4.8: Converting Regular Expressions and FA to Grammars

In this section, we give simple algorithms for converting regular expressions and finite automata to grammars.

Since we have algorithms for converting between regular expressions and finite automata, it is tempting to only define one of these algorithms. But better results can be obtained by defining direct conversions.

Converting Regular Expressions to Grammars

Regular expressions are converted to grammars using a recursive algorithm that makes use of some of the operations on grammars that were defined in Section 4.7.

Converting Regular Expressions to Grammars

Regular expressions are converted to grammars using a recursive algorithm that makes use of some of the operations on grammars that were defined in Section 4.7.

The structure of the algorithm is very similar to the structure of our algorithm for converting regular expressions to finite automata. This gives us a function **regToGram** $\in$ **Reg** $\rightarrow$ **Gram**.

The algorithm is implemented in Forlan by the function

```
val fromReg : reg -> gram
```

of the `Gram` module. It's available in the top-level environment with the name `regToGram`.

Converting Regular Expressions to Grammars

Here is how we can convert the regular expression $01 + 10(11)^*$ to a grammar using Forlan:

```
- val gram = regToGram(Reg.input "");
@ 01 + 10(11)*
@ .
val gram = - : gram
- Gram.output
= ("", Gram.renameVariablesCanonically gram);
{variables} A, B, C, D, E, F {start variable} A
{productions}
A -> B | C; B -> 01; C -> DE; D -> 10; E -> % | FE;
F -> 11
val it = () : unit
```

Converting Finite Automata to Grammars

Suppose M is an FA. We define a function/algorithm **faToGram** $\in$ **FA** $\to$ **Gram** by, for all FAs M, **faToGram** M is the grammar G defined below. If $Q_M \cap$ **alphabet** $M = \emptyset$, then G is defined by

- $Q_G = Q_M$;
- $s_G = s_M$;
- $P_G = \{ \qquad\qquad | \, q, x \to r \in T_M \} \cup \qquad\qquad$.

Otherwise, we first rename the states of M using a uniform number of $\langle$ and $\rangle$ pairs, so as to avoid conflicts with the elements of M's alphabet.

Converting Finite Automata to Grammars

Suppose M is an FA. We define a function/algorithm $\textbf{faToGram} \in \textbf{FA} \to \textbf{Gram}$ by, for all FAs M, $\textbf{faToGram}\, M$ is the grammar G defined below. If $Q_M \cap \textbf{alphabet}\, M = \emptyset$, then G is defined by

- $Q_G = Q_M$;
- $s_G = s_M$;
- $P_G = \{\, q \to xr \mid q, x \to r \in T_M \,\} \cup$.

Otherwise, we first rename the states of M using a uniform number of $\langle$ and $\rangle$ pairs, so as to avoid conflicts with the elements of M's alphabet.

Converting Finite Automata to Grammars

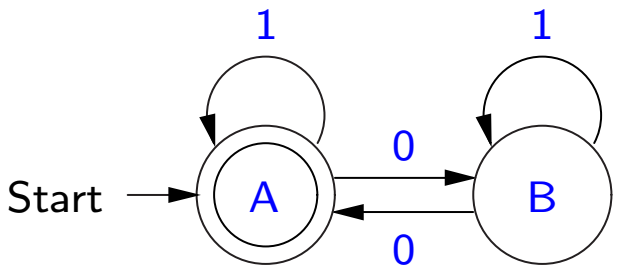
Suppose M is an FA. We define a function/algorithm **faToGram** $\in$ **FA** $\to$ **Gram** by, for all FAs M, **faToGram** M is the grammar G defined below. If $Q_M \cap$ **alphabet** $M = \emptyset$, then G is defined by

- $Q_G = Q_M$;
- $s_G = s_M$;
- $P_G = \{\, q \to xr \mid q, x \to r \in T_M \,\} \cup \{\, q \to \% \mid q \in A_M \,\}$.

Otherwise, we first rename the states of M using a uniform number of $\langle$ and $\rangle$ pairs, so as to avoid conflicts with the elements of M's alphabet.

Converting Finite Automata to Grammars

For example, suppose M is the DFA



Our algorithm converts M into the grammar

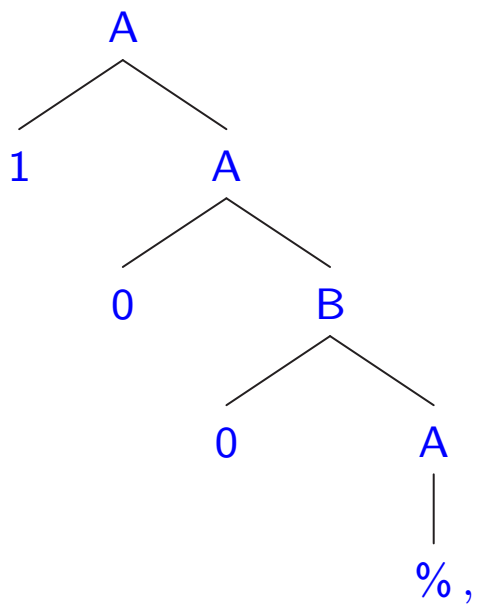
$$A \to \% \mid 0B \mid 1A,$$
$$B \to 0A \mid 1B.$$

Converting FAs to Grammars

Consider, e.g., the valid labeled path for M

$$\mathsf{A} \overset{1}{\Rightarrow} \mathsf{A} \overset{0}{\Rightarrow} \mathsf{B} \overset{0}{\Rightarrow} \mathsf{A},$$

which explains why $100 \in L(M)$. It corresponds to the valid parse tree for G



which explains why $100 \in L(G)$.

Converting FA's to Grammars

If we have converted an FA M to a grammar G, we can prove $L(M) \subseteq L(G)$ by induction on the lengths of labeled paths, and we can prove $L(G) \subseteq L(M)$ by induction on parse trees. Thus we have $L(G) = L(M)$.

Converting FA's to Grammars

The Forlan module `Gram` contains the function

```
val fromFA : fa -> gram
```

which implements our algorithm for converting finite automata to grammars. It's available in the top-level environment with the name `faToGram`.

Converting FA's to Grammars

Suppose `fa` of type `fa` is bound to M. Here is how we can convert M to a grammar using Forlan:

```
- val gram = faToGram fa;
val gram = - : gram
- Gram.output("", gram);
{variables} A, B {start variable} A
{productions} A -> % | 0B | 1A; B -> 0A | 1B
val it = () : unit
```

Consequences of Conversion Functions

Because of the existence of our conversion functions, we have that every regular language is a context-free language.

Consequences of Conversion Functions

Because of the existence of our conversion functions, we have that every regular language is a context-free language.

On the other hand, the language $\{ 0^n 1^n \mid n \in \mathbb{N} \}$ is context-free, because of the grammar

$$A \to \% \mid 0A1,$$

but is not regular, as we proved in Section 3.14.

Consequences of Conversion Functions

Because of the existence of our conversion functions, we have that every regular language is a context-free language.

On the other hand, the language $\{ 0^n 1^n \mid n \in \mathbb{N} \}$ is context-free, because of the grammar

$$A \to \% \mid 0A1,$$

but is not regular, as we proved in Section 3.14.

Summarizing, we have:

Theorem 4.8.2
The regular languages are a proper subset of the context-free languages: **RegLan** $\subsetneq$ **CFLan**.