Course MA2C03, Michaelmas Term 2014
Section 3: Formal Languages

David R. Wilkins

Copyright © David R. Wilkins 2000–2014

Contents

3 Formal Languages

3.1 Alphabets and Words

Let A be a finite set. We shall refer to this set A as an *alphabet* and we may refer to the elements of A as *letters*. (For example, the set A might be the set of letters in the English language, or in any other world language, or the set $\{0, 1\}$ of binary digits, or the set $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ of decimal digits.)

For any natural number n, we define a *word* of length n over the alphabet A to be a string of the form $a_1 a_2 \dots a_n$ in which $a_i \in A$ for $i = 1, 2, \dots, n$. We shall denote by A^n the set of all words of length n over the alphabet A. In particular, $A^1 = A$.

Example Let $A = \{a, b, c\}$. Then

$$\begin{aligned} A^2 &= \{aa, ab, ac, ba, bb, bc, ca, cb, cc\} \\ A^3 &= \{aaa, aab, aac, aba, abb, abc, \dots, ccc\} \end{aligned}$$

Remark The set A^n can be identified with the Cartesian product $A \times A \times \cdots \times A$ of n copies of the set A. The elements of this Cartesian product are ordered n-tuples $(a_1, a_2, \dots, a_n)$ whose components $a_1, a_2, \dots, a_n$ are elements of the set A. However, in the interests of brevity, it is convenient to drop the parentheses and commas, denoting any element $(a_1, a_2, \dots, a_n)$ of this Cartesian product by the corresponding string or word $a_1 a_2 \dots a_n$ of length n.

We denote by A^+ the union of the sets A^n for all natural numbers n, so that

$$A^+ = \bigcup_{n=1}^{\infty} A^n = A \cup A^2 \cup A^3 \cup A^4 \cup \cdots.$$

The set A^+ is thus the set of all words of positive length over the alphabet n.

Example Let $A = \{0, 1\}$. Then A^+ is the set of all binary strings whose length is finite and non-zero, and contains strings such as 1, 0, 10, 101, 010, 010101, and 000010010.

We introduce also an *empty word* ε, which we regard as a word of length zero. This may be thought of as the empty string, not involving any of the letters from the alphabet A. We define $A^0 = \{\varepsilon\}$, and we denote by A^* the set $\{\varepsilon\} \cup A^+$ obtained by adjoining the empty word ε to the set A^+ of words of positive length over the alphabet A. Thus

$$A^* = \{\varepsilon\} \cup A^+ = \bigcup_{n=0}^{\infty} A^i = \{\varepsilon\} \cup A \cup A^2 \cup A^3 \cup A^4 \cup \cdots.$$

Each word in A^* has a length which is a non-negative integer. The empty word is the only word in A^* whose length is zero.

We denote the length of a word w by $|w|$.

Definition Let A be a finite set, and let w_1 and w_2 be words over the alphabet A, with $w_1 = a_1a_2 \dots a_m$ and $w_2 = b_1b_2 \dots b_n$. The *concatenation* of the words w_1 and w_2 is the word $w_1 \circ w_2$, where

$$w_1 \circ w_2 = a_1a_2 \dots a_mb_1b_2 \dots b_n.$$

The concatenation $w_1 \circ w_2$ of two words w_1 and w_2 may also be denoted by w_1w_2.

Example Let A be the set of lower case letters in the English alphabet, and let w_1 and w_2 be the words 'book' and 'case' respectively. Then $w_1 \circ w_2$ is the word 'bookcase' and $w_2 \circ w_1$ is the word 'casebook'. Note that, in this instance, $w_1 \circ w_2 \neq w_2 \circ w_1$.

Note that $|w_1 \circ w_2| = |w_1| + |w_2|$ for all words w_1 and w_2 over some alphabet.

The operation $\circ$ of concatenation on the set of words over some alphabet is not commutative if that alphabet has more than one element. Indeed if a and b are distinct elements of this alphabet then $a \circ b$ is the string ab, and $b \circ a$ is the string ba, and therefore $a \circ b \neq b \circ a$.

Let w_1, w_2 and w_3 be words over some alphabet A. Then $(w_1 \circ w_2) \circ w_3 = w_1 \circ (w_2 \circ w_3)$. Indeed if

$$w_1 = a_1a_2 \dots a_m, \quad w_2 = b_1b_2 \dots b_n, \quad w_3 = c_1c_2 \dots c_p,$$

then $(w_1 \circ w_2) \circ w_3$ and $w_1 \circ (w_2 \circ w_3)$ both denote the word

$$a_1a_2 \dots a_mb_1b_2 \dots b_nc_1c_2 \dots c_p.$$

The empty word ε has the property that $\varepsilon \circ w = w \circ \varepsilon = w$ for all words w over an alphabet A.

The following theorem follows directly from these observations.

Theorem 3.1 *Let A be a finite set. Then $(A^*, \circ)$ is a monoid, where the set A^* is the set of words over the alphabet A, and $\circ$ is the operation of concatenation of words. The identity element of this monoid is the empty word ε.*

Definition Let A be a finite set. A *language* over A is a subset of A^*. A language L over A is said to be a *formal language* if there is some finite set of rules or algorithm that will generate all the words that belong to L and no others.

Let A be a finite set. The union and intersection of any finite collection of languages over A are themselves languages over A. In particular, the union $L_1 \cup L_2$ and intersection $L_1 \cap L_2$ of two languages L_1 and L_2 over A is a language over A.

The *concatenation* of languages L_1 and L_2 over A is the language $L_1 \circ L_2$, where

$$L_1 \circ L_2 = \{w_1 \circ w_2 \in A^* : w_1 \in L_1 \text{ and } w_2 \in L_2\}.$$

The concatenation $L_1 \circ L_2$ of the languages L_1 and L_2 may also be denoted by $L_1 L_2$.

Given any language L, we can form languages L^n for all natural numbers n, where $L^1 = L$ and $L^n = L \circ L^{n-1}$ for all natural numbers n. The associativity of the concatenation operation ensures that $L^m \circ L^n = L^{m+n}$ for all natural numbers m and n. We define $L^0 = \{\epsilon\}$. The language L then determines languages L^+ and L^* over A, where

$$L^+ = \bigcup_{n=1}^{\infty} L^n = L \cup L^2 \cup L^3 \cup L^4 \cup \cdots$$

and

$$L^* = \{\varepsilon\} \cup L^+ = \bigcup_{n=0}^{\infty} L^n = \{\varepsilon\} \cup L \cup L^2 \cup L^3 \cup L^4 \cup \cdots.$$

3.2 Simple Grammars to Generate English Sentences

We describe a simple grammar to construct a certain collection of English sentences. This grammar is specified by the following collection of *productions* or replacement rules:

$$\begin{aligned}
\langle \text{S} \rangle &\to \langle \text{NP} \rangle \langle \text{VP} \rangle \\
\langle \text{VP} \rangle &\to \langle \text{V} \rangle \langle \text{NP} \rangle \\
\langle \text{NP} \rangle &\to \langle \text{T} \rangle \langle \text{QN} \rangle \\
\langle \text{QN} \rangle &\to \langle \text{Adj} \rangle \langle \text{QN} \rangle \\
\langle \text{QN} \rangle &\to \langle \text{N} \rangle \\
\langle \text{T} \rangle &\to \text{the} \\
\langle \text{N} \rangle &\to \text{cat}
\end{aligned}$$

$$
\begin{aligned}
\langle \mathrm{N} \rangle &\rightarrow \text{dog} \\
\langle \mathrm{Adj} \rangle &\rightarrow \text{black} \\
\langle \mathrm{Adj} \rangle &\rightarrow \text{old} \\
\langle \mathrm{Adj} \rangle &\rightarrow \text{small} \\
\langle \mathrm{V} \rangle &\rightarrow \text{saw} \\
\langle \mathrm{V} \rangle &\rightarrow \text{chased}
\end{aligned}
$$

Here $\langle \mathrm{S} \rangle$, $\langle \mathrm{VP} \rangle$, $\langle \mathrm{NP} \rangle$, $\langle \mathrm{QN} \rangle$, $\langle \mathrm{T} \rangle$, $\langle \mathrm{N} \rangle$, $\langle \mathrm{Adj} \rangle$, $\langle \mathrm{V} \rangle$, represent certain phrases or words that may form part of sentences generated by the grammar. They may be interpreted as follows:—

- $\langle \mathrm{S} \rangle$ represents the sentence to be generated by the grammar;
- $\langle \mathrm{VP} \rangle$ represents a 'verb phrase';
- $\langle \mathrm{NP} \rangle$ represents a 'noun phrase';
- $\langle \mathrm{QN} \rangle$ represents a noun optionally preceded by one or more adjectives;
- $\langle \mathrm{T} \rangle$ represents the definite article 'the' (which will subsequently replace it);
- $\langle \mathrm{N} \rangle$ represents a noun chosen from the set the set $\{\text{dog}, \text{cat}\}$;
- $\langle \mathrm{Adj} \rangle$ represents an adjective chosen from the set $\{\text{black}, \text{old}, \text{small}\}$;
- $\langle \mathrm{V} \rangle$ represents a verb chosen from the set $\{\text{saw}, \text{chased}\}$;

Each of these productions specifies that the entity on the left hand side of the arrow may be replaced by the string on the right hand side of the arrow.

We may apply these productions successively, starting with the symbol $\langle \mathrm{S} \rangle$, in order to obtain the sentences generated by the grammar.

