4.7: Closure Properties of Context-free Languages

In this section, we define union, concatenation and closure operations/algorithms on grammars. As a result, we will have that the context-free languages are closed under union, concatenation and closure.

In Section 4.10, we will see that the context-free languages aren't closed under intersection, complementation and set difference.

But we are able to define operations/algorithms for:

- intersecting a grammar and an empty-string finite automaton; and
- subtracting a deterministic finite automaton from a grammar.

Thus, if L_1 is a context-free language, and L_2 is a regular language, we will have that $L_1 \cap L_2$ and $L_1 - L_2$ are context-free.

The book shows several additional closure properties of context-free languages, in addition to giving the corresponding operations on grammars.

Basic Grammars and Operations on Grammars

The grammar, **emptyStr**, with variable A and production $\mathsf{A} \rightarrow \%$ generates the language $\{\%\}$.

The grammar, **emptySet**, with variable A and no productions generates the language $\emptyset$.

If w is a string, then the grammar with variable A and production $\mathsf{A} \rightarrow w$ generates the language $\{w\}$. Actually, we must be careful to chose a variable that doesn't occur in w. We can do that by adding as many nested `<` and `>` around `A` as needed (`A`, `<A>`, `<<A>>`, etc.).

This defines functions **strToGram** $\in$ **Str** $\rightarrow$ **Gram** and **symToGram** $\in$ **Sym** $\rightarrow$ **Gram**.

Union Operation

Suppose G_1 and G_2 are grammars. We can define a grammar H such that $L(H) = L(G_1) \cup L(G_2)$ by unioning together the variables and productions of G_1 and G_2, and adding a new start variable q, along with productions

$$q \to s_{G_1} \mid s_{G_2}.$$

What do we have to know about G_1, G_2 and q for the above to be valid?

- $Q_{G_1} \cap Q_{G_2} = \emptyset$ and $q \notin Q_{G_1} \cup Q_{G_2}$; and
- $\textbf{alphabet}\, G_1 \cap Q_{G_2} = \emptyset$, $\textbf{alphabet}\, G_2 \cap Q_{G_1} = \emptyset$ and $q \notin \textbf{alphabet}\, G_1 \cup \textbf{alphabet}\, G_2$.

Union Operation

Our official union operation for grammars renames the variables of G_1 and G_2, and chooses the start variable q, in a uniform way that makes the preceding properties hold.

This gives us a function $\textbf{union} \in \textbf{Gram} \times \textbf{Gram} \to \textbf{Gram}$.

We do something similar when defining the other closure operations. In what follows, though, we'll ignore this issue, so as to keep things simple.

Concatenation and Closure Operations

Suppose G_1 and G_2 are grammars. We can define a grammar H such that $L(H) = L(G_1)L(G_2)$ by unioning together the variables and productions of G_1 and G_2, and adding a new start variable q, along with production

$$q \rightarrow s_{G_1} s_{G_2}.$$

This gives us a function **concat** $\in$ **Gram** $\times$ **Gram** $\rightarrow$ **Gram**.

Suppose G is a grammar. We can define a grammar H such that $L(H) = L(G)^*$ by adding to the variables and productions of G a new start variable q, along with productions

$$q \rightarrow \% \mid s_G q.$$

This gives us a function **closure** $\in$ **Gram** $\rightarrow$ **Gram**.

Intersection of Grammar and EFA

We now consider an algorithm for intersecting a grammar G with an EFA M, resulting in **simplify** H, where the grammar H is defined as follows.

For all $p \in Q_G$ and $q, r \in Q_M$, H has a variable $\langle p, q, r \rangle$ that generates

$$\{\, w \in (\textbf{alphabet}\, G)^* \mid w \in \Pi_{G,p} \text{ and } r \in \Delta_M(\{q\}, w) \,\}.$$

The remaining variable of H is A, which is its start variable. For each $r \in A_M$, H has a production

$$\mathsf{A} \to \langle s_G, s_M, r \rangle.$$

For each $\%$-production $p \to \%$ of G and $q, r \in Q_M$, if $r \in \Delta_M(\{q\}, \%)$, then H will have the production

$$\langle p, q, r \rangle \to \%.$$

Intersection of Grammar and EFA

To say what the remaining productions of H are, define a function

$$f \in (\textbf{alphabet}\, G \cup Q_G) \times Q_M \times Q_M \to \textbf{alphabet}\, G \cup Q_H$$

by: for all $a \in \textbf{alphabet}\, G \cup Q_G$ and $q, r \in Q_M$,

$$f(a, q, r) = \begin{cases} a, & \text{if } a \in \textbf{alphabet}\, G, \text{ and} \\ \langle a, q, r \rangle, & \text{if } a \in Q_G. \end{cases}$$

For all $p \in Q_G$, $n \in \mathbb{N} - \{0\}$, $a_1, \ldots, a_n \in \textbf{Sym}$ and $q_1, \ldots, q_{n+1} \in Q_M$, if

- $p \to a_1 \cdots a_n \in P_G$, and
- for all $i \in [1 : n]$, if $a_i \in \textbf{alphabet}\, G$, then $q_{i+1} \in \Delta_M(\{q_i\}, a_i)$,

then we let

$$\langle p, q_1, q_{n+1} \rangle \to f(a_1, q_1, q_2) \cdots f(a_n, q_n, q_{n+1})$$

be production of H.

