4.9: Chomsky Normal Form

In this section, we study a special form of grammars called Chomsky Normal Form (CNF), named for the linguist Noam Chomsky.

4.9: Chomsky Normal Form

In this section, we study a special form of grammars called Chomsky Normal Form (CNF), named for the linguist Noam Chomsky.

Grammars in CNF have very nice formal properties. In particular, valid parse trees for grammars in CNF are very close to being binary trees.

4.9: Chomsky Normal Form

In this section, we study a special form of grammars called Chomsky Normal Form (CNF), named for the linguist Noam Chomsky.

Grammars in CNF have very nice formal properties. In particular, valid parse trees for grammars in CNF are very close to being binary trees.

Any grammar that doesn't generate $\%$ can be put in CNF. And, if G is a grammar that does generate $\%$, it can be turned into a grammar in CNF that generates $L(G) - \{\%\}$. In the next section, we will use this fact when proving the pumping lemma for context-free languages, a method for showing the certain languages are not context-free.

4.9: Chomsky Normal Form

In this section, we study a special form of grammars called Chomsky Normal Form (CNF), named for the linguist Noam Chomsky.

Grammars in CNF have very nice formal properties. In particular, valid parse trees for grammars in CNF are very close to being binary trees.

Any grammar that doesn't generate $\%$ can be put in CNF. And, if G is a grammar that does generate $\%$, it can be turned into a grammar in CNF that generates $L(G) - \{\%\}$. In the next section, we will use this fact when proving the pumping lemma for context-free languages, a method for showing the certain languages are not context-free.

When converting a grammar to CNF, we will first eliminate productions of the form $q \rightarrow \%$ and $q \rightarrow r$.

Eliminating %-Productions

A %-*production* is a production of the form $q \to \%$. We will show by example how to turn a grammar G into a simplified grammar with no %-productions that generates $L(G) - \{\%\}$.

Eliminating %-Productions

A %-*production* is a production of the form $q \to \%$. We will show by example how to turn a grammar G into a simplified grammar with no %-productions that generates $L(G) - \{\%\}$.

Suppose G is the grammar

$$
\begin{aligned}
\mathsf{A} &\to \mathsf{0A1} \mid \mathsf{BB}, \\
\mathsf{B} &\to \% \mid \mathsf{2B}.
\end{aligned}
$$

First, we determine which variables q are *nullable* in the sense that they generate %.

Clearly,

Eliminating % *-Productions*

A %-*production* is a production of the form $q \to \%$. We will show by example how to turn a grammar G into a simplified grammar with no %-productions that generates $L(G) - \{\%\}$.

Suppose G is the grammar

$$\mathsf{A} \to \mathsf{0A1} \mid \mathsf{BB},$$
$$\mathsf{B} \to \% \mid \mathsf{2B}.$$

First, we determine which variables q are *nullable* in the sense that they generate %.

Clearly, B is nullable. And, since $\mathsf{A} \to \mathsf{BB} \in P_G$, it follows that A is nullable.

Eliminating %-Productions

Since A is nullable, we replace the production $A \to 0A1$ with the productions

Eliminating %-Productions

Since A is nullable, we replace the production $A \to 0A1$ with the productions $A \to 0A1$ and $A \to 01$. The idea is that this second production will make up for the fact that A won't be nullable in the new grammar.

Eliminating %-Productions

Since A is nullable, we replace the production $A \to 0A1$ with the productions $A \to 0A1$ and $A \to 01$. The idea is that this second production will make up for the fact that A won't be nullable in the new grammar.

Since B is nullable, we replace the production $A \to BB$ with the productions

Eliminating %-Productions

Since A is nullable, we replace the production $A \to 0A1$ with the productions $A \to 0A1$ and $A \to 01$. The idea is that this second production will make up for the fact that A won't be nullable in the new grammar.

Since B is nullable, we replace the production $A \to BB$ with the productions $A \to BB$ and $A \to B$ (the result of deleting either one of the B's).

Eliminating %-Productions

Since A is nullable, we replace the production $\mathsf{A} \to \mathsf{0A1}$ with the productions $\mathsf{A} \to \mathsf{0A1}$ and $\mathsf{A} \to \mathsf{01}$. The idea is that this second production will make up for the fact that A won't be nullable in the new grammar.

Since B is nullable, we replace the production $\mathsf{A} \to \mathsf{BB}$ with the productions $\mathsf{A} \to \mathsf{BB}$ and $\mathsf{A} \to \mathsf{B}$ (the result of deleting either one of the B's).

