Formal Languages, Automata and Codes

Oleg Gutik

Lecture 2

2 THREE BASIC CONCEPTS

Three fundamental ideas are the major themes of this course: **languages**, **grammars**, and **automata**. In the course we shall explore many results about these concepts and about their relationship to each other. First, we must understand the meaning of the terms.

Three fundamental ideas are the major themes of this course: **languages**, **grammars**, and **automata**. In the course we shall explore many results about these concepts and about their relationship to each other. First, we must understand the meaning of the terms.

Three fundamental ideas are the major themes of this course: **languages**, **grammars**, and **automata**. In the course we shall explore many results about these concepts and about their relationship to each other. First, we must understand the meaning of the terms.

Three fundamental ideas are the major themes of this course: **languages**, **grammars**, and **automata**. In the course we shall explore many results about these concepts and about their relationship to each other. First, we must understand the meaning of the terms.

Three fundamental ideas are the major themes of this course: **languages**, **grammars**, and **automata**. In the course we shall explore many results about these concepts and about their relationship to each other. First, we must understand the meaning of the terms.

Three fundamental ideas are the major themes of this course: **languages**, **grammars**, and **automata**. In the course we shall explore many results about these concepts and about their relationship to each other. First, we must understand the meaning of the terms.

We are all familiar with the notion of natural languages, such as English, Ukrainian, Polish, German, and French. Still, most of us would probably find it difficult to say exactly what the word "language" means. Dictionaries define the term informally as a system suitable for the expression of certain ideas, facts, or concepts, including a set of symbols and rules for their manipulation. While this gives us an intuitive idea of what a language is, it is not sufficient as a definition for the study of formal languages. We need a precise definition for the term.

We start with a finite, nonempty set Σ of symbols, called an *alphabet*. From the individual symbols we construct *strings* (or *words*), which are finite sequences of symbols from the alphabet. For example, if the alphabet $\Sigma = \{a, b\}$, then $abab$ and $aaabbba$ are strings on Σ. With few exceptions, we will use lowercase letters $a, b, c, \ldots$ for elements of Σ and $u, v, w, \ldots$ for string names. We shall write, for example,

$$w = abaaa$$

to indicate that the string named w has the specific value $abaaa$.

We are all familiar with the notion of natural languages, such as English, Ukrainian, Polish, German, and French. Still, most of us would probably find it difficult to say exactly what the word "language" means. Dictionaries define the term informally as a system suitable for the expression of certain ideas, facts, or concepts, including a set of symbols and rules for their manipulation. While this gives us an intuitive idea of what a language is, it is not sufficient as a definition for the study of formal languages. We need a precise definition for the term.

We start with a finite, nonempty set Σ of symbols, called an *alphabet*. From the individual symbols we construct *strings* (or *words*), which are finite sequences of symbols from the alphabet. For example, if the alphabet $\Sigma = \{a, b\}$, then $abab$ and $aaabbba$ are strings on Σ. With few exceptions, we will use lowercase letters $a, b, c, \ldots$ for elements of Σ and $u, v, w, \ldots$ for string names. We shall write, for example,

$$w = abaaa$$

to indicate that the string named w has the specific value $abaaa$.

We are all familiar with the notion of natural languages, such as English, Ukrainian, Polish, German, and French. Still, most of us would probably find it difficult to say exactly what the word "language" means. Dictionaries define the term informally as a system suitable for the expression of certain ideas, facts, or concepts, including a set of symbols and rules for their manipulation. While this gives us an intuitive idea of what a language is, it is not sufficient as a definition for the study of formal languages. We need a precise definition for the term.

We start with a finite, nonempty set Σ of symbols, called an *alphabet*. From the individual symbols we construct *strings* (or *words*), which are finite sequences of symbols from the alphabet. For example, if the alphabet $\Sigma = \{a, b\}$, then $abab$ and $aaabbba$ are strings on Σ. With few exceptions, we will use lowercase letters $a, b, c, \ldots$ for elements of Σ and $u, v, w, \ldots$ for string names. We shall write, for example,

$$w = abaaa$$

to indicate that the string named w has the specific value $abaaa$.

We are all familiar with the notion of natural languages, such as English, Ukrainian, Polish, German, and French. Still, most of us would probably find it difficult to say exactly what the word "language" means. Dictionaries define the term informally as a system suitable for the expression of certain ideas, facts, or concepts, including a set of symbols and rules for their manipulation. While this gives us an intuitive idea of what a language is, it is not sufficient as a definition for the study of formal languages. We need a precise definition for the term.

We start with a finite, nonempty set Σ of symbols, called an *alphabet*. From the individual symbols we construct *strings* (or *words*), which are finite sequences of symbols from the alphabet. For example, if the alphabet $\Sigma = \{a, b\}$, then $abab$ and $aaabbba$ are strings on Σ. With few exceptions, we will use lowercase letters $a, b, c, \ldots$ for elements of Σ and $u, v, w, \ldots$ for string names. We shall write, for example,

$$w = abaaa$$

to indicate that the string named w has the specific value $abaaa$.

We are all familiar with the notion of natural languages, such as English, Ukrainian, Polish, German, and French. Still, most of us would probably find it difficult to say exactly what the word "language" means. Dictionaries define the term informally as a system suitable for the expression of certain ideas, facts, or concepts, including a set of symbols and rules for their manipulation. While this gives us an intuitive idea of what a language is, it is not sufficient as a definition for the study of formal languages. We need a precise definition for the term.

We start with a finite, nonempty set Σ of symbols, called an *alphabet*. From the individual symbols we construct *strings* (or *words*), which are finite sequences of symbols from the alphabet. For example, if the alphabet $\Sigma = \{a, b\}$, then $abab$ and $aaabbba$ are strings on Σ. With few exceptions, we will use lowercase letters $a, b, c, \ldots$ for elements of Σ and $u, v, w, \ldots$ for string names. We shall write, for example,

$$w = abaaa$$

to indicate that the string named w has the specific value $abaaa$.

We are all familiar with the notion of natural languages, such as English, Ukrainian, Polish, German, and French. Still, most of us would probably find it difficult to say exactly what the word "language" means. Dictionaries define the term informally as a system suitable for the expression of certain ideas, facts, or concepts, including a set of symbols and rules for their manipulation. While this gives us an intuitive idea of what a language is, it is not sufficient as a definition for the study of formal languages. We need a precise definition for the term.

We start with a finite, nonempty set Σ of symbols, called an *alphabet*. From the individual symbols we construct *strings* (or *words*), which are finite sequences of symbols from the alphabet. For example, if the alphabet $\Sigma = \{a, b\}$, then $abab$ and $aaabbba$ are strings on Σ. With few exceptions, we will use lowercase letters $a, b, c, \ldots$ for elements of Σ and $u, v, w, \ldots$ for string names. We shall write, for example,

$$w = abaaa$$

to indicate that the string named w has the specific value $abaaa$.

We are all familiar with the notion of natural languages, such as English, Ukrainian, Polish, German, and French. Still, most of us would probably find it difficult to say exactly what the word “language” means. Dictionaries define the term informally as a system suitable for the expression of certain ideas, facts, or concepts, including a set of symbols and rules for their manipulation. While this gives us an intuitive idea of what a language is, it is not sufficient as a definition for the study of formal languages. We need a precise definition for the term.

We start with a finite, nonempty set Σ of symbols, called an *alphabet*. From the individual symbols we construct *strings* (or *words*), which are finite sequences of symbols from the alphabet. For example, if the alphabet $\Sigma = \{a, b\}$, then $abab$ and $aaabbba$ are strings on Σ. With few exceptions, we will use lowercase letters $a, b, c, \ldots$ for elements of Σ and $u, v, w, \ldots$ for string names. We shall write, for example,

$$w = abaaa$$

to indicate that the string named w has the specific value $abaaa$.

We are all familiar with the notion of natural languages, such as English, Ukrainian, Polish, German, and French. Still, most of us would probably find it difficult to say exactly what the word “language” means. Dictionaries define the term informally as a system suitable for the expression of certain ideas, facts, or concepts, including a set of symbols and rules for their manipulation. While this gives us an intuitive idea of what a language is, it is not sufficient as a definition for the study of formal languages. We need a precise definition for the term.

We start with a finite, nonempty set Σ of symbols, called an *alphabet*. From the individual symbols we construct *strings* (or *words*), which are finite sequences of symbols from the alphabet. For example, if the alphabet $\Sigma = \{a, b\}$, then $abab$ and $aaabbba$ are strings on Σ. With few exceptions, we will use lowercase letters $a, b, c, \ldots$ for elements of Σ and $u, v, w, \ldots$ for string names. We shall write, for example,

$$w = abaaa$$

to indicate that the string named w has the specific value $abaaa$.

We are all familiar with the notion of natural languages, such as English, Ukrainian, Polish, German, and French. Still, most of us would probably find it difficult to say exactly what the word "language" means. Dictionaries define the term informally as a system suitable for the expression of certain ideas, facts, or concepts, including a set of symbols and rules for their manipulation. While this gives us an intuitive idea of what a language is, it is not sufficient as a definition for the study of formal languages. We need a precise definition for the term.

We start with a finite, nonempty set Σ of symbols, called an *alphabet*. From the individual symbols we construct *strings* (or *words*), which are finite sequences of symbols from the alphabet. For example, if the alphabet $\Sigma = \{a, b\}$, then $abab$ and $aaabbba$ are strings on Σ. With few exceptions, we will use lowercase letters $a, b, c, \ldots$ for elements of Σ and $u, v, w, \ldots$ for string names. We shall write, for example,

$$w = abaaa$$

to indicate that the string named w has the specific value $abaaa$.

We are all familiar with the notion of natural languages, such as English, Ukrainian, Polish, German, and French. Still, most of us would probably find it difficult to say exactly what the word "language" means. Dictionaries define the term informally as a system suitable for the expression of certain ideas, facts, or concepts, including a set of symbols and rules for their manipulation. While this gives us an intuitive idea of what a language is, it is not sufficient as a definition for the study of formal languages. We need a precise definition for the term.

We start with a finite, nonempty set Σ of symbols, called an *alphabet*. From the individual symbols we construct *strings* (or *words*), which are finite sequences of symbols from the alphabet. For example, if the alphabet $\Sigma = \{a, b\}$, then $abab$ and $aaabbba$ are strings on Σ. With few exceptions, we will use lowercase letters $a, b, c, \ldots$ for elements of Σ and $u, v, w, \ldots$ for string names. We shall write, for example,

$$w = abaaa$$

to indicate that the string named w has the specific value $abaaa$.

We are all familiar with the notion of natural languages, such as English, Ukrainian, Polish, German, and French. Still, most of us would probably find it difficult to say exactly what the word "language" means. Dictionaries define the term informally as a system suitable for the expression of certain ideas, facts, or concepts, including a set of symbols and rules for their manipulation. While this gives us an intuitive idea of what a language is, it is not sufficient as a definition for the study of formal languages. We need a precise definition for the term.

We start with a finite, nonempty set Σ of symbols, called an *alphabet*. From the individual symbols we construct *strings* (or *words*), which are finite sequences of symbols from the alphabet. For example, if the alphabet $\Sigma = \{a, b\}$, then $abab$ and $aaabbba$ are strings on Σ. With few exceptions, we will use lowercase letters $a, b, c, \ldots$ for elements of Σ and $u, v, w, \ldots$ for string names. We shall write, for example,

$$w = abaaa$$

to indicate that the string named w has the specific value $abaaa$.

We are all familiar with the notion of natural languages, such as English, Ukrainian, Polish, German, and French. Still, most of us would probably find it difficult to say exactly what the word "language" means. Dictionaries define the term informally as a system suitable for the expression of certain ideas, facts, or concepts, including a set of symbols and rules for their manipulation. While this gives us an intuitive idea of what a language is, it is not sufficient as a definition for the study of formal languages. We need a precise definition for the term.

We start with a finite, nonempty set Σ of symbols, called an *alphabet*. From the individual symbols we construct *strings* (or *words*), which are finite sequences of symbols from the alphabet. For example, if the alphabet $\Sigma = \{a, b\}$, then $abab$ and $aaabbba$ are strings on Σ. With few exceptions, we will use lowercase letters $a, b, c, \ldots$ for elements of Σ and $u, v, w, \ldots$ for string names. We shall write, for example,

$$w = abaaa$$

to indicate that the string named w has the specific value $abaaa$.

We are all familiar with the notion of natural languages, such as English, Ukrainian, Polish, German, and French. Still, most of us would probably find it difficult to say exactly what the word "language" means. Dictionaries define the term informally as a system suitable for the expression of certain ideas, facts, or concepts, including a set of symbols and rules for their manipulation. While this gives us an intuitive idea of what a language is, it is not sufficient as a definition for the study of formal languages. We need a precise definition for the term.

We start with a finite, nonempty set Σ of symbols, called an *alphabet*. From the individual symbols we construct *strings* (or *words*), which are finite sequences of symbols from the alphabet. For example, if the alphabet $\Sigma = \{a, b\}$, then $abab$ and $aaabbba$ are strings on Σ. With few exceptions, we will use lowercase letters $a, b, c, \ldots$ for elements of Σ and $u, v, w, \ldots$ for string names. We shall write, for example,

$$w = abaaa$$

to indicate that the string named w has the specific value $abaaa$.

We are all familiar with the notion of natural languages, such as English, Ukrainian, Polish, German, and French. Still, most of us would probably find it difficult to say exactly what the word "language" means. Dictionaries define the term informally as a system suitable for the expression of certain ideas, facts, or concepts, including a set of symbols and rules for their manipulation. While this gives us an intuitive idea of what a language is, it is not sufficient as a definition for the study of formal languages. We need a precise definition for the term.

We start with a finite, nonempty set Σ of symbols, called an *alphabet*. From the individual symbols we construct *strings* (or *words*), which are finite sequences of symbols from the alphabet. For example, if the alphabet $\Sigma = \{a, b\}$, then $abab$ and $aaabbba$ are strings on Σ. With few exceptions, we will use lowercase letters $a, b, c, \ldots$ for elements of Σ and $u, v, w, \ldots$ for string names. We shall write, for example,

$$w = abaaa$$

to indicate that the string named w has the specific value $abaaa$.

We are all familiar with the notion of natural languages, such as English, Ukrainian, Polish, German, and French. Still, most of us would probably find it difficult to say exactly what the word "language" means. Dictionaries define the term informally as a system suitable for the expression of certain ideas, facts, or concepts, including a set of symbols and rules for their manipulation. While this gives us an intuitive idea of what a language is, it is not sufficient as a definition for the study of formal languages. We need a precise definition for the term.

We start with a finite, nonempty set Σ of symbols, called an *alphabet*. From the individual symbols we construct *strings* (or *words*), which are finite sequences of symbols from the alphabet. For example, if the alphabet $\Sigma = \{a, b\}$, then $abab$ and $aaabbba$ are strings on Σ. With few exceptions, we will use lowercase letters $a, b, c, \ldots$ for elements of Σ and $u, v, w, \ldots$ for string names. We shall write, for example,

$$w = abaaa$$

to indicate that the string named w has the specific value $abaaa$.

We are all familiar with the notion of natural languages, such as English, Ukrainian, Polish, German, and French. Still, most of us would probably find it difficult to say exactly what the word "language" means. Dictionaries define the term informally as a system suitable for the expression of certain ideas, facts, or concepts, including a set of symbols and rules for their manipulation. While this gives us an intuitive idea of what a language is, it is not sufficient as a definition for the study of formal languages. We need a precise definition for the term.

We start with a finite, nonempty set Σ of symbols, called an *alphabet*. From the individual symbols we construct *strings* (or *words*), which are finite sequences of symbols from the alphabet. For example, if the alphabet $\Sigma = \{a, b\}$, then $abab$ and $aaabbba$ are strings on Σ. With few exceptions, we will use lowercase letters $a, b, c, \ldots$ for elements of Σ and $u, v, w, \ldots$ for string names. We shall write, for example,

$$w = abaaa$$

to indicate that the string named w has the specific value $abaaa$.

The concatenation of two strings w and v is the string obtained by appending the symbols of v to the right end of w, that is, if

$$w = a_1 a_2 \cdots a_n$$

and

$$v = b_1 b_2 \cdots b_m,$$

then the concatenation of w and v, denoted by wv, is

$$wv = a_1 a_2 \cdots a_n b_1 b_2 \cdots b_m.$$

The *reverse* of a string is obtained by writing the symbols in reverse order; if w is a string as shown above, then its reverse w^R is

$$w^R = a_n \cdots a_2 a_1.$$

The *length* of a string w, denoted by $|w|$, is the number of symbols in the string. We shall frequently need to refer to the *empty string*, which is a string with no symbols at all. It will be denoted by λ. The following simple relations

$$|\lambda| = 0,$$
$$\lambda w = w\lambda = w$$

hold for all string w.

The concatenation of two strings w and v is the string obtained by appending the symbols of v to the right end of w, that is, if

$$w = a_1 a_2 \cdots a_n$$

and

$$v = b_1 b_2 \cdots b_m,$$

then the concatenation of w and v, denoted by wv, is

$$wv = a_1 a_2 \cdots a_n b_1 b_2 \cdots b_m.$$

The *reverse* of a string is obtained by writing the symbols in reverse order; if w is a string as shown above, then its reverse w^R is

$$w^R = a_n \cdots a_2 a_1.$$

The *length* of a string w, denoted by $|w|$, is the number of symbols in the string. We shall frequently need to refer to the *empty string*, which is a string with no symbols at all. It will be denoted by λ. The following simple relations

$$|\lambda| = 0,$$
$$\lambda w = w\lambda = w$$

hold for all string w.

The concatenation of two strings w and v is the string obtained by appending the symbols of v to the right end of w, that is, if

$$w = a_1 a_2 \cdots a_n$$

and

$$v = b_1 b_2 \cdots b_m,$$

then the concatenation of w and v, denoted by wv, is

$$wv = a_1 a_2 \cdots a_n b_1 b_2 \cdots b_m.$$

The *reverse* of a string is obtained by writing the symbols in reverse order; if w is a string as shown above, then its reverse w^R is

$$w^R = a_n \cdots a_2 a_1.$$

The *length* of a string w, denoted by $|w|$, is the number of symbols in the string. We shall frequently need to refer to the *empty string*, which is a string with no symbols at all. It will be denoted by λ. The following simple relations

$$|\lambda| = 0,$$

$$\lambda w = w\lambda = w$$

hold for all string w.

The concatenation of two strings w and v is the string obtained by appending the symbols of v to the right end of w, that is, if

$$w = a_1 a_2 \cdots a_n$$

and

$$v = b_1 b_2 \cdots b_m,$$

then the concatenation of w and v, denoted by wv, is

$$wv = a_1 a_2 \cdots a_n b_1 b_2 \cdots b_m.$$

The *reverse* of a string is obtained by writing the symbols in reverse order; if w is a string as shown above, then its reverse w^R is

$$w^R = a_n \cdots a_2 a_1.$$

The *length* of a string w, denoted by $|w|$, is the number of symbols in the string. We shall frequently need to refer to the *empty string*, which is a string with no symbols at all. It will be denoted by λ. The following simple relations

$$|\lambda| = 0,$$

$$\lambda w = w\lambda = w$$

hold for all string w.

The concatenation of two strings w and v is the string obtained by appending the symbols of v to the right end of w, that is, if

$$w = a_1 a_2 \cdots a_n$$

and

$$v = b_1 b_2 \cdots b_m,$$

then the concatenation of w and v, denoted by wv, is

$$wv = a_1 a_2 \cdots a_n b_1 b_2 \cdots b_m.$$

The *reverse* of a string is obtained by writing the symbols in reverse order; if w is a string as shown above, then its reverse w^R is

$$w^R = a_n \cdots a_2 a_1.$$

The *length* of a string w, denoted by $|w|$, is the number of symbols in the string. We shall frequently need to refer to the *empty string*, which is a string with no symbols at all. It will be denoted by λ. The following simple relations

$$|\lambda| = 0,$$

$$\lambda w = w\lambda = w$$

hold for all string w.

The concatenation of two strings w and v is the string obtained by appending the symbols of v to the right end of w, that is, if

$$w = a_1 a_2 \cdots a_n$$

and

$$v = b_1 b_2 \cdots b_m,$$

then the concatenation of w and v, denoted by wv, is

$$wv = a_1 a_2 \cdots a_n b_1 b_2 \cdots b_m.$$

The *reverse* of a string is obtained by writing the symbols in reverse order; if w is a string as shown above, then its reverse w^R is

$$w^R = a_n \cdots a_2 a_1.$$

The *length* of a string w, denoted by $|w|$, is the number of symbols in the string. We shall frequently need to refer to the *empty string*, which is a string with no symbols at all. It will be denoted by λ. The following simple relations

$$|\lambda| = 0,$$
$$\lambda w = w\lambda = w$$

hold for all string w.

The concatenation of two strings w and v is the string obtained by appending the symbols of v to the right end of w, that is, if

$$w = a_1 a_2 \cdots a_n$$

and

$$v = b_1 b_2 \cdots b_m,$$

then the concatenation of w and v, denoted by wv, is

$$wv = a_1 a_2 \cdots a_n b_1 b_2 \cdots b_m.$$

The *reverse* of a string is obtained by writing the symbols in reverse order; if w is a string as shown above, then its reverse w^R is

$$w^R = a_n \cdots a_2 a_1.$$

The *length* of a string w, denoted by $|w|$, is the number of symbols in the string. We shall frequently need to refer to the *empty string*, which is a string with no symbols at all. It will be denoted by λ. The following simple relations

$$|\lambda| = 0,$$
$$\lambda w = w\lambda = w$$

hold for all string w.