Example We may generate the sentence 'The dog chased the old black cat'

as follows:

⟨S⟩	⇒	⟨NP⟩⟨VP⟩	(⟨S⟩	→	⟨NP⟩⟨VP⟩)
	⇒	⟨T⟩⟨QN⟩⟨VP⟩	(⟨NP⟩	→	⟨T⟩⟨QN⟩)
	⇒	the ⟨QN⟩⟨VP⟩	(⟨T⟩	→	the)
	⇒	the ⟨N⟩⟨VP⟩	(⟨QN⟩	→	⟨N⟩)
	⇒	the dog ⟨VP⟩	(⟨N⟩	→	dog)
	⇒	the dog ⟨V⟩⟨NP⟩	(⟨VP⟩	→	⟨V⟩⟨NP⟩)
	⇒	the dog chased ⟨NP⟩	(⟨V⟩	→	chased)
	⇒	the dog chased ⟨T⟩⟨QN⟩	(⟨NP⟩	→	⟨T⟩⟨QN⟩)
	⇒	the dog chased the ⟨QN⟩	(⟨T⟩	→	the)
	⇒	the dog chased the ⟨Adj⟩⟨QN⟩	(⟨QN⟩	→	⟨Adj⟩⟨QN⟩)
	⇒	the dog chased the old ⟨QN⟩	(⟨Adj⟩	→	old)
	⇒	the dog chased the old ⟨Adj⟩⟨QN⟩	(⟨QN⟩	→	⟨Adj⟩⟨QN⟩)
	⇒	the dog chased the old black ⟨QN⟩	(⟨Adj⟩	→	black)
	⇒	the dog chased the old black ⟨N⟩	(⟨QN⟩	→	⟨N⟩)
	⇒	the dog chased the old black cat.	(⟨N⟩	→	cat)

The production used at each step of the derivation is specified on the right. At each stage a single instance of ⟨S⟩, ⟨VP⟩, ⟨NP⟩, ⟨QN⟩, ⟨T⟩, ⟨N⟩, ⟨Adj⟩ or ⟨V⟩ is replaced by the appropriate string.

In grammars such as the one we are studying, words such as 'the', 'black', 'old', 'small', 'cat', 'dog', 'chased', 'saw' are referred to as *terminals*. Entities such as ⟨S⟩, ⟨VP⟩, ⟨NP⟩, ⟨QN⟩, ⟨T⟩, ⟨N⟩, ⟨Adj⟩ and ⟨V⟩ are referred to as *nonterminals*. Each production in a *context-free grammar* specifies that a single occurrence of the nonterminal specified to the left of the arrow may be replaced by the string specified to the right of the arrow. Such replacements are applied, one at a time, to transform strings made up of terminals and nonterminals to other strings of the same kind.

Grammars defined by means of productions may be specified in a more compact form, introduced by John Backus and Peter Naur, and employed in a report on the high-level programming language ALGOL-60 published in 1960. Our grammar for simple English sentences may be presented in Backus-Naur form as follows:

⟨S⟩	→	⟨NP⟩⟨VP⟩
⟨VP⟩	→	⟨V⟩⟨NP⟩
⟨NP⟩	→	⟨T⟩⟨QN⟩
⟨QN⟩	→	⟨Adj⟩⟨QN⟩ | ⟨N⟩
⟨T⟩	→	the

$$
\begin{aligned}
\langle \mathrm{N} \rangle &\rightarrow \text{cat} \mid \text{dog} \\
\langle \mathrm{Adj} \rangle &\rightarrow \text{black} \mid \text{old} \mid \text{small} \\
\langle \mathrm{V} \rangle &\rightarrow \text{saw} \mid \text{chased}
\end{aligned}
$$

Each item in this list specifies that the nonterminal occurring to the left of the arrow may be replaced by one of various alternatives; the alternatives are separated by the meta-character |.

One may verify that the sentences generated by the grammar are of the following form: the sentence consists of the definite article 'the', optionally followed by one or more of the adjectives 'black', 'old' and 'small', followed by one of the nouns 'cat' and 'dog', followed by one of the verbs 'saw' and 'chased', followed by the definite article 'the', optionally followed by one or more of the listed adjectives, followed by one of the listed nouns. From this description one may verify that the sentences generated by simple grammar we have described are the same as those generated by the grammar specified in Backus-Naur form as follows:—

$$
\begin{aligned}
\langle \mathrm{S} \rangle &\rightarrow \text{the } \langle \mathrm{T1} \rangle \\
\langle \mathrm{T1} \rangle &\rightarrow \text{black } \langle \mathrm{T1} \rangle \mid \text{old } \langle \mathrm{T1} \rangle \mid \text{small } \langle \mathrm{T1} \rangle \mid \text{cat } \langle \mathrm{T2} \rangle \mid \text{dog } \langle \mathrm{T2} \rangle \\
\langle \mathrm{T2} \rangle &\rightarrow \text{saw } \langle \mathrm{T3} \rangle \mid \text{chased } \langle \mathrm{T3} \rangle \\
\langle \mathrm{T3} \rangle &\rightarrow \text{the } \langle \mathrm{T4} \rangle \\
\langle \mathrm{T4} \rangle &\rightarrow \text{black } \langle \mathrm{T4} \rangle \mid \text{old } \langle \mathrm{T4} \rangle \mid \text{small } \langle \mathrm{T4} \rangle \mid \text{cat } \langle \mathrm{F} \rangle \mid \text{dog } \langle \mathrm{F} \rangle \\
\langle \mathrm{F} \rangle &\rightarrow \varepsilon
\end{aligned}
$$

This grammar is an example of a *regular grammar*. A regular grammar is determined by productions in which a single nonterminal is replaced, either by a terminal followed by a nonterminal, or by a single terminal, or by the empty string ε.

Example The sentence 'The dog chased the old black cat' is generated in the regular grammar presented above as follows:

$$
\begin{aligned}
\langle \mathrm{S} \rangle &\Rightarrow \text{the } \langle \mathrm{T1} \rangle \\
&\Rightarrow \text{the dog } \langle \mathrm{T2} \rangle \\
&\Rightarrow \text{the dog chased } \langle \mathrm{T3} \rangle \\
&\Rightarrow \text{the dog chased the } \langle \mathrm{T4} \rangle \\
&\Rightarrow \text{the dog chased the old } \langle \mathrm{T4} \rangle \\
&\Rightarrow \text{the dog chased the old black } \langle \mathrm{T4} \rangle \\
&\Rightarrow \text{the dog chased the old black cat } \langle \mathrm{F} \rangle \\
&\Rightarrow \text{the dog chased the old black cat.}
\end{aligned}
$$

If a language is generated by a regular grammar then it is possible to construct a *finite state acceptor* for that language. This is a *finite state machine* which may be used to determine whether or not a given string belongs to the language.

We describe a finite state acceptor for the collection of sentences generated by the regular grammar described above. This machine has a finite number of internal states. One of these states is the *initial state* of the machine. Some of the states of a finite state acceptor are regarded as *final states*. Words taken from the list

the, black, old, small, cat, dog, saw, chased

are successively input into the machine. Each time one of these words is input the machine either remains in the state it is currently in, or it makes a transition to some other internal state, determined by the current state and the input word. We consider such a machine with seven internal states, which we label as S, T1, T2, T3, T4, F and E. The following table describes the effect of inputting each word:

	the	black	old	small	cat	dog	saw	chased
S	T1	E	E	E	E	E	E	E
T1	E	T1	T1	T1	T2	T2	E	E
T2	E	E	E	E	E	E	T3	T3
T3	T4	E	E	E	E	E	E	E
T4	E	T4	T4	T4	F	F	E	E
F	E	E	E	E	E	E	E	E
E	E	E	E	E	E	E	E	E

The words label the columns of the table, the internal states label the rows, and each entry in the table specifies the state that results when the current state is that labelling the row and the input word is that labelling the column. For example, if the machine is in state T1, and if the input word is 'cat', then the internal state of the machine is changed to the state T2. The state E may be regarded as an 'error state': the machine enters this state whenever a word is input that may not occur at the relevant position in the sentence. Moreover, once the machine is in the state E, it remains in this state, no matter which word is input. The state S is the initial or starting state. There is a single final state, which is the state F. A finite sequence of words is accepted by the machine if and only if successively inputting these words causes the machine to move from state S to state F.

3.3 Well-Formed Formulae in Logic

We shall investigate the grammar of well-formed formulae in the Propositional Calculus. We begin with a discussion of basic principles of this calculus.

Let p and q be *Boolean variables*. Such variables may represent propositions that might be true in certain circumstances, or false in other circumstances. A Boolean variable may therefore take on one of two values: true (**T**) or false (**F**).

Example Let the Boolean variable p represent the proposition $(x > y)$, where x and y are arbitrary real numbers. If $x = 10$ and $y = 5$ then p is true. But if $x = 5$ and $y = 10$ then p is false.

In the Propositional Calculus we also have two Boolean constants. We shall denote one of these by **T**: it may represent a proposition that is true in all circumstances. We shall denote the other constant by **F**: it may represent a proposition that is false in all circumstances.

In the Propositional Calculus we may build more complicated formulae out of simpler formulae using the operations of *negation* $\neg$, *conjunction* $\wedge$ and and *disjunction* $\vee$. Negation is a unary operation, whereas conjunction and disjunction are binary operations.

Let p be a Boolean variable. The *negation* $\neg p$ of p has the property that it is true whenever p is false, and it is false whenever p is true. The relationship between p and $\neg p$ is therefore expressed by the following truth table:—

p	$\neg p$
T	**F**
F	**T**

Let p and q be Boolean variables. The *conjunction* $p \wedge q$ of p and q is true if and only if both p and q are true. The *disjunction* $p \vee q$ of p and q is true if and only if at least one of p and q is true. The relationship between p, q, $p \wedge q$ and $p \vee q$ are therefore expressed by the following truth tables:—

p	q	$p \wedge q$
T	**T**	**T**
T	**F**	**F**
F	**T**	**F**
F	**F**	**F**

p	q	$p \vee q$
T	**T**	**T**
T	**F**	**T**
F	**T**	**T**
F	**F**	**F**

The conjunction $p \wedge q$ may be thought of as representing the proposition p AND q. Similarly the disjunction $p \vee q$ may be thought of as representing the proposition p OR q.