The production $\mathsf{B} \to \%$ is deleted.

Eliminating %-Productions

Since A is nullable, we replace the production $A \rightarrow 0A1$ with the productions $A \rightarrow 0A1$ and $A \rightarrow 01$. The idea is that this second production will make up for the fact that A won't be nullable in the new grammar.

Since B is nullable, we replace the production $A \rightarrow BB$ with the productions $A \rightarrow BB$ and $A \rightarrow B$ (the result of deleting either one of the B's).

The production $B \rightarrow \%$ is deleted.

Since B is nullable, we replace the production $B \rightarrow 2B$ with the productions

Eliminating %-Productions

Since A is nullable, we replace the production A → 0A1 with the productions A → 0A1 and A → 01. The idea is that this second production will make up for the fact that A won't be nullable in the new grammar.

Since B is nullable, we replace the production A → BB with the productions A → BB and A → B (the result of deleting either one of the B's).

The production B → % is deleted.

Since B is nullable, we replace the production B → 2B with the productions B → 2B and B → 2.

Eliminating % *-Productions*

Since A is nullable, we replace the production $A \to 0A1$ with the productions $A \to 0A1$ and $A \to 01$. The idea is that this second production will make up for the fact that A won't be nullable in the new grammar.

Since B is nullable, we replace the production $A \to BB$ with the productions $A \to BB$ and $A \to B$ (the result of deleting either one of the B's).

The production $B \to \%$ is deleted.

Since B is nullable, we replace the production $B \to 2B$ with the productions $B \to 2B$ and $B \to 2$.

(If a production has n occurrences of nullable variables in its right side, then there will be 2^n new right sides, corresponding to all ways of deleting or not deleting those n variable occurrences. But if a right side of % would result, we don't include it, and some may be duplicates.)

Eliminating %-Productions

This give us the grammar

$$\begin{aligned} \mathsf{A} &\rightarrow \mathsf{0A1} \mid \mathsf{01} \mid \mathsf{BB} \mid \mathsf{B}, \\ \mathsf{B} &\rightarrow \mathsf{2B} \mid \mathsf{2}. \end{aligned}$$

In general, we finish by simplifying our new grammar. The new grammar of our example is already simplified, however.

Eliminating Unit Productions

A *unit production* for a grammar G is a production of the form $q \rightarrow r$, where r is a variable (possibly equal to q). We now show by example how to turn a grammar G into a simplified grammar with no $\%$-productions or unit productions that generates $L(G) - \{\%\}$.

Eliminating Unit Productions

A *unit production* for a grammar G is a production of the form $q \to r$, where r is a variable (possibly equal to q). We now show by example how to turn a grammar G into a simplified grammar with no %-productions or unit productions that generates $L(G) - \{\%\}$.

Suppose G is the grammar

$$\begin{aligned} \mathsf{A} &\to \mathsf{0A1} \mid \mathsf{01} \mid \mathsf{BB} \mid \mathsf{B}, \\ \mathsf{B} &\to \mathsf{2B} \mid \mathsf{2}. \end{aligned}$$

We begin by applying our algorithm for eliminating %-productions to our grammar; the algorithm has no effect in this case.

Eliminating Unit Productions

Our new grammar will have the same variables and start variable as G. Its set of productions is the set of all $q \to w$ such that q is a variable of G, $w \in \textbf{Str}$ doesn't consist of a single variable of G, and there is a variable r such that

- r is parsable from q, and
- $r \to w$ is a production of G.

(Determining whether r is parsable from q is easy, since

Eliminating Unit Productions

Our new grammar will have the same variables and start variable as G. Its set of productions is the set of all $q \to w$ such that q is a variable of G, $w \in \mathbf{Str}$ doesn't consist of a single variable of G, and there is a variable r such that

- r is parsable from q, and
- $r \to w$ is a production of G.

(Determining whether r is parsable from q is easy, since we are working with a grammar with no %-productions.)

This process results in the grammar

$$\mathsf{A} \to \mathsf{0A1} \mid \mathsf{01} \mid \mathsf{BB} \mid \mathsf{2B} \mid \mathsf{2},$$
$$\mathsf{B} \to \mathsf{2B} \mid \mathsf{2}.$$

Finally, we simplify our grammar, which gets rid of the production

Eliminating Unit Productions

Our new grammar will have the same variables and start variable as G. Its set of productions is the set of all $q \to w$ such that q is a variable of G, $w \in \mathbf{Str}$ doesn't consist of a single variable of G, and there is a variable r such that

- r is parsable from q, and
- $r \to w$ is a production of G.