The concatenation of two strings w and v is the string obtained by appending the symbols of v to the right end of w, that is, if

$$w = a_1 a_2 \cdots a_n$$

and

$$v = b_1 b_2 \cdots b_m,$$

then the concatenation of w and v, denoted by wv, is

$$wv = a_1 a_2 \cdots a_n b_1 b_2 \cdots b_m.$$

The *reverse* of a string is obtained by writing the symbols in reverse order; if w is a string as shown above, then its reverse w^R is

$$w^R = a_n \cdots a_2 a_1.$$

The *length* of a string w, denoted by $|w|$, is the number of symbols in the string. We shall frequently need to refer to the *empty string*, which is a string with no symbols at all. It will be denoted by λ. The following simple relations

$$|\lambda| = 0,$$

$$\lambda w = w\lambda = w$$

hold for all string w.

The concatenation of two strings w and v is the string obtained by appending the symbols of v to the right end of w, that is, if

$$w = a_1 a_2 \cdots a_n$$

and

$$v = b_1 b_2 \cdots b_m,$$

then the concatenation of w and v, denoted by wv, is

$$wv = a_1 a_2 \cdots a_n b_1 b_2 \cdots b_m.$$

The *reverse* of a string is obtained by writing the symbols in reverse order; if w is a string as shown above, then its reverse w^R is

$$w^R = a_n \cdots a_2 a_1.$$

The *length* of a string w, denoted by $|w|$, is the number of symbols in the string. We shall frequently need to refer to the *empty string*, which is a string with no symbols at all. It will be denoted by λ. The following simple relations

$$|\lambda| = 0,$$

$$\lambda w = w\lambda = w$$

hold for all string w.

The concatenation of two strings w and v is the string obtained by appending the symbols of v to the right end of w, that is, if

$$w = a_1 a_2 \cdots a_n$$

and

$$v = b_1 b_2 \cdots b_m,$$

then the concatenation of w and v, denoted by wv, is

$$wv = a_1 a_2 \cdots a_n b_1 b_2 \cdots b_m.$$

The *reverse* of a string is obtained by writing the symbols in reverse order; if w is a string as shown above, then its reverse w^R is

$$w^R = a_n \cdots a_2 a_1.$$

The *length* of a string w, denoted by $|w|$, is the number of symbols in the string. We shall frequently need to refer to the *empty string*, which is a string with no symbols at all. It will be denoted by λ. The following simple relations

$$|\lambda| = 0,$$

$$\lambda w = w\lambda = w$$

hold for all string w.

The concatenation of two strings w and v is the string obtained by appending the symbols of v to the right end of w, that is, if

$$w = a_1 a_2 \cdots a_n$$

and

$$v = b_1 b_2 \cdots b_m,$$

then the concatenation of w and v, denoted by wv, is

$$wv = a_1 a_2 \cdots a_n b_1 b_2 \cdots b_m.$$

The *reverse* of a string is obtained by writing the symbols in reverse order; if w is a string as shown above, then its reverse w^R is

$$w^R = a_n \cdots a_2 a_1.$$

The *length* of a string w, denoted by $|w|$, is the number of symbols in the string. We shall frequently need to refer to the *empty string*, which is a string with no symbols at all. It will be denoted by λ. The following simple relations

$$|\lambda| = 0,$$

$$\lambda w = w\lambda = w$$

hold for all string w.

The concatenation of two strings w and v is the string obtained by appending the symbols of v to the right end of w, that is, if

$$w = a_1 a_2 \cdots a_n$$

and

$$v = b_1 b_2 \cdots b_m,$$

then the concatenation of w and v, denoted by wv, is

$$wv = a_1 a_2 \cdots a_n b_1 b_2 \cdots b_m.$$

The *reverse* of a string is obtained by writing the symbols in reverse order; if w is a string as shown above, then its reverse w^R is

$$w^R = a_n \cdots a_2 a_1.$$

The *length* of a string w, denoted by $|w|$, is the number of symbols in the string. We shall frequently need to refer to the *empty string*, which is a string with no symbols at all. It will be denoted by λ. The following simple relations

$$|\lambda| = 0,$$

$$\lambda w = w\lambda = w$$

hold for all string w.

The concatenation of two strings w and v is the string obtained by appending the symbols of v to the right end of w, that is, if

$$w = a_1 a_2 \cdots a_n$$

and

$$v = b_1 b_2 \cdots b_m,$$

then the concatenation of w and v, denoted by wv, is

$$wv = a_1 a_2 \cdots a_n b_1 b_2 \cdots b_m.$$

The *reverse* of a string is obtained by writing the symbols in reverse order; if w is a string as shown above, then its reverse w^R is

$$w^R = a_n \cdots a_2 a_1.$$

The *length* of a string w, denoted by $|w|$, is the number of symbols in the string. We shall frequently need to refer to the *empty string*, which is a string with no symbols at all. It will be denoted by λ. The following simple relations

$$|\lambda| = 0,$$

$$\lambda w = w\lambda = w$$

hold for all string w.

The concatenation of two strings w and v is the string obtained by appending the symbols of v to the right end of w, that is, if

$$w = a_1 a_2 \cdots a_n$$

and

$$v = b_1 b_2 \cdots b_m,$$

then the concatenation of w and v, denoted by wv, is

$$wv = a_1 a_2 \cdots a_n b_1 b_2 \cdots b_m.$$

The *reverse* of a string is obtained by writing the symbols in reverse order; if w is a string as shown above, then its reverse w^R is

$$w^R = a_n \cdots a_2 a_1.$$

The *length* of a string w, denoted by $|w|$, is the number of symbols in the string. We shall frequently need to refer to the *empty string*, which is a string with no symbols at all. It will be denoted by λ. The following simple relations

$$|\lambda| = 0,$$

$$\lambda w = w\lambda = w$$

hold for all string w.

The concatenation of two strings w and v is the string obtained by appending the symbols of v to the right end of w, that is, if

$$w = a_1 a_2 \cdots a_n$$

and

$$v = b_1 b_2 \cdots b_m,$$

then the concatenation of w and v, denoted by wv, is

$$wv = a_1 a_2 \cdots a_n b_1 b_2 \cdots b_m.$$

The *reverse* of a string is obtained by writing the symbols in reverse order; if w is a string as shown above, then its reverse w^R is

$$w^R = a_n \cdots a_2 a_1.$$

The *length* of a string w, denoted by $|w|$, is the number of symbols in the string. We shall frequently need to refer to the *empty string*, which is a string with no symbols at all. It will be denoted by λ. The following simple relations

$$|\lambda| = 0,$$

$$\lambda w = w\lambda = w$$

hold for all string w.

The concatenation of two strings w and v is the string obtained by appending the symbols of v to the right end of w, that is, if

$$w = a_1 a_2 \cdots a_n$$

and

$$v = b_1 b_2 \cdots b_m,$$

then the concatenation of w and v, denoted by wv, is

$$wv = a_1 a_2 \cdots a_n b_1 b_2 \cdots b_m.$$

The *reverse* of a string is obtained by writing the symbols in reverse order; if w is a string as shown above, then its reverse w^R is

$$w^R = a_n \cdots a_2 a_1.$$

The *length* of a string w, denoted by $|w|$, is the number of symbols in the string. We shall frequently need to refer to the *empty string*, which is a string with no symbols at all. It will be denoted by λ. The following simple relations

$$|\lambda| = 0,$$

$$\lambda w = w\lambda = w$$

hold for all string w.

The concatenation of two strings w and v is the string obtained by appending the symbols of v to the right end of w, that is, if

$$w = a_1 a_2 \cdots a_n$$

and

$$v = b_1 b_2 \cdots b_m,$$

then the concatenation of w and v, denoted by wv, is

$$wv = a_1 a_2 \cdots a_n b_1 b_2 \cdots b_m.$$

The *reverse* of a string is obtained by writing the symbols in reverse order; if w is a string as shown above, then its reverse w^R is

$$w^R = a_n \cdots a_2 a_1.$$

The *length* of a string w, denoted by $|w|$, is the number of symbols in the string. We shall frequently need to refer to the *empty string*, which is a string with no symbols at all. It will be denoted by λ. The following simple relations

$$|\lambda| = 0,$$

$$\lambda w = w\lambda = w$$

hold for all string w.

The concatenation of two strings w and v is the string obtained by appending the symbols of v to the right end of w, that is, if

$$w = a_1 a_2 \cdots a_n$$

and

$$v = b_1 b_2 \cdots b_m,$$

then the concatenation of w and v, denoted by wv, is

$$wv = a_1 a_2 \cdots a_n b_1 b_2 \cdots b_m.$$

The *reverse* of a string is obtained by writing the symbols in reverse order; if w is a string as shown above, then its reverse w^R is

$$w^R = a_n \cdots a_2 a_1.$$

The *length* of a string w, denoted by $|w|$, is the number of symbols in the string. We shall frequently need to refer to the *empty string*, which is a string with no symbols at all. It will be denoted by λ. The following simple relations

$$|\lambda| = 0,$$

$$\lambda w = w\lambda = w$$

hold for all string w.

The concatenation of two strings w and v is the string obtained by appending the symbols of v to the right end of w, that is, if

$$w = a_1 a_2 \cdots a_n$$

and

$$v = b_1 b_2 \cdots b_m,$$

then the concatenation of w and v, denoted by wv, is

$$wv = a_1 a_2 \cdots a_n b_1 b_2 \cdots b_m.$$

The *reverse* of a string is obtained by writing the symbols in reverse order; if w is a string as shown above, then its reverse w^R is

$$w^R = a_n \cdots a_2 a_1.$$

The *length* of a string w, denoted by $|w|$, is the number of symbols in the string. We shall frequently need to refer to the *empty string*, which is a string with no symbols at all. It will be denoted by λ. The following simple relations

$$|\lambda| = 0,$$
$$\lambda w = w\lambda = w$$

hold for all string w.

Any string of consecutive symbols in some string w is said to be a *substring* of w. If

$$w = vu,$$

then the substrings v and u are said to be a *prefix* and a *suffix* of w, respectively. For example, if $w = abbab$, then

$$\{\lambda, a, ab, abb, abba, abbab\}$$

is the set of all prefixes of w, while bab, ab, b are some of its suffixes. Simple properties of strings, such as their length, are very intuitive and probably need little elaboration. For example, if u and v are strings, then the length of their concatenation is the sum of the individual lengths, that is,

$$|uv| = |u| + |v|. \quad (1)$$

But although this relationship is obvious, it is useful to be able to make it precise and prove it. The techniques for doing so are important in more complicated situations.

Any string of consecutive symbols in some string w is said to be a *substring* of w. If

$$w = vu,$$

then the substrings v and u are said to be a *prefix* and a *suffix* of w, respectively. For example, if $w = abbab$, then

$$\{\lambda, a, ab, abb, abba, abbab\}$$

is the set of all prefixes of w, while bab, ab, b are some of its suffixes. Simple properties of strings, such as their length, are very intuitive and probably need little elaboration. For example, if u and v are strings, then the length of their concatenation is the sum of the individual lengths, that is,

$$|uv| = |u| + |v|. \tag{1}$$

But although this relationship is obvious, it is useful to be able to make it precise and prove it. The techniques for doing so are important in more complicated situations.

Any string of consecutive symbols in some string w is said to be a *substring* of w. If

$$w = vu,$$

then the substrings v and u are said to be a *prefix* and a *suffix* of w, respectively. For example, if $w = abbab$, then

$$\{\lambda, a, ab, abb, abba, abbab\}$$

is the set of all prefixes of w, while bab, ab, b are some of its suffixes. Simple properties of strings, such as their length, are very intuitive and probably need little elaboration. For example, if u and v are strings, then the length of their concatenation is the sum of the individual lengths, that is,

$$|uv| = |u| + |v|. \tag{1}$$

But although this relationship is obvious, it is useful to be able to make it precise and prove it. The techniques for doing so are important in more complicated situations.

Any string of consecutive symbols in some string w is said to be a *substring* of w. If

$$w = vu,$$

then the substrings v and u are said to be a *prefix* and a *suffix* of w, respectively. For example, if $w = abbab$, then

$$\{\lambda, a, ab, abb, abba, abbab\}$$

is the set of all prefixes of w, while bab, ab, b are some of its suffixes. Simple properties of strings, such as their length, are very intuitive and probably need little elaboration. For example, if u and v are strings, then the length of their concatenation is the sum of the individual lengths, that is,

$$|uv| = |u| + |v|. \tag{1}$$

But although this relationship is obvious, it is useful to be able to make it precise and prove it. The techniques for doing so are important in more complicated situations.

Any string of consecutive symbols in some string w is said to be a *substring* of w. If

$$w = vu,$$

then the substrings v and u are said to be a *prefix* and a *suffix* of w, respectively. For example, if $w = abbab$, then

$$\{\lambda, a, ab, abb, abba, abbab\}$$

is the set of all prefixes of w, while bab, ab, b are some of its suffixes. Simple properties of strings, such as their length, are very intuitive and probably need little elaboration. For example, if u and v are strings, then the length of their concatenation is the sum of the individual lengths, that is,

$$|uv| = |u| + |v|. \tag{1}$$

But although this relationship is obvious, it is useful to be able to make it precise and prove it. The techniques for doing so are important in more complicated situations.

Any string of consecutive symbols in some string w is said to be a *substring* of w. If

$$w = vu,$$

then the substrings v and u are said to be a *prefix* and a *suffix* of w, respectively. For example, if $w = abbab$, then

$$\{\lambda, a, ab, abb, abba, abbab\}$$

is the set of all prefixes of w, while bab, ab, b are some of its suffixes. Simple properties of strings, such as their length, are very intuitive and probably need little elaboration. For example, if u and v are strings, then the length of their concatenation is the sum of the individual lengths, that is,

$$|uv| = |u| + |v|. \tag{1}$$

But although this relationship is obvious, it is useful to be able to make it precise and prove it. The techniques for doing so are important in more complicated situations.

Any string of consecutive symbols in some string w is said to be a *substring* of w. If

$$w = vu,$$

then the substrings v and u are said to be a *prefix* and a *suffix* of w, respectively. For example, if $w = abbab$, then

$$\{\lambda, a, ab, abb, abba, abbab\}$$

is the set of all prefixes of w, while bab, ab, b are some of its suffixes. Simple properties of strings, such as their length, are very intuitive and probably need little elaboration. For example, if u and v are strings, then the length of their concatenation is the sum of the individual lengths, that is,

$$|uv| = |u| + |v|. \qquad (1)$$

But although this relationship is obvious, it is useful to be able to make it precise and prove it. The techniques for doing so are important in more complicated situations.

Any string of consecutive symbols in some string w is said to be a *substring* of w. If

$$w = vu,$$

then the substrings v and u are said to be a *prefix* and a *suffix* of w, respectively. For example, if $w = abbab$, then

$$\{\lambda, a, ab, abb, abba, abbab\}$$

is the set of all prefixes of w, while bab, ab, b are some of its suffixes. Simple properties of strings, such as their length, are very intuitive and probably need little elaboration. For example, if u and v are strings, then the length of their concatenation is the sum of the individual lengths, that is,

$$|uv| = |u| + |v|. \tag{1}$$

But although this relationship is obvious, it is useful to be able to make it precise and prove it. The techniques for doing so are important in more complicated situations.

Any string of consecutive symbols in some string w is said to be a *substring* of w. If

$$w = vu,$$

then the substrings v and u are said to be a *prefix* and a *suffix* of w, respectively. For example, if $w = abbab$, then

$$\{\lambda, a, ab, abb, abba, abbab\}$$

is the set of all prefixes of w, while bab, ab, b are some of its suffixes. Simple properties of strings, such as their length, are very intuitive and probably need little elaboration. For example, if u and v are strings, then the length of their concatenation is the sum of the individual lengths, that is,

$$|uv| = |u| + |v|. \tag{1}$$

But although this relationship is obvious, it is useful to be able to make it precise and prove it. The techniques for doing so are important in more complicated situations.

Any string of consecutive symbols in some string w is said to be a *substring* of w. If

$$w = vu,$$

then the substrings v and u are said to be a *prefix* and a *suffix* of w, respectively. For example, if $w = abbab$, then

$$\{\lambda, a, ab, abb, abba, abbab\}$$

is the set of all prefixes of w, while bab, ab, b are some of its suffixes. Simple properties of strings, such as their length, are very intuitive and probably need little elaboration. For example, if u and v are strings, then the length of their concatenation is the sum of the individual lengths, that is,

$$|uv| = |u| + |v|. \quad (1)$$

But although this relationship is obvious, it is useful to be able to make it precise and prove it. The techniques for doing so are important in more complicated situations.

Any string of consecutive symbols in some string w is said to be a *substring* of w. If

$$w = vu,$$

then the substrings v and u are said to be a *prefix* and a *suffix* of w, respectively. For example, if $w = abbab$, then

$$\{\lambda, a, ab, abb, abba, abbab\}$$

is the set of all prefixes of w, while bab, ab, b are some of its suffixes. Simple properties of strings, such as their length, are very intuitive and probably need little elaboration. For example, if u and v are strings, then the length of their concatenation is the sum of the individual lengths, that is,

$$|uv| = |u| + |v|. \tag{1}$$

But although this relationship is obvious, it is useful to be able to make it precise and prove it. The techniques for doing so are important in more complicated situations.

Any string of consecutive symbols in some string w is said to be a *substring* of w. If

$$w = vu,$$

then the substrings v and u are said to be a *prefix* and a *suffix* of w, respectively. For example, if $w = abbab$, then

$$\{\lambda, a, ab, abb, abba, abbab\}$$

is the set of all prefixes of w, while bab, ab, b are some of its suffixes. Simple properties of strings, such as their length, are very intuitive and probably need little elaboration. For example, if u and v are strings, then the length of their concatenation is the sum of the individual lengths, that is,

$$|uv| = |u| + |v|. \tag{1}$$

But although this relationship is obvious, it is useful to be able to make it precise and prove it. The techniques for doing so are important in more complicated situations.

Any string of consecutive symbols in some string w is said to be a *substring* of w. If

$$w = vu,$$

then the substrings v and u are said to be a *prefix* and a *suffix* of w, respectively. For example, if $w = abbab$, then

$$\{\lambda, a, ab, abb, abba, abbab\}$$

is the set of all prefixes of w, while bab, ab, b are some of its suffixes. Simple properties of strings, such as their length, are very intuitive and probably need little elaboration. For example, if u and v are strings, then the length of their concatenation is the sum of the individual lengths, that is,

$$|uv| = |u| + |v|. \tag{1}$$

But although this relationship is obvious, it is useful to be able to make it precise and prove it. The techniques for doing so are important in more complicated situations.

Any string of consecutive symbols in some string w is said to be a *substring* of w. If

$$w = vu,$$

then the substrings v and u are said to be a *prefix* and a *suffix* of w, respectively. For example, if $w = abbab$, then

$$\{\lambda, a, ab, abb, abba, abbab\}$$

is the set of all prefixes of w, while bab, ab, b are some of its suffixes. Simple properties of strings, such as their length, are very intuitive and probably need little elaboration. For example, if u and v are strings, then the length of their concatenation is the sum of the individual lengths, that is,

$$|uv| = |u| + |v|. \tag{1}$$

But although this relationship is obvious, it is useful to be able to make it precise and prove it. The techniques for doing so are important in more complicated situations.

Any string of consecutive symbols in some string w is said to be a *substring* of w. If

$$w = vu,$$

then the substrings v and u are said to be a *prefix* and a *suffix* of w, respectively. For example, if $w = abbab$, then

$$\{\lambda, a, ab, abb, abba, abbab\}$$

is the set of all prefixes of w, while bab, ab, b are some of its suffixes. Simple properties of strings, such as their length, are very intuitive and probably need little elaboration. For example, if u and v are strings, then the length of their concatenation is the sum of the individual lengths, that is,

$$|uv| = |u| + |v|. \tag{1}$$

But although this relationship is obvious, it is useful to be able to make it precise and prove it. The techniques for doing so are important in more complicated situations.

Any string of consecutive symbols in some string w is said to be a *substring* of w. If

$$w = vu,$$

then the substrings v and u are said to be a *prefix* and a *suffix* of w, respectively. For example, if $w = abbab$, then

$$\{\lambda, a, ab, abb, abba, abbab\}$$

is the set of all prefixes of w, while bab, ab, b are some of its suffixes. Simple properties of strings, such as their length, are very intuitive and probably need little elaboration. For example, if u and v are strings, then the length of their concatenation is the sum of the individual lengths, that is,

$$|uv| = |u| + |v|. \tag{1}$$

But although this relationship is obvious, it is useful to be able to make it precise and prove it. The techniques for doing so are important in more complicated situations.

Any string of consecutive symbols in some string w is said to be a *substring* of w. If

$$w = vu,$$

then the substrings v and u are said to be a *prefix* and a *suffix* of w, respectively. For example, if $w = abbab$, then

$$\{\lambda, a, ab, abb, abba, abbab\}$$

is the set of all prefixes of w, while bab, ab, b are some of its suffixes. Simple properties of strings, such as their length, are very intuitive and probably need little elaboration. For example, if u and v are strings, then the length of their concatenation is the sum of the individual lengths, that is,

$$|uv| = |u| + |v|. \tag{1}$$

But although this relationship is obvious, it is useful to be able to make it precise and prove it. The techniques for doing so are important in more complicated situations.

