TDDD14 / TDDD85 – Lecture 10

Equivalence between CFG and PDA

August Ernstsson, 2024 (based on lecture notes by Jonas Wallgren)

LINKÖPING UNIVERSITY

From last week

Definition 1: Greibach normal form

- A grammar is in Greibach normal form (GNF)
  - **if all** productions have the form $A \rightarrow aB_1B_2B_3\ldots$
  - (Sequence of B's may be empty)

- Every CFG can be converted to a CFG in GNF.

- Greibach normal form will be important in lecture 10.

Simplification – ε productions and unit productions

- *ε productions*: A → ε

- *Unit productions*: A → B

- These rules can be convenient when defining a grammar.
- But: needlessly complicates analysis or implementation.

Definition 1: Pushdown Automata

- A PDA is a septuple $\langle Q, \Sigma, \Gamma, \delta, s, \bot, F \rangle$
  - Q = set of states
  - Σ = (input) alphabet
  - Γ = stack alphabet
  - s = start state $\in Q$
  - F = final states $\subseteq Q$
  - $\bot$ = start stack symbol
  - δ = transition relation $\subseteq (Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma) \times (Q \times \Gamma*))$

The transition relation δ

- δ = transition relation $\subseteq (Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma) \times (Q \times \Gamma*))$

- $\langle\langle p, a, A\rangle, \langle q, B_1, B_2 \ldots B_n\rangle\rangle \in \delta$ means:
  - In state p with A on top of the stack:
    - Read a,
    - go to state q,
    - change the stack top A to $B_1B_2\ldots B_n$ (the left end being the new stack top).

The transition relation δ

- δ = transition relation $\subseteq (Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma) \times (Q \times \Gamma*))$

- $\langle\langle p, \varepsilon, A\rangle, \langle q, B_1, B_2 \ldots B_n\rangle\rangle \in \delta$ means:
  - In state p with A on top of the stack, without reading anything:
    - Go to state q,
    - change the stack top A to $B_1B_2 \ldots B_n$ (the left end being the new stack top).

Acceptance

- Two different modes of acceptance are used in the literature.
- A PDA can accept a string if
  - the *stack is empty*, or
  - if it reaches a *final state*.

- **Definition 4**
  - A PDA accepts the string x if $\langle s, x, \bot \rangle \rightarrow^* \langle q, \varepsilon, \gamma \rangle$ when $q \in F$.
- **Definition 5**
  - A PDA accepts the string x if $\langle s, x, \bot \rangle \rightarrow^* \langle q, \varepsilon, \varepsilon \rangle$ when $q \in Q$

Configurations

- What we handled in Example 2 are called *configurations*.

- **Definition 2**. A configuration is a triple $\langle q, x, \gamma \rangle$, where:
  - q = state $\in Q$
  - x = string $\in \Sigma*$
  - γ = stack $\in \Gamma*$

From Example 2:
$\langle q1, 01c01, R \rangle \rightarrow \langle q1, 1c01, BR \rangle$

- **Definition 3**. The next-configuration relation $\rightarrow$
  - If $\langle \langle p, a, A \rangle, \langle q, \gamma \rangle \rangle \in \delta$ then $\langle p, ay, A\beta \rangle \rightarrow \langle q, y, \gamma\beta \rangle \rangle$
  - If $\langle \langle p, \varepsilon, A \rangle, \langle q, \gamma \rangle \rangle \in \delta$ then $\langle p, y, A\beta \rangle \rightarrow \langle q, y, \gamma\beta \rangle \rangle$

Notation

$\rightarrow$ in CFGs, A $\rightarrow$ B etc.

$\rightarrow$ in PDA state transitions

$\rightarrow_M$ in PDA state transitions, PDA M

$\rightarrow^n$ in PDA state transitions, n steps

$\rightarrow^*$ in PDA state transitions, any number of steps

$\Rightarrow$ derivation of a string in CFG

$\Rightarrow^*$ derivation of a string in CFG, n steps

$\Rightarrow_{lm}$ leftmost derivation of a string in CFG

Let's start!

LINKÖPING UNIVERSITY

Today's topic

- In this lecture, we will show that context free grammars and pushdown automata *have the same power*.
  - Equivalence between CFG and PDA.

- Comment on DPDAs if there is time.
  - We will return to PDAs later when we compare different classes of languages.

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, 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.
- The importance doesn't lie in the proofs but in the existence of the conversions and their consequences.

CFG to PDA

- 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,
- e.g. every rule in P has the form $A \rightarrow cB_1B_2...B_n$ (where $c \in \Sigma$, $B_i \in N$, $n \geqslant 0$).
- Now, create a PDA $M = (\{q\}, \Sigma, N, \delta, q, S, \varnothing\rangle$, where
  - There is just one state.
  - 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 $\langle\langle q, c, A\rangle, \langle q, B_1, B_2...B_n\rangle\rangle$

Example 1

- A grammar on Greibach normal form for arithmetic expressions with start symbol E,

  E → aX | bX | aY | bY | aYX | bYX | a | b

  T → aY | bY | a | b

  X → +E

  Y → *T

- If we construct a PDA as above 5 of the 14 elements in δ will be:
  - ⟨⟨q, a, E⟩, ⟨q, Y X⟩⟩,
  - ⟨⟨q, *, Y ⟩, ⟨q, T ⟩⟩,
  - ⟨⟨q, +, X⟩, ⟨q, E⟩⟩,
  - ⟨⟨q, b, T ⟩, ⟨q, ε⟩⟩,
  - ⟨⟨q, a, E⟩, ⟨q, ε⟩⟩.