We may build up more complicated formulae using these basic operations of negation, conjunction and disjunction. For example, if p, q and r are Boolean variables, and if the Boolean variable s is used to represent the conjunction $p \wedge q$ of p and q, then we may write

$$\neg s = \neg(p \wedge q), \quad s \wedge r = (p \wedge q) \wedge r,$$

$$s \vee r = (p \wedge q) \vee r, \quad r \vee s = r \vee (p \wedge q), \quad \text{etc.}$$

Consider the propositions represented by the formulae $(p \wedge q) \wedge r$ and $p \wedge (q \wedge r)$. These two propositions are true if and only if each of the propositions p, q and r is true. They are false if any one of the propositions p, q and r is false. It follows that no ambiguity will result if we write $p \wedge q \wedge r$ in place of $(p \wedge q) \wedge r$ or $p \wedge (q \wedge r)$. More generally, we can define the conjunction of any finite number of propositions: the *conjunction* $p_1 \wedge p_2 \wedge \cdots \wedge p_n$ of propositions $p_1, p_2, \ldots, p_n$ is true if and only if every one of the propositions $p_1, p_2, \ldots, p_n$ is true. We may define in a similar fashion the disjunction of any finite number of propositions: the *disjunction* $p_1 \vee p_2 \vee \cdots \vee p_n$ of propositions $p_1, p_2, \ldots, p_n$ is true if and only if at least one of the propositions $p_1, p_2, \ldots, p_n$ is true.

Let us now consider what meaning, if any, one might assign to a formula such as '$p \wedge q \vee r$'. One might consider interpreting a formula of this form either as $(p \wedge q) \vee r$ or as $p \wedge (q \vee r)$. The following truth table exhibits the dependence of the truth values of these two latter formulae on those of p, q and r:—

p	q	r	$p \wedge q$	$q \vee r$	$(p \wedge q) \vee r$	$p \wedge (q \vee r)$
T	**T**	**T**	**T**	**T**	**T**	**T**
T	**T**	**F**	**T**	**T**	**T**	**T**
T	**F**	**T**	**F**	**T**	**T**	**T**
T	**F**	**F**	**F**	**F**	**F**	**F**
F	**T**	**T**	**F**	**T**	**T**	**F**
F	**T**	**F**	**F**	**T**	**F**	**F**
F	**F**	**T**	**F**	**T**	**T**	**F**
F	**F**	**F**	**F**	**F**	**F**	**F**

We see from this truth table that $(p \wedge q) \vee r$ and $p \wedge (q \vee r)$ do not represent equivalent propositions. For example, if p is false, q is true and r is true, then $(p \wedge q) \vee r$ is true, but $p \wedge (q \vee r)$ is false.

We would need to resolve this ambiguity in some way if we were to admit expressions such as '$p \wedge q \vee q$'. One approach would be to assign higher precedence to one or other of the binary operations $\wedge$ and $\vee$. (This would correspond to the convention in evaluating expressions in ordinary arithmetic

and algebra, where multiplication is assigned a higher precedence than addition.) A second possible approach would involve assigning equal precedence to the two operations $\wedge$ and $\vee$ whilst adopting the convention that evaluations of formulae in the Propositional Calculus involving these operations proceed from left to right in the absence of any parentheses indicating the order in which the operations are to be performed. (This would correspond to a standard convention in evaluating expressions in ordinary arithmetic and algebra involving additions and subtractions.)

However a sensible approach would involve regarding a formula such as '$p \wedge q \vee r$' as being ill-formed, on the grounds that it is inherently ambiguous in the absence of parentheses that would specify the order in which the operations $\wedge$ and $\vee$ are to be performed. One would not then seek to assign any truth value to such a formula, any more than one would seek to assign a truth value to a jumble of symbols such as '$p) \wedge \vee(((qr$'.

We are then led to the problem of providing a formal specification to determine which strings of characters involving 'p', 'q', 'r' etc., '$\neg$', '$\wedge$', '$\vee$' '(' and ')' are to be regarded as well-formed formulae in the Propositional Calculus. A related problem is that of designing an algorithm to determine whether or not a string involving these characters is to be regarded as a well-formed formula of the Propositional Calculus.

A formula of the Propositional Calculus consists of a string of characters taken from some finite set. This set would contain characters such as $\neg$, $\wedge$ and $\vee$ to denote the operations of negation, conjunction and disjunction respectively. It might also contain parentheses '(' and ')' that can be used to determine in the usual fashion the order in which the binary operations are to be performed and the subformulae to which they are to be applied. We could introduce characters **T** and **F** to denote the Boolean constants 'true' and 'false' respectively. It remains to consider how Boolean variables are to be represented. We could certainly use single letters p, q, r, s. But this would only enable us to write down formulae with at most four distinct propositional variables. Were we to use single letters from the English alphabet in both upper and lower case to denote propositional variables, this would restrict us to formulae with at most fifty-two distinct Boolean variables. But there should be no limit to the number of distinct Boolean variables that we could introduce into a well-formed formula. We therefore need a scheme for representing unlimited quantities of Boolean variables within our formula. We could do this by using p', p'', p''', p'''' etc., in addition to single letters such as p. Here p'''', for example, is to be regarded as a string of length 5, consisting of the letter p, followed by four instances of the prime character $'$. Accordingly we shall specify that a propositional variable is to be represented by the letters p, q, r, s, either alone or else followed by a string consisting

of any number of prime characters. (The choice of which letters to use is of course arbitrary; any similar choice would serve just as well.) The formulae of the Propositional Calculus may then be regarded as strings (or words) over the alphabet A, where

$$A = \{\neg, \wedge, \vee, (,), \mathbf{T}, \mathbf{F}, p, q, r, s, '\}.$$

The elements of A^* (i.e., the words over the alphabet A) are then strings of characters taken from the set A. Some of them, such as '$(p \wedge p') \vee (p'' \wedge p''')$', represent propositions whose truth values are determined unambiguously from the truth values of the Boolean variables occurring within them. Others, such as '$)p\neg) \wedge \vee \vee (\mathbf{T}p'''$' are pure gibberish. Our task is then to provide some sort of formal specification which determines which of the strings belonging to A^* are to be regarded as well-formed formulae of the Propositional Calculus. The collection of well-formed formulae is then a language over the alphabet A.

The method by which we specify the well-formed formulae is an example of a *context-free grammar*. For this, we introduce a set N of *nonterminals*. We represent each nonterminal by an appropriate identifier enclosed within angle brackets '$\langle$' and '$\rangle$'. For example we shall use the nonterminal '$\langle$wff$\rangle$' to represent an arbitrary well-formed formula. Other nonterminals shall be used to represent well-formed formulae that are of a special form. For example, the nonterminal '$\langle$negation$\rangle$' will represent a well-formed formula that is the negation of some other well-formed formula. Other nonterminals may represent things such as the letters p, q, r and s of the English alphabet.

We shall refer to elements of the set A as *terminals*. (In any context-free grammar, the *terminals* are the elements of the alphabet over which the language specified by that grammar is defined.)

The context-free grammar will then consist of a finite collection of *productions*. Each production specifies that a certain nonterminal may be replaced by some string whose elements are terminals or nonterminals. A word in A^* is then said to be *generated* by the grammar if some succession of replacements determined by productions in the grammar transforms the nonterminal $\langle$wff$\rangle$ into the given word.

Our grammar will include three productions in which the nonterminal $\langle$wff$\rangle$ occurs on the left hand side. These are

$$\begin{aligned} \langle\text{wff}\rangle &\rightarrow (\langle\text{wff}\rangle) \\ \langle\text{wff}\rangle &\rightarrow \langle\text{atom}\rangle \\ \langle\text{wff}\rangle &\rightarrow \langle\text{compound}\rangle \end{aligned}$$

The effect of including the first of these productions is to ensure that, whenever a well-formed formula is enclosed within parentheses, the resulting for-

mula is also well-formed. For example, the formulae $(p \wedge q)$, $((p \wedge q))$, $(((p \wedge q)))$, etc., are obtained in this way from the well-formed formula $p \wedge q$, and our grammar will therefore ensure that these formulae are also well-formed.