Example 1.8

Show that formula

$$|uv| = |u| + |v|. \quad (1)$$

holds for any strings u and v. To prove this, we first need a definition of the length of a string. We make such a definition in a recursive fashion by

$$|a| = 1,$$
$$|wa| = |w| + 1,$$

for all $a \in \Sigma$ and w any string on Σ. This definition is a formal statement of our intuitive understanding of the length of a string: The length of a single symbol is one, and the length of any string is increased by one if we add another symbol to it. With this formal definition, we are ready to prove equality (1) by induction characters.

Example 1.8

Show that formula

$$|uv| = |u| + |v|. \tag{1}$$

holds for any strings u and v. To prove this, we first need a definition of the length of a string. We make such a definition in a recursive fashion by

$$|a| = 1,$$
$$|wa| = |w| + 1,$$

for all $a \in \Sigma$ and w any string on Σ. This definition is a formal statement of our intuitive understanding of the length of a string: The length of a single symbol is one, and the length of any string is increased by one if we add another symbol to it. With this formal definition, we are ready to prove equality (1) by induction characters.

Example 1.8

Show that formula

$$|uv| = |u| + |v|. \tag{1}$$

holds for any strings u and v. To prove this, we first need a definition of the length of a string. We make such a definition in a recursive fashion by

$$|a| = 1,$$
$$|wa| = |w| + 1,$$

for all $a \in \Sigma$ and w any string on Σ. This definition is a formal statement of our intuitive understanding of the length of a string: The length of a single symbol is one, and the length of any string is increased by one if we add another symbol to it. With this formal definition, we are ready to prove equality (1) by induction characters.

Example 1.8

Show that formula

$$|uv| = |u| + |v|. \tag{1}$$

holds for any strings u and v. To prove this, we first need a definition of the length of a string. We make such a definition in a recursive fashion by

$$|a| = 1,$$
$$|wa| = |w| + 1,$$

for all $a \in \Sigma$ and w any string on Σ. This definition is a formal statement of our intuitive understanding of the length of a string: The length of a single symbol is one, and the length of any string is increased by one if we add another symbol to it. With this formal definition, we are ready to prove equality (1) by induction characters.

Example 1.8

Show that formula

$$|uv| = |u| + |v|. \tag{1}$$

holds for any strings u and v. To prove this, we first need a definition of the length of a string. We make such a definition in a recursive fashion by

$$|a| = 1,$$
$$|wa| = |w| + 1,$$

for all $a \in \Sigma$ and w any string on Σ. This definition is a formal statement of our intuitive understanding of the length of a string: The length of a single symbol is one, and the length of any string is increased by one if we add another symbol to it. With this formal definition, we are ready to prove equality (1) by induction characters.

Example 1.8

Show that formula

$$|uv| = |u| + |v|. \quad (1)$$

holds for any strings u and v. To prove this, we first need a definition of the length of a string. We make such a definition in a recursive fashion by

$$|a| = 1,$$
$$|wa| = |w| + 1,$$

for all $a \in \Sigma$ and w any string on Σ. This definition is a formal statement of our intuitive understanding of the length of a string: The length of a single symbol is one, and the length of any string is increased by one if we add another symbol to it. With this formal definition, we are ready to prove equality (1) by induction characters.

Example 1.8

Show that formula

$$|uv| = |u| + |v|. \tag{1}$$

holds for any strings u and v. To prove this, we first need a definition of the length of a string. We make such a definition in a recursive fashion by

$$|a| = 1,$$
$$|wa| = |w| + 1,$$

for all $a \in \Sigma$ and w any string on Σ. This definition is a formal statement of our intuitive understanding of the length of a string: The length of a single symbol is one, and the length of any string is increased by one if we add another symbol to it. With this formal definition, we are ready to prove equality (1) by induction characters.

Example 1.8

Show that formula

$$|uv| = |u| + |v|. \tag{1}$$

holds for any strings u and v. To prove this, we first need a definition of the length of a string. We make such a definition in a recursive fashion by

$$|a| = 1,$$
$$|wa| = |w| + 1,$$

for all $a \in \Sigma$ and w any string on Σ. This definition is a formal statement of our intuitive understanding of the length of a string: The length of a single symbol is one, and the length of any string is increased by one if we add another symbol to it. With this formal definition, we are ready to prove equality (1) by induction characters.

Example 1.8

Show that formula

$$|uv| = |u| + |v|. \tag{1}$$

holds for any strings u and v. To prove this, we first need a definition of the length of a string. We make such a definition in a recursive fashion by

$$|a| = 1,$$
$$|wa| = |w| + 1,$$

for all $a \in \Sigma$ and w any string on Σ. This definition is a formal statement of our intuitive understanding of the length of a string: The length of a single symbol is one, and the length of any string is increased by one if we add another symbol to it. With this formal definition, we are ready to prove equality (1) by induction characters.

Example 1.8

Show that formula

$$|uv| = |u| + |v|. \quad (1)$$

holds for any strings u and v. To prove this, we first need a definition of the length of a string. We make such a definition in a recursive fashion by

$$|a| = 1,$$
$$|wa| = |w| + 1,$$

for all $a \in \Sigma$ and w any string on Σ. This definition is a formal statement of our intuitive understanding of the length of a string: The length of a single symbol is one, and the length of any string is increased by one if we add another symbol to it. With this formal definition, we are ready to prove equality (1) by induction characters.

Example 1.8

Show that formula

$$|uv| = |u| + |v|. \tag{1}$$

holds for any strings u and v. To prove this, we first need a definition of the length of a string. We make such a definition in a recursive fashion by

$$|a| = 1,$$
$$|wa| = |w| + 1,$$

for all $a \in \Sigma$ and w any string on Σ. This definition is a formal statement of our intuitive understanding of the length of a string: The length of a single symbol is one, and the length of any string is increased by one if we add another symbol to it. With this formal definition, we are ready to prove equality (1) by induction characters.

Example 1.8

Show that formula

$$|uv| = |u| + |v|. \tag{1}$$

holds for any strings u and v. To prove this, we first need a definition of the length of a string. We make such a definition in a recursive fashion by

$$|a| = 1,$$
$$|wa| = |w| + 1,$$

for all $a \in \Sigma$ and w any string on Σ. This definition is a formal statement of our intuitive understanding of the length of a string: The length of a single symbol is one, and the length of any string is increased by one if we add another symbol to it. With this formal definition, we are ready to prove equality (1) by induction characters.

Example 1.8

Show that formula

$$|uv| = |u| + |v|. \tag{1}$$

holds for any strings u and v. To prove this, we first need a definition of the length of a string. We make such a definition in a recursive fashion by

$$|a| = 1,$$
$$|wa| = |w| + 1,$$

for all $a \in \Sigma$ and w any string on Σ. This definition is a formal statement of our intuitive understanding of the length of a string: The length of a single symbol is one, and the length of any string is increased by one if we add another symbol to it. With this formal definition, we are ready to prove equality (1) by induction characters.

Example 1.8 (continuation)

By definition, the equality

$$|uv| = |u| + |v|. \tag{1}$$

holds for all strings u of any length and all strings v of length 1, so we have a basis. As an inductive assumption, we take that equality (1) holds for all strings u of any length and all strings v of length $1, 2, \ldots, n$. Now take any string v of length $n+1$ and write it as $v = wa$. Then,

$$|v| = |w| + 1,$$
$$|uv| = |uwa| = |uw| + 1.$$

By the inductive hypothesis (which is applicable since w is of length n),

$$|uw| = |u| + |w|$$

so that

$$|uv| = |u| + |w| + 1 = |u| + |v|.$$

Therefore, equality (1) holds for all u and all v of length up to $n+1$, completing the inductive step and the argument. ∎

Example 1.8 (continuation)

By definition, the equality

$$|uv| = |u| + |v|. \tag{1}$$

holds for all strings u of any length and all strings v of length 1, so we have a basis. As an inductive assumption, we take that equality (1) holds for all strings u of any length and all strings v of length $1, 2, \ldots, n$. Now take any string v of length $n+1$ and write it as $v = wa$. Then,

$$|v| = |w| + 1,$$
$$|uv| = |uwa| = |uw| + 1.$$

By the inductive hypothesis (which is applicable since w is of length n),

$$|uw| = |u| + |w|$$

so that

$$|uv| = |u| + |w| + 1 = |u| + |v|.$$

Therefore, equality (1) holds for all u and all v of length up to $n+1$, completing the inductive step and the argument. ∎

Example 1.8 (continuation)

By definition, the equality

$$|uv| = |u| + |v|. \tag{1}$$

holds for all strings u of any length and all strings v of length 1, so we have a basis. As an inductive assumption, we take that equality (1) holds for all strings u of any length and all strings v of length $1, 2, \ldots, n$. Now take any string v of length $n+1$ and write it as $v = wa$. Then,

$$|v| = |w| + 1,$$
$$|uv| = |uwa| = |uw| + 1.$$

By the inductive hypothesis (which is applicable since w is of length n),

$$|uw| = |u| + |w|$$

so that

$$|uv| = |u| + |w| + 1 = |u| + |v|.$$

Therefore, equality (1) holds for all u and all v of length up to $n+1$, completing the inductive step and the argument. ■

Example 1.8 (continuation)

By definition, the equality

$$|uv| = |u| + |v|. \tag{1}$$

holds for all strings u of any length and all strings v of length 1, so we have a basis. As an inductive assumption, we take that equality (1) holds for all strings u of any length and all strings v of length $1, 2, \ldots, n$. Now take any string v of length $n+1$ and write it as $v = wa$. Then,

$$|v| = |w| + 1,$$
$$|uv| = |uwa| = |uw| + 1.$$

By the inductive hypothesis (which is applicable since w is of length n),

$$|uw| = |u| + |w|$$

so that

$$|uv| = |u| + |w| + 1 = |u| + |v|.$$

Therefore, equality (1) holds for all u and all v of length up to $n+1$, completing the inductive step and the argument. ∎

Example 1.8 (continuation)

By definition, the equality

$$|uv| = |u| + |v|. \tag{1}$$

holds for all strings u of any length and all strings v of length 1, so we have a basis. As an inductive assumption, we take that equality (1) holds for all strings u of any length and all strings v of length $1, 2, \ldots, n$. Now take any string v of length $n+1$ and write it as $v = wa$. Then,

$$|v| = |w| + 1,$$
$$|uv| = |uwa| = |uw| + 1.$$

By the inductive hypothesis (which is applicable since w is of length n),

$$|uw| = |u| + |w|$$

so that

$$|uv| = |u| + |w| + 1 = |u| + |v|.$$

Therefore, equality (1) holds for all u and all v of length up to $n+1$, completing the inductive step and the argument. ∎

Example 1.8 (continuation)

By definition, the equality

$$|uv| = |u| + |v|. \tag{1}$$

holds for all strings u of any length and all strings v of length 1, so we have a basis. As an inductive assumption, we take that equality (1) holds for all strings u of any length and all strings v of length $1, 2, \ldots, n$. Now take any string v of length $n+1$ and write it as $v = wa$. Then,

$$|v| = |w| + 1,$$
$$|uv| = |uwa| = |uw| + 1.$$

By the inductive hypothesis (which is applicable since w is of length n),

$$|uw| = |u| + |w|$$

so that

$$|uv| = |u| + |w| + 1 = |u| + |v|.$$

Therefore, equality (1) holds for all u and all v of length up to $n+1$, completing the inductive step and the argument. ■

Example 1.8 (continuation)

By definition, the equality

$$|uv| = |u| + |v|. \tag{1}$$

holds for all strings u of any length and all strings v of length 1, so we have a basis. As an inductive assumption, we take that equality (1) holds for all strings u of any length and all strings v of length $1, 2, \ldots, n$. Now take any string v of length $n+1$ and write it as $v = wa$. Then,

$$|v| = |w| + 1,$$
$$|uv| = |uwa| = |uw| + 1.$$

By the inductive hypothesis (which is applicable since w is of length n),

$$|uw| = |u| + |w|$$

so that

$$|uv| = |u| + |w| + 1 = |u| + |v|.$$

Therefore, equality (1) holds for all u and all v of length up to $n+1$, completing the inductive step and the argument. ∎

Example 1.8 (continuation)

By definition, the equality

$$|uv| = |u| + |v|. \tag{1}$$

holds for all strings u of any length and all strings v of length 1, so we have a basis. As an inductive assumption, we take that equality (1) holds for all strings u of any length and all strings v of length $1, 2, \ldots, n$. Now take any string v of length $n+1$ and write it as $v = wa$. Then,

$$|v| = |w| + 1,$$
$$|uv| = |uwa| = |uw| + 1.$$

By the inductive hypothesis (which is applicable since w is of length n),

$$|uw| = |u| + |w|$$

so that

$$|uv| = |u| + |w| + 1 = |u| + |v|.$$

Therefore, equality (1) holds for all u and all v of length up to $n+1$, completing the inductive step and the argument. ∎

Example 1.8 (continuation)

By definition, the equality

$$|uv| = |u| + |v|. \tag{1}$$

holds for all strings u of any length and all strings v of length 1, so we have a basis. As an inductive assumption, we take that equality (1) holds for all strings u of any length and all strings v of length $1, 2, \ldots, n$. Now take any string v of length $n + 1$ and write it as $v = wa$. Then,

$$|v| = |w| + 1,$$
$$|uv| = |uwa| = |uw| + 1.$$

By the inductive hypothesis (which is applicable since w is of length n),

$$|uw| = |u| + |w|$$

so that

$$|uv| = |u| + |w| + 1 = |u| + |v|.$$

Therefore, equality (1) holds for all u and all v of length up to $n + 1$, completing the inductive step and the argument. ■

Example 1.8 (continuation)

By definition, the equality

$$|uv| = |u| + |v|. \qquad (1)$$

holds for all strings u of any length and all strings v of length 1, so we have a basis. As an inductive assumption, we take that equality (1) holds for all strings u of any length and all strings v of length $1, 2, \ldots, n$. Now take any string v of length $n+1$ and write it as $v = wa$. Then,

$$|v| = |w| + 1,$$
$$|uv| = |uwa| = |uw| + 1.$$

By the inductive hypothesis (which is applicable since w is of length n),

$$|uw| = |u| + |w|$$

so that

$$|uv| = |u| + |w| + 1 = |u| + |v|.$$

Therefore, equality (1) holds for all u and all v of length up to $n+1$, completing the inductive step and the argument. ∎

Example 1.8 (continuation)

By definition, the equality

$$|uv| = |u| + |v|. \tag{1}$$

holds for all strings u of any length and all strings v of length 1, so we have a basis. As an inductive assumption, we take that equality (1) holds for all strings u of any length and all strings v of length $1, 2, \ldots, n$. Now take any string v of length $n+1$ and write it as $v = wa$. Then,

$$|v| = |w| + 1,$$
$$|uv| = |uwa| = |uw| + 1.$$

By the inductive hypothesis (which is applicable since w is of length n),

$$|uw| = |u| + |w|$$

so that

$$|uv| = |u| + |w| + 1 = |u| + |v|.$$

Therefore, equality (1) holds for all u and all v of length up to $n+1$, completing the inductive step and the argument. ■

Example 1.8 (continuation)

By definition, the equality

$$|uv| = |u| + |v|. \tag{1}$$

holds for all strings u of any length and all strings v of length 1, so we have a basis. As an inductive assumption, we take that equality (1) holds for all strings u of any length and all strings v of length $1, 2, \ldots, n$. Now take any string v of length $n + 1$ and write it as $v = wa$. Then,

$$|v| = |w| + 1,$$
$$|uv| = |uwa| = |uw| + 1.$$

By the inductive hypothesis (which is applicable since w is of length n),

$$|uw| = |u| + |w|$$

so that

$$|uv| = |u| + |w| + 1 = |u| + |v|.$$

Therefore, equality (1) holds for all u and all v of length up to $n + 1$, completing the inductive step and the argument. ■

Example 1.8 (continuation)

By definition, the equality

$$|uv| = |u| + |v|. \tag{1}$$

holds for all strings u of any length and all strings v of length 1, so we have a basis. As an inductive assumption, we take that equality (1) holds for all strings u of any length and all strings v of length $1, 2, \ldots, n$. Now take any string v of length $n+1$ and write it as $v = wa$. Then,

$$|v| = |w| + 1,$$
$$|uv| = |uwa| = |uw| + 1.$$

By the inductive hypothesis (which is applicable since w is of length n),

$$|uw| = |u| + |w|$$

so that

$$|uv| = |u| + |w| + 1 = |u| + |v|.$$

Therefore, equality (1) holds for all u and all v of length up to $n+1$, completing the inductive step and the argument. ■

Example 1.8 (continuation)

By definition, the equality

$$|uv| = |u| + |v|. \tag{1}$$

holds for all strings u of any length and all strings v of length 1, so we have a basis. As an inductive assumption, we take that equality (1) holds for all strings u of any length and all strings v of length $1, 2, \dots, n$. Now take any string v of length $n + 1$ and write it as $v = wa$. Then,

$$|v| = |w| + 1,$$
$$|uv| = |uwa| = |uw| + 1.$$

By the inductive hypothesis (which is applicable since w is of length n),

$$|uw| = |u| + |w|$$

so that

$$|uv| = |u| + |w| + 1 = |u| + |v|.$$

Therefore, equality (1) holds for all u and all v of length up to $n + 1$, completing the inductive step and the argument. ■

Example 1.8 (continuation)

By definition, the equality

$$|uv| = |u| + |v|. \tag{1}$$

holds for all strings u of any length and all strings v of length 1, so we have a basis. As an inductive assumption, we take that equality (1) holds for all strings u of any length and all strings v of length $1, 2, \ldots, n$. Now take any string v of length $n + 1$ and write it as $v = wa$. Then,

$$|v| = |w| + 1,$$
$$|uv| = |uwa| = |uw| + 1.$$

By the inductive hypothesis (which is applicable since w is of length n),

$$|uw| = |u| + |w|$$

so that

$$|uv| = |u| + |w| + 1 = |u| + |v|.$$

Therefore, equality (1) holds for all u and all v of length up to $n + 1$, completing the inductive step and the argument. ∎

Example 1.8 (continuation)

By definition, the equality

$$|uv| = |u| + |v|. \tag{1}$$

holds for all strings u of any length and all strings v of length 1, so we have a basis. As an inductive assumption, we take that equality (1) holds for all strings u of any length and all strings v of length $1, 2, \ldots, n$. Now take any string v of length $n + 1$ and write it as $v = wa$. Then,

$$|v| = |w| + 1,$$
$$|uv| = |uwa| = |uw| + 1.$$

By the inductive hypothesis (which is applicable since w is of length n),

$$|uw| = |u| + |w|$$

so that

$$|uv| = |u| + |w| + 1 = |u| + |v|.$$

Therefore, equality (1) holds for all u and all v of length up to $n + 1$, completing the inductive step and the argument. ■

Example 1.8 (continuation)

By definition, the equality

$$|uv| = |u| + |v|. \tag{1}$$

holds for all strings u of any length and all strings v of length 1, so we have a basis. As an inductive assumption, we take that equality (1) holds for all strings u of any length and all strings v of length $1, 2, \ldots, n$. Now take any string v of length $n + 1$ and write it as $v = wa$. Then,

$$|v| = |w| + 1,$$
$$|uv| = |uwa| = |uw| + 1.$$

By the inductive hypothesis (which is applicable since w is of length n),

$$|uw| = |u| + |w|$$

so that

$$|uv| = |u| + |w| + 1 = |u| + |v|.$$

Therefore, equality (1) holds for all u and all v of length up to $n + 1$, completing the inductive step and the argument. ∎

Example 1.8 (continuation)

By definition, the equality

$$|uv| = |u| + |v|. \tag{1}$$

holds for all strings u of any length and all strings v of length 1, so we have a basis. As an inductive assumption, we take that equality (1) holds for all strings u of any length and all strings v of length $1, 2, \ldots, n$. Now take any string v of length $n+1$ and write it as $v = wa$. Then,

$$|v| = |w| + 1,$$
$$|uv| = |uwa| = |uw| + 1.$$

By the inductive hypothesis (which is applicable since w is of length n),

$$|uw| = |u| + |w|$$

so that

$$|uv| = |u| + |w| + 1 = |u| + |v|.$$

Therefore, equality (1) holds for all u and all v of length up to $n+1$, completing the inductive step and the argument. ∎

Example 1.8 (continuation)

By definition, the equality

$$|uv| = |u| + |v|. \tag{1}$$

holds for all strings u of any length and all strings v of length 1, so we have a basis. As an inductive assumption, we take that equality (1) holds for all strings u of any length and all strings v of length $1, 2, \ldots, n$. Now take any string v of length $n+1$ and write it as $v = wa$. Then,

$$|v| = |w| + 1,$$
$$|uv| = |uwa| = |uw| + 1.$$

By the inductive hypothesis (which is applicable since w is of length n),

$$|uw| = |u| + |w|$$

so that

$$|uv| = |u| + |w| + 1 = |u| + |v|.$$

Therefore, equality (1) holds for all u and all v of length up to $n+1$, completing the inductive step and the argument. ■

If w is a string, then w^n stands for the string obtained by repeating w n times. As a special case, we define

$$w^0 = \lambda,$$

for all strings w.

