normal form so both each derivation step and each next-configuration step handles one nonterminal, so this could formally be proved by induction over the number of steps.

[2]Remember that $\Rightarrow_{lm}$ is leftmost derivation.

From this it follows $S \overset{*}{\Rightarrow}_{lm} x \Leftrightarrow \langle q, x, S\rangle \overset{*}{\rightarrow} \langle q, \varepsilon, \varepsilon\rangle$, i.e. if a string can be derived by the CFG it is accepted by the PDA. $\Box$

The construction presented above especially eases the induction proof. In general you don't need the grammar to be on Greibach normal form. If you just want to do a transformation of a grammar into a PDA you can allow terminals on the stack. For every grammar rule $A \rightarrow \alpha$ let $\langle\langle q, \varepsilon, A\rangle, \langle q, \alpha\rangle\rangle$ be an element in δ and for every nonterminal a let $\langle\langle q, a, a\rangle, \langle q, \varepsilon\rangle\rangle$ be an element in δ.

3 PDA→CFG

In this section we prove that given a PDA we can construct a CFG that describes the language of the PDA. First we will transform a one-state PDA to a grammar, then we will convert a many-state one to the one-state form.

3.1 A special case

Given a one-state PDA. The only important property of the PDA is its single state. Its Σ and Γ may overlap. The state may be final or not.

For every element $\langle\langle q, c, A\rangle, \langle q, B_1 B_2 \ldots B_n\rangle\rangle \in \delta$ let the grammar have the rule $A \rightarrow cB_1 B_2 \ldots B_n$. Let the start stack symbol of the PDA be the start symbol of the CFG.

3.2 The general case

Let the starting point be the PDA $M = \langle Q, \Sigma, \Gamma, \delta, s, \perp, \{t\}\rangle$ with one final state. We see from the construction of M' in lecture 9 that a PDA M can be converted into a PDA M' with just one final state.

Remember from lecture 4 that when converting an NFA to a DFA a state in the resulting DFA can represent the possiblility of being in several states in the NFA. In converting a many-state PDA to a one-state one we will in some similar manner put much information into the stack symbols. The stack symbols will have names with a structure representing their intended use.

If we with M have $\langle p, x, A\rangle \overset{n}{\underset{M}{\rightarrow}} \langle q, \varepsilon, \varepsilon\rangle$ we will with our new M' have $\langle *, x, [pAq]\rangle \overset{n}{\underset{M'}{\rightarrow}} \langle *, \varepsilon, \varepsilon\rangle$. Note that $[pAq]$ is just a name. The PDA doesn't look at its parts, it can't reconize them or separate them. But the name is constructed in such a way that we can handle it easily.

Now we can define $M' = \langle\{*\}, \Sigma, \Gamma', \delta', *, [s\perp t], \emptyset\rangle$, where $\Gamma = Q \times \Gamma \times Q$ and $*$ is an arbitrarily chosen state name, just to be different from any other state name. For every element $\langle\langle p, c, A\rangle, \langle q_0, B_1 B_2 \ldots B_n\rangle\rangle \in \delta$ we will have $\langle\langle *, c, [pAq_n]\rangle, \langle *, [pBq_1][q_1 Bq_2] \ldots [q_{n-1} Bq_n]\rangle \in \delta'$ for *all* $q_1, q_2, \ldots q_n \in Q$. Yes, there should be an element in δ' for *every* possible combination of n states in M.

Lemma 1. *If a language is accepted by a many-state PDA it is accepted by a one-state one.*

Proof. Construct M' as above. Then $\langle p, x, B_1B_2\dots B_n\rangle \xrightarrow[M]{n} \langle q_n, \varepsilon, \varepsilon\rangle \Leftrightarrow$
$\Leftrightarrow \langle *, x, [pB_1q_1][q_1B_2q_2]\dots[q_{n-1}pB_1q_n]\rangle \xrightarrow[M']{n} \langle *, \varepsilon, \varepsilon\rangle$ The stack symbols in M' are named after each step in M, so this could formally be proved by induction over n.

From this it follows $\langle s, x, \perp\rangle \xrightarrow[M]{*} \langle t, \varepsilon, \varepsilon\rangle \Leftrightarrow \langle *, x, [s\perp t]\rangle \xrightarrow[M']{*} \langle *, \varepsilon, \varepsilon\rangle$, i.e. if a string is accepted by a many-states PDA it is accepted by a one-state one. □

Theorem 2. *If a language is accepted by a PDA then it is the language af a CFG.*

Proof. According to **Lemma 1** a many-states PDA can be converted to a one-state one. According to section 3.1 a one-state PDA can be transformed into a CFG. □

4 DPDA

In a DFA the transition function must be just that, a (total) function that is defined for all arguments: In any state, for any symbol read, there must be a state to go to. Sometimes you, somewhat informally, allow the transition to be a partial function, i.e. undefined for some arguments. See e.g. the solution to exercise 2 in the first set of home assignments. For PDAs that is the normal way to handle it.

Definition 1. *A deterministic pushdown automaton, DPDA, is a PDA where δ is a partial function.*

So, there is for every state p

- either at most one $\langle\langle p, a, A\rangle, \langle q, \beta\rangle\rangle$ for each symbol a
- or a $\langle\langle p, \varepsilon, A\rangle, \langle q', \beta'\rangle\rangle$ in δ.

If a DPDA accepts x with empty stack it can't accept xy (y$\neq \varepsilon$) with empty stack. In other words: A string in the accepted language cannot be a proper prefix of another string in the language. That is called *the prefix property.* We will in coming lectures see how that is handled by introducing a special end marker. If the original grammar is $S \to \varepsilon | aS$ (with strings like a, aa, and aaa where every string is a prefix of other ones) then the rule $S' \to S\$$ is added (leading to the strings a$, aa$, aaa$ etc).

5 More to think about

1. In Section 2 we, given a CFG, constructed a (nondeterministic) PDA recognising the same language as the grammar. Can you find any part in the construction where nondeterminism is used, or seems to be important?

2. Assume that we write a computer program which simulates a PDA (using a brute force approach which explores all possible transitions). Using this computer program and the construction in Section 2 we then have a method for recognising a language represented by a context-free grammar, by (1) converting the grammar to a PDA, and (2) simulating the PDA on an input string. Can you think of any weaknesses of this approach? What happens if we have a complicated grammar (corresponding to the syntax of a programming language) and an input string consisting of thousands or millions of symbols (corresponding to a program)?