If a well-formed formula is not merely other well-formed formulae enclosed within parentheses, then it may be regarded as being either an atomic formula or a compound formula. The atomic formulae are the Boolean constants **T** and **F** and variables such as p, q, r, s, p', p'' etc. The compound formulae are those that are constructed out of shorter well-formed formulae using the operations of negation, conjunction and disjunction. The production

$$\langle \text{wff} \rangle \to \langle \text{atom} \rangle$$

allows us to replace the nonterminal $\langle \text{wff} \rangle$ by $\langle \text{atom} \rangle$ when the well-formed formula we are seeking to generate is a Boolean constant or a Boolean variable. The production

$$\langle \text{wff} \rangle \to \langle \text{compound} \rangle$$

allows us to replace the nonterminal $\langle \text{wff} \rangle$ by $\langle \text{compound} \rangle$ when the well-formed formula we are seeking to generate is a compound formula. Successive applications of these three productions to the nonterminal $\langle \text{wff} \rangle$ yield strings such as

$$\langle \text{atom} \rangle, \quad \langle \text{compound} \rangle, \quad (\langle \text{atom} \rangle), \quad ((((\langle \text{compound} \rangle)))).$$

The sequence of steps that transform $\langle \text{wff} \rangle$ into $((((\langle \text{compound} \rangle))))$ may be presented as follows:

$$\begin{aligned} \langle \text{wff} \rangle &\Rightarrow (\langle \text{wff} \rangle) \\ &\Rightarrow ((\langle \text{wff} \rangle)) \\ &\Rightarrow ((((\langle \text{wff} \rangle)))) \\ &\Rightarrow ((((\langle \text{compound} \rangle)))) \end{aligned}$$

The first three steps apply the production $\langle \text{wff} \rangle \to (\langle \text{wff} \rangle)$ to the nonterminal $\langle \text{wff} \rangle$ in the relevant formula, replacing this nonterminal by $(\langle \text{wff} \rangle)$. The final step applies the production $\langle \text{wff} \rangle \to \langle \text{compound} \rangle$ to the nonterminal $\langle \text{wff} \rangle$ in the penultimate formula, replacing this nonterminal by $\langle \text{compound} \rangle$. (Where the symbol $\Rightarrow$ occurs between two strings, this indicates that the first string can be transformed into the second string on applying one of the productions of the grammar to a single nonterminal occuring in the first string to replace that nonterminal by the appropriate string.) We write

$$\langle \text{wff} \rangle \stackrel{*}{\Rightarrow} ((((\langle \text{compound} \rangle))))$$

to indicate that the string on the left can be transformed into the string on the right through the successive application of a finite number of productions belonging to the grammar. (In general, where the symbol $\stackrel{*}{\Rightarrow}$ is placed between two strings, this indicates either that the first string is identical to the second, or else that the first string can be transformed into the second through the successive application of a finite number of productions belonging to the relevant grammar.)

The three productions which we can apply to the nonterminal ⟨wff⟩ can be specified more compactly in *Backus-Naur form* by means of the following:

$$\langle \text{wff} \rangle \to (\langle \text{wff} \rangle) \mid \langle \text{atom} \rangle \mid \langle \text{compound} \rangle$$

This indicates that the nonterminal ⟨wff⟩ occurring on the left hand side may be replaced by any one of a list of alternatives presented on the right hand side. The meta-character | is used to separate the alternatives within this list.

We next consider the productions for producing atomic formulae. Any atomic formula represents either a Boolean constant or a Boolean variable. Moreover the Boolean constants are **T** and **F**. This leads us to introduce the following four productions:

$$\begin{aligned} \langle \text{atom} \rangle &\to \langle \text{constant} \rangle \\ \langle \text{atom} \rangle &\to \langle \text{variable} \rangle \\ \langle \text{constant} \rangle &\to \mathbf{T} \\ \langle \text{constant} \rangle &\to \mathbf{F} \end{aligned}$$

These four productions may be specified in Backus-Naur form by the following:

$$\begin{aligned} \langle \text{atom} \rangle &\to \langle \text{constant} \rangle \mid \langle \text{variable} \rangle \\ \langle \text{constant} \rangle &\to \mathbf{T} \mid \mathbf{F} \end{aligned}$$

Using these productions we find that

$$\langle \text{atom} \rangle \Rightarrow \langle \text{constant} \rangle \Rightarrow \mathbf{T}, \quad \langle \text{atom} \rangle \Rightarrow \langle \text{constant} \rangle \Rightarrow \mathbf{F},$$

and thus

$$\langle \text{atom} \rangle \stackrel{*}{\Rightarrow} \mathbf{T}, \quad \langle \text{atom} \rangle \stackrel{*}{\Rightarrow} \mathbf{F}.$$

We also need to specify the productions that transform the nonterminal ⟨variable⟩ into the formulae

$$p, q, r, s, p', q', r', s', p'', q'', r'', s'', p''', q''', r''', s''', \ldots .$$

These transformations may be accomplished using the following productions:

$$\begin{aligned} \langle \text{variable} \rangle &\rightarrow \langle \text{variable} \rangle' \\ \langle \text{variable} \rangle &\rightarrow \langle \text{letter} \rangle \\ \langle \text{letter} \rangle &\rightarrow p \\ \langle \text{letter} \rangle &\rightarrow q \\ \langle \text{letter} \rangle &\rightarrow r \\ \langle \text{letter} \rangle &\rightarrow s \end{aligned}$$

These productions can be specified in Backus-Naur form as follows:

$$\begin{aligned} \langle \text{variable} \rangle &\rightarrow \langle \text{letter} \rangle \mid \langle \text{variable} \rangle' \\ \langle \text{letter} \rangle &\rightarrow p \mid q \mid r \mid s \end{aligned}$$

For example, the formula p''' is generated from the nonterminal $\langle \text{variable} \rangle$ by the following sequence of transformations:

$$\langle \text{variable} \rangle \Rightarrow \langle \text{variable} \rangle' \Rightarrow \langle \text{variable} \rangle'' \Rightarrow \langle \text{variable} \rangle''' \Rightarrow \langle \text{letter} \rangle''' \Rightarrow p'''.$$

The first three transformations use the production $\langle \text{variable} \rangle \rightarrow \langle \text{variable} \rangle'$, the fourth uses the production $\langle \text{variable} \rangle \rightarrow \langle \text{letter} \rangle$, and the final transformation uses the production $\langle \text{letter} \rangle \rightarrow p$. Thus $\langle \text{variable} \rangle \stackrel{*}{\Rightarrow} p'''$. The formulae representing Boolean constants and variables can now all be generated from the nonterminal $\langle \text{wff} \rangle$. For example, $\langle \text{wff} \rangle \stackrel{*}{\Rightarrow} q'$, since

$$\langle \text{wff} \rangle \Rightarrow \langle \text{atom} \rangle \Rightarrow \langle \text{variable} \rangle \Rightarrow \langle \text{variable} \rangle' \Rightarrow \langle \text{letter} \rangle' \Rightarrow q'.$$

Our grammar now has productions to generate any atomic formula. We now need to add productions that will generate compound formulae. A compound formula is either the negation of some other well-formed formula, or the conjunction of two well-formed formulae, or the disjunction of two such formulae. This leads us to introduce the productions

$$\begin{aligned} \langle \text{compound} \rangle &\rightarrow \langle \text{negation} \rangle \\ \langle \text{compound} \rangle &\rightarrow \langle \text{conjunction} \rangle \\ \langle \text{compound} \rangle &\rightarrow \langle \text{disjunction} \rangle \end{aligned}$$

which may be specified in Backus-Naur form as

$$\langle \text{compound} \rangle \rightarrow \langle \text{negation} \rangle \mid \langle \text{conjunction} \rangle \mid \langle \text{disjunction} \rangle$$

Let us consider negations. If F is any well-formed formula, then $\neg(F)$ is a well-formed formula that represents the negation of the formula F. Also we

shall regard $\neg F$ as being a well-formed formula representing the negation of F in the cases when F is atomic or when F is itself a negation. However we will adopt the convention that the negation operation $\neg$ has higher precedence than either $\wedge$ or $\vee$. Thus if F is of the form $G \wedge H$ then $\neg G \wedge H$ will be equivalent to $(\neg G) \wedge H$ and will not represent the negation $\neg(G \wedge H)$ of F. Similarly $\neg G \vee H$ is to be interpreted as $(\neg G) \vee H$, and is not equivalent to $\neg(G \vee H)$. Therefore we only denote the negation of a well-formed formula F by $\neg F$ in the cases when F is atomic or a negation, and not in the cases when F is a conjunction or disjunction. We therefore introduce the following productions into our grammar:

$$\begin{aligned} \langle \text{negation} \rangle &\rightarrow \neg \langle \text{nf} \rangle \\ \langle \text{nf} \rangle &\rightarrow \langle \text{atom} \rangle \\ \langle \text{nf} \rangle &\rightarrow \langle \text{negation} \rangle \\ \langle \text{nf} \rangle &\rightarrow (\langle \text{wff} \rangle) \end{aligned}$$

These productions are specified in Backus-Naur form as follows:

$$\begin{aligned} \langle \text{negation} \rangle &\rightarrow \neg \langle \text{nf} \rangle \\ \langle \text{nf} \rangle &\rightarrow \langle \text{atom} \rangle \mid \langle \text{negation} \rangle \mid (\langle \text{wff} \rangle) \end{aligned}$$

Next let us consider conjunctions. If G and H are well-formed formulae then $(G) \wedge (H)$ is a well-formed formula representing the conjunction of G and H. We may replace $(G) \wedge (H)$ by $G \wedge (H)$ without introducing any ambiguity in the cases when G is atomic, the negation of a well-formed formula or a conjunction of well-formed formulae. (Our rules of precedence ensure that $\neg G \wedge H$ is interpreted as $(\neg G) \wedge H$ and not as $\neg(G \wedge H)$, since we regard negation has having higher precedence than conjunction.) We shall not allow ourselves to replace $(G) \wedge (H)$ by $G \wedge (H)$ when G is a disjunction of well-formed formulae. Similarly we may replace $(G) \wedge (H)$ and $G \wedge (H)$ by $(G) \wedge H$ and $G \wedge H$ respectively when the formula H is an atomic formula or the negation of a well-formed formula, or a conjunction of well-formed formulae, but not when H is a disjunction of well-formed formulae. These considerations are respected if we introduce the productions

$$\begin{aligned} \langle \text{conjunction} \rangle &\rightarrow \langle \text{cf} \rangle \wedge \langle \text{cf} \rangle \\ \langle \text{cf} \rangle &\rightarrow \langle \text{atom} \rangle \\ \langle \text{cf} \rangle &\rightarrow \langle \text{negation} \rangle \\ \langle \text{cf} \rangle &\rightarrow \langle \text{conjunction} \rangle \\ \langle \text{cf} \rangle &\rightarrow (\langle \text{wff} \rangle) \end{aligned}$$

which may be specified in Backus-Naur form as follows:

$$\begin{aligned} \langle\text{conjunction}\rangle &\rightarrow \langle\text{cf}\rangle \wedge \langle\text{cf}\rangle \\ \langle\text{cf}\rangle &\rightarrow \langle\text{atom}\rangle \mid \langle\text{negation}\rangle \mid \langle\text{conjunction}\rangle \mid (\langle\text{wff}\rangle) \end{aligned}$$