(Determining whether r is parsable from q is easy, since we are working with a grammar with no %-productions.)
This process results in the grammar

$$\mathsf{A} \to \mathsf{0A1} \mid \mathsf{01} \mid \mathsf{BB} \mid \mathsf{2B} \mid \mathsf{2},$$
$$\mathsf{B} \to \mathsf{2B} \mid \mathsf{2}.$$

Finally, we simplify our grammar, which gets rid of the production $\mathsf{A} \to \mathsf{2B}$.

Eliminating `%`-Productions and Unit Productions in Forlan

The Forlan module `Gram` defines the following functions:

```
val eliminateEmptyProductions        : gram -> gram
val eliminateEmptyAndUnitProductions : gram -> gram
```

For example, if `gram` is the grammar

$$\mathsf{A} \to \mathsf{0A1} \mid \mathsf{BB},$$
$$\mathsf{B} \to \% \mid \mathsf{2B}.$$

then we can proceed as follows.

Elimination in Forlan

```
- val gram' = Gram.eliminateEmptyProductions gram;
val gram' = - : gram
- Gram.output("", gram');
{variables} A, B {start variable} A
{productions} A -> B | 01 | BB | 0A1; B -> 2 | 2B
val it = () : unit
- val gram'' =
=       Gram.eliminateEmptyAndUnitProductions gram;
val gram'' = - : gram
- Gram.output("", gram'');
{variables} A, B {start variable} A
{productions} A -> 2 | 01 | BB | 0A1; B -> 2 | 2B
val it = () : unit
```

Generating a Grammar's Language When Finite

We can now give an algorithm that takes in a grammar G and generates $L(G)$, when it is finite, and reports that $L(G)$ is infinite, otherwise.

Generating a Grammar's Language When Finite

We can now give an algorithm that takes in a grammar G and generates $L(G)$, when it is finite, and reports that $L(G)$ is infinite, otherwise.

The algorithm begins by letting G' be the result of eliminating %-productions and unit productions from G. Thus G' is simplified and generates $L(G) - \{\%\}$.

Generating a Grammar's Language When Finite

We can now give an algorithm that takes in a grammar G and generates $L(G)$, when it is finite, and reports that $L(G)$ is infinite, otherwise.

The algorithm begins by letting G' be the result of eliminating $\%$-productions and unit productions from G. Thus G' is simplified and generates $L(G) - \{\%\}$.

If there is recursion in the productions of G'—either direct or mutual—then there is a variable q of G' and a valid parse tree pt for G', such that the height of pt is at least one, q is the root label of pt, and the yield of pt has the form xqy, for strings x and y, each of whose symbols is in **alphabet** $G' \cup Q_{G'}$. Because G' lacks $\%$- and unit-productions, it follows that $x \neq \%$ or $y \neq \%$.

Generating a Grammar's Language When Finite

We can now give an algorithm that takes in a grammar G and generates $L(G)$, when it is finite, and reports that $L(G)$ is infinite, otherwise.

The algorithm begins by letting G' be the result of eliminating %-productions and unit productions from G. Thus G' is simplified and generates $L(G) - \{\%\}$.

If there is recursion in the productions of G'—either direct or mutual—then there is a variable q of G' and a valid parse tree pt for G', such that the height of pt is at least one, q is the root label of pt, and the yield of pt has the form xqy, for strings x and y, each of whose symbols is in $\textbf{alphabet}\, G' \cup Q_{G'}$. Because G' lacks %- and unit-productions, it follows that $x \neq \%$ or $y \neq \%$.

Because each variable of G' is generating, we can turn pt into a valid parse tree pt' whose root label is q, and whose yield has the form uqv, for $u, v \in (\textbf{alphabet}\, G')^*$, where $u \neq \%$ or $v \neq \%$.

Generating a Grammar's Language When Finite

Thus we have that uqv is parsable from q in G', and an easy mathemtical induction shows that $u^n q v^n$ is parsable from q in G', for all $n \in \mathbb{N}$. Because $u \neq \%$ or $v \neq \%$, and q is generating, it follows that there are infinitely many strings that are generated from q in G'. And, since q is reachable, and every variable of G' is generating, it follows that $L(G')$, and thus $L(G)$, is infinite.

