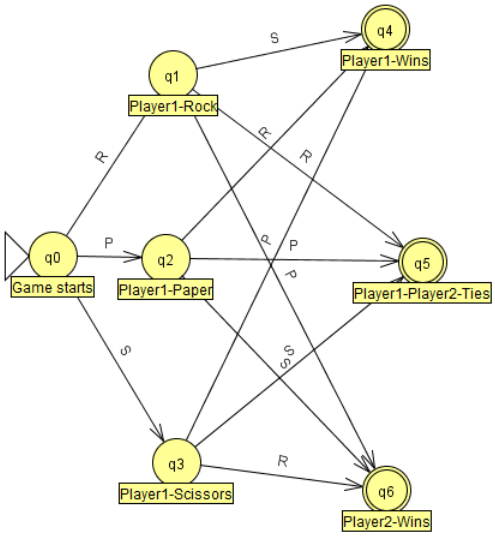
Converting DFA to Regular Grammar

Pre-requisite knowledge: deterministic finite automata, non-deterministic finite automata, regular expressions, regular languages, and regular grammars.

A DFA may be easily converted into a regular grammar. Let us consider the following DFA which models one round of the classic game Rock-Paper-Scissors. The DFA accepts two-lettered inputs where the first letter is Player 1's input and the second letter is Player 2's input. The input alphabet is {R, P, S} representing rock, paper, and scissors. For example, RP represents rock for Player 1 and paper for Player 2; this string ends in a final state in which Player 2 wins since rock crushes scissors.



Try It! Open the file RockPaperScissors_DFA.jflap in JFLAP. Using *Input* > *Multiple Run*, run the valid and invalid inputs: PP, SP, RS, RSP, RSSP, and RT.

Converting a DFA, defined by M = (Q, Σ, δ, q_0, F), to a regular grammar, defined by G = (V, T, S, P) is straight forward. The rules are summarized below:

1. The start symbol of the grammar is q_0, the non-terminal corresponding to the start state of the DFA.
2. For each transition from state q_i to state q_j on some symbol 'a', create a production rule of the form: $q_i \rightarrow a\ q_j$.
3. For each state q_i of the DFA which is a final state, create a production rule of the form: $q_i \rightarrow \lambda$.

Next, we work on converting the DFA into a grammar. Using the DFA above, we convert each transition (between two states) into a production rule for the resulting regular grammar. Since there are twelve transitions, we end up with twelve production rules:

$q_0 \rightarrow R\ q_1 \mid P\ q_2 \mid S\ q_3$,

$q_1 \rightarrow S\ q_4 \mid R\ q_5 \mid P\ q_6$,

$q_2 \rightarrow R\ q_4 \mid P\ q_5 \mid S\ q_6$,

$q_3 \rightarrow P\ q_4 \mid S\ q_5 \mid R\ q_6$.

To add the transitions to the three final states, we also include the following transitions:

$q_4 \rightarrow \lambda$,

$q_5 \rightarrow \lambda$,

$q_6 \rightarrow \lambda$.

To complete the regular grammar definition, G = (V, T, S, P), we have

$V = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}$,

$T = \{R, P, S\}$

$S = q_0$,

$P = \{q_0 \rightarrow R\ q_1 \mid P\ q_2 \mid S\ q_3, q_1 \rightarrow S\ q_4 \mid R\ q_5 \mid P\ q_6, q_2 \rightarrow R\ q_4 \mid P\ q_5 \mid S\ q_6, q_3 \rightarrow P\ q_4 \mid S\ q_5 \mid R\ q_6, q_4 \rightarrow \lambda, q_5 \rightarrow \lambda, q_6 \rightarrow \lambda\}$.

Questions to think about:

1. Does the grammar accept PR?

   Answer: Yes the grammar accepts PR using the derivation $q_0 \Rightarrow P\ q_2 \Rightarrow PR\ q_4 \Rightarrow PR$.

2. Does this grammar accept PRP?

   Answer: No, the grammar does not accept PRP.

3. Is the resulting grammar a regular grammar?

   Answer: Recall that a regular grammar is either right- or left-linear and since the resulting grammar is right-linear, the grammar is a regular grammar.

4. Many of you may have seen the episodes of the television series “Big Bang Theory” where they mention an extension to the Rock-Paper-Scissors game called Rock-Paper-Scissors-Lizard-Spock (http://en.wikipedia.org/wiki/Rock-paper-scissors-lizard-Spock). Create a DFA for this expanded game. Convert the DFA to a regular grammar as we did in this module.

Try it! Open the file RockPaperScissors_DFA.jflap if it is not loaded in JFLAP. Convert the DFA to a grammar by selecting *Convert* > *Convert to Grammar*. You may perform the conversion one step at a time. Click *Step* and the first grammar rule is generated on the right-hand pane, (q3) -> R(q6). Repeat the process a few more times paying close attention to each new rule generated. Finish the conversion

by clicking *Step to Completion*. The resulting grammar should be similar to the one that was developed earlier with each variable surrounded by ( ).

Questions to think about

1. Can all DFAs be converted to a regular grammar (right-linear) with the given algorithm?

   Answer: Yes. Using the algorithm, any DFA may be converted to a regular grammar. Every DFA has exactly one start state; this translates to the start variable for the grammar. Each transition in the DFA becomes one production rule in the grammar. A DFA must have at least one final state which allows for the derivation to terminate.

Reference:

Peter Linz, "An Introduction to Formal Languages and Automata" 5th edition, Jones and Bartlett, 2011.