Example 1, continued

$E \to aX \mid bX \mid aY \mid bY \mid aYX \mid bYX \mid a \mid b$

$T \to aY \mid bY \mid a \mid b$

$X \to +E$

$Y \to *T$

- Performing the following derivation:
  - $E \Rightarrow aYX \Rightarrow a * T\,X \Rightarrow a * bX \Rightarrow a * b + E \Rightarrow a * b + a$
- Corresponds to the following next-configuration steps using the PDA:
  - $\langle q, a * b + a, E \rangle \to$
  - $\langle q, * b + a, Y\,X \rangle \to$
  - $\langle q, b + a, T\,X \rangle \to$
  - $\langle q, + a, X \rangle \to$
  - $\langle q, a, E \rangle \to$
  - $\langle q, \varepsilon, \varepsilon \rangle$
  - and the string is accepted with empty stack!

Theorem 1

- **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 outlined earlier.
  - Then $A \ {}_n\Rightarrow_{lm} z\gamma \Leftrightarrow \langle q, zy, A\rangle \ {}_n\rightarrow \langle q, y, \gamma\rangle$.
    - The grammar is on Greibach normal form:
      - Each derivation step and each next-config. step handles one non-terminal
      - Can formally be proved by induction over the number of steps!
  - From this it follows $S \ {}_*\Rightarrow_{lm} x \Leftrightarrow \langle q, x, S\rangle \ {}_*\rightarrow \langle q, \varepsilon, \varepsilon\rangle$,
    - i.e. if a string can be derived by the CFG it is accepted by the PDA.

Theorem 1, notes

- 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.
  - For every nonterminal a: let $\langle\langle q, a, a\rangle, \langle q, \varepsilon\rangle\rangle$ be an element in δ.

PDA to CFG

- In this section we prove that given a PDA we can construct a CFG that describes the language of the PDA.
  - First we transform a one-state PDA to a grammar
  - Then we will convert a many-state one to the one-state form.
- 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 $\langle\langle q, c, A\rangle, \langle q, B_1B_2...B_n\rangle\rangle \in \delta$
    - Let the grammar have the rule $A \rightarrow cB_1B_2...B_n$.
    - Let the start stack symbol of the PDA be the start symbol of the CFG.

PDA to CFG: General case

- Let the starting point be the PDA $M = \langle Q, \Sigma, \Gamma, \delta, s, \bot, \{t\} \rangle$ 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.
- 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 $\langle p, x, A \rangle \ {}_n\!\!\rightarrow_M \langle q, \varepsilon, \varepsilon \rangle$ we will with our new M' have $\langle *, x, [pAq] \rangle \ {}_n\!\!\rightarrow_{M'} \langle *, \varepsilon, \varepsilon \rangle$.
  - 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 $M' = \langle \{*\}, \Sigma, \Gamma', \delta', *, [s\bot t], \varnothing \rangle$, 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 $\langle\langle p, c, A\rangle, \langle q_0, B_1B_2...B_n\rangle\rangle \in \delta$ we will have $\langle\langle *, c, [pAq_n]\rangle, \langle *, [pBq_1][q_1Bq_2]...[q_{n-1}Bq_n]\rangle \in \delta'$ for all $q_1, q_2,...q_n \in Q$.
  - Yes, there should be an element in δ' for every possible combination of n states in M.

PDA to CFG: General case, cont.

- **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 $\langle p, x, B_1B_2...B_n\rangle \ {}_n\!\rightarrow_M \langle q_n, \varepsilon, \varepsilon\rangle \Leftrightarrow \langle *, x, [pB_1q_1][q_1B_2q_2] \ldots [q_{n-1}pB_1q_n]\rangle \ {}_n\!\rightarrow_{M'} \langle *, \varepsilon, \varepsilon\rangle$
  - 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 $\langle s, x, \perp\rangle \ {}_*\!\rightarrow_M \langle t, \varepsilon, \varepsilon\rangle \Leftrightarrow \langle *, x, [s\perp t]\rangle \ {}_*\!\rightarrow_{M'} \langle *, \varepsilon, \varepsilon\rangle$,
  - 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 of 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.

Deterministic PDA

- 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 we, informally, allow the transition to be a partial function.
- **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.
  - *Prefix property*: A string in the accepted language cannot be a proper prefix of another string in the language.

To think about

- 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?
- Assume that we write a computer program which simulates a PDA (using a brute force approach which explores all possible transitions). Now we 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?

Coming up soon …

- The next week:
  - **Wednesday**: Closure properties and pumping lemma for CFLs

- The week after next:
  - **Monday**: LL(1) and LR(0) parsing
  - **Friday**: LR(1) parsing (Note: Will likely be moved here!)

Thanks for today!

LINKÖPING UNIVERSITY