With these productions we find, for example, that

$$\begin{aligned} \langle\text{conjunction}\rangle &\Rightarrow \langle\text{cf}\rangle \wedge \langle\text{cf}\rangle \\ &\Rightarrow \langle\text{atom}\rangle \wedge \langle\text{cf}\rangle \\ &\Rightarrow \langle\text{atom}\rangle \wedge \langle\text{negation}\rangle \\ &\Rightarrow \langle\text{atom}\rangle \wedge \neg\langle\text{nf}\rangle \\ &\Rightarrow \langle\text{atom}\rangle \wedge \neg\langle\text{atom}\rangle \end{aligned}$$

and

$$\begin{aligned} \langle\text{conjunction}\rangle &\Rightarrow \langle\text{cf}\rangle \wedge \langle\text{cf}\rangle \\ &\Rightarrow \langle\text{atom}\rangle \wedge \langle\text{cf}\rangle \\ &\Rightarrow \langle\text{atom}\rangle \wedge \langle\text{conjunction}\rangle \\ &\Rightarrow \langle\text{atom}\rangle \wedge \langle\text{cf}\rangle \wedge \langle\text{cf}\rangle \\ &\Rightarrow \langle\text{atom}\rangle \wedge \langle\text{atom}\rangle \wedge \langle\text{cf}\rangle \\ &\Rightarrow \langle\text{atom}\rangle \wedge \langle\text{atom}\rangle \wedge \langle\text{atom}\rangle. \end{aligned}$$

We can then apply further productions in order to obtain formulae such as $p \wedge \neg q$ and $p' \wedge p'' \wedge p'''$.

Finally we have to specify the productions for handling disjunction. These are analogous to those for conjunctions, and are the following:—

$$\begin{aligned} \langle\text{disjunction}\rangle &\rightarrow \langle\text{df}\rangle \vee \langle\text{df}\rangle \\ \langle\text{df}\rangle &\rightarrow \langle\text{atom}\rangle \\ \langle\text{df}\rangle &\rightarrow \langle\text{negation}\rangle \\ \langle\text{df}\rangle &\rightarrow \langle\text{disjunction}\rangle \\ \langle\text{df}\rangle &\rightarrow (\langle\text{wff}\rangle) \end{aligned}$$

These can be presented in Backus-Naur form as follows:

$$\begin{aligned} \langle\text{disjunction}\rangle &\rightarrow \langle\text{df}\rangle \vee \langle\text{df}\rangle \\ \langle\text{df}\rangle &\rightarrow \langle\text{atom}\rangle \mid \langle\text{negation}\rangle \mid \langle\text{disjunction}\rangle \mid (\langle\text{wff}\rangle) \end{aligned}$$

With these productions we can generate formulae such as $p \vee (q \wedge r)$ by applying productions successively as follows:

$$\begin{aligned}
\langle\text{disjunction}\rangle &\Rightarrow \langle\text{df}\rangle \vee \langle\text{df}\rangle \\
&\Rightarrow \langle\text{atom}\rangle \vee \langle\text{df}\rangle \\
&\Rightarrow \langle\text{atom}\rangle \vee (\langle\text{wff}\rangle) \\
&\Rightarrow \langle\text{atom}\rangle \vee (\langle\text{compound}\rangle) \\
&\Rightarrow \langle\text{atom}\rangle \vee (\langle\text{conjunction}\rangle) \\
&\Rightarrow \langle\text{atom}\rangle \vee (\langle\text{cf}\rangle \wedge \langle\text{cf}\rangle) \\
&\Rightarrow \langle\text{atom}\rangle \vee (\langle\text{atom}\rangle \wedge \langle\text{cf}\rangle) \\
&\Rightarrow \langle\text{atom}\rangle \vee (\langle\text{atom}\rangle \wedge \langle\text{atom}\rangle)
\end{aligned}$$

This completes our construction of a context-free grammar that generates the well-formed formulae of the Propositional Calculus.

This grammar is specified in Backus-Naur form by the following:—

$$\begin{aligned}
\langle\text{wff}\rangle &\rightarrow (\langle\text{wff}\rangle) \mid \langle\text{atom}\rangle \mid \langle\text{compound}\rangle \\
\langle\text{atom}\rangle &\rightarrow \langle\text{constant}\rangle \mid \langle\text{variable}\rangle \\
\langle\text{compound}\rangle &\rightarrow \langle\text{negation}\rangle \mid \langle\text{conjunction}\rangle \mid \langle\text{disjunction}\rangle \\
\langle\text{negation}\rangle &\rightarrow \neg\langle\text{nf}\rangle \\
\langle\text{nf}\rangle &\rightarrow \langle\text{atom}\rangle \mid \langle\text{negation}\rangle \mid (\langle\text{wff}\rangle) \\
\langle\text{conjunction}\rangle &\rightarrow \langle\text{cf}\rangle \wedge \langle\text{cf}\rangle \\
\langle\text{cf}\rangle &\rightarrow \langle\text{atom}\rangle \mid \langle\text{negation}\rangle \mid \langle\text{conjunction}\rangle \mid (\langle\text{wff}\rangle) \\
\langle\text{disjunction}\rangle &\rightarrow \langle\text{df}\rangle \vee \langle\text{df}\rangle \\
\langle\text{df}\rangle &\rightarrow \langle\text{atom}\rangle \mid \langle\text{negation}\rangle \mid \langle\text{disjunction}\rangle \mid (\langle\text{wff}\rangle) \\
\langle\text{constant}\rangle &\rightarrow \mathbf{T} \mid \mathbf{F} \\
\langle\text{variable}\rangle &\rightarrow \langle\text{letter}\rangle \mid \langle\text{variable}\rangle' \\
\langle\text{letter}\rangle &\rightarrow p \mid q \mid r \mid s
\end{aligned}$$

The following are the productions of this grammar:—

$$\begin{aligned}
\langle\text{wff}\rangle &\rightarrow (\langle\text{wff}\rangle) \\
\langle\text{wff}\rangle &\rightarrow \langle\text{atom}\rangle \\
\langle\text{wff}\rangle &\rightarrow \langle\text{compound}\rangle \\
\langle\text{atom}\rangle &\rightarrow \langle\text{constant}\rangle \\
\langle\text{atom}\rangle &\rightarrow \langle\text{variable}\rangle \\
\langle\text{compound}\rangle &\rightarrow \langle\text{negation}\rangle
\end{aligned}$$

$$
\begin{aligned}
\langle \text{compound} \rangle &\to \langle \text{conjunction} \rangle \\
\langle \text{compound} \rangle &\to \langle \text{disjunction} \rangle \\
\langle \text{negation} \rangle &\to \neg \langle \text{nf} \rangle \\
\langle \text{nf} \rangle &\to \langle \text{atom} \rangle \\
\langle \text{nf} \rangle &\to \langle \text{negation} \rangle \\
\langle \text{nf} \rangle &\to (\langle \text{wff} \rangle) \\
\langle \text{conjunction} \rangle &\to \langle \text{cf} \rangle \wedge \langle \text{cf} \rangle \\
\langle \text{cf} \rangle &\to \langle \text{atom} \rangle \\
\langle \text{cf} \rangle &\to \langle \text{negation} \rangle \\
\langle \text{cf} \rangle &\to \langle \text{conjunction} \rangle \\
\langle \text{cf} \rangle &\to (\langle \text{wff} \rangle) \\
\langle \text{disjunction} \rangle &\to \langle \text{df} \rangle \vee \langle \text{df} \rangle \\
\langle \text{df} \rangle &\to \langle \text{atom} \rangle \\
\langle \text{df} \rangle &\to \langle \text{negation} \rangle \\
\langle \text{df} \rangle &\to \langle \text{disjunction} \rangle \\
\langle \text{df} \rangle &\to (\langle \text{wff} \rangle) \\
\langle \text{constant} \rangle &\to \mathbf{T} \\
\langle \text{constant} \rangle &\to \mathbf{F} \\
\langle \text{variable} \rangle &\to \langle \text{variable} \rangle' \\
\langle \text{variable} \rangle &\to \langle \text{letter} \rangle \\
\langle \text{letter} \rangle &\to p \\
\langle \text{letter} \rangle &\to q \\
\langle \text{letter} \rangle &\to r \\
\langle \text{letter} \rangle &\to s
\end{aligned}
$$

The well-formed formulae of the Propositional Calculus are those words F over the alphabet

$$\{\neg, \wedge, \vee, (,), \mathbf{T}, \mathbf{F}, p, q, r, s, '\}$$

for which $\langle \text{wff} \rangle \stackrel{*}{\Rightarrow} F$.

