Today's lecture: Non-Deterministic Pushdown Automata

Reading:

Today: Non-Deterministic Pushdown Automata
Read: *Kozen* lecture 23 or *Sipser* 2.2.

October 17: From CFLs to PDAs
Read: *Kozen* lecture 24 (or *Sipser* 2.2).

October 19: From PDAs to CFLs
Read: *Kozen* lecture 25 (or *Sipser* 2.2).

October 21: Deterministic PDAs
Read: *Kozen* lectures E and F.

October 24: Parsing
Read: *Kozen* lecture 26 (not in *Sipser*)

October 26: Midterm: in class.

October 28: A Parsing Algorithm
Read: *Kozen* lecture 27 (not in *Sipser*)

October 31: LALR Parsing
Read: TBD

I. Non-Deterministic Pushdown Automata

A. The key ideas

1. A NFA augmented with a stack (i.e. a Last-In, First-Out store)

2. What ingredients do we need?

a. The usual stuff for an NFA:

i. A set of states: Q.

ii. An input alphabet: Σ.

iii. An initial state, $q_0 \in Q$.

Note that we previously defined NFAs to have a set of initial states Q_0. It is straightforward to show that an NFA with a single initial state can simulate an NFA with multiple possible initial states –

Thus, the restriction of a single starting state doesn't change anything. In just a moment, we'll define the stack to have a unique initial configuration. For simplicity, we do the same with the NFA.

iv. A set of accepting states $F \subseteq Q$.

Note that a single accepting state would work just as well. This time, we're staying closer to the traditional NFA.

b. The stuff for a stack:

i. We need to be able to push things onto the stack and pop things off. Thus, we need a stack alphabet: Γ.

ii. It's convenient to know when we're down to the last symbol on the stack. So, we'll assume there is a special symbol: $\perp \in \Gamma$.

The stack will initially hold $\perp$ and nothing else.

c. A transition relation to tie all of the pieces together: δ.

i. Updates the state and stack according to the current state, stack contents, and the current input symbol.

ii. Sufficient just to consider top-of-stack symbol.

How would you show that this is equivalent to being able to consider the top k symbols for any fixed k?

iii. Needs to be able to replace the top-of-stack symbol with zero or more symbols.

Can't just be one, or the number of symbols on the stack would never change. Being able to replace the top-of-stack symbol with 0, 1, or 2 symbols is sufficient.

How would you show this?

We allow arbitrary pre-defined strings of symbols to be pushed onto the stack to avoid adding a special restriction.

iv. This means that the transition relation reads the current state, the top-of-stack-symbol, and the current input character. It removes the top-of-stack symbol, moves the NFA to a new state, and pushes a string of zero or more symbols onto the stack.

v. Formally: $\delta : (Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma) \times (Q \times \Gamma^*)$

B. The formal definition

1. A NPDA is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, s, \perp, F)$, where each of the components are as defined above.

2. Acceptance condition(s)

a. Final state: the machine accepts if the machine can reach an accepting state of the NFA after having read the entire input.

b. Empty stack: the machine accepts if the machine can reach a configuration where the stack is empty after having read the entire input.

c. The two acceptance conditions give rise to the same set of languages. Next week, we will show (of course) that this is exactly the CFLs.

C. Configurations: How do we describe the progress of a NPDA while reading an input string?

1. With a tuple, of course.

2. In this case, a 3-tuple: (p, x, u), where

$$\begin{array}{lcll} p & \in & Q, & \text{the current state of the NFA} \\ x & \in & \Sigma^*, & \text{the remaining input to be read} \\ u & \in & \Gamma^*, & \text{the string of symbols on the stack, top-of-stack first} \end{array}$$

II. Examples

A. Expressions

1. A grammar for expressions

$$
\begin{array}{rcl}
G & = & (N, \Sigma, P, S) \\
N & = & \{\textit{expr}, \textit{term}, \textit{factor}\} \\
\Sigma & = & \{\textbf{ID}, \textbf{CONST}, +, -, *, /, \textbf{[}, \textbf{]}\} \\
P & = & \{ \quad \begin{array}{rcll}
expr & \rightarrow & & \textit{term} \\
 & & | & \textit{expr}+\textit{term} \\
 & & | & \textit{expr}-\textit{term} \\
term & \rightarrow & & \textit{factor} \\
 & & | & \textit{term}*\textit{factor} \\
 & & | & \textit{term}/\textit{factor} \\
factor & \rightarrow & & \textbf{ID} \\
 & & | & \textbf{CONST} \\
 & & | & -factor \\
 & & | & \textbf{[}\textit{expr}\textbf{]}
\end{array} \\
 & & \} \\
S & = & \textit{expr}
\end{array}
$$

2. A NPDA for this grammar

3. Parsing an expression

B. $x | \neg(\exists w.\ x = ww)$