CS350 - Equivalence of PDAs and CFLs

Objectives: In this activity you will learn how PDAs and CFLs are equivalent

To see that PDAs and CFLs are equivalent we must show that every CFG must have an equivalent PDA and that every PDA must have an equivalent CFG.

We will start by taking a CFG and construct a PDA to recognize the same language.

Download the files ex10.1 and ex10.2 and start JFLAP. Open the file ex10.1 containing a CFG in the grammar editor. Select under the Convert menu `Transform CFG to PDA (LL)`. You will be supplied with a basic PDA with three states. It starts by pushing the start symbol of the grammar onto the stack. In the second state it simply matches an input symbol with the symbol at the top of the stack. It then non-deterministically transitions on an empty stack to an accepting state.
The goal for an input string is the following: If $q2$ is reached, then all the symbols in the input string were pushed on the stack using the productions in the grammar and popped off the stack in the same order they appear in the input string. We have already added the "popped off" loop transitions on $q1$.
Select a production rule and push the `Create Selected` button. This will add an additional transition to the PDA. Continue with the process until the PDA is complete.
Test out both the original CFG and the constructed PDA until you assure yourself that they do indeed accept the same language.

Explain in your own words how the additional PDA transitions are added and why this creates an equivalent PDA.

We must now show that every NPDA must have an equivalent CFG.

The idea behind the conversion from an NPDA to an equivalent CFG is to convert each transition into one or more productions that mimic the behavior. To simplify the process, JFLAP will require that the NPDA be in a certain format. If it is not, the user must convert it to the appropriate format. All transitions must pop exactly one symbol and push exactly zero or two symbols. In other words, with each transition the stack will increase or decrease its size by one. The NPDA must have one final state and least one transition into the final state that pops Z off the stack.

For those transitions that pop one symbol and push no symbols, one production is generated to mimic this behavior. For those transitions that pop one symbol and push two symbols, a lot of productions are generated to indicate possible ways these two symbols that are pushed could eventually be popped off the stack. Many useless productions will be generated in addition to the correct productions.

Open file ex10.2 in the PDA editor. We want a CFG which recognizes this language. Select `Convert to grammar`. Oops! You get an error message because the NPDA is not in the correct format. Correct this by adding another state and replace the transition with

two: One that goes from $q1$ to the new state and pushes two symbols `xb` where `x` is just an added symbol: `(b,a`xb);. Then another which goes from the new state to $q2$ and pops the `x`: (λ,x;λ).

Try the conversion again with the fixed NPDA. We are now ready to convert to a CFG. Variables in the grammar will be represented in the form (q_iA, q_j) where q_i and q_j represent states from the NPDA and A is a stack symbol. The meaning of this variable is "if when moving along a path from state q_i to state q_j, the stack is exactly the same except that A is removed from the stack". The start variable S will be represented by our goal (q_0Zq_4);

The algorithm is kind of messy but basically, for each transition from `(qi,a,A)` to q_j which pops the A, we generate one production $(q_i, A, q_j) \rightarrow a$
For each transition of the form `(qi,a,A)` to q_j which pops the A and pushes BC, we generate the productions $(q_iAq_k) \rightarrow a(q_jBq_l)(q_lCq_k)$ for all q_k and q_l.

This should create a grammar that starts with the goal and works backwards to generate the input. Test it out with the Hints from JFLAP. Then export the grammar and test both the original NPDA and the grammar to assure yourself that they are equivalent.

Explain in your own words why such a generated grammar would give the same language as the PDA.

Since we are able to convert a CFG to an NPDA and conversely, convert anNPDA to a CFG, both represent the context-free languages!