Example We verify that the formula

$$(p \wedge r') \vee ((\neg r'') \wedge q \wedge \neg r''')$$

is a well-formed formula of the Propositional Calculus. This formula may be obtained from the nonterminal $\langle \text{wff} \rangle$ by applying successive productions as

follows:

$$
\begin{aligned}
\langle \text{wff} \rangle &\Rightarrow \langle \text{compound} \rangle \\
&\Rightarrow \langle \text{disjunction} \rangle \\
&\Rightarrow \langle \text{df} \rangle \vee \langle \text{df} \rangle \\
&\Rightarrow (\langle \text{wff} \rangle) \vee \langle \text{df} \rangle \\
&\Rightarrow (\langle \text{compound} \rangle) \vee \langle \text{df} \rangle \\
&\Rightarrow (\langle \text{conjunction} \rangle) \vee \langle \text{df} \rangle \\
&\Rightarrow (\langle \text{cf} \rangle \wedge \langle \text{cf} \rangle) \vee \langle \text{df} \rangle \\
&\Rightarrow (\langle \text{atom} \rangle \wedge \langle \text{cf} \rangle) \vee \langle \text{df} \rangle \\
&\Rightarrow (\langle \text{variable} \rangle \wedge \langle \text{cf} \rangle) \vee \langle \text{df} \rangle \\
&\Rightarrow (\langle \text{letter} \rangle \wedge \langle \text{cf} \rangle) \vee \langle \text{df} \rangle \\
&\Rightarrow (p \wedge \langle \text{cf} \rangle) \vee \langle \text{df} \rangle \\
&\Rightarrow (p \wedge \langle \text{atom} \rangle) \vee \langle \text{df} \rangle \\
&\Rightarrow (p \wedge \langle \text{variable} \rangle) \vee \langle \text{df} \rangle \\
&\Rightarrow (p \wedge \langle \text{variable} \rangle') \vee \langle \text{df} \rangle \\
&\Rightarrow (p \wedge \langle \text{letter} \rangle') \vee \langle \text{df} \rangle \\
&\Rightarrow (p \wedge r') \vee \langle \text{df} \rangle \\
&\Rightarrow (p \wedge r') \vee (\langle \text{wff} \rangle) \\
&\Rightarrow (p \wedge r') \vee (\langle \text{compound} \rangle) \\
&\Rightarrow (p \wedge r') \vee (\langle \text{conjunction} \rangle) \\
&\Rightarrow (p \wedge r') \vee (\langle \text{cf} \rangle \wedge \langle \text{cf} \rangle) \\
&\Rightarrow (p \wedge r') \vee ((\langle \text{wff} \rangle) \wedge \langle \text{cf} \rangle) \\
&\Rightarrow (p \wedge r') \vee ((\langle \text{compound} \rangle) \wedge \langle \text{cf} \rangle) \\
&\Rightarrow (p \wedge r') \vee ((\langle \text{negation} \rangle) \wedge \langle \text{cf} \rangle) \\
&\Rightarrow (p \wedge r') \vee ((\neg \langle \text{nf} \rangle) \wedge \langle \text{cf} \rangle) \\
&\Rightarrow (p \wedge r') \vee ((\neg \langle \text{atom} \rangle) \wedge \langle \text{cf} \rangle) \\
&\Rightarrow (p \wedge r') \vee ((\neg \langle \text{variable} \rangle) \wedge \langle \text{cf} \rangle) \\
&\Rightarrow (p \wedge r') \vee ((\neg \langle \text{variable} \rangle') \wedge \langle \text{cf} \rangle) \\
&\Rightarrow (p \wedge r') \vee ((\neg \langle \text{variable} \rangle'') \wedge \langle \text{cf} \rangle) \\
&\Rightarrow (p \wedge r') \vee ((\neg \langle \text{letter} \rangle'') \wedge \langle \text{cf} \rangle) \\
&\Rightarrow (p \wedge r') \vee ((\neg r'') \wedge \langle \text{cf} \rangle) \\
&\Rightarrow (p \wedge r') \vee ((\neg r'') \wedge \langle \text{conjunction} \rangle) \\
&\Rightarrow (p \wedge r') \vee ((\neg r'') \wedge \langle \text{cf} \rangle \wedge \langle \text{cf} \rangle) \\
&\Rightarrow (p \wedge r') \vee ((\neg r'') \wedge \langle \text{atom} \rangle \wedge \langle \text{cf} \rangle)
\end{aligned}
$$

$$
\begin{aligned}
&\Rightarrow\ (p \wedge r') \vee ((\neg r'') \wedge \langle \text{variable} \rangle \wedge \langle \text{cf} \rangle)\\
&\Rightarrow\ (p \wedge r') \vee ((\neg r'') \wedge \langle \text{letter} \rangle \wedge \langle \text{cf} \rangle)\\
&\Rightarrow\ (p \wedge r') \vee ((\neg r'') \wedge q \wedge \langle \text{cf} \rangle)\\
&\Rightarrow\ (p \wedge r') \vee ((\neg r'') \wedge q \wedge \langle \text{negation} \rangle)\\
&\Rightarrow\ (p \wedge r') \vee ((\neg r'') \wedge q \wedge \neg\langle \text{nf} \rangle)\\
&\Rightarrow\ (p \wedge r') \vee ((\neg r'') \wedge q \wedge \neg\langle \text{atom} \rangle)\\
&\Rightarrow\ (p \wedge r') \vee ((\neg r'') \wedge q \wedge \neg\langle \text{variable} \rangle)\\
&\Rightarrow\ (p \wedge r') \vee ((\neg r'') \wedge q \wedge \neg\langle \text{variable} \rangle')\\
&\Rightarrow\ (p \wedge r') \vee ((\neg r'') \wedge q \wedge \neg\langle \text{variable} \rangle'')\\
&\Rightarrow\ (p \wedge r') \vee ((\neg r'') \wedge q \wedge \neg\langle \text{variable} \rangle''')\\
&\Rightarrow\ (p \wedge r') \vee ((\neg r'') \wedge q \wedge \neg\langle \text{letter} \rangle''')\\
&\Rightarrow\ (p \wedge r') \vee ((\neg r'') \wedge q \wedge \neg r''')
\end{aligned}
$$

Remark The grammar we have constructed to describe the well-formed formulae of the Propositional Calculus generates formulae such as $(((p \wedge q)))$ and $p \vee ((((((q))))))$ which are unambiguous but which contain superfluous parentheses which serve no useful purpose. It is possible to modify the grammar to ensure that the modified grammar does not generate formulae with such superfluous parentheses. In particular we can ensure that a formula generated by the modified grammar is never enclosed within parentheses, that no Boolean constant or variable within the formula is enclosed by itself within parentheses, and that no subformula is enclosed by itself within two or more sets of parentheses. This modified grammar is expressed in Backus-Naur form as follows:—

$$
\begin{aligned}
\langle \text{wff} \rangle &\ \rightarrow\ \langle \text{atom} \rangle \mid \langle \text{compound} \rangle\\
\langle \text{atom} \rangle &\ \rightarrow\ \langle \text{constant} \rangle \mid \langle \text{variable} \rangle\\
\langle \text{compound} \rangle &\ \rightarrow\ \langle \text{negation} \rangle \mid \langle \text{conjunction} \rangle \mid \langle \text{disjunction} \rangle\\
\langle \text{negation} \rangle &\ \rightarrow\ \neg\langle \text{nf} \rangle\\
\langle \text{nf} \rangle &\ \rightarrow\ \langle \text{atom} \rangle \mid \langle \text{negation} \rangle \mid (\langle \text{compound} \rangle)\\
\langle \text{conjunction} \rangle &\ \rightarrow\ \langle \text{cf} \rangle \wedge \langle \text{cf} \rangle\\
\langle \text{cf} \rangle &\ \rightarrow\ \langle \text{atom} \rangle \mid \langle \text{negation} \rangle \mid \langle \text{conjunction} \rangle \mid (\langle \text{compound} \rangle)\\
\langle \text{disjunction} \rangle &\ \rightarrow\ \langle \text{df} \rangle \vee \langle \text{df} \rangle\\
\langle \text{df} \rangle &\ \rightarrow\ \langle \text{atom} \rangle \mid \langle \text{negation} \rangle \mid \langle \text{disjunction} \rangle \mid (\langle \text{compound} \rangle)\\
\langle \text{constant} \rangle &\ \rightarrow\ \mathbf{T} \mid \mathbf{F}\\
\langle \text{variable} \rangle &\ \rightarrow\ \langle \text{letter} \rangle \mid \langle \text{variable} \rangle'\\
\langle \text{letter} \rangle &\ \rightarrow\ p \mid q \mid r \mid s
\end{aligned}
$$

3.4 Context-Free Grammars

We have discussed examples of context-free grammars. We now present and discuss a formal definition of such grammars.

Definition A *context-free grammar* $(V, A, \langle S \rangle, P)$ consists of a finite set V, a subset A of V, an element $\langle S \rangle$ of $V \setminus A$, and a finite subset P of $(V \setminus A) \times V^*$.

Let $(V, A, \langle S \rangle, P)$ be a context-free grammar. The elements of A are referred to as *terminals*. Let $N = V \setminus A$. The elements of N are referred to as *nonterminals*. The nonterminal $\langle S \rangle$ is the *start symbol*. The set N of nonterminals is non-empty since $\langle S \rangle \in N$.

The finite set P specifies the *productions* of the grammar. An element of P is an ordered pair of the form $(\langle T \rangle, w)$ where $\langle T \rangle \in N$ is a nonterminal and $w \in V^*$ is a word over the alphabet V (i.e, a finite string, where each constituent of the string is either a terminal or a nonterminal). We denote by

$$\langle T \rangle \to w$$

the production specified by an ordered pair $(\langle T \rangle, w)$ belonging to the set P.

Definition Let w' and w'' be words over the alphabet V. We say that w' *directly yields* w'' if there exist words u and v over the alphabet V and a production $\langle T \rangle \to w$ of the grammar such that $w' = u\langle T \rangle v$ and $w'' = uwv$. (Either or both of the words u and v may be the empty word.)