If Σ is an alphabet, then we use Σ^* to denote the set of strings obtained by concatenating zero or more symbols from Σ. The set Σ^* always contains the empty string λ. To exclude the empty string, we define

$$\Sigma^+ = \Sigma^* - \{\lambda\}.$$

While Σ is finite by assumption, Σ^* and Σ^+ are always infinite because there is no limit on the length of the strings in these sets. A *language* is defined very generally as a subset of Σ^*. A string in a language L will be called a *sentence* of L. This definition is quite broad; any set of strings on an alphabet Σ can be considered as a language. Later we will study methods by which specific languages can be defined and described; this will enable us to give some structure to this rather broad concept. For the moment, though, we shall just look at a few specific examples.

If w is a string, then w^n stands for the string obtained by repeating w n times. As a special case, we define

$$w^0 = \lambda,$$

for all strings w.

If Σ is an alphabet, then we use Σ^* to denote the set of strings obtained by concatenating zero or more symbols from Σ. The set Σ^* always contains the empty string λ. To exclude the empty string, we define

$$\Sigma^+ = \Sigma^* - \{\lambda\}.$$

While Σ is finite by assumption, Σ^* and Σ^+ are always infinite because there is no limit on the length of the strings in these sets. A *language* is defined very generally as a subset of Σ^*. A string in a language L will be called a *sentence* of L. This definition is quite broad; any set of strings on an alphabet Σ can be considered as a language. Later we will study methods by which specific languages can be defined and described; this will enable us to give some structure to this rather broad concept. For the moment, though, we shall just look at a few specific examples.

If w is a string, then w^n stands for the string obtained by repeating w n times. As a special case, we define

$$w^0 = \lambda,$$

for all strings w.

If Σ is an alphabet, then we use Σ^* to denote the set of strings obtained by concatenating zero or more symbols from Σ. The set Σ^* always contains the empty string λ. To exclude the empty string, we define

$$\Sigma^+ = \Sigma^* - \{\lambda\}.$$

While Σ is finite by assumption, Σ^* and Σ^+ are always infinite because there is no limit on the length of the strings in these sets. A *language* is defined very generally as a subset of Σ^*. A string in a language L will be called a *sentence* of L. This definition is quite broad; any set of strings on an alphabet Σ can be considered as a language. Later we will study methods by which specific languages can be defined and described; this will enable us to give some structure to this rather broad concept. For the moment, though, we shall just look at a few specific examples.

If w is a string, then w^n stands for the string obtained by repeating w n times. As a special case, we define

$$w^0 = \lambda,$$

for all strings w.

If Σ is an alphabet, then we use Σ^* to denote the set of strings obtained by concatenating zero or more symbols from Σ. The set Σ^* always contains the empty string λ. To exclude the empty string, we define

$$\Sigma^+ = \Sigma^* - \{\lambda\}.$$

While Σ is finite by assumption, Σ^* and Σ^+ are always infinite because there is no limit on the length of the strings in these sets. A *language* is defined very generally as a subset of Σ^*. A string in a language L will be called a *sentence* of L. This definition is quite broad; any set of strings on an alphabet Σ can be considered as a language. Later we will study methods by which specific languages can be defined and described; this will enable us to give some structure to this rather broad concept. For the moment, though, we shall just look at a few specific examples.

If w is a string, then w^n stands for the string obtained by repeating w n times. As a special case, we define

$$w^0 = \lambda,$$

for all strings w.

If Σ is an alphabet, then we use Σ^* to denote the set of strings obtained by concatenating zero or more symbols from Σ. The set Σ^* always contains the empty string λ. To exclude the empty string, we define

$$\Sigma^+ = \Sigma^* - \{\lambda\}.$$

While Σ is finite by assumption, Σ^* and Σ^+ are always infinite because there is no limit on the length of the strings in these sets. A *language* is defined very generally as a subset of Σ^*. A string in a language L will be called a *sentence* of L. This definition is quite broad; any set of strings on an alphabet Σ can be considered as a language. Later we will study methods by which specific languages can be defined and described; this will enable us to give some structure to this rather broad concept. For the moment, though, we shall just look at a few specific examples.

If w is a string, then w^n stands for the string obtained by repeating w n times. As a special case, we define

$$w^0 = \lambda,$$

for all strings w.

If Σ is an alphabet, then we use Σ^* to denote the set of strings obtained by concatenating zero or more symbols from Σ. The set Σ^* always contains the empty string λ. To exclude the empty string, we define

$$\Sigma^+ = \Sigma^* - \{\lambda\}.$$

While Σ is finite by assumption, Σ^* and Σ^+ are always infinite because there is no limit on the length of the strings in these sets. A *language* is defined very generally as a subset of Σ^*. A string in a language L will be called a *sentence* of L. This definition is quite broad; any set of strings on an alphabet Σ can be considered as a language. Later we will study methods by which specific languages can be defined and described; this will enable us to give some structure to this rather broad concept. For the moment, though, we shall just look at a few specific examples.

If w is a string, then w^n stands for the string obtained by repeating w n times. As a special case, we define

$$w^0 = \lambda,$$

for all strings w.

If Σ is an alphabet, then we use Σ^* to denote the set of strings obtained by concatenating zero or more symbols from Σ. The set Σ^* always contains the empty string λ. To exclude the empty string, we define

$$\Sigma^+ = \Sigma^* - \{\lambda\}.$$

While Σ is finite by assumption, Σ^* and Σ^+ are always infinite because there is no limit on the length of the strings in these sets. A *language* is defined very generally as a subset of Σ^*. A string in a language L will be called a *sentence* of L. This definition is quite broad; any set of strings on an alphabet Σ can be considered as a language. Later we will study methods by which specific languages can be defined and described; this will enable us to give some structure to this rather broad concept. For the moment, though, we shall just look at a few specific examples.

If w is a string, then w^n stands for the string obtained by repeating w n times. As a special case, we define

$$w^0 = \lambda,$$

for all strings w.
If Σ is an alphabet, then we use Σ^* to denote the set of strings obtained by concatenating zero or more symbols from Σ. The set Σ^* always contains the empty string λ. To exclude the empty string, we define

$$\Sigma^+ = \Sigma^* - \{\lambda\}.$$

While Σ is finite by assumption, Σ^* and Σ^+ are always infinite because there is no limit on the length of the strings in these sets. A *language* is defined very generally as a subset of Σ^*. A string in a language L will be called a *sentence* of L. This definition is quite broad; any set of strings on an alphabet Σ can be considered as a language. Later we will study methods by which specific languages can be defined and described; this will enable us to give some structure to this rather broad concept. For the moment, though, we shall just look at a few specific examples.

If w is a string, then w^n stands for the string obtained by repeating w n times. As a special case, we define

$$w^0 = \lambda,$$

for all strings w.

If Σ is an alphabet, then we use Σ^* to denote the set of strings obtained by concatenating zero or more symbols from Σ. The set Σ^* always contains the empty string λ. To exclude the empty string, we define

$$\Sigma^+ = \Sigma^* - \{\lambda\}.$$

While Σ is finite by assumption, Σ^* and Σ^+ are always infinite because there is no limit on the length of the strings in these sets. A *language* is defined very generally as a subset of Σ^*. A string in a language L will be called a *sentence* of L. This definition is quite broad; any set of strings on an alphabet Σ can be considered as a language. Later we will study methods by which specific languages can be defined and described; this will enable us to give some structure to this rather broad concept. For the moment, though, we shall just look at a few specific examples.

If w is a string, then w^n stands for the string obtained by repeating w n times. As a special case, we define

$$w^0 = \lambda,$$

for all strings w.

If Σ is an alphabet, then we use Σ^* to denote the set of strings obtained by concatenating zero or more symbols from Σ. The set Σ^* always contains the empty string λ. To exclude the empty string, we define

$$\Sigma^+ = \Sigma^* - \{\lambda\}.$$

While Σ is finite by assumption, Σ^* and Σ^+ are always infinite because there is no limit on the length of the strings in these sets. A *language* is defined very generally as a subset of Σ^*. A string in a language L will be called a *sentence* of L. This definition is quite broad; any set of strings on an alphabet Σ can be considered as a language. Later we will study methods by which specific languages can be defined and described; this will enable us to give some structure to this rather broad concept. For the moment, though, we shall just look at a few specific examples.

If w is a string, then w^n stands for the string obtained by repeating w n times. As a special case, we define

$$w^0 = \lambda,$$

for all strings w.
If Σ is an alphabet, then we use Σ^* to denote the set of strings obtained by concatenating zero or more symbols from Σ. The set Σ^* always contains the empty string λ. To exclude the empty string, we define

$$\Sigma^+ = \Sigma^* - \{\lambda\}.$$

While Σ is finite by assumption, Σ^* and Σ^+ are always infinite because there is no limit on the length of the strings in these sets. A *language* is defined very generally as a subset of Σ^*. A string in a language L will be called a *sentence* of L. This definition is quite broad; any set of strings on an alphabet Σ can be considered as a language. Later we will study methods by which specific languages can be defined and described; this will enable us to give some structure to this rather broad concept. For the moment, though, we shall just look at a few specific examples.

If w is a string, then w^n stands for the string obtained by repeating w n times. As a special case, we define

$$w^0 = \lambda,$$

for all strings w.

If Σ is an alphabet, then we use Σ^* to denote the set of strings obtained by concatenating zero or more symbols from Σ. The set Σ^* always contains the empty string λ. To exclude the empty string, we define

$$\Sigma^+ = \Sigma^* - \{\lambda\}.$$

While Σ is finite by assumption, Σ^* and Σ^+ are always infinite because there is no limit on the length of the strings in these sets. A *language* is defined very generally as a subset of Σ^*. A string in a language L will be called a *sentence* of L. This definition is quite broad; any set of strings on an alphabet Σ can be considered as a language. Later we will study methods by which specific languages can be defined and described; this will enable us to give some structure to this rather broad concept. For the moment, though, we shall just look at a few specific examples.

If w is a string, then w^n stands for the string obtained by repeating w n times. As a special case, we define

$$w^0 = \lambda,$$

for all strings w.

If Σ is an alphabet, then we use Σ^* to denote the set of strings obtained by concatenating zero or more symbols from Σ. The set Σ^* always contains the empty string λ. To exclude the empty string, we define

$$\Sigma^+ = \Sigma^* - \{\lambda\}.$$

While Σ is finite by assumption, Σ^* and Σ^+ are always infinite because there is no limit on the length of the strings in these sets. A *language* is defined very generally as a subset of Σ^*. A string in a language L will be called a *sentence* of L. This definition is quite broad; any set of strings on an alphabet Σ can be considered as a language. Later we will study methods by which specific languages can be defined and described; this will enable us to give some structure to this rather broad concept. For the moment, though, we shall just look at a few specific examples.

If w is a string, then w^n stands for the string obtained by repeating w n times. As a special case, we define

$$w^0 = \lambda,$$

for all strings w.

If Σ is an alphabet, then we use Σ^* to denote the set of strings obtained by concatenating zero or more symbols from Σ. The set Σ^* always contains the empty string λ. To exclude the empty string, we define

$$\Sigma^+ = \Sigma^* - \{\lambda\}.$$

While Σ is finite by assumption, Σ^* and Σ^+ are always infinite because there is no limit on the length of the strings in these sets. A *language* is defined very generally as a subset of Σ^*. A string in a language L will be called a *sentence* of L. This definition is quite broad; any set of strings on an alphabet Σ can be considered as a language. Later we will study methods by which specific languages can be defined and described; this will enable us to give some structure to this rather broad concept. For the moment, though, we shall just look at a few specific examples.

If w is a string, then w^n stands for the string obtained by repeating w n times. As a special case, we define

$$w^0 = \lambda,$$

for all strings w.

If Σ is an alphabet, then we use Σ^* to denote the set of strings obtained by concatenating zero or more symbols from Σ. The set Σ^* always contains the empty string λ. To exclude the empty string, we define

$$\Sigma^+ = \Sigma^* - \{\lambda\}.$$

While Σ is finite by assumption, Σ^* and Σ^+ are always infinite because there is no limit on the length of the strings in these sets. A *language* is defined very generally as a subset of Σ^*. A string in a language L will be called a *sentence* of L. This definition is quite broad; any set of strings on an alphabet Σ can be considered as a language. Later we will study methods by which specific languages can be defined and described; this will enable us to give some structure to this rather broad concept. For the moment, though, we shall just look at a few specific examples.

If w is a string, then w^n stands for the string obtained by repeating w n times. As a special case, we define

$$w^0 = \lambda,$$

for all strings w.

If Σ is an alphabet, then we use Σ^* to denote the set of strings obtained by concatenating zero or more symbols from Σ. The set Σ^* always contains the empty string λ. To exclude the empty string, we define

$$\Sigma^+ = \Sigma^* - \{\lambda\}.$$

While Σ is finite by assumption, Σ^* and Σ^+ are always infinite because there is no limit on the length of the strings in these sets. A *language* is defined very generally as a subset of Σ^*. A string in a language L will be called a *sentence* of L. This definition is quite broad; any set of strings on an alphabet Σ can be considered as a language. Later we will study methods by which specific languages can be defined and described; this will enable us to give some structure to this rather broad concept. For the moment, though, we shall just look at a few specific examples.

If w is a string, then w^n stands for the string obtained by repeating w n times. As a special case, we define

$$w^0 = \lambda,$$

for all strings w.

If Σ is an alphabet, then we use Σ^* to denote the set of strings obtained by concatenating zero or more symbols from Σ. The set Σ^* always contains the empty string λ. To exclude the empty string, we define

$$\Sigma^+ = \Sigma^* - \{\lambda\}.$$

While Σ is finite by assumption, Σ^* and Σ^+ are always infinite because there is no limit on the length of the strings in these sets. A *language* is defined very generally as a subset of Σ^*. A string in a language L will be called a *sentence* of L. This definition is quite broad; any set of strings on an alphabet Σ can be considered as a language. Later we will study methods by which specific languages can be defined and described; this will enable us to give some structure to this rather broad concept. For the moment, though, we shall just look at a few specific examples.

If w is a string, then w^n stands for the string obtained by repeating w n times. As a special case, we define

$$w^0 = \lambda,$$

for all strings w.

If Σ is an alphabet, then we use Σ^* to denote the set of strings obtained by concatenating zero or more symbols from Σ. The set Σ^* always contains the empty string λ. To exclude the empty string, we define

$$\Sigma^+ = \Sigma^* - \{\lambda\}.$$

While Σ is finite by assumption, Σ^* and Σ^+ are always infinite because there is no limit on the length of the strings in these sets. A *language* is defined very generally as a subset of Σ^*. A string in a language L will be called a *sentence* of L. This definition is quite broad; any set of strings on an alphabet Σ can be considered as a language. Later we will study methods by which specific languages can be defined and described; this will enable us to give some structure to this rather broad concept. For the moment, though, we shall just look at a few specific examples.

Example 1.9

Let $\Sigma = \{a, b\}$. Then

$$\Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}.$$

The set

$$\{a, aa, aab\}$$

is a language on Σ. Since it has a finite number of sentences, we call it a *finite language*. The set

$$L = \{a^n b^n : n \geq 0\}$$

is also a language on Σ. The strings $aabb$ and $aaaabbbb$ are in the language L, but the string abb is not in L. This language is infinite. Most interesting languages are infinite.

Example 1.9

Let $\Sigma = \{a, b\}$. Then

$$\Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}.$$

The set

$$\{a, aa, aab\}$$

is a language on Σ. Since it has a finite number of sentences, we call it a *finite language*. The set

$$L = \{a^n b^n : n \geq 0\}$$

is also a language on Σ. The strings $aabb$ and $aaaabbbb$ are in the language L, but the string abb is not in L. This language is infinite. Most interesting languages are infinite. ■

Example 1.9

Let $\Sigma = \{a, b\}$. Then

$$\Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}.$$

The set

$$\{a, aa, aab\}$$

is a language on Σ. Since it has a finite number of sentences, we call it a *finite language*. The set

$$L = \{a^n b^n : n \geq 0\}$$

is also a language on Σ. The strings $aabb$ and $aaaabbbb$ are in the language L, but the string abb is not in L. This language is infinite. Most interesting languages are infinite. ■

Example 1.9

Let $\Sigma = \{a, b\}$. Then

$$\Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}.$$

The set

$$\{a, aa, aab\}$$

is a language on Σ. Since it has a finite number of sentences, we call it a *finite language*. The set

$$L = \{a^n b^n : n \geq 0\}$$

is also a language on Σ. The strings $aabb$ and $aaaabbbb$ are in the language L, but the string abb is not in L. This language is infinite. Most interesting languages are infinite. ■

Example 1.9

Let $\Sigma = \{a, b\}$. Then

$$\Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}.$$

The set

$$\{a, aa, aab\}$$

is a language on Σ. Since it has a finite number of sentences, we call it a *finite language*. The set

$$L = \{a^n b^n : n \geq 0\}$$

is also a language on Σ. The strings $aabb$ and $aaaabbbb$ are in the language L, but the string abb is not in L. This language is infinite. Most interesting languages are infinite. ■

Example 1.9

Let $\Sigma = \{a, b\}$. Then

$$\Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}.$$

The set

$$\{a, aa, aab\}$$

is a language on Σ. Since it has a finite number of sentences, we call it a *finite language*. The set

$$L = \{a^n b^n : n \geq 0\}$$

is also a language on Σ. The strings $aabb$ and $aaaabbbb$ are in the language L, but the string abb is not in L. This language is infinite. Most interesting languages are infinite. ■

Example 1.9

Let $\Sigma = \{a, b\}$. Then

$$\Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, \dots\}.$$

The set

$$\{a, aa, aab\}$$

is a language on Σ. Since it has a finite number of sentences, we call it a *finite language*. The set

$$L = \{a^n b^n : n \geq 0\}$$

is also a language on Σ. The strings $aabb$ and $aaaabbbb$ are in the language L, but the string abb is not in L. This language is infinite. Most interesting languages are infinite. ∎

Example 1.9

Let $\Sigma = \{a, b\}$. Then

$$\Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}.$$

The set

$$\{a, aa, aab\}$$

is a language on Σ. Since it has a finite number of sentences, we call it a *finite language*. The set

$$L = \{a^n b^n : n \geq 0\}$$

is also a language on Σ. The strings $aabb$ and $aaaabbbb$ are in the language L, but the string abb is not in L. This language is infinite. Most interesting languages are infinite. ■

Example 1.9

Let $\Sigma = \{a, b\}$. Then

$$\Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}.$$

The set

$$\{a, aa, aab\}$$

is a language on Σ. Since it has a finite number of sentences, we call it a *finite language*. The set

$$L = \{a^n b^n : n \geq 0\}$$

is also a language on Σ. The strings $aabb$ and $aaaabbbb$ are in the language L, but the string abb is not in L. This language is infinite. Most interesting languages are infinite. ■

Example 1.9

Let $\Sigma = \{a, b\}$. Then

$$\Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, \dots\}.$$

The set

$$\{a, aa, aab\}$$

is a language on Σ. Since it has a finite number of sentences, we call it a *finite language*. The set

$$L = \{a^n b^n : n \geq 0\}$$

is also a language on Σ. The strings $aabb$ and $aaaabbbb$ are in the language L, but the string abb is not in L. This language is infinite. Most interesting languages are infinite. ■

Example 1.9

Let $\Sigma = \{a, b\}$. Then

$$\Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}.$$

The set

$$\{a, aa, aab\}$$

is a language on Σ. Since it has a finite number of sentences, we call it a *finite language*. The set

$$L = \{a^n b^n : n \geq 0\}$$

is also a language on Σ. The strings $aabb$ and $aaaabbbb$ are in the language L, but the string abb is not in L. This language is infinite. Most interesting languages are infinite. ■

Example 1.9

Let $\Sigma = \{a, b\}$. Then

$$\Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}.$$

The set

$$\{a, aa, aab\}$$

is a language on Σ. Since it has a finite number of sentences, we call it a *finite language*. The set

$$L = \{a^n b^n : n \geq 0\}$$

is also a language on Σ. The strings $aabb$ and $aaaabbbb$ are in the language L, but the string abb is not in L. This language is infinite. Most interesting languages are infinite. ∎

Example 1.9

Let $\Sigma = \{a, b\}$. Then

$$\Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}.$$

The set

$$\{a, aa, aab\}$$

is a language on Σ. Since it has a finite number of sentences, we call it a *finite language*. The set

$$L = \{a^n b^n : n \geq 0\}$$

is also a language on Σ. The strings $aabb$ and $aaaabbbb$ are in the language L, but the string abb is not in L. This language is infinite. Most interesting languages are infinite.

Example 1.9

Let $\Sigma = \{a, b\}$. Then

$$\Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}.$$

The set

$$\{a, aa, aab\}$$

is a language on Σ. Since it has a finite number of sentences, we call it a *finite language*. The set

$$L = \{a^n b^n : n \geq 0\}$$

is also a language on Σ. The strings $aabb$ and $aaaabbbb$ are in the language L, but the string abb is not in L. This language is infinite. Most interesting languages are infinite.

Example 1.9

Let $\Sigma = \{a, b\}$. Then

$$\Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}.$$

The set

$$\{a, aa, aab\}$$

is a language on Σ. Since it has a finite number of sentences, we call it a *finite language*. The set

$$L = \{a^n b^n : n \geq 0\}$$

is also a language on Σ. The strings $aabb$ and $aaaabbbb$ are in the language L, but the string abb is not in L. This language is infinite. Most interesting languages are infinite.