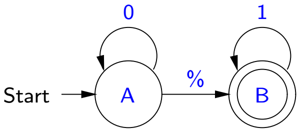
Intersection of Grammar and EFA

This gives us a function **inter** $\in$ **Gram** $\times$ **EFA** $\to$ **Gram**.

For example, let G be the grammar

$$\mathsf{A} \to \% \mid \mathsf{0A1A} \mid \mathsf{1A0A},$$

and M be the EFA



so that G generates all elements of $\{0, 1\}^*$ with an equal number of 0's and 1's, and M accepts $\{0\}^*\{1\}^*$.

Intersection of Grammar and EFA

Then **simplify** H is

$$\begin{aligned}
\mathsf{A} &\to \langle \mathsf{A},\mathsf{A},\mathsf{B}\rangle,\\
\langle \mathsf{A},\mathsf{A},\mathsf{A}\rangle &\to \%,\\
\langle \mathsf{A},\mathsf{A},\mathsf{B}\rangle &\to \%,\\
\langle \mathsf{A},\mathsf{A},\mathsf{B}\rangle &\to 0\langle \mathsf{A},\mathsf{A},\mathsf{A}\rangle 1\langle \mathsf{A},\mathsf{B},\mathsf{B}\rangle,\\
\langle \mathsf{A},\mathsf{A},\mathsf{B}\rangle &\to 0\langle \mathsf{A},\mathsf{A},\mathsf{B}\rangle 1\langle \mathsf{A},\mathsf{B},\mathsf{B}\rangle,\\
\langle \mathsf{A},\mathsf{A},\mathsf{B}\rangle &\to 0\langle \mathsf{A},\mathsf{B},\mathsf{B}\rangle 1\langle \mathsf{A},\mathsf{B},\mathsf{B}\rangle,\\
\langle \mathsf{A},\mathsf{B},\mathsf{B}\rangle &\to \%.
\end{aligned}$$

Note that simplification eliminated the variable $\langle \mathsf{A},\mathsf{B},\mathsf{A}\rangle$. If we hand simplify further, we can turn this into:

$$\begin{aligned}
\mathsf{A} &\to \langle \mathsf{A},\mathsf{A},\mathsf{B}\rangle,\\
\langle \mathsf{A},\mathsf{A},\mathsf{B}\rangle &\to \% \mid 0\langle \mathsf{A},\mathsf{A},\mathsf{B}\rangle 1
\end{aligned}$$

Intersection of Grammar and EFA

To prove that our intersection algorithm is correct, we'll need two lemmas.

Lemma 4.7.1
For $p \in Q_G$, let the property $P_p(w)$, for $w \in \Pi_{G,p}$, be:

for all $q, r \in Q_M$, if $r \in \Delta_M(\{q\}, w)$, then $w \in \Pi_{H,\langle p,q,r\rangle}$.

Then, for all $p \in Q_G$, for all $w \in \Pi_{G,p}$, $P_p(w)$.

Proof. By induction on Π. In (2), we use the fact that, if $n \in \mathbb{N} - \{0\}$, $q_1, q_{n+1} \in Q_M$, $w_1, \ldots, w_n \in \mathbf{Str}$ and $q_{n+1} \in \Delta_M(\{q_1\}, w_1 \cdots w_n)$, then there are $q_2, \ldots, q_n \in Q_M$ such that $q_{i+1} \in \Delta_M(\{q_i\}, w_i)$, for all $i \in [1 : n]$. (This is true because M is an EFA; if M were an FA, we wouldn't be able to conclude this.) $\square$

Lemma 4.7.2

Let the property $P_{\mathrm{A}}(w)$, *for* $w \in \Pi_{H,\mathrm{A}}$, *be*

$$w \in L(G) \text{ and } w \in L(M).$$

For $p \in Q_G$ *and* $q, r \in Q_M$, *let the property* $P_{\langle p,q,r \rangle}(w)$, *for* $w \in \Pi_{H,\langle p,q,r \rangle}$, *be*

$$w \in \Pi_{G,p} \text{ and } r \in \Delta_M(\{q\}, w).$$

Then:

(1) *For all* $w \in \Pi_{H,\mathrm{A}}$, $P_{\mathrm{A}}(w)$.

(2) *For all* $p \in Q_G$ *and* $q, r \in Q_M$, *for all* $w \in \Pi_{H,\langle p,q,r \rangle}$, $P_{\langle p,q,r \rangle}(w)$.