We see therefore that a word w' over the alphabet V directly yields another such word w'' if and only if there exists a production $\langle T \rangle \to w$ in the grammar such that w'' may be obtained from w' by replacing a single occurrence of the nonterminal $\langle T \rangle$ within w' by the word w. If the word w' directly yields w'', then we denote this fact by writing

$$w' \Rightarrow w''.$$

Definition Let w' and w'' be words over the alphabet V. We say that w' *yields* w'' if either $w' = w''$ or else there exist words $w_0, w_1, \ldots, w_n$ over the alphabet V such that $w_0 = w'$, $w_n = w''$ and $w_{i-1} \Rightarrow w_i$ for all integers i between 1 and n.

If the word w' yields w'', then we denote this fact by writing

$$w' \stackrel{*}{\Rightarrow} w''.$$

Definition Let $(V, A, \langle S\rangle, P)$ be a context-free grammar. The *language* generated by this grammar is the subset L of A^* defined by

$$L = \{w \in A^* : \langle S\rangle \stackrel{*}{\Rightarrow} w\}.$$

We see therefore that the language L generated by a context-free grammar $(V, A, \langle S\rangle, P)$ consists of the set of all finite strings consisting entirely of terminals that may be obtained from the start symbol $\langle S\rangle$ by applying a finite sequence of productions of the grammar, where the application of a production causes a single nonterminal to be replaced by the string in V^* specified by that production.

3.5 Phrase Structure Grammars

There is a class of formal grammars that includes all context-free grammars. The grammars of this more general type are known as *phrase structure grammars*

Definition A *phrase structure grammar* $(V, A, \langle S\rangle, P)$ consists of a finite set V, a subset A of V, an element $\langle S\rangle$ of $V \setminus A$, and a finite subset P of $(V^* \setminus A^*) \times V^*$.

As in the case of context-free grammars, the elements of A are referred to as *terminals*, the elements of $V \setminus A$ are referred to as *nonterminals*, the nonterminal $\langle S\rangle$ is the *start symbol* and the elements of P specify the *productions* of the grammar. The production specified by an element (r, w) of P is denoted by $r \to w$. However the left hand side r of a production $r \to w$ in a phrase structure grammar need not consist solely of a single nonterminal, but may be a finite string r of elements of V^*, provided that this string r contains at least one nonterminal. (Note that V^* denotes the set of all finite words over the alphabet V whose elements are terminals and nonterminals, A^* denotes the set of all finite words consisting entirely of terminals, and thus $V^* \setminus A^*$ denotes the set of all finite words belonging to V^* which contain at least one nonterminal.)

Definition Let w' and w'' be words over the alphabet V. We say that w' *directly yields* w'' if there exist words u and v over the alphabet V and a production $r \to w$ such that $w' = urv$ and $w'' = uwv$. (Either or both of the words u and v may be the empty word.)

We see therefore that a word w' over the alphabet V directly yields another such word w'' if and only if there exists a production $r \to w$ in the

grammar such that w'' may be obtained from w' by replacing a single occurrence of r as a substring of w' by the string w. If the word w' directly yields w'', then we denote this fact by writing

$$w_1 \Rightarrow w_2.$$

Definition Let w' and w'' be words over the alphabet V. We say that w' *yields* w'' if either $w' = w''$ or else there exist words $w_0, w_1, \ldots, w_n$ over the alphabet w such that $w_0 = w'$, $w_n = w''$ and $w_{i-1} \Rightarrow w_i$ for all integers i between 1 and n.

If the word w' yields w'', then we denote this fact by writing

$$w' \stackrel{*}{\Rightarrow} w''.$$

Definition Let $(V, A, \langle S\rangle, P)$ be a phrase structure grammar. The *language* generated by this grammar is the subset L of A^* defined by

$$L = \{w \in A^* : \langle S\rangle \stackrel{*}{\Rightarrow} w\}.$$

3.6 Regular Languages

Definition Let A be a finite set, and let A^* be the set of words over the alphabet A. A subset L of A^* is said to be a *regular language* over the alphabet A if $L = L_m$ for some finite sequence $L_1, L_2, \ldots, L_m$ of subsets of A^* with the property that, for each integer i between 1 and m, the set L_i satisfies at least one of the following conditions:—

(i) L_i is a finite set;

(ii) $L_i = L_j^*$ for some integer j satisfying $1 \leq j < i$;

(iii) $L_i = L_j \circ L_k$ for some integers j and k satisfying $1 \leq j < i$, $1 \leq k < i$;

(iv) $L_i = L_j \cup L_k$ for some integers j and k satisfying $1 \leq j < i$, $1 \leq k < i$.

(Here $L_j \circ L_k$ denotes the set of all words over the alphabet A that are concatenations of the form $w'w''$ with $w' \in L_j$ and $w'' \in L_k$.)

Let A be a finite set, and let A^* be the set of all words over the alphabet A. The regular languages over the alphabet A constitute the smallest collection $\mathcal{C}$ of subsets of A^* which satisfies the following properties:—

(i) all finite subsets of A^* belong to $\mathcal{C}$;

(ii) if M is a subset of A^* belonging to $\mathcal{C}$ then so is M^*;

(iii) if M and N are subsets of A^* belonging to $\mathcal{C}$ then so is $M \circ N$.

(iv) if M and N are subsets of A^* belonging to $\mathcal{C}$ then so is $M \cup N$.

Indeed the collection of regular languages over the alphabet A has all four properties. Moreover any collection of languages over the alphabet A with these properties includes all regular languages. Indeed if L is a regular language then $L = L_m$ for some finite sequence $L_1, L_2, \ldots, L_m$ of subsets of A^* which each set L_i is either a finite set, or of the form L_j^* for some set L_j with $j < i$, or of the form $L_j \circ L_k$ for some sets L_j and L_k with $j < i$ and $k < i$, or of the form $L_j \cup L_k$ for some sets L_j and L_k with $j < i$ and $k < i$. It follows from this that if a collection $\mathcal{C}$ of subsets of A^* with the four properties given above contains L_j for all integers j satisfying $1 \leq j < i$ then it also contains L_i. Therefore all of the sets $L_1, L_2, \ldots, L_m$ must belong to the collection $\mathcal{C}$, and, in particular, the regular language L must belong to $\mathcal{C}$.

Example Let L be the set of decimal representations of integers. We give each integer a unique decimal representation in L, so that the decimal representation of any positive integer in L begins with a non-zero digit, the decimal representation of zero is '0', and the decimal representation of any negative number begins with a minus sign followed by a non-zero digit. The set L is a language over the alphabet A, where

$$A = \{-, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}.$$

Moreover $L = L_9$, where

$$\begin{aligned}
L_1 &= \{0\}, \\
L_2 &= \{1, 2, 3, 4, 5, 6, 7, 8, 9\}, \\
L_3 &= L_1 \cup L_2, \\
L_4 &= L_3^*, \\
L_5 &= L_2 \circ L_4, \\
L_6 &= \{-\}, \\
L_7 &= L_6 \circ L_5, \\
L_8 &= L_5 \cup L_7, \\
L_9 &= L_8 \cup L_1.
\end{aligned}$$

Note that L_3 is the set of decimal digits, L_4 is the set of finite strings of decimal digits (including the empty string), L_5 is the set of decimal representations of positive integers, L_7 is the set of decimal representations of

negative integers, and L_8 is the set of decimal representations of non-zero integers. We conclude that L is a regular language over the alphabet A.

The regular languages may be characterised as those languages that are generated by *regular grammars*. They may also be characterised as those languages that are recognized by *finite state acceptors*. We shall give below formal definitions of *regular grammars* and *finite state acceptors*.

3.7 Regular Grammars

Definition A context-free grammar is said to be a *regular grammar* if every production is of one of the three forms

$$\begin{array}{lrcl} \text{(i)} & \langle A \rangle & \rightarrow & b\langle B \rangle, \\ \text{(ii)} & \langle A \rangle & \rightarrow & b, \\ \text{(iii)} & \langle A \rangle & \rightarrow & \varepsilon, \end{array}$$

where $\langle A \rangle$ and $\langle B \rangle$ represent nonterminals, b represents a terminal, and ε denotes the empty word. A regular grammar is said to be in *normal form* if all its productions are of types (i) and (iii).

Lemma 3.2 *Any language generated by a regular grammar may be generated by a regular grammar in normal form.*

Proof Let L be a language over an alphabet A, and let $(V, A, \langle S \rangle, P)$ be a regular grammar generating the language L. We may construct a new regular grammar in normal form by first adding a nonterminal $\langle F \rangle$ that does not already belong to V, and then replacing any production which is of the form $\langle A \rangle \rightarrow b$ for some nonterminal $\langle A \rangle$ and terminal b by the pair of productions $\langle A \rangle \rightarrow b\langle F \rangle$ and $\langle F \rangle \rightarrow \varepsilon$. (Indeed the replacement of $\langle A \rangle$ by b in any word may be accomplished in two steps, by first replacing $\langle A \rangle$ by $b\langle F \rangle$ and then replacing $\langle F \rangle$ by the empty word ε.) The resultant regular grammar is in normal form and generates the same language as the given regular grammar. ∎

Example Let L be the set of decimal representations of integers, in which the most significant digit of a non-zero integer is non-zero, and in which zero is represented by '0'. L is a regular language over the alphabet A, where

$$A = \{-, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}.$$

This language is generated by the regular grammar whose nonterminals are $\langle S\rangle$, $\langle A\rangle$, $\langle B\rangle$ and whose productions are

$$\begin{array}{lcl}
\langle S\rangle & \to & -\langle A\rangle \\
\langle S\rangle & \to & 0 \\
\langle S\rangle & \to & 1\langle B\rangle \\
\langle S\rangle & \to & 2\langle B\rangle \\
 & \vdots & \\
\langle S\rangle & \to & 9\langle B\rangle \\
\langle A\rangle & \to & 1\langle B\rangle \\
\langle A\rangle & \to & 2\langle B\rangle \\
 & \vdots & \\
\langle A\rangle & \to & 9\langle B\rangle \\
\langle B\rangle & \to & 0\langle B\rangle \\
\langle B\rangle & \to & 1\langle B\rangle \\
\langle B\rangle & \to & 2\langle B\rangle \\
 & \vdots & \\
\langle B\rangle & \to & 9\langle B\rangle \\
\langle B\rangle & \to & \varepsilon
\end{array}$$

This regular grammar is not in normal form, but to obtain a regular grammar in normal form it suffices to introduce a new nonterminal $\langle C\rangle$, and replace the production $\langle S\rangle \to 0$ by the two productions

$$\begin{array}{lcl}
\langle S\rangle & \to & 0\langle C\rangle \\
\langle C\rangle & \to & \varepsilon
\end{array}$$

3.8 Finite State Acceptors

Definition A *finite state acceptor* (S, A, i, t, F) consists of finite sets S and A, an element i of S, a function $t\colon S \times A \to S$ from $S \times A$ to S and a subset F of S. The set S is the *set of states*, the set A is the *input alphabet*, the element i of S is the *starting state*, the function $t\colon S \times A \to S$ is the *transition mapping* and F is the set of *finishing states*.