Example 1.9

Let $\Sigma = \{a, b\}$. Then

$$\Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}.$$

The set

$$\{a, aa, aab\}$$

is a language on Σ. Since it has a finite number of sentences, we call it a *finite language*. The set

$$L = \{a^n b^n : n \geq 0\}$$

is also a language on Σ. The strings $aabb$ and $aaaabbbb$ are in the language L, but the string abb is not in L. This language is infinite. Most interesting languages are infinite.

Example 1.9

Let $\Sigma = \{a, b\}$. Then

$$\Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}.$$

The set

$$\{a, aa, aab\}$$

is a language on Σ. Since it has a finite number of sentences, we call it a *finite language*. The set

$$L = \{a^n b^n : n \geq 0\}$$

is also a language on Σ. The strings $aabb$ and $aaaabbbb$ are in the language L, but the string abb is not in L. This language is infinite. Most interesting languages are infinite. ■

Since languages are sets, the union, intersection, and difference of two languages are immediately defined. The complement of a language is defined with respect to Σ^*; that is, the complement of L is

$$\overline{L} = \Sigma^* - L.$$

The *reverse of a language* is the set of all string reversals, that is,

$$L^R = \left\{ w^R : w \in L \right\}.$$

The *concatenation of two languages* L_1 and L_2 is the set of all strings obtained by concatenating any element of L_1 with any element of L_2; specifically,

$$L_1 L_2 = \{xy : x \in L_1, y \in L_2\}.$$

We define L^n as L concatenated with itself n times, with the special cases

$$L^0 = \{\lambda\}$$

and

$$L^1 = L$$

or every language L.

Since languages are sets, the union, intersection, and difference of two languages are immediately defined. The complement of a language is defined with respect to Σ^*; that is, the complement of L is

$$\overline{L} = \Sigma^* - L.$$

The *reverse of a language* is the set of all string reversals, that is,

$$L^R = \left\{w^R : w \in L\right\}.$$

The *concatenation of two languages* L_1 and L_2 is the set of all strings obtained by concatenating any element of L_1 with any element of L_2; specifically,

$$L_1L_2 = \{xy : x \in L_1, y \in L_2\}.$$

We define L^n as L concatenated with itself n times, with the special cases

$$L^0 = \{\lambda\}$$

and

$$L^1 = L$$

or every language L.

Since languages are sets, the union, intersection, and difference of two languages are immediately defined. The complement of a language is defined with respect to Σ^*; that is, the complement of L is

$$\overline{L} = \Sigma^* - L.$$

The *reverse of a language* is the set of all string reversals, that is,

$$L^R = \left\{ w^R \colon w \in L \right\}.$$

The *concatenation of two languages* L_1 and L_2 is the set of all strings obtained by concatenating any element of L_1 with any element of L_2; specifically,

$$L_1 L_2 = \{ xy \colon x \in L_1, y \in L_2 \}.$$

We define L^n as L concatenated with itself n times, with the special cases

$$L^0 = \{\lambda\}$$

and

$$L^1 = L$$

or every language L.

Since languages are sets, the union, intersection, and difference of two languages are immediately defined. The complement of a language is defined with respect to Σ^*; that is, the complement of L is

$$\overline{L} = \Sigma^* - L.$$

The *reverse of a language* is the set of all string reversals, that is,

$$L^R = \left\{ w^R : w \in L \right\}.$$

The *concatenation of two languages* L_1 and L_2 is the set of all strings obtained by concatenating any element of L_1 with any element of L_2; specifically,

$$L_1 L_2 = \{ xy : x \in L_1, y \in L_2 \}.$$

We define L^n as L concatenated with itself n times, with the special cases

$$L^0 = \{\lambda\}$$

and

$$L^1 = L$$

or every language L.

Since languages are sets, the union, intersection, and difference of two languages are immediately defined. The complement of a language is defined with respect to Σ^*; that is, the complement of L is

$$\overline{L} = \Sigma^* - L.$$

The *reverse of a language* is the set of all string reversals, that is,

$$L^R = \left\{ w^R : w \in L \right\}.$$

The *concatenation of two languages* L_1 and L_2 is the set of all strings obtained by concatenating any element of L_1 with any element of L_2; specifically,

$$L_1 L_2 = \{ xy : x \in L_1, y \in L_2 \}.$$

We define L^n as L concatenated with itself n times, with the special cases

$$L^0 = \{\lambda\}$$

and

$$L^1 = L$$

or every language L.

Since languages are sets, the union, intersection, and difference of two languages are immediately defined. The complement of a language is defined with respect to Σ^*; that is, the complement of L is

$$\overline{L} = \Sigma^* - L.$$

The *reverse of a language* is the set of all string reversals, that is,

$$L^R = \left\{w^R : w \in L\right\}.$$

The *concatenation of two languages* L_1 and L_2 is the set of all strings obtained by concatenating any element of L_1 with any element of L_2; specifically,

$$L_1L_2 = \{xy : x \in L_1, y \in L_2\}.$$

We define L^n as L concatenated with itself n times, with the special cases

$$L^0 = \{\lambda\}$$

and

$$L^1 = L$$

or every language L.

Since languages are sets, the union, intersection, and difference of two languages are immediately defined. The complement of a language is defined with respect to Σ^*; that is, the complement of L is

$$\overline{L} = \Sigma^* - L.$$

The *reverse of a language* is the set of all string reversals, that is,

$$L^R = \left\{w^R : w \in L\right\}.$$

The *concatenation of two languages* L_1 and L_2 is the set of all strings obtained by concatenating any element of L_1 with any element of L_2; specifically,

$$L_1 L_2 = \{xy : x \in L_1, y \in L_2\}.$$

We define L^n as L concatenated with itself n times, with the special cases

$$L^0 = \{\lambda\}$$

and

$$L^1 = L$$

or every language L.

Since languages are sets, the union, intersection, and difference of two languages are immediately defined. The complement of a language is defined with respect to Σ^*; that is, the complement of L is

$$\overline{L} = \Sigma^* - L.$$

The *reverse of a language* is the set of all string reversals, that is,

$$L^R = \left\{ w^R : w \in L \right\}.$$

The *concatenation of two languages* L_1 and L_2 is the set of all strings obtained by concatenating any element of L_1 with any element of L_2; specifically,

$$L_1 L_2 = \{xy : x \in L_1, y \in L_2\}.$$

We define L^n as L concatenated with itself n times, with the special cases

$$L^0 = \{\lambda\}$$

and

$$L^1 = L$$

or every language L.

Since languages are sets, the union, intersection, and difference of two languages are immediately defined. The complement of a language is defined with respect to Σ^*; that is, the complement of L is

$$\overline{L} = \Sigma^* - L.$$

The *reverse of a language* is the set of all string reversals, that is,

$$L^R = \left\{ w^R : w \in L \right\}.$$

The *concatenation of two languages* L_1 and L_2 is the set of all strings obtained by concatenating any element of L_1 with any element of L_2; specifically,

$$L_1 L_2 = \{xy : x \in L_1, y \in L_2\}.$$

We define L^n as L concatenated with itself n times, with the special cases

$$L^0 = \{\lambda\}$$

and

$$L^1 = L$$

or every language L.

Since languages are sets, the union, intersection, and difference of two languages are immediately defined. The complement of a language is defined with respect to Σ^*; that is, the complement of L is

$$\overline{L} = \Sigma^* - L.$$

The *reverse of a language* is the set of all string reversals, that is,

$$L^R = \left\{ w^R : w \in L \right\}.$$

The *concatenation of two languages* L_1 and L_2 is the set of all strings obtained by concatenating any element of L_1 with any element of L_2; specifically,

$$L_1 L_2 = \{xy : x \in L_1, y \in L_2\}.$$

We define L^n as L concatenated with itself n times, with the special cases

$$L^0 = \{\lambda\}$$

and

$$L^1 = L$$

or every language L.

Since languages are sets, the union, intersection, and difference of two languages are immediately defined. The complement of a language is defined with respect to Σ^*; that is, the complement of L is

$$\overline{L} = \Sigma^* - L.$$

The *reverse of a language* is the set of all string reversals, that is,

$$L^R = \left\{ w^R : w \in L \right\}.$$

The *concatenation of two languages* L_1 and L_2 is the set of all strings obtained by concatenating any element of L_1 with any element of L_2; specifically,

$$L_1 L_2 = \{ xy : x \in L_1, y \in L_2 \}.$$

We define L^n as L concatenated with itself n times, with the special cases

$$L^0 = \{\lambda\}$$

and

$$L^1 = L$$

or every language L.

Since languages are sets, the union, intersection, and difference of two languages are immediately defined. The complement of a language is defined with respect to Σ^*; that is, the complement of L is

$$\overline{L} = \Sigma^* - L.$$

The *reverse of a language* is the set of all string reversals, that is,

$$L^R = \left\{ w^R \colon w \in L \right\}.$$

The *concatenation of two languages* L_1 and L_2 is the set of all strings obtained by concatenating any element of L_1 with any element of L_2; specifically,

$$L_1 L_2 = \{ xy \colon x \in L_1, y \in L_2 \}.$$

We define L^n as L concatenated with itself n times, with the special cases

$$L^0 = \{\lambda\}$$

and

$$L^1 = L$$

or every language L.

Since languages are sets, the union, intersection, and difference of two languages are immediately defined. The complement of a language is defined with respect to Σ^*; that is, the complement of L is

$$\overline{L} = \Sigma^* - L.$$

The *reverse of a language* is the set of all string reversals, that is,

$$L^R = \left\{ w^R \colon w \in L \right\}.$$

The *concatenation of two languages* L_1 and L_2 is the set of all strings obtained by concatenating any element of L_1 with any element of L_2; specifically,

$$L_1 L_2 = \{xy \colon x \in L_1, y \in L_2\}.$$

We define L^n as L concatenated with itself n times, with the special cases

$$L^0 = \{\lambda\}$$

and

$$L^1 = L$$

or every language L.

Since languages are sets, the union, intersection, and difference of two languages are immediately defined. The complement of a language is defined with respect to Σ^*; that is, the complement of L is

$$\overline{L} = \Sigma^* - L.$$

The *reverse of a language* is the set of all string reversals, that is,

$$L^R = \left\{w^R : w \in L\right\}.$$

The *concatenation of two languages* L_1 and L_2 is the set of all strings obtained by concatenating any element of L_1 with any element of L_2; specifically,

$$L_1 L_2 = \{xy : x \in L_1, y \in L_2\}.$$

We define L^n as L concatenated with itself n times, with the special cases

$$L^0 = \{\lambda\}$$

and

$$L^1 = L$$

or every language L.

Since languages are sets, the union, intersection, and difference of two languages are immediately defined. The complement of a language is defined with respect to Σ^*; that is, the complement of L is

$$\overline{L} = \Sigma^* - L.$$

The *reverse of a language* is the set of all string reversals, that is,

$$L^R = \left\{w^R : w \in L\right\}.$$

The *concatenation of two languages* L_1 and L_2 is the set of all strings obtained by concatenating any element of L_1 with any element of L_2; specifically,

$$L_1 L_2 = \{xy : x \in L_1, y \in L_2\}.$$

We define L^n as L concatenated with itself n times, with the special cases

$$L^0 = \{\lambda\}$$

and

$$L^1 = L$$

or every language L.

Since languages are sets, the union, intersection, and difference of two languages are immediately defined. The complement of a language is defined with respect to Σ^*; that is, the complement of L is

$$\overline{L} = \Sigma^* - L.$$

The *reverse of a language* is the set of all string reversals, that is,

$$L^R = \left\{ w^R : w \in L \right\}.$$

The *concatenation of two languages* L_1 and L_2 is the set of all strings obtained by concatenating any element of L_1 with any element of L_2; specifically,

$$L_1 L_2 = \{xy : x \in L_1, y \in L_2\}.$$

We define L^n as L concatenated with itself n times, with the special cases

$$L^0 = \{\lambda\}$$

and

$$L^1 = L$$

or every language L.

Finally, we define the *star-closure of a language* L as

$$L^* = L^0 \cup L_1 \cup L_2 \cup \cdots$$

and the *positive closure* of L as

$$L^+ = L^1 \cup L_2 \cup L_3 \cup \cdots .$$

Example 1.10

If

$$L = \{a^n b^n : n \geq 0\},$$

then

$$L^2 = \{a^n b^n a^m b^m : n \geq 0, m \geq 0\}.$$

Note that n and m in the above are unrelated; the string $aabbaaabbb$ is in L^2.

The reverse of L is easily described in set notation as

$$L^R = \{b^n a^n : n \geq 0\},$$

but it is considerably harder to describe $\overline{L}$ or L^* this way. A few tries will quickly convince you of the limitation of set notation for the specification of complicated languages.

Finally, we define the *star-closure of a language* L as

$$L^* = L^0 \cup L_1 \cup L_2 \cup \cdots$$

and the *positive closure* of L as

$$L^+ = L^1 \cup L_2 \cup L_3 \cup \cdots .$$

Example 1.10

If

$$L = \{a^n b^n : n \geq 0\},$$

then

$$L^2 = \{a^n b^n a^m b^m : n \geq 0, m \geq 0\}.$$

Note that n and m in the above are unrelated; the string $aabbaaabbb$ is in L^2.

The reverse of L is easily described in set notation as

$$L^R = \{b^n a^n : n \geq 0\},$$

but it is considerably harder to describe $\overline{L}$ or L^* this way. A few tries will quickly convince you of the limitation of set notation for the specification of complicated languages.

Finally, we define the *star-closure of a language* L as

$$L^* = L^0 \cup L_1 \cup L_2 \cup \cdots$$

and the *positive closure* of L as

$$L^+ = L^1 \cup L_2 \cup L_3 \cup \cdots .$$

Example 1.10

If

$$L = \{a^n b^n : n \geq 0\},$$

then

$$L^2 = \{a^n b^n a^m b^m : n \geq 0, m \geq 0\}.$$

Note that n and m in the above are unrelated; the string $aabbaaabbb$ is in L^2. The reverse of L is easily described in set notation as

$$L^R = \{b^n a^n : n \geq 0\},$$

but it is considerably harder to describe $\overline{L}$ or L^* this way. A few tries will quickly convince you of the limitation of set notation for the specification of complicated languages.

Finally, we define the *star-closure of a language* L as

$$L^* = L^0 \cup L_1 \cup L_2 \cup \cdots$$

and the *positive closure* of L as

$$L^+ = L^1 \cup L_2 \cup L_3 \cup \cdots .$$

Example 1.10

If

$$L = \{a^n b^n : n \geq 0\},$$

then

$$L^2 = \{a^n b^n a^m b^m : n \geq 0, m \geq 0\}.$$

Note that n and m in the above are unrelated; the string $aabbaaabbb$ is in L^2.

The reverse of L is easily described in set notation as

$$L^R = \{b^n a^n : n \geq 0\},$$

but it is considerably harder to describe $\overline{L}$ or L^* this way. A few tries will quickly convince you of the limitation of set notation for the specification of complicated languages.

Finally, we define the *star-closure of a language* L as

$$L^* = L^0 \cup L_1 \cup L_2 \cup \cdots$$

and the *positive closure* of L as

$$L^+ = L^1 \cup L_2 \cup L_3 \cup \cdots .$$

Example 1.10

If

$$L = \{a^n b^n : n \geq 0\},$$

then

$$L^2 = \{a^n b^n a^m b^m : n \geq 0, m \geq 0\}.$$

Note that n and m in the above are unrelated; the string $aabbaaabbb$ is in L^2.
The reverse of L is easily described in set notation as

$$L^R = \{b^n a^n : n \geq 0\},$$

but it is considerably harder to describe $\overline{L}$ or L^* this way. A few tries will quickly convince you of the limitation of set notation for the specification of complicated languages.

Finally, we define the *star-closure of a language* L as

$$L^* = L^0 \cup L_1 \cup L_2 \cup \cdots$$

and the *positive closure* of L as

$$L^+ = L^1 \cup L_2 \cup L_3 \cup \cdots .$$

Example 1.10

If

$$L = \{a^n b^n : n \geq 0\},$$

then

$$L^2 = \{a^n b^n a^m b^m : n \geq 0, m \geq 0\}.$$

Note that n and m in the above are unrelated; the string $aabbaaabbb$ is in L^2.

The reverse of L is easily described in set notation as

$$L^R = \{b^n a^n : n \geq 0\},$$

but it is considerably harder to describe $\overline{L}$ or L^* this way. A few tries will quickly convince you of the limitation of set notation for the specification of complicated languages.

Finally, we define the *star-closure of a language* L as

$$L^* = L^0 \cup L_1 \cup L_2 \cup \cdots$$

and the *positive closure* of L as

$$L^+ = L^1 \cup L_2 \cup L_3 \cup \cdots .$$

Example 1.10

If

$$L = \{a^n b^n : n \geq 0\},$$

then

$$L^2 = \{a^n b^n a^m b^m : n \geq 0, m \geq 0\}.$$

Note that n and m in the above are unrelated; the string $aabbaaabbb$ is in L^2.

The reverse of L is easily described in set notation as

$$L^R = \{b^n a^n : n \geq 0\},$$

but it is considerably harder to describe $\overline{L}$ or L^* this way. A few tries will quickly convince you of the limitation of set notation for the specification of complicated languages.

Finally, we define the *star-closure of a language* L as

$$L^* = L^0 \cup L_1 \cup L_2 \cup \cdots$$

and the *positive closure* of L as

$$L^+ = L^1 \cup L_2 \cup L_3 \cup \cdots .$$

Example 1.10

If

$$L = \{a^n b^n : n \geq 0\},$$

then

$$L^2 = \{a^n b^n a^m b^m : n \geq 0, m \geq 0\}.$$

Note that n and m in the above are unrelated; the string $aabbaaabbb$ is in L^2.

The reverse of L is easily described in set notation as

$$L^R = \{b^n a^n : n \geq 0\},$$

but it is considerably harder to describe $\overline{L}$ or L^* this way. A few tries will quickly convince you of the limitation of set notation for the specification of complicated languages.

Finally, we define the *star-closure of a language* L as

$$L^* = L^0 \cup L_1 \cup L_2 \cup \cdots$$

and the *positive closure* of L as

$$L^+ = L^1 \cup L_2 \cup L_3 \cup \cdots .$$

Example 1.10

If

$$L = \{a^n b^n : n \geq 0\},$$

then

$$L^2 = \{a^n b^n a^m b^m : n \geq 0, m \geq 0\}.$$

Note that n and m in the above are unrelated; the string $aabbaaabbb$ is in L^2.

The reverse of L is easily described in set notation as

$$L^R = \{b^n a^n : n \geq 0\},$$

but it is considerably harder to describe $\overline{L}$ or L^* this way. A few tries will quickly convince you of the limitation of set notation for the specification of complicated languages.

Finally, we define the *star-closure of a language* L as

$$L^* = L^0 \cup L_1 \cup L_2 \cup \cdots$$

and the *positive closure* of L as

$$L^+ = L^1 \cup L_2 \cup L_3 \cup \cdots .$$

Example 1.10

If

$$L = \{a^n b^n : n \geq 0\},$$

then

$$L^2 = \{a^n b^n a^m b^m : n \geq 0, m \geq 0\}.$$

Note that n and m in the above are unrelated; the string $aabbaaabbb$ is in L^2.

The reverse of L is easily described in set notation as

$$L^R = \{b^n a^n : n \geq 0\},$$

but it is considerably harder to describe $\overline{L}$ or L^* this way. A few tries will quickly convince you of the limitation of set notation for the specification of complicated languages.

Finally, we define the *star-closure of a language* L as

$$L^* = L^0 \cup L_1 \cup L_2 \cup \cdots$$

and the *positive closure* of L as

$$L^+ = L^1 \cup L_2 \cup L_3 \cup \cdots .$$

Example 1.10

If

$$L = \{a^n b^n : n \geq 0\},$$

then

$$L^2 = \{a^n b^n a^m b^m : n \geq 0, m \geq 0\}.$$

Note that n and m in the above are unrelated; the string $aabbaaabbb$ is in L^2. The reverse of L is easily described in set notation as

$$L^R = \{b^n a^n : n \geq 0\},$$

but it is considerably harder to describe $\overline{L}$ or L^* this way. A few tries will quickly convince you of the limitation of set notation for the specification of complicated languages.

Finally, we define the *star-closure of a language* L as

$$L^* = L^0 \cup L_1 \cup L_2 \cup \cdots$$

and the *positive closure* of L as

$$L^+ = L^1 \cup L_2 \cup L_3 \cup \cdots .$$

Example 1.10

If

$$L = \{a^n b^n : n \geq 0\},$$

then

$$L^2 = \{a^n b^n a^m b^m : n \geq 0, m \geq 0\}.$$

Note that n and m in the above are unrelated; the string $aabbaaabbb$ is in L^2.

The reverse of L is easily described in set notation as

$$L^R = \{b^n a^n : n \geq 0\},$$

but it is considerably harder to describe $\overline{L}$ or L^* this way. A few tries will quickly convince you of the limitation of set notation for the specification of complicated languages.

Finally, we define the *star-closure of a language* L as

$$L^* = L^0 \cup L_1 \cup L_2 \cup \cdots$$

and the *positive closure* of L as

$$L^+ = L^1 \cup L_2 \cup L_3 \cup \cdots .$$

Example 1.10

If

$$L = \{a^n b^n : n \geq 0\},$$

then

$$L^2 = \{a^n b^n a^m b^m : n \geq 0, m \geq 0\}.$$

Note that n and m in the above are unrelated; the string $aabbaaabbb$ is in L^2.