Proof. We proceed by induction on Π. □

Lemma 4.7.3
$L(H) = L(G) \cap L(M)$.

Proof. $L(H) \subseteq L(G) \cap L(M)$ follows by Lemma 4.7.2(1).

For the other inclusion, suppose $w \in L(G) \cap L(M)$, so that $w \in \Pi_{G,s_M}$ and $r \in \Delta_M(\{s_m\}, w)$, for some $r \in A_M$. By Lemma 4.7.1, it follows that $w \in \Pi_{H,\langle s_G, s_M, r\rangle}$. But because $r \in A_M$, we have that $\mathsf{A} \to \langle s_G, s_M, r\rangle$ is a production of H. Thus $w \in \Pi_{H,\mathsf{A}} = L(H)$. $\square$

Difference of Grammar and DFA

Given a grammar G and a DFA M, we can define the difference of G and M to be

$$\textbf{inter}(G, \textbf{complement}(M, \textbf{alphabet}\ G)).$$

This is analogous to what we did when defining the difference of DFAs.

This gives us a function $\textbf{minus} \in \textbf{Gram} \times \textbf{DFA} \to \textbf{Gram}$.

Summary of Closure Properties

Theorem 4.7.1

Suppose $L, L_1, L_2 \in$ **CFLan** *and* $L' \in$ **RegLan**. *Then:*

(1) $L_1 \cup L_2 \in$ **CFLan**;

(2) $L_1 L_2 \in$ **CFLan**;

(3) $L^* \in$ **CFLan**;

(4) $L \cap L' \in$ **CFLan**; *and*

(5) $L - L' \in$ **CFLan**.

Operations on Grammars in Forlan

The Forlan module `Gram` defines the following constants and operations on grammars:

```
val emptyStr   : gram
val emptySet   : gram
val fromStr    : str -> gram
val fromSym    : sym -> gram
val union      : gram * gram -> gram
val concat     : gram * gram -> gram
val closure    : gram -> gram
val fromStrSet : str set -> gram
val inter      : gram * efa -> gram
val minus      : gram * dfa -> gram
```

The functions `fromStr` and `fromSym` and are also available in the top-level environment with the names

```
val strToGram : str -> gram
val symToGram : sym -> gram
```

Forlan Examples

For example, we can construct a grammar G such that $L(G) = \{01\} \cup \{10\}\{11\}^*$, as follows.

```
- val gram1 = strToGram(Str.fromString "01");
val gram1 = - : gram
- val gram2 = strToGram(Str.fromString "10");
val gram2 = - : gram
- val gram3 = strToGram(Str.fromString "11");
val gram3 = - : gram
- val gram =
= Gram.union(gram1,
=            Gram.concat(gram2,
=                        Gram.closure gram3));
val gram = - : gram
```

Forlan Examples

```
- val gram' = Gram.renameVariablesCanonically gram;
val gram' = - : gram
- Gram.output("", 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
```

Forlan Examples

We can use `Gram.fromStrSet` as follows:

```
- val gram'' =
=   Gram.fromStrSet
=   (StrSet.fromString "0, 01, 010, 0101");
val gram'' = - : gram
- val gram''' = Gram.renameVariablesCanonically
gram'';
val gram''' = - : gram
- Gram.output("", gram''');
{variables} A, B, C, D, E {start variable} A
{productions}
A -> B | C | D | E; B -> 0; C -> 01; D -> 010;
E -> 0101
val it = () : unit
```

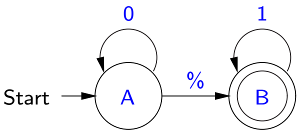
Forlan Examples

And here are examples of how we can use `Gram.inter` and `Gram.minus`.

Let `gram` be the grammar

$$\mathsf{A} \rightarrow \% \mid \mathsf{0A1A} \mid \mathsf{1A0A},$$

and `efa` be the EFA



Forlan Examples

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

Forlan Examples

```
- val gram'' =
=       Gram.eliminateVariable
=       (Gram.eliminateVariable
=        (gram', Sym.fromString "<A,A,A>"),
=        Sym.fromString "<A,B,B>");
val gram'' = - : gram
- val gram''' =
=       Gram.renameVariablesCanonically
=       (Gram.restart(Gram.simplify gram''));
val gram''' = - : gram
- Gram.output("", gram''');
{variables} A {start variable} A
{productions} A -> % | 0A1
val it = () : unit
```

Forlan Examples

```
- val dfa =
=       DFA.renameStatesCanonically
=       (DFA.minimize(nfaToDFA(efaToNFA efa)));
val dfa = - : dfa
- val gram'' = Gram.minus(gram, dfa);
val gram'' = - : gram
- Gram.generated gram'' (Str.fromString "0101");
val it = true : bool
- Gram.generated gram'' (Str.fromString "0011");
val it = false : bool
```