A finite state acceptor is a particular type of *finite state machine*. Such a machine has a finite number of internal states. Data is input discretely,

and each datum causes the machine either to remain in the same internal state or else to make a transition to some other internal state determined solely by the current state and the input datum. In a finite state acceptor (S, A, i, t, F) the set S represents of the internal states of the machine and is finite, and each datum is an element of the input alphabet A. The machine is initially in the starting state i. The transition function t specifies how the internal state of the machine changes on inputting a datum: if the machine is currently in state s, and if the input datum is a, then the internal state of the machine becomes s' where $s' = t(s, a)$. Any finite state acceptor determines a language L over the alphabet A, consisting of those words $a_1 a_2 \dots a_n$ which, when the elements $a_1, a_2, \dots, a_n$ of A are successively input, cause the machine initially in the starting state to end up in one of the finishing states specified by the subset F of S.

Definition Let (S, A, i, t, F) be a finite state acceptor, and let A^* denote the set of words over the input alphabet A. A word $a_1 a_2 \dots a_n$ of length n over the alphabet A is said to be *recognized* or *accepted* by the finite state acceptor if there are states $s_0, s_1, s_2, \dots, s_n$ belonging to S such that $s_0 = i$, $s_n \in F$, and $s_i = t(s_{i-1}, a_i)$ for each integer i between 1 and n.

Definition Let (S, A, i, t, F) be a finite state acceptor. A language L over the alphabet A is said to be *recognized* or *accepted* by the finite state acceptor if L is the set consisting of all words recognized by the finite state acceptor.

It can be proved that a language over some alphabet A is a regular language if and only if that language is recognized by some finite state acceptor with input alphabet A.

Example Let L be the set of decimal representations of integers, in which the most significant digit of a non-zero integer is non-zero, and in which zero is represented by '0'. L is a regular language over the alphabet A, where

$$A = \{-, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}.$$

Let $S = \{i, a, b, c, e\}$, let $F = \{b, c\}$, and let $t\colon S \times A \to S$ be the transition mapping determined by the following *transition table*:—

	-	0	1	2	3	4	5	6	7	8	9
i	a	c	b	b	b	b	b	b	b	b	b
a	e	e	b	b	b	b	b	b	b	b	b
b	e	b	b	b	b	b	b	b	b	b	b
c	e	e	e	e	e	e	e	e	e	e	e
e	e	e	e	e	e	e	e	e	e	e	e

(Here the value of $t(s, \alpha)$ for a state s and character α is listed in the row of the table labelled by s and the column labelled by α.) Then (S, A, i, t, F) is a finite state acceptor for the regular language L.

Problems

1. Devise a context-free grammar to generate the language over the alphabet $\{0, 1\}$ consisting of the strings

$$01,\ \ 0011,\ \ 000111,\ \ 00001111, \ldots$$

(i.e., consisting of m zeros, for some non-negative integer m, followed by m ones). You should specify the nonterminals of the grammar, the start symbol and the productions of the grammar.

2. Consider the context free-grammar over the alphabet $\{x, y, (,)\}$ with nonterminals $\langle S\rangle$ and $\langle A\rangle$, start symbol $\langle S\rangle$ and productions

$$\langle S\rangle \to \langle A\rangle\langle A\rangle, \quad \langle A\rangle \to (\langle S\rangle), \quad \langle A\rangle \to x, \quad \langle A\rangle \to y.$$

Show that the string $(x(yx))y$ belongs to the language over the alphabet $\{x, y, (,)\}$ generated by this grammar. Does the string $(x(xy$ belong to this language? [Briefly justify your answer.]

3. Describe the formal language over the alphabet $\{0, 1\}$ generated by the context-free grammar whose only non-terminal is $\langle S\rangle$, whose start symbol is $\langle S\rangle$ and whose productions are the following:

$$\begin{aligned} \langle S\rangle &\to 0 \\ \langle S\rangle &\to 0\langle S\rangle \\ \langle S\rangle &\to \langle S\rangle 1 \end{aligned}$$

Is this context-free grammar a regular grammar?

4. Describe the language over the alphabet $\{0, 1\}$ generated by the context-free grammar whose non-terminals are $\langle S\rangle$ and $\langle A\rangle$, whose start symbol is $\langle S\rangle$ and whose prodcutions are

$$\langle S\rangle \to \langle S\,\rangle\langle A\rangle, \quad \langle S\rangle \to 1, \quad \langle A\rangle \to 01.$$

Is the context-free grammar a regular grammar?

5. Describe the formal language over the alphabet $\{0, 1\}$ generated by the context-free grammar whose only non-terminal is $\langle S \rangle$, whose start symbol is $\langle S \rangle$ and whose productions are the following:

$$\begin{aligned} \langle S \rangle &\rightarrow 0 \\ \langle S \rangle &\rightarrow 00\langle S \rangle \\ \langle S \rangle &\rightarrow \langle S \rangle 11 \end{aligned}$$

   Is this context-free grammar a regular grammar?

6. Describe the formal language over the alphabet $\{0, 1\}$ generated by the context-free grammar whose only non-terminal is $\langle S \rangle$, whose start symbol is $\langle S \rangle$ and whose productions are the following:

$$\begin{aligned} \langle S \rangle &\rightarrow 0 \\ \langle S \rangle &\rightarrow 1\langle S \rangle 1 \end{aligned}$$

   (i.e., describe the structure of the binary strings generated by the grammar). Is this context-free grammar a regular grammar?

7. Construct a regular grammar that generates the language L over the alphabet $\{0, 1\}$, where

$$L = \{1, 1000, 1000000, 1000000000, \ldots\},$$

   so that a string of binary digits belongs to L if and only if it consists of the digit 1 followed by a string of $3n$ zeroes, for some non-negative integer n. You should specify your formal grammar in Backus-Naur form.

8. (a) Devise a regular grammar to generate the language over the alphabet $\{a, (,), 0, 1\}$ consisting of all strings such as `a(001)` and `a(1001010)` in which the initial substring `a(` is followed by a non-empty string of binary digits, which is followed by the character `)`.

   (b) Devise a finite state acceptor that accepts (i.e., determines) the language described in (a). You should specify the states of the machine, the start state, the finishing state(s), and the transition table that defines the machine.

9. (a) Devise a finite state acceptor that accepts (i.e., determines) the language over the alphabet $\{x, y, z\}$ consisting of all strings

$$xyz, xyzxyz, xyzxyzxyz, xyzxyzxyzxyz, \ldots$$

that are the concatenation of n copies of the string xyz for some positive integer n. You should specify the states of the machine, the start state, the finishing state(s), and the transition table that defines the machine.

(b) Devise a regular grammar to generate the language described in (a).

10. (a) Give the definition of a finite state acceptor that accepts (or determines) the language over the alphabet $\{a, b, c\}$ consisting of all finite strings, such as *ab*, *aabbb*, *aaaaab*, *abc*, *aabbbc*, that consist of one or more occurrences of the character a, followed by one or more occurrences of the character b, optionally followed by a single occurrence of the character c. You should specify the states of the machine, the start state, the finishing state or states, and the transition table that defines the machine.

(b) Devise a regular grammar to generate the language described in (a).

11. (a) Let L be the language over the alphabet $\{0, 1\}$ consisting of those finite strings of binary digits in which neither 010 nor 101 occurs as a substring. Give the description of a finite state acceptor for the language L, specifying the starting state, the finishing state or states, and the transition table for this finite state acceptor.

(b) Construct a regular context-free grammar that generates the langle L described in (a).

12. (a) Give the specification of a finite state acceptor for the language over the alphabet $\{a, b, c\}$ consisting of all finite strings, such as *aabbc*, *aabbbc* and *aaabbbc*, that consist of two or more occurrences of the character a, followed by two or more occurrences of the character b, followed by a single occurrence of the character c. You should in particular specify the starting state, the finishing state or states, and the transition table for this finite state acceptor.

(b) Give the specification of a regular grammar to generate the language over the alphabet $\{a, b, c\}$ that was defined in (a).

13. (a) Give the specification of a finite state acceptor that accepts the language over the alphabet $\{0, 1\}$ consisting of all words where the number of occurrences of the digit 0 within the word is a multiple of 3. (In particular you should specify the set of states, the starting state, the finishing states, and the transition table that determines the transition function of the finite state acceptor.)

    (b) Devise a regular grammar to generate the language specified in (a). (In particular, you should specify the nonterminals, the start state and the productions of the grammar.)