The reverse of L is easily described in set notation as

$$L^R = \{b^n a^n : n \geq 0\},$$

but it is considerably harder to describe $\overline{L}$ or L^* this way. A few tries will quickly convince you of the limitation of set notation for the specification of complicated languages.

Finally, we define the *star-closure of a language* L as

$$L^* = L^0 \cup L_1 \cup L_2 \cup \cdots$$

and the *positive closure* of L as

$$L^+ = L^1 \cup L_2 \cup L_3 \cup \cdots .$$

Example 1.10

If

$$L = \{a^n b^n : n \geq 0\},$$

then

$$L^2 = \{a^n b^n a^m b^m : n \geq 0, m \geq 0\}.$$

Note that n and m in the above are unrelated; the string $aabbaaabbb$ is in L^2.

The reverse of L is easily described in set notation as

$$L^R = \{b^n a^n : n \geq 0\},$$

but it is considerably harder to describe $\overline{L}$ or L^* this way. A few tries will quickly convince you of the limitation of set notation for the specification of complicated languages.

Finally, we define the *star-closure of a language* L as

$$L^* = L^0 \cup L_1 \cup L_2 \cup \cdots$$

and the *positive closure* of L as

$$L^+ = L^1 \cup L_2 \cup L_3 \cup \cdots .$$

Example 1.10

If

$$L = \{a^n b^n : n \geq 0\},$$

then

$$L^2 = \{a^n b^n a^m b^m : n \geq 0, m \geq 0\}.$$

Note that n and m in the above are unrelated; the string $aabbaaabbb$ is in L^2.

The reverse of L is easily described in set notation as

$$L^R = \{b^n a^n : n \geq 0\},$$

but it is considerably harder to describe $\overline{L}$ or L^* this way. A few tries will quickly convince you of the limitation of set notation for the specification of complicated languages.

Finally, we define the *star-closure of a language* L as

$$L^* = L^0 \cup L_1 \cup L_2 \cup \cdots$$

and the *positive closure* of L as

$$L^+ = L^1 \cup L_2 \cup L_3 \cup \cdots .$$

Example 1.10

If

$$L = \{a^n b^n : n \geq 0\},$$

then

$$L^2 = \{a^n b^n a^m b^m : n \geq 0, m \geq 0\}.$$

Note that n and m in the above are unrelated; the string $aabbaaabbb$ is in L^2.

The reverse of L is easily described in set notation as

$$L^R = \{b^n a^n : n \geq 0\},$$

but it is considerably harder to describe $\overline{L}$ or L^* this way. A few tries will quickly convince you of the limitation of set notation for the specification of complicated languages.

Finally, we define the *star-closure of a language* L as

$$L^* = L^0 \cup L_1 \cup L_2 \cup \cdots$$

and the *positive closure* of L as

$$L^+ = L^1 \cup L_2 \cup L_3 \cup \cdots .$$

Example 1.10

If

$$L = \{a^n b^n : n \geq 0\},$$

then

$$L^2 = \{a^n b^n a^m b^m : n \geq 0, m \geq 0\}.$$

Note that n and m in the above are unrelated; the string $aabbaaabbb$ is in L^2.

The reverse of L is easily described in set notation as

$$L^R = \{b^n a^n : n \geq 0\},$$

but it is considerably harder to describe $\overline{L}$ or L^* this way. A few tries will quickly convince you of the limitation of set notation for the specification of complicated languages. ■

To study languages mathematically, we need a mechanism to describe them. Everyday language is imprecise and ambiguous, so informal descriptions in English are often inadequate. The set notation used in Examples 1.9 and 1.10 is more suitable, but limited. As we proceed we will learn about several language-definition mechanisms that are useful in different circumstances. Here we introduce a common and powerful one, the notion of a *grammar*.

A grammar for the English language tells us whether a particular sentence is well formed or not. A typical rule of English grammar is "a sentence can consist of a noun phrase followed by a predicate". More concisely we write this as

$$\langle \text{sentence} \rangle \to \langle \text{noun\_phrase} \rangle \langle \text{predicate} \rangle,$$

with the obvious interpretation. This is, of course, not enough to deal with actual sentences. We must now provide definitions for the newly introduced constructs $\langle \text{noun\_phrase} \rangle$ and $\langle \text{predicate} \rangle$. If we do so by

$$\langle \text{noun\_phrase} \rangle \to \langle \text{article} \rangle \langle \text{noun} \rangle,$$
$$\langle \text{predicate} \rangle \to \langle \text{verb} \rangle,$$

and if we associate the actual words "a" and "the" with $\langle \text{article} \rangle$, "boy" and "dog" with $\langle \text{noun} \rangle$, and "runs" and "walks" with $\langle \text{verb} \rangle$, then the grammar tells us that the sentences "a boy runs" and "the dog walks" are properly formed. If we were to give a complete grammar, then in theory, every proper sentence could be explained this way.

To study languages mathematically, we need a mechanism to describe them. Everyday language is imprecise and ambiguous, so informal descriptions in English are often inadequate. The set notation used in Examples 1.9 and 1.10 is more suitable, but limited. As we proceed we will learn about several language-definition mechanisms that are useful in different circumstances. Here we introduce a common and powerful one, the notion of a *grammar*.

A grammar for the English language tells us whether a particular sentence is well formed or not. A typical rule of English grammar is "a sentence can consist of a noun phrase followed by a predicate". More concisely we write this as

$$\langle \text{sentence} \rangle \to \langle \text{noun\_phrase} \rangle \langle \text{predicate} \rangle,$$

with the obvious interpretation. This is, of course, not enough to deal with actual sentences. We must now provide definitions for the newly introduced constructs $\langle \text{noun\_phrase} \rangle$ and $\langle \text{predicate} \rangle$. If we do so by

$$\langle \text{noun\_phrase} \rangle \to \langle \text{article} \rangle \langle \text{noun} \rangle,$$
$$\langle \text{predicate} \rangle \to \langle \text{verb} \rangle,$$

and if we associate the actual words "a" and "the" with $\langle \text{article} \rangle$, "boy" and "dog" with $\langle \text{noun} \rangle$, and "runs" and "walks" with $\langle \text{verb} \rangle$, then the grammar tells us that the sentences "a boy runs" and "the dog walks" are properly formed. If we were to give a complete grammar, then in theory, every proper sentence could be explained this way.

To study languages mathematically, we need a mechanism to describe them. Everyday language is imprecise and ambiguous, so informal descriptions in English are often inadequate. The set notation used in Examples 1.9 and 1.10 is more suitable, but limited. As we proceed we will learn about several language-definition mechanisms that are useful in different circumstances. Here we introduce a common and powerful one, the notion of a *grammar*.

A grammar for the English language tells us whether a particular sentence is well formed or not. A typical rule of English grammar is "a sentence can consist of a noun phrase followed by a predicate". More concisely we write this as

$$\langle \text{sentence} \rangle \to \langle \text{noun\_phrase} \rangle \langle \text{predicate} \rangle,$$

with the obvious interpretation. This is, of course, not enough to deal with actual sentences. We must now provide definitions for the newly introduced constructs $\langle \text{noun\_phrase} \rangle$ and $\langle \text{predicate} \rangle$. If we do so by

$$\langle \text{noun\_phrase} \rangle \to \langle \text{article} \rangle \langle \text{noun} \rangle,$$
$$\langle \text{predicate} \rangle \to \langle \text{verb} \rangle,$$

and if we associate the actual words "a" and "the" with $\langle \text{article} \rangle$, "boy" and "dog" with $\langle \text{noun} \rangle$, and "runs" and "walks" with $\langle \text{verb} \rangle$, then the grammar tells us that the sentences "a boy runs" and "the dog walks" are properly formed. If we were to give a complete grammar, then in theory, every proper sentence could be explained this way.

To study languages mathematically, we need a mechanism to describe them. Everyday language is imprecise and ambiguous, so informal descriptions in English are often inadequate. The set notation used in Examples 1.9 and 1.10 is more suitable, but limited. As we proceed we will learn about several language-definition mechanisms that are useful in different circumstances. Here we introduce a common and powerful one, the notion of a *grammar*.

A grammar for the English language tells us whether a particular sentence is well formed or not. A typical rule of English grammar is "a sentence can consist of a noun phrase followed by a predicate". More concisely we write this as

$$\langle \text{sentence} \rangle \to \langle \text{noun\_phrase} \rangle \langle \text{predicate} \rangle,$$

with the obvious interpretation. This is, of course, not enough to deal with actual sentences. We must now provide definitions for the newly introduced constructs $\langle \text{noun\_phrase} \rangle$ and $\langle \text{predicate} \rangle$. If we do so by

$$\langle \text{noun\_phrase} \rangle \to \langle \text{article} \rangle \langle \text{noun} \rangle,$$
$$\langle \text{predicate} \rangle \to \langle \text{verb} \rangle,$$

and if we associate the actual words "a" and "the" with $\langle \text{article} \rangle$, "boy" and "dog" with $\langle \text{noun} \rangle$, and "runs" and "walks" with $\langle \text{verb} \rangle$, then the grammar tells us that the sentences "a boy runs" and "the dog walks" are properly formed. If we were to give a complete grammar, then in theory, every proper sentence could be explained this way.

To study languages mathematically, we need a mechanism to describe them. Everyday language is imprecise and ambiguous, so informal descriptions in English are often inadequate. The set notation used in Examples 1.9 and 1.10 is more suitable, but limited. As we proceed we will learn about several language-definition mechanisms that are useful in different circumstances. Here we introduce a common and powerful one, the notion of a *grammar*.

A grammar for the English language tells us whether a particular sentence is well formed or not. A typical rule of English grammar is "a sentence can consist of a noun phrase followed by a predicate". More concisely we write this as

$$\langle \text{sentence} \rangle \to \langle \text{noun\_phrase} \rangle \langle \text{predicate} \rangle,$$

with the obvious interpretation. This is, of course, not enough to deal with actual sentences. We must now provide definitions for the newly introduced constructs $\langle \text{noun\_phrase} \rangle$ and $\langle \text{predicate} \rangle$. If we do so by

$$\langle \text{noun\_phrase} \rangle \to \langle \text{article} \rangle \langle \text{noun} \rangle,$$
$$\langle \text{predicate} \rangle \to \langle \text{verb} \rangle,$$

and if we associate the actual words "a" and "the" with $\langle \text{article} \rangle$, "boy" and "dog" with $\langle \text{noun} \rangle$, and "runs" and "walks" with $\langle \text{verb} \rangle$, then the grammar tells us that the sentences "a boy runs" and "the dog walks" are properly formed. If we were to give a complete grammar, then in theory, every proper sentence could be explained this way.

To study languages mathematically, we need a mechanism to describe them. Everyday language is imprecise and ambiguous, so informal descriptions in English are often inadequate. The set notation used in Examples 1.9 and 1.10 is more suitable, but limited. As we proceed we will learn about several language-definition mechanisms that are useful in different circumstances. Here we introduce a common and powerful one, the notion of a *grammar*.

A grammar for the English language tells us whether a particular sentence is well formed or not. A typical rule of English grammar is "a sentence can consist of a noun phrase followed by a predicate". More concisely we write this as

$$\langle \text{sentence} \rangle \rightarrow \langle \text{noun\_phrase} \rangle \langle \text{predicate} \rangle,$$

with the obvious interpretation. This is, of course, not enough to deal with actual sentences. We must now provide definitions for the newly introduced constructs $\langle \text{noun\_phrase} \rangle$ and $\langle \text{predicate} \rangle$. If we do so by

$$\langle \text{noun\_phrase} \rangle \rightarrow \langle \text{article} \rangle \langle \text{noun} \rangle,$$
$$\langle \text{predicate} \rangle \rightarrow \langle \text{verb} \rangle,$$

and if we associate the actual words "a" and "the" with $\langle \text{article} \rangle$, "boy" and "dog" with $\langle \text{noun} \rangle$, and "runs" and "walks" with $\langle \text{verb} \rangle$, then the grammar tells us that the sentences "a boy runs" and "the dog walks" are properly formed. If we were to give a complete grammar, then in theory, every proper sentence could be explained this way.

To study languages mathematically, we need a mechanism to describe them. Everyday language is imprecise and ambiguous, so informal descriptions in English are often inadequate. The set notation used in Examples 1.9 and 1.10 is more suitable, but limited. As we proceed we will learn about several language-definition mechanisms that are useful in different circumstances. Here we introduce a common and powerful one, the notion of a *grammar*.

A grammar for the English language tells us whether a particular sentence is well formed or not. A typical rule of English grammar is "a sentence can consist of a noun phrase followed by a predicate". More concisely we write this as

$$\langle \text{sentence} \rangle \to \langle \text{noun\_phrase} \rangle \langle \text{predicate} \rangle,$$

with the obvious interpretation. This is, of course, not enough to deal with actual sentences. We must now provide definitions for the newly introduced constructs $\langle \text{noun\_phrase} \rangle$ and $\langle \text{predicate} \rangle$. If we do so by

$$\langle \text{noun\_phrase} \rangle \to \langle \text{article} \rangle \langle \text{noun} \rangle,$$
$$\langle \text{predicate} \rangle \to \langle \text{verb} \rangle,$$

and if we associate the actual words "a" and "the" with $\langle \text{article} \rangle$, "boy" and "dog" with $\langle \text{noun} \rangle$, and "runs" and "walks" with $\langle \text{verb} \rangle$, then the grammar tells us that the sentences "a boy runs" and "the dog walks" are properly formed. If we were to give a complete grammar, then in theory, every proper sentence could be explained this way.

To study languages mathematically, we need a mechanism to describe them. Everyday language is imprecise and ambiguous, so informal descriptions in English are often inadequate. The set notation used in Examples 1.9 and 1.10 is more suitable, but limited. As we proceed we will learn about several language-definition mechanisms that are useful in different circumstances. Here we introduce a common and powerful one, the notion of a *grammar*.

A grammar for the English language tells us whether a particular sentence is well formed or not. A typical rule of English grammar is "a sentence can consist of a noun phrase followed by a predicate". More concisely we write this as

$$\langle \text{sentence} \rangle \rightarrow \langle \text{noun\_phrase} \rangle \langle \text{predicate} \rangle,$$

with the obvious interpretation. This is, of course, not enough to deal with actual sentences. We must now provide definitions for the newly introduced constructs $\langle \text{noun\_phrase} \rangle$ and $\langle \text{predicate} \rangle$. If we do so by

$$\langle \text{noun\_phrase} \rangle \rightarrow \langle \text{article} \rangle \langle \text{noun} \rangle,$$
$$\langle \text{predicate} \rangle \rightarrow \langle \text{verb} \rangle,$$

and if we associate the actual words "a" and "the" with $\langle \text{article} \rangle$, "boy" and "dog" with $\langle \text{noun} \rangle$, and "runs" and "walks" with $\langle \text{verb} \rangle$, then the grammar tells us that the sentences "a boy runs" and "the dog walks" are properly formed. If we were to give a complete grammar, then in theory, every proper sentence could be explained this way.

To study languages mathematically, we need a mechanism to describe them. Everyday language is imprecise and ambiguous, so informal descriptions in English are often inadequate. The set notation used in Examples 1.9 and 1.10 is more suitable, but limited. As we proceed we will learn about several language-definition mechanisms that are useful in different circumstances. Here we introduce a common and powerful one, the notion of a *grammar*.

A grammar for the English language tells us whether a particular sentence is well formed or not. A typical rule of English grammar is "a sentence can consist of a noun phrase followed by a predicate". More concisely we write this as

$$\langle \text{sentence} \rangle \rightarrow \langle \text{noun\_phrase} \rangle \langle \text{predicate} \rangle,$$

with the obvious interpretation. This is, of course, not enough to deal with actual sentences. We must now provide definitions for the newly introduced constructs $\langle \text{noun\_phrase} \rangle$ and $\langle \text{predicate} \rangle$. If we do so by

$$\langle \text{noun\_phrase} \rangle \rightarrow \langle \text{article} \rangle \langle \text{noun} \rangle,$$
$$\langle \text{predicate} \rangle \rightarrow \langle \text{verb} \rangle,$$

and if we associate the actual words "a" and "the" with $\langle \text{article} \rangle$, "boy" and "dog" with $\langle \text{noun} \rangle$, and "runs" and "walks" with $\langle \text{verb} \rangle$, then the grammar tells us that the sentences "a boy runs" and "the dog walks" are properly formed. If we were to give a complete grammar, then in theory, every proper sentence could be explained this way.

To study languages mathematically, we need a mechanism to describe them. Everyday language is imprecise and ambiguous, so informal descriptions in English are often inadequate. The set notation used in Examples 1.9 and 1.10 is more suitable, but limited. As we proceed we will learn about several language-definition mechanisms that are useful in different circumstances. Here we introduce a common and powerful one, the notion of a *grammar*.

A grammar for the English language tells us whether a particular sentence is well formed or not. A typical rule of English grammar is "a sentence can consist of a noun phrase followed by a predicate". More concisely we write this as

$$\langle \text{sentence} \rangle \to \langle \text{noun\_phrase} \rangle \langle \text{predicate} \rangle,$$

with the obvious interpretation. This is, of course, not enough to deal with actual sentences. We must now provide definitions for the newly introduced constructs $\langle \text{noun\_phrase} \rangle$ and $\langle \text{predicate} \rangle$. If we do so by

$$\langle \text{noun\_phrase} \rangle \to \langle \text{article} \rangle \langle \text{noun} \rangle,$$
$$\langle \text{predicate} \rangle \to \langle \text{verb} \rangle,$$

and if we associate the actual words "a" and "the" with $\langle \text{article} \rangle$, "boy" and "dog" with $\langle \text{noun} \rangle$, and "runs" and "walks" with $\langle \text{verb} \rangle$, then the grammar tells us that the sentences "a boy runs" and "the dog walks" are properly formed. If we were to give a complete grammar, then in theory, every proper sentence could be explained this way.

To study languages mathematically, we need a mechanism to describe them. Everyday language is imprecise and ambiguous, so informal descriptions in English are often inadequate. The set notation used in Examples 1.9 and 1.10 is more suitable, but limited. As we proceed we will learn about several language-definition mechanisms that are useful in different circumstances. Here we introduce a common and powerful one, the notion of a *grammar*.

A grammar for the English language tells us whether a particular sentence is well formed or not. A typical rule of English grammar is "a sentence can consist of a noun phrase followed by a predicate". More concisely we write this as

$$\langle \text{sentence} \rangle \to \langle \text{noun\_phrase} \rangle \langle \text{predicate} \rangle,$$

with the obvious interpretation. This is, of course, not enough to deal with actual sentences. We must now provide definitions for the newly introduced constructs $\langle \text{noun\_phrase} \rangle$ and $\langle \text{predicate} \rangle$. If we do so by

$$\langle \text{noun\_phrase} \rangle \to \langle \text{article} \rangle \langle \text{noun} \rangle,$$
$$\langle \text{predicate} \rangle \to \langle \text{verb} \rangle,$$

and if we associate the actual words "a" and "the" with $\langle \text{article} \rangle$, "boy" and "dog" with $\langle \text{noun} \rangle$, and "runs" and "walks" with $\langle \text{verb} \rangle$, then the grammar tells us that the sentences "a boy runs" and "the dog walks" are properly formed. If we were to give a complete grammar, then in theory, every proper sentence could be explained this way.

To study languages mathematically, we need a mechanism to describe them. Everyday language is imprecise and ambiguous, so informal descriptions in English are often inadequate. The set notation used in Examples 1.9 and 1.10 is more suitable, but limited. As we proceed we will learn about several language-definition mechanisms that are useful in different circumstances. Here we introduce a common and powerful one, the notion of a *grammar*.

A grammar for the English language tells us whether a particular sentence is well formed or not. A typical rule of English grammar is "a sentence can consist of a noun phrase followed by a predicate". More concisely we write this as

$$\langle \text{sentence} \rangle \to \langle \text{noun\_phrase} \rangle \langle \text{predicate} \rangle,$$

with the obvious interpretation. This is, of course, not enough to deal with actual sentences. We must now provide definitions for the newly introduced constructs $\langle \text{noun\_phrase} \rangle$ and $\langle \text{predicate} \rangle$. If we do so by

$$\langle \text{noun\_phrase} \rangle \to \langle \text{article} \rangle \langle \text{noun} \rangle,$$
$$\langle \text{predicate} \rangle \to \langle \text{verb} \rangle,$$

and if we associate the actual words "a" and "the" with $\langle \text{article} \rangle$, "boy" and "dog" with $\langle \text{noun} \rangle$, and "runs" and "walks" with $\langle \text{verb} \rangle$, then the grammar tells us that the sentences "a boy runs" and "the dog walks" are properly formed. If we were to give a complete grammar, then in theory, every proper sentence could be explained this way.

To study languages mathematically, we need a mechanism to describe them. Everyday language is imprecise and ambiguous, so informal descriptions in English are often inadequate. The set notation used in Examples 1.9 and 1.10 is more suitable, but limited. As we proceed we will learn about several language-definition mechanisms that are useful in different circumstances. Here we introduce a common and powerful one, the notion of a *grammar*.