Generating a Grammar's Language When Finite

Thus we have that uqv is parsable from q in G', and an easy mathemtical induction shows that $u^n q v^n$ is parsable from q in G', for all $n \in \mathbb{N}$. Because $u \neq \%$ or $v \neq \%$, and q is generating, it follows that there are infinitely many strings that are generated from q in G'. And, since q is reachable, and every variable of G' is generating, it follows that $L(G')$, and thus $L(G)$, is infinite.

And when G' has no recursion in its productions, we can calculate $L(G')$ from the bottom-up, and add $\%$ iff G generates $\%$.

Generating a Grammar's Language in Forlan

The Forlan module `Gram` defines the following function:

```
val toStrSet : gram -> str set
```

Suppose `gram` is the grammar

$$
\begin{aligned}
&\mathsf{A} \to \mathsf{BB},\\
&\mathsf{B} \to \mathsf{CC},\\
&\mathsf{C} \to \% \mid \mathsf{0} \mid \mathsf{1},
\end{aligned}
$$

and `gram'` is the grammar

$$
\begin{aligned}
&\mathsf{A} \to \mathsf{BB},\\
&\mathsf{B} \to \mathsf{CC},\\
&\mathsf{C} \to \% \mid \mathsf{0} \mid \mathsf{1} \mid \mathsf{A}.
\end{aligned}
$$

Then we can proceed as follows.

Generating a Grammar's Language in Forlan

```
- StrSet.output("", Gram.toStrSet gram);
%, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101,
110, 111, 0000, 0001, 0010, 0011, 0100, 0101, 0110,
0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111
val it = () : unit
- StrSet.output("", Gram.toStrSet gram');
language is infinite

uncaught exception Error
```

Generating a Grammar's Language in Forlan

Suppose we have a grammar G and a natural number n. How can we generate the set of all elements of $L(G)$ of length n?

Generating a Grammar's Language in Forlan

Suppose we have a grammar G and a natural number n. How can we generate the set of all elements of $L(G)$ of length n?

Of course, we could generate all strings over the alphabet of G of length n, and use our algorithm for checking whether a grammar generates a string to filter-out those strings that are not generated by G.

Generating a Grammar's Language in Forlan

Suppose we have a grammar G and a natural number n. How can we generate the set of all elements of $L(G)$ of length n?

Of course, we could generate all strings over the alphabet of G of length n, and use our algorithm for checking whether a grammar generates a string to filter-out those strings that are not generated by G.

Alternatively, we can start by creating an EFA M accepting all strings over the alphabet of G with length n. Then, we can intersect G with M, and apply `Gram.toStrSet` to the resulting grammar.

Chomsky Normal Form

A grammar G is in *Chomsky Normal Form* (CNF) iff each of its productions has one of the following forms:

- $q \to a$, where a is not a variable; and
- $q \to pr$, where p and r are variables.

We explain by example how a grammar G can be turned into a simplified grammar in CNF that generates $L(G) - \{\%\}$.

Chomsky Normal Form

A grammar G is in *Chomsky Normal Form* (CNF) iff each of its productions has one of the following forms:

- $q \to a$, where a is not a variable; and
- $q \to pr$, where p and r are variables.

We explain by example how a grammar G can be turned into a simplified grammar in CNF that generates $L(G) - \{\%\}$.

Suppose G is the grammar

$$\mathsf{A} \to \mathsf{0A1} \mid \mathsf{01} \mid \mathsf{BB} \mid \mathsf{2},$$
$$\mathsf{B} \to \mathsf{2B} \mid \mathsf{2}.$$

We begin by applying our algorithm for eliminating $\%$-productions and unit productions to this grammar. In this case, it has no effect.

Conversion into CNF

Since the productions $\mathsf{A} \rightarrow \mathsf{BB}$, $\mathsf{A} \rightarrow 2$ and $\mathsf{B} \rightarrow 2$ are legal CNF productions, we simply transfer them to our new grammar.

Conversion into CNF

Since the productions $\mathsf{A} \to \mathsf{BB}$, $\mathsf{A} \to 2$ and $\mathsf{B} \to 2$ are legal CNF productions, we simply transfer them to our new grammar.

Next we add the variables $\langle 0 \rangle$, $\langle 1 \rangle$ and $\langle 2 \rangle$ to our grammar, along with the productions

$$\langle 0 \rangle \to 0, \quad \langle 1 \rangle \to 1, \quad \langle 2 \rangle \to 2.$$

Conversion into CNF

Since the productions $\mathsf{A} \to \mathsf{BB}$, $\mathsf{A} \to 2$ and $\mathsf{B} \to 2$ are legal CNF productions, we simply transfer them to our new grammar.