A grammar for the English language tells us whether a particular sentence is well formed or not. A typical rule of English grammar is "a sentence can consist of a noun phrase followed by a predicate". More concisely we write this as

$$\langle \text{sentence} \rangle \to \langle \text{noun\_phrase} \rangle \langle \text{predicate} \rangle,$$

with the obvious interpretation. This is, of course, not enough to deal with actual sentences. We must now provide definitions for the newly introduced constructs $\langle \text{noun\_phrase} \rangle$ and $\langle \text{predicate} \rangle$. If we do so by

$$\langle \text{noun\_phrase} \rangle \to \langle \text{article} \rangle \langle \text{noun} \rangle,$$
$$\langle \text{predicate} \rangle \to \langle \text{verb} \rangle,$$

and if we associate the actual words "a" and "the" with $\langle \text{article} \rangle$, "boy" and "dog" with $\langle \text{noun} \rangle$, and "runs" and "walks" with $\langle \text{verb} \rangle$, then the grammar tells us that the sentences "a boy runs" and "the dog walks" are properly formed. If we were to give a complete grammar, then in theory, every proper sentence could be explained this way.

To study languages mathematically, we need a mechanism to describe them. Everyday language is imprecise and ambiguous, so informal descriptions in English are often inadequate. The set notation used in Examples 1.9 and 1.10 is more suitable, but limited. As we proceed we will learn about several language-definition mechanisms that are useful in different circumstances. Here we introduce a common and powerful one, the notion of a *grammar*.

A grammar for the English language tells us whether a particular sentence is well formed or not. A typical rule of English grammar is "a sentence can consist of a noun phrase followed by a predicate". More concisely we write this as

$$\langle \text{sentence} \rangle \rightarrow \langle \text{noun\_phrase} \rangle \langle \text{predicate} \rangle,$$

with the obvious interpretation. This is, of course, not enough to deal with actual sentences. We must now provide definitions for the newly introduced constructs $\langle \text{noun\_phrase} \rangle$ and $\langle \text{predicate} \rangle$. If we do so by

$$\langle \text{noun\_phrase} \rangle \rightarrow \langle \text{article} \rangle \langle \text{noun} \rangle,$$
$$\langle \text{predicate} \rangle \rightarrow \langle \text{verb} \rangle,$$

and if we associate the actual words "a" and "the" with $\langle \text{article} \rangle$, "boy" and "dog" with $\langle \text{noun} \rangle$, and "runs" and "walks" with $\langle \text{verb} \rangle$, then the grammar tells us that the sentences "a boy runs" and "the dog walks" are properly formed. If we were to give a complete grammar, then in theory, every proper sentence could be explained this way.

To study languages mathematically, we need a mechanism to describe them. Everyday language is imprecise and ambiguous, so informal descriptions in English are often inadequate. The set notation used in Examples 1.9 and 1.10 is more suitable, but limited. As we proceed we will learn about several language-definition mechanisms that are useful in different circumstances. Here we introduce a common and powerful one, the notion of a *grammar*.

A grammar for the English language tells us whether a particular sentence is well formed or not. A typical rule of English grammar is "a sentence can consist of a noun phrase followed by a predicate". More concisely we write this as

$$\langle \text{sentence} \rangle \to \langle \text{noun\_phrase} \rangle \langle \text{predicate} \rangle,$$

with the obvious interpretation. This is, of course, not enough to deal with actual sentences. We must now provide definitions for the newly introduced constructs $\langle \text{noun\_phrase} \rangle$ and $\langle \text{predicate} \rangle$. If we do so by

$$\langle \text{noun\_phrase} \rangle \to \langle \text{article} \rangle \langle \text{noun} \rangle,$$
$$\langle \text{predicate} \rangle \to \langle \text{verb} \rangle,$$

and if we associate the actual words "a" and "the" with $\langle \text{article} \rangle$, "boy" and "dog" with $\langle \text{noun} \rangle$, and "runs" and "walks" with $\langle \text{verb} \rangle$, then the grammar tells us that the sentences "a boy runs" and "the dog walks" are properly formed. If we were to give a complete grammar, then in theory, every proper sentence could be explained this way.

To study languages mathematically, we need a mechanism to describe them. Everyday language is imprecise and ambiguous, so informal descriptions in English are often inadequate. The set notation used in Examples 1.9 and 1.10 is more suitable, but limited. As we proceed we will learn about several language-definition mechanisms that are useful in different circumstances. Here we introduce a common and powerful one, the notion of a *grammar*.

A grammar for the English language tells us whether a particular sentence is well formed or not. A typical rule of English grammar is "a sentence can consist of a noun phrase followed by a predicate". More concisely we write this as

$$\langle \text{sentence} \rangle \rightarrow \langle \text{noun\_phrase} \rangle \langle \text{predicate} \rangle,$$

with the obvious interpretation. This is, of course, not enough to deal with actual sentences. We must now provide definitions for the newly introduced constructs $\langle \text{noun\_phrase} \rangle$ and $\langle \text{predicate} \rangle$. If we do so by

$$\langle \text{noun\_phrase} \rangle \rightarrow \langle \text{article} \rangle \langle \text{noun} \rangle,$$
$$\langle \text{predicate} \rangle \rightarrow \langle \text{verb} \rangle,$$

and if we associate the actual words "a" and "the" with $\langle \text{article} \rangle$, "boy" and "dog" with $\langle \text{noun} \rangle$, and "runs" and "walks" with $\langle \text{verb} \rangle$, then the grammar tells us that the sentences "a boy runs" and "the dog walks" are properly formed. If we were to give a complete grammar, then in theory, every proper sentence could be explained this way.

To study languages mathematically, we need a mechanism to describe them. Everyday language is imprecise and ambiguous, so informal descriptions in English are often inadequate. The set notation used in Examples 1.9 and 1.10 is more suitable, but limited. As we proceed we will learn about several language-definition mechanisms that are useful in different circumstances. Here we introduce a common and powerful one, the notion of a *grammar*.

A grammar for the English language tells us whether a particular sentence is well formed or not. A typical rule of English grammar is "a sentence can consist of a noun phrase followed by a predicate". More concisely we write this as

$$\langle \text{sentence} \rangle \to \langle \text{noun\_phrase} \rangle \langle \text{predicate} \rangle,$$

with the obvious interpretation. This is, of course, not enough to deal with actual sentences. We must now provide definitions for the newly introduced constructs $\langle \text{noun\_phrase} \rangle$ and $\langle \text{predicate} \rangle$. If we do so by

$$\langle \text{noun\_phrase} \rangle \to \langle \text{article} \rangle \langle \text{noun} \rangle,$$
$$\langle \text{predicate} \rangle \to \langle \text{verb} \rangle,$$

and if we associate the actual words "a" and "the" with $\langle \text{article} \rangle$, "boy" and "dog" with $\langle \text{noun} \rangle$, and "runs" and "walks" with $\langle \text{verb} \rangle$, then the grammar tells us that the sentences "a boy runs" and "the dog walks" are properly formed. If we were to give a complete grammar, then in theory, every proper sentence could be explained this way.

To study languages mathematically, we need a mechanism to describe them. Everyday language is imprecise and ambiguous, so informal descriptions in English are often inadequate. The set notation used in Examples 1.9 and 1.10 is more suitable, but limited. As we proceed we will learn about several language-definition mechanisms that are useful in different circumstances. Here we introduce a common and powerful one, the notion of a *grammar*.

A grammar for the English language tells us whether a particular sentence is well formed or not. A typical rule of English grammar is "a sentence can consist of a noun phrase followed by a predicate". More concisely we write this as

$$\langle \text{sentence} \rangle \to \langle \text{noun\_phrase} \rangle \langle \text{predicate} \rangle,$$

with the obvious interpretation. This is, of course, not enough to deal with actual sentences. We must now provide definitions for the newly introduced constructs $\langle \text{noun\_phrase} \rangle$ and $\langle \text{predicate} \rangle$. If we do so by

$$\langle \text{noun\_phrase} \rangle \to \langle \text{article} \rangle \langle \text{noun} \rangle,$$
$$\langle \text{predicate} \rangle \to \langle \text{verb} \rangle,$$

and if we associate the actual words "a" and "the" with $\langle \text{article} \rangle$, "boy" and "dog" with $\langle \text{noun} \rangle$, and "runs" and "walks" with $\langle \text{verb} \rangle$, then the grammar tells us that the sentences "a boy runs" and "the dog walks" are properly formed. If we were to give a complete grammar, then in theory, every proper sentence could be explained this way.

To study languages mathematically, we need a mechanism to describe them. Everyday language is imprecise and ambiguous, so informal descriptions in English are often inadequate. The set notation used in Examples 1.9 and 1.10 is more suitable, but limited. As we proceed we will learn about several language-definition mechanisms that are useful in different circumstances. Here we introduce a common and powerful one, the notion of a *grammar*.

A grammar for the English language tells us whether a particular sentence is well formed or not. A typical rule of English grammar is "a sentence can consist of a noun phrase followed by a predicate". More concisely we write this as

$$\langle \text{sentence} \rangle \rightarrow \langle \text{noun\_phrase} \rangle \langle \text{predicate} \rangle,$$

with the obvious interpretation. This is, of course, not enough to deal with actual sentences. We must now provide definitions for the newly introduced constructs $\langle \text{noun\_phrase} \rangle$ and $\langle \text{predicate} \rangle$. If we do so by

$$\langle \text{noun\_phrase} \rangle \rightarrow \langle \text{article} \rangle \langle \text{noun} \rangle,$$
$$\langle \text{predicate} \rangle \rightarrow \langle \text{verb} \rangle,$$

and if we associate the actual words "a" and "the" with $\langle \text{article} \rangle$, "boy" and "dog" with $\langle \text{noun} \rangle$, and "runs" and "walks" with $\langle \text{verb} \rangle$, then the grammar tells us that the sentences "a boy runs" and "the dog walks" are properly formed. If we were to give a complete grammar, then in theory, every proper sentence could be explained this way.

To study languages mathematically, we need a mechanism to describe them. Everyday language is imprecise and ambiguous, so informal descriptions in English are often inadequate. The set notation used in Examples 1.9 and 1.10 is more suitable, but limited. As we proceed we will learn about several language-definition mechanisms that are useful in different circumstances. Here we introduce a common and powerful one, the notion of a *grammar*.

A grammar for the English language tells us whether a particular sentence is well formed or not. A typical rule of English grammar is "a sentence can consist of a noun phrase followed by a predicate". More concisely we write this as

$$\langle \text{sentence} \rangle \to \langle \text{noun\_phrase} \rangle \langle \text{predicate} \rangle,$$

with the obvious interpretation. This is, of course, not enough to deal with actual sentences. We must now provide definitions for the newly introduced constructs $\langle \text{noun\_phrase} \rangle$ and $\langle \text{predicate} \rangle$. If we do so by

$$\langle \text{noun\_phrase} \rangle \to \langle \text{article} \rangle \langle \text{noun} \rangle,$$
$$\langle \text{predicate} \rangle \to \langle \text{verb} \rangle,$$

and if we associate the actual words "a" and "the" with $\langle \text{article} \rangle$, "boy" and "dog" with $\langle \text{noun} \rangle$, and "runs" and "walks" with $\langle \text{verb} \rangle$, then the grammar tells us that the sentences "a boy runs" and "the dog walks" are properly formed. If we were to give a complete grammar, then in theory, every proper sentence could be explained this way.

To study languages mathematically, we need a mechanism to describe them. Everyday language is imprecise and ambiguous, so informal descriptions in English are often inadequate. The set notation used in Examples 1.9 and 1.10 is more suitable, but limited. As we proceed we will learn about several language-definition mechanisms that are useful in different circumstances. Here we introduce a common and powerful one, the notion of a *grammar*.

A grammar for the English language tells us whether a particular sentence is well formed or not. A typical rule of English grammar is "a sentence can consist of a noun phrase followed by a predicate". More concisely we write this as

$$\langle \text{sentence} \rangle \rightarrow \langle \text{noun\_phrase} \rangle \langle \text{predicate} \rangle,$$

with the obvious interpretation. This is, of course, not enough to deal with actual sentences. We must now provide definitions for the newly introduced constructs $\langle \text{noun\_phrase} \rangle$ and $\langle \text{predicate} \rangle$. If we do so by

$$\langle \text{noun\_phrase} \rangle \rightarrow \langle \text{article} \rangle \langle \text{noun} \rangle,$$
$$\langle \text{predicate} \rangle \rightarrow \langle \text{verb} \rangle,$$

and if we associate the actual words "a" and "the" with $\langle \text{article} \rangle$, "boy" and "dog" with $\langle \text{noun} \rangle$, and "runs" and "walks" with $\langle \text{verb} \rangle$, then the grammar tells us that the sentences "a boy runs" and "the dog walks" are properly formed. If we were to give a complete grammar, then in theory, every proper sentence could be explained this way.

To study languages mathematically, we need a mechanism to describe them. Everyday language is imprecise and ambiguous, so informal descriptions in English are often inadequate. The set notation used in Examples 1.9 and 1.10 is more suitable, but limited. As we proceed we will learn about several language-definition mechanisms that are useful in different circumstances. Here we introduce a common and powerful one, the notion of a *grammar*.

A grammar for the English language tells us whether a particular sentence is well formed or not. A typical rule of English grammar is "a sentence can consist of a noun phrase followed by a predicate". More concisely we write this as

$$\langle \text{sentence} \rangle \rightarrow \langle \text{noun\_phrase} \rangle \langle \text{predicate} \rangle,$$

with the obvious interpretation. This is, of course, not enough to deal with actual sentences. We must now provide definitions for the newly introduced constructs $\langle \text{noun\_phrase} \rangle$ and $\langle \text{predicate} \rangle$. If we do so by

$$\langle \text{noun\_phrase} \rangle \rightarrow \langle \text{article} \rangle \langle \text{noun} \rangle,$$
$$\langle \text{predicate} \rangle \rightarrow \langle \text{verb} \rangle,$$

and if we associate the actual words "a" and "the" with $\langle \text{article} \rangle$, "boy" and "dog" with $\langle \text{noun} \rangle$, and "runs" and "walks" with $\langle \text{verb} \rangle$, then the grammar tells us that the sentences "a boy runs" and "the dog walks" are properly formed. If we were to give a complete grammar, then in theory, every proper sentence could be explained this way.

This example illustrates the definition of a general concept in terms of simple ones. We start with the top-level concept, here $\langle \text{sentence} \rangle$, and successively reduce it to the irreducible building blocks of the language. The generalization of these ideas leads us to formal grammars.

Definition 1.1

A *grammar* G is defined as a quadruple

$$G = (V, T, S, P),$$

where

- V is a finite set of objects called *variables*,
- T is a finite set of objects called *terminal symbols*,
- $S \in V$ is a special symbol called *the start variable*,
- P is a finite set of *productions*.

It will be assumed without further mention that the sets V and T are nonempty and disjoint.

This example illustrates the definition of a general concept in terms of simple ones. We start with the top-level concept, here $\langle \text{sentence} \rangle$, and successively reduce it to the irreducible building blocks of the language. The generalization of these ideas leads us to formal grammars.

Definition 1.1

A *grammar* G is defined as a quadruple

$$G = (V, T, S, P),$$

where

- V is a finite set of objects called *variables*,
- T is a finite set of objects called *terminal symbols*,
- $S \in V$ is a special symbol called the *start variable*,
- P is a finite set of *productions*.

It will be assumed without further mention that the sets V and T are nonempty and disjoint.

This example illustrates the definition of a general concept in terms of simple ones. We start with the top-level concept, here $\langle \text{sentence} \rangle$, and successively reduce it to the irreducible building blocks of the language. The generalization of these ideas leads us to formal grammars.

Definition 1.1

A *grammar* G is defined as a quadruple

$$G = (V, T, S, P),$$

where

- V is a finite set of objects called *variables*,
- T is a finite set of objects called *terminal symbols*,
- $S \in V$ is a special symbol called the *start variable*,
- P is a finite set of *productions*.

It will be assumed without further mention that the sets V and T are nonempty and disjoint.

This example illustrates the definition of a general concept in terms of simple ones. We start with the top-level concept, here $\langle \text{sentence} \rangle$, and successively reduce it to the irreducible building blocks of the language. The generalization of these ideas leads us to formal grammars.

Definition 1.1

A *grammar* G is defined as a quadruple

$$G = (V, T, S, P),$$

where

- V is a finite set of objects called *variables*,
- T is a finite set of objects called *terminal symbols*,
- $S \in V$ is a special symbol called the *start* variable,
- P is a finite set of *productions*.

It will be assumed without further mention that the sets V and T are nonempty and disjoint.

This example illustrates the definition of a general concept in terms of simple ones. We start with the top-level concept, here $\langle \text{sentence} \rangle$, and successively reduce it to the irreducible building blocks of the language. The generalization of these ideas leads us to formal grammars.

Definition 1.1

A *grammar* G is defined as a quadruple

$$G = (V, T, S, P),$$

where

- V is a finite set of objects called *variables*,
- T is a finite set of objects called *terminal symbols*,
- $S \in V$ is a special symbol called the *start variable*,
- P is a finite set of *productions*.

It will be assumed without further mention that the sets V and T are nonempty and disjoint.

This example illustrates the definition of a general concept in terms of simple ones. We start with the top-level concept, here $\langle \mathrm{sentence} \rangle$, and successively reduce it to the irreducible building blocks of the language. The generalization of these ideas leads us to formal grammars.

Definition 1.1

A *grammar* G is defined as a quadruple

$$G = (V, T, S, P),$$

where

- V is a finite set of objects called *variables*,
- T is a finite set of objects called *terminal symbols*,
- $S \in V$ is a special symbol called the *start variable*,
- P is a finite set of *productions*.

It will be assumed without further mention that the sets V and T are nonempty and disjoint.

This example illustrates the definition of a general concept in terms of simple ones. We start with the top-level concept, here $\langle \text{sentence} \rangle$, and successively reduce it to the irreducible building blocks of the language. The generalization of these ideas leads us to formal grammars.

Definition 1.1

A *grammar* G is defined as a quadruple

$$G = (V, T, S, P),$$

where

- V is a finite set of objects called *variables*;
- T is a finite set of objects called *terminal symbols*;
- $S \in V$ is a special symbol called the *start variable*;
- P is a finite set of *productions*.

It will be assumed without further mention that the sets V and T are nonempty and disjoint.

This example illustrates the definition of a general concept in terms of simple ones. We start with the top-level concept, here $\langle \text{sentence} \rangle$, and successively reduce it to the irreducible building blocks of the language. The generalization of these ideas leads us to formal grammars.

Definition 1.1

A *grammar* G is defined as a quadruple

$$G = (V, T, S, P),$$

where

- V is a finite set of objects called *variables*;
- T is a finite set of objects called *terminal symbols*;
- $S \in V$ is a special symbol called the *start variable*;
- P is a finite set of *productions*.

It will be assumed without further mention that the sets V and T are nonempty and disjoint.

This example illustrates the definition of a general concept in terms of simple ones. We start with the top-level concept, here $\langle\text{sentence}\rangle$, and successively reduce it to the irreducible building blocks of the language. The generalization of these ideas leads us to formal grammars.

Definition 1.1

A *grammar* G is defined as a quadruple

$$G = (V, T, S, P),$$

where

- V is a finite set of objects called *variables*;
- T is a finite set of objects called *terminal symbols*;
- $S \in V$ is a special symbol called the *start variable*;
- P is a finite set of *productions*.

It will be assumed without further mention that the sets V and T are nonempty and disjoint.

This example illustrates the definition of a general concept in terms of simple ones. We start with the top-level concept, here $\langle \text{sentence} \rangle$, and successively reduce it to the irreducible building blocks of the language. The generalization of these ideas leads us to formal grammars.

Definition 1.1

A *grammar* G is defined as a quadruple

$$G = (V, T, S, P),$$

where

- V is a finite set of objects called *variables*;
- T is a finite set of objects called *terminal symbols*;
- $S \in V$ is a special symbol called the *start variable*;
- P is a finite set of *productions*.

It will be assumed without further mention that the sets V and T are nonempty and disjoint.

This example illustrates the definition of a general concept in terms of simple ones. We start with the top-level concept, here $\langle \text{sentence} \rangle$, and successively reduce it to the irreducible building blocks of the language. The generalization of these ideas leads us to formal grammars.

Definition 1.1

A *grammar* G is defined as a quadruple

$$G = (V, T, S, P),$$

where

- V is a finite set of objects called *variables*;
- T is a finite set of objects called *terminal symbols*;
- $S \in V$ is a special symbol called the *start variable*;
- P is a finite set of *productions*.

It will be assumed without further mention that the sets V and T are nonempty and disjoint.

This example illustrates the definition of a general concept in terms of simple ones. We start with the top-level concept, here $\langle\text{sentence}\rangle$, and successively reduce it to the irreducible building blocks of the language. The generalization of these ideas leads us to formal grammars.

Definition 1.1

A *grammar* G is defined as a quadruple

$$G = (V, T, S, P),$$

where

- V is a finite set of objects called *variables*;
- T is a finite set of objects called *terminal symbols*;
- $S \in V$ is a special symbol called the *start variable*;
- P is a finite set of *productions*.

It will be assumed without further mention that the sets V and T are nonempty and disjoint.

This example illustrates the definition of a general concept in terms of simple ones. We start with the top-level concept, here $\langle \text{sentence} \rangle$, and successively reduce it to the irreducible building blocks of the language. The generalization of these ideas leads us to formal grammars.