Next we add the variables $\langle 0 \rangle$, $\langle 1 \rangle$ and $\langle 2 \rangle$ to our grammar, along with the productions

$$\langle 0 \rangle \to 0, \quad \langle 1 \rangle \to 1, \quad \langle 2 \rangle \to 2.$$

Now, we can replace the production $\mathsf{A} \to 01$ with $\mathsf{A} \to \langle 0 \rangle \langle 1 \rangle$. And, we can replace the production $\mathsf{B} \to 2\mathsf{B}$ with the production $\mathsf{B} \to \langle 2 \rangle \mathsf{B}$.

Conversion into CNF

Since the productions $A \to BB$, $A \to 2$ and $B \to 2$ are legal CNF productions, we simply transfer them to our new grammar.

Next we add the variables $\langle 0 \rangle$, $\langle 1 \rangle$ and $\langle 2 \rangle$ to our grammar, along with the productions

$$\langle 0 \rangle \to 0, \quad \langle 1 \rangle \to 1, \quad \langle 2 \rangle \to 2.$$

Now, we can replace the production $A \to 01$ with $A \to \langle 0 \rangle \langle 1 \rangle$. And, we can replace the production $B \to 2B$ with the production $B \to \langle 2 \rangle B$.

Finally, we replace the production $A \to 0A1$ with the productions

Conversion into CNF

Since the productions $A \to BB$, $A \to 2$ and $B \to 2$ are legal CNF productions, we simply transfer them to our new grammar.

Next we add the variables $\langle 0 \rangle$, $\langle 1 \rangle$ and $\langle 2 \rangle$ to our grammar, along with the productions

$$\langle 0 \rangle \to 0, \quad \langle 1 \rangle \to 1, \quad \langle 2 \rangle \to 2.$$

Now, we can replace the production $A \to 01$ with $A \to \langle 0 \rangle\langle 1 \rangle$. And, we can replace the production $B \to 2B$ with the production $B \to \langle 2 \rangle B$.

Finally, we replace the production $A \to 0A1$ with the productions

$$A \to \langle 0 \rangle C, \quad C \to A\langle 1 \rangle,$$

and add C to the set of variables of our new grammar.

Conversion into CNF

Summarizing, our new grammar is

$$\begin{aligned}
\mathsf{A} &\to \mathsf{BB} \mid 2 \mid \langle 0\rangle\langle 1\rangle \mid \langle 0\rangle\mathsf{C},\\
\mathsf{B} &\to 2 \mid \langle 2\rangle\mathsf{B},\\
\langle 0\rangle &\to 0,\\
\langle 1\rangle &\to 1,\\
\langle 2\rangle &\to 2,\\
\mathsf{C} &\to \mathsf{A}\langle 1\rangle.
\end{aligned}$$

The official version of our algorithm names variables in a different way.

Converting into CNF in Forlan

The Forlan module `Gram` defines the following function:

```
val chomskyNormalForm : gram -> gram
```

Suppose `gram` of type `gram` is bound to the grammar with variables A and B, start variable A, and productions

$$\mathsf{A} \to \mathsf{0A1} \mid \mathsf{BB},$$
$$\mathsf{B} \to \% \mid \mathsf{2B}.$$

CNF in Forlan

Here is how Forlan can be used to turn this grammar into a CNF grammar that generates the nonempty strings that are generated by `gram`:

```
- val gram' = Gram.chomskyNormalForm gram;
val gram' = - : gram
- Gram.output("", gram');
{variables} <1,A>, <1,B>, <2,0>, <2,1>, <2,2>, <3,A1>
{start variable} <1,A>
{productions}
<1,A> -> 2 | <1,B><1,B> | <2,0><2,1> | <2,0><3,A1>;
<1,B> -> 2 | <2,2><1,B>; <2,0> -> 0; <2,1> -> 1;
<2,2> -> 2; <3,A1> -> <1,A><2,1>
val it = () : unit
```

CNF in Forlan

```
- val gram'' = Gram.renameVariablesCanonically gram';
val gram'' = - : gram
- Gram.output("", gram'');
{variables} A, B, C, D, E, F {start variable} A
{productions}
A -> 2 | BB | CD | CF; B -> 2 | EB; C -> 0; D -> 1;
E -> 2; F -> AD
val it = () : unit
```