Definition 1.1

A *grammar* G is defined as a quadruple

$$G = (V, T, S, P),$$

where

- V is a finite set of objects called *variables*;
- T is a finite set of objects called *terminal symbols*;
- $S \in V$ is a special symbol called the *start variable*;
- P is a finite set of *productions*.

It will be assumed without further mention that the sets V and T are nonempty and disjoint.

This example illustrates the definition of a general concept in terms of simple ones. We start with the top-level concept, here $\langle \text{sentence} \rangle$, and successively reduce it to the irreducible building blocks of the language. The generalization of these ideas leads us to formal grammars.

Definition 1.1

A *grammar* G is defined as a quadruple

$$G = (V, T, S, P),$$

where

- V is a finite set of objects called *variables*;
- T is a finite set of objects called *terminal symbols*;
- $S \in V$ is a special symbol called the *start variable*;
- P is a finite set of *productions*.

It will be assumed without further mention that the sets V and T are nonempty and disjoint.

This example illustrates the definition of a general concept in terms of simple ones. We start with the top-level concept, here $\langle \text{sentence} \rangle$, and successively reduce it to the irreducible building blocks of the language. The generalization of these ideas leads us to formal grammars.

Definition 1.1

A *grammar* G is defined as a quadruple

$$G = (V, T, S, P),$$

where

- V is a finite set of objects called *variables*;
- T is a finite set of objects called *terminal symbols*;
- $S \in V$ is a special symbol called the *start variable*;
- P is a finite set of *productions*.

It will be assumed without further mention that the sets V and T are nonempty and disjoint.

This example illustrates the definition of a general concept in terms of simple ones. We start with the top-level concept, here $\langle \text{sentence} \rangle$, and successively reduce it to the irreducible building blocks of the language. The generalization of these ideas leads us to formal grammars.

Definition 1.1

A *grammar* G is defined as a quadruple

$$G = (V, T, S, P),$$

where

- V is a finite set of objects called *variables*;
- T is a finite set of objects called *terminal symbols*;
- $S \in V$ is a special symbol called the *start variable*;
- P is a finite set of *productions*.

It will be assumed without further mention that the sets V and T are nonempty and disjoint.

The production rules are the heart of a grammar; they specify how the grammar transforms one string into another, and through this they define a language associated with the grammar. In our discussion we will assume that all production rules are of the form

$$x \to y,$$

where x is an element of $(V \cup T)^+$ and y is in $(V \cup T)^*$. The productions are applied in the following manner: Given a string w of the form

$$w = uxv,$$

we say the production $x \to y$ is applicable to this string, and we may use it to replace x with y, thereby obtaining a new string

$$z = uyv.$$

This is written as

$$w \Rightarrow z.$$

We say that w *derives* z or that z is *derived* from w. Successive strings are derived by applying the productions of the grammar in arbitrary order. A production can be used whenever it is applicable, and it can be applied as often as desired. If

$$w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n,$$

we say that w_1 *derives* w_n and write

$$w_1 \stackrel{*}{\Rightarrow} w_n.$$

The star $*$ indicates that an unspecified number of steps (including zero) can be taken to derive w_n from w_1.

The production rules are the heart of a grammar; they specify how the grammar transforms one string into another, and through this they define a language associated with the grammar. In our discussion we will assume that all production rules are of the form

$$x \to y,$$

where x is an element of $(V \cup T)^+$ and y is in $(V \cup T)^*$. The productions are applied in the following manner: Given a string w of the form

$$w = uxv,$$

we say the production $x \to y$ is applicable to this string, and we may use it to replace x with y, thereby obtaining a new string

$$z = uyv.$$

This is written as

$$w \Rightarrow z.$$

We say that w *derives* z or that z is *derived* from w. Successive strings are derived by applying the productions of the grammar in arbitrary order. A production can be used whenever it is applicable, and it can be applied as often as desired. If

$$w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n,$$

we say that w_1 *derives* w_n and write

$$w_1 \stackrel{*}{\Rightarrow} w_n.$$

The star $*$ indicates that an unspecified number of steps (including zero) can be taken to derive w_n from w_1.

The production rules are the heart of a grammar; they specify how the grammar transforms one string into another, and through this they define a language associated with the grammar. In our discussion we will assume that all production rules are of the form

$$x \to y,$$

where x is an element of $(V \cup T)^+$ and y is in $(V \cup T)^*$. The productions are applied in the following manner: Given a string w of the form

$$w = uxv,$$

we say the production $x \to y$ is applicable to this string, and we may use it to replace x with y, thereby obtaining a new string

$$z = uyv.$$

This is written as

$$w \Rightarrow z.$$

We say that w *derives* z or that z is *derived* from w. Successive strings are derived by applying the productions of the grammar in arbitrary order. A production can be used whenever it is applicable, and it can be applied as often as desired. If

$$w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n,$$

we say that w_1 *derives* w_n and write

$$w_1 \stackrel{*}{\Rightarrow} w_n.$$

The star $*$ indicates that an unspecified number of steps (including zero) can be taken to derive w_n from w_1.

The production rules are the heart of a grammar; they specify how the grammar transforms one string into another, and through this they define a language associated with the grammar. In our discussion we will assume that all production rules are of the form

$$x \to y,$$

where x is an element of $(V \cup T)^+$ and y is in $(V \cup T)^*$. The productions are applied in the following manner: Given a string w of the form

$$w = uxv,$$

we say the production $x \to y$ is applicable to this string, and we may use it to replace x with y, thereby obtaining a new string

$$z = uyv.$$

This is written as

$$w \Rightarrow z.$$

We say that w *derives* z or that z is *derived* from w. Successive strings are derived by applying the productions of the grammar in arbitrary order. A production can be used whenever it is applicable, and it can be applied as often as desired. If

$$w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n,$$

we say that w_1 *derives* w_n and write

$$w_1 \stackrel{*}{\Rightarrow} w_n.$$

The star $*$ indicates that an unspecified number of steps (including zero) can be taken to derive w_n from w_1.

The production rules are the heart of a grammar; they specify how the grammar transforms one string into another, and through this they define a language associated with the grammar. In our discussion we will assume that all production rules are of the form

$$x \to y,$$

where x is an element of $(V \cup T)^+$ and y is in $(V \cup T)^*$. The productions are applied in the following manner: Given a string w of the form

$$w = uxv,$$

we say the production $x \to y$ is applicable to this string, and we may use it to replace x with y, thereby obtaining a new string

$$z = uyv.$$

This is written as

$$w \Rightarrow z.$$

We say that w *derives* z or that z is *derived* from w. Successive strings are derived by applying the productions of the grammar in arbitrary order. A production can be used whenever it is applicable, and it can be applied as often as desired. If

$$w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n,$$

we say that w_1 *derives* w_n and write

$$w_1 \stackrel{*}{\Rightarrow} w_n.$$

The star $*$ indicates that an unspecified number of steps (including zero) can be taken to derive w_n from w_1.

The production rules are the heart of a grammar; they specify how the grammar transforms one string into another, and through this they define a language associated with the grammar. In our discussion we will assume that all production rules are of the form

$$x \to y,$$

where x is an element of $(V \cup T)^+$ and y is in $(V \cup T)^*$. The productions are applied in the following manner: Given a string w of the form

$$w = uxv,$$

we say the production $x \to y$ is applicable to this string, and we may use it to replace x with y, thereby obtaining a new string

$$z = uyv.$$

This is written as

$$w \Rightarrow z.$$

We say that w *derives* z or that z is *derived* from w. Successive strings are derived by applying the productions of the grammar in arbitrary order. A production can be used whenever it is applicable, and it can be applied as often as desired. If

$$w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n,$$

we say that w_1 *derives* w_n and write

$$w_1 \overset{*}{\Rightarrow} w_n.$$

The star $*$ indicates that an unspecified number of steps (including zero) can be taken to derive w_n from w_1.

The production rules are the heart of a grammar; they specify how the grammar transforms one string into another, and through this they define a language associated with the grammar. In our discussion we will assume that all production rules are of the form

$$x \to y,$$

where x is an element of $(V \cup T)^+$ and y is in $(V \cup T)^*$. The productions are applied in the following manner: Given a string w of the form

$$w = uxv,$$

we say the production $x \to y$ is applicable to this string, and we may use it to replace x with y, thereby obtaining a new string

$$z = uyv.$$

This is written as

$$w \Rightarrow z.$$

We say that w *derives* z or that z is *derived* from w. Successive strings are derived by applying the productions of the grammar in arbitrary order. A production can be used whenever it is applicable, and it can be applied as often as desired. If

$$w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n,$$

we say that w_1 *derives* w_n and write

$$w_1 \overset{*}{\Rightarrow} w_n.$$

The star $*$ indicates that an unspecified number of steps (including zero) can be taken to derive w_n from w_1.

The production rules are the heart of a grammar; they specify how the grammar transforms one string into another, and through this they define a language associated with the grammar. In our discussion we will assume that all production rules are of the form

$$x \to y,$$

where x is an element of $(V \cup T)^+$ and y is in $(V \cup T)^*$. The productions are applied in the following manner: Given a string w of the form

$$w = uxv,$$

we say the production $x \to y$ is applicable to this string, and we may use it to replace x with y, thereby obtaining a new string

$$z = uyv.$$

This is written as

$$w \Rightarrow z.$$

We say that w *derives* z or that z is *derived* from w. Successive strings are derived by applying the productions of the grammar in arbitrary order. A production can be used whenever it is applicable, and it can be applied as often as desired. If

$$w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n,$$

we say that w_1 *derives* w_n and write

$$w_1 \stackrel{*}{\Rightarrow} w_n.$$

The star $*$ indicates that an unspecified number of steps (including zero) can be taken to derive w_n from w_1.

The production rules are the heart of a grammar; they specify how the grammar transforms one string into another, and through this they define a language associated with the grammar. In our discussion we will assume that all production rules are of the form

$$x \to y,$$

where x is an element of $(V \cup T)^+$ and y is in $(V \cup T)^*$. The productions are applied in the following manner: Given a string w of the form

$$w = uxv,$$

we say the production $x \to y$ is applicable to this string, and we may use it to replace x with y, thereby obtaining a new string

$$z = uyv.$$

This is written as

$$w \Rightarrow z.$$

We say that w *derives* z or that z is *derived* from w. Successive strings are derived by applying the productions of the grammar in arbitrary order. A production can be used whenever it is applicable, and it can be applied as often as desired. If

$$w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n,$$

we say that w_1 *derives* w_n and write

$$w_1 \stackrel{*}{\Rightarrow} w_n.$$

The star $*$ indicates that an unspecified number of steps (including zero) can be taken to derive w_n from w_1.

The production rules are the heart of a grammar; they specify how the grammar transforms one string into another, and through this they define a language associated with the grammar. In our discussion we will assume that all production rules are of the form

$$x \to y,$$

where x is an element of $(V \cup T)^+$ and y is in $(V \cup T)^*$. The productions are applied in the following manner: Given a string w of the form

$$w = uxv,$$

we say the production $x \to y$ is applicable to this string, and we may use it to replace x with y, thereby obtaining a new string

$$z = uyv.$$

This is written as

$$w \Rightarrow z.$$

We say that w *derives* z or that z is *derived* from w. Successive strings are derived by applying the productions of the grammar in arbitrary order. A production can be used whenever it is applicable, and it can be applied as often as desired. If

$$w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n,$$

we say that w_1 *derives* w_n and write

$$w_1 \overset{*}{\Rightarrow} w_n.$$

The star $*$ indicates that an unspecified number of steps (including zero) can be taken to derive w_n from w_1.

The production rules are the heart of a grammar; they specify how the grammar transforms one string into another, and through this they define a language associated with the grammar. In our discussion we will assume that all production rules are of the form

$$x \to y,$$

where x is an element of $(V \cup T)^+$ and y is in $(V \cup T)^*$. The productions are applied in the following manner: Given a string w of the form

$$w = uxv,$$

we say the production $x \to y$ is applicable to this string, and we may use it to replace x with y, thereby obtaining a new string

$$z = uyv.$$

This is written as

$$w \Rightarrow z.$$

We say that w *derives* z or that z is *derived* from w. Successive strings are derived by applying the productions of the grammar in arbitrary order. A production can be used whenever it is applicable, and it can be applied as often as desired. If

$$w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n,$$

we say that w_1 *derives* w_n and write

$$w_1 \overset{*}{\Rightarrow} w_n.$$

The star $*$ indicates that an unspecified number of steps (including zero) can be taken to derive w_n from w_1.

The production rules are the heart of a grammar; they specify how the grammar transforms one string into another, and through this they define a language associated with the grammar. In our discussion we will assume that all production rules are of the form

$$x \to y,$$

where x is an element of $(V \cup T)^+$ and y is in $(V \cup T)^*$. The productions are applied in the following manner: Given a string w of the form

$$w = uxv,$$

we say the production $x \to y$ is applicable to this string, and we may use it to replace x with y, thereby obtaining a new string

$$z = uyv.$$

This is written as

$$w \Rightarrow z.$$

We say that w *derives* z or that z is *derived* from w. Successive strings are derived by applying the productions of the grammar in arbitrary order. A production can be used whenever it is applicable, and it can be applied as often as desired. If

$$w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n,$$

we say that w_1 *derives* w_n and write

$$w_1 \stackrel{*}{\Rightarrow} w_n.$$

The star $*$ indicates that an unspecified number of steps (including zero) can be taken to derive w_n from w_1.

The production rules are the heart of a grammar; they specify how the grammar transforms one string into another, and through this they define a language associated with the grammar. In our discussion we will assume that all production rules are of the form

$$x \to y,$$

where x is an element of $(V \cup T)^+$ and y is in $(V \cup T)^*$. The productions are applied in the following manner: Given a string w of the form

$$w = uxv,$$

we say the production $x \to y$ is applicable to this string, and we may use it to replace x with y, thereby obtaining a new string

$$z = uyv.$$

This is written as

$$w \Rightarrow z.$$

We say that w *derives* z or that z is *derived* from w. Successive strings are derived by applying the productions of the grammar in arbitrary order. A production can be used whenever it is applicable, and it can be applied as often as desired. If

$$w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n,$$

we say that w_1 *derives* w_n and write

$$w_1 \overset{*}{\Rightarrow} w_n.$$

The star $*$ indicates that an unspecified number of steps (including zero) can be taken to derive w_n from w_1.

The production rules are the heart of a grammar; they specify how the grammar transforms one string into another, and through this they define a language associated with the grammar. In our discussion we will assume that all production rules are of the form

$$x \to y,$$

where x is an element of $(V \cup T)^+$ and y is in $(V \cup T)^*$. The productions are applied in the following manner: Given a string w of the form

$$w = uxv,$$

we say the production $x \to y$ is applicable to this string, and we may use it to replace x with y, thereby obtaining a new string

$$z = uyv.$$

This is written as

$$w \Rightarrow z.$$

We say that w *derives* z or that z is *derived* from w. Successive strings are derived by applying the productions of the grammar in arbitrary order. A production can be used whenever it is applicable, and it can be applied as often as desired. If

$$w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n,$$

we say that w_1 *derives* w_n and write

$$w_1 \overset{*}{\Rightarrow} w_n.$$

The star $*$ indicates that an unspecified number of steps (including zero) can be taken to derive w_n from w_1.

The production rules are the heart of a grammar; they specify how the grammar transforms one string into another, and through this they define a language associated with the grammar. In our discussion we will assume that all production rules are of the form

$$x \to y,$$

where x is an element of $(V \cup T)^+$ and y is in $(V \cup T)^*$. The productions are applied in the following manner: Given a string w of the form

$$w = uxv,$$

we say the production $x \to y$ is applicable to this string, and we may use it to replace x with y, thereby obtaining a new string

$$z = uyv.$$

This is written as

$$w \Rightarrow z.$$

We say that w *derives* z or that z is *derived* from w. Successive strings are derived by applying the productions of the grammar in arbitrary order. A production can be used whenever it is applicable, and it can be applied as often as desired. If

$$w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n,$$

we say that w_1 *derives* w_n and write

$$w_1 \stackrel{*}{\Rightarrow} w_n.$$

The star $*$ indicates that an unspecified number of steps (including zero) can be taken to derive w_n from w_1.

The production rules are the heart of a grammar; they specify how the grammar transforms one string into another, and through this they define a language associated with the grammar. In our discussion we will assume that all production rules are of the form

$$x \to y,$$

where x is an element of $(V \cup T)^+$ and y is in $(V \cup T)^*$. The productions are applied in the following manner: Given a string w of the form

$$w = uxv,$$

we say the production $x \to y$ is applicable to this string, and we may use it to replace x with y, thereby obtaining a new string

$$z = uyv.$$

This is written as

$$w \Rightarrow z.$$

We say that w *derives* z or that z is *derived* from w. Successive strings are derived by applying the productions of the grammar in arbitrary order. A production can be used whenever it is applicable, and it can be applied as often as desired. If

$$w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n,$$

we say that w_1 *derives* w_n and write

$$w_1 \stackrel{*}{\Rightarrow} w_n.$$

The star $*$ indicates that an unspecified number of steps (including zero) can be taken to derive w_n from w_1.

The production rules are the heart of a grammar; they specify how the grammar transforms one string into another, and through this they define a language associated with the grammar. In our discussion we will assume that all production rules are of the form

$$x \to y,$$

where x is an element of $(V \cup T)^+$ and y is in $(V \cup T)^*$. The productions are applied in the following manner: Given a string w of the form

$$w = uxv,$$

we say the production $x \to y$ is applicable to this string, and we may use it to replace x with y, thereby obtaining a new string

$$z = uyv.$$

This is written as

$$w \Rightarrow z.$$

We say that w *derives* z or that z is *derived* from w. Successive strings are derived by applying the productions of the grammar in arbitrary order. A production can be used whenever it is applicable, and it can be applied as often as desired. If

$$w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n,$$

we say that w_1 *derives* w_n and write

$$w_1 \stackrel{*}{\Rightarrow} w_n.$$

The star $*$ indicates that an unspecified number of steps (including zero) can be taken to derive w_n from w_1.

The production rules are the heart of a grammar; they specify how the grammar transforms one string into another, and through this they define a language associated with the grammar. In our discussion we will assume that all production rules are of the form

$$x \to y,$$

where x is an element of $(V \cup T)^+$ and y is in $(V \cup T)^*$. The productions are applied in the following manner: Given a string w of the form

$$w = uxv,$$

we say the production $x \to y$ is applicable to this string, and we may use it to replace x with y, thereby obtaining a new string

$$z = uyv.$$

This is written as

$$w \Rightarrow z.$$

We say that w *derives* z or that z is *derived* from w. Successive strings are derived by applying the productions of the grammar in arbitrary order. A production can be used whenever it is applicable, and it can be applied as often as desired. If

$$w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n,$$

we say that w_1 *derives* w_n and write

$$w_1 \overset{*}{\Rightarrow} w_n.$$

The star $*$ indicates that an unspecified number of steps (including zero) can be taken to derive w_n from w_1.

The production rules are the heart of a grammar; they specify how the grammar transforms one string into another, and through this they define a language associated with the grammar. In our discussion we will assume that all production rules are of the form

$$x \to y,$$

where x is an element of $(V \cup T)^+$ and y is in $(V \cup T)^*$. The productions are applied in the following manner: Given a string w of the form

$$w = uxv,$$

we say the production $x \to y$ is applicable to this string, and we may use it to replace x with y, thereby obtaining a new string

$$z = uyv.$$

This is written as

$$w \Rightarrow z.$$

We say that w *derives* z or that z is *derived* from w. Successive strings are derived by applying the productions of the grammar in arbitrary order. A production can be used whenever it is applicable, and it can be applied as often as desired. If

$$w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n,$$

we say that w_1 *derives* w_n and write

$$w_1 \stackrel{*}{\Rightarrow} w_n.$$

The star $*$ indicates that an unspecified number of steps (including zero) can be taken to derive w_n from w_1.

The production rules are the heart of a grammar; they specify how the grammar transforms one string into another, and through this they define a language associated with the grammar. In our discussion we will assume that all production rules are of the form

$$x \to y,$$

where x is an element of $(V \cup T)^+$ and y is in $(V \cup T)^*$. The productions are applied in the following manner: Given a string w of the form

$$w = uxv,$$

we say the production $x \to y$ is applicable to this string, and we may use it to replace x with y, thereby obtaining a new string

$$z = uyv.$$

This is written as

$$w \Rightarrow z.$$

We say that w *derives* z or that z is *derived* from w. Successive strings are derived by applying the productions of the grammar in arbitrary order. A production can be used whenever it is applicable, and it can be applied as often as desired. If

$$w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n,$$

we say that w_1 *derives* w_n and write

$$w_1 \overset{*}{\Rightarrow} w_n.$$

The star $*$ indicates that an unspecified number of steps (including zero) can be taken to derive w_n from w_1.

The production rules are the heart of a grammar; they specify how the grammar transforms one string into another, and through this they define a language associated with the grammar. In our discussion we will assume that all production rules are of the form

$$x \to y,$$

where x is an element of $(V \cup T)^+$ and y is in $(V \cup T)^*$. The productions are applied in the following manner: Given a string w of the form

$$w = uxv,$$

we say the production $x \to y$ is applicable to this string, and we may use it to replace x with y, thereby obtaining a new string

$$z = uyv.$$

This is written as

$$w \Rightarrow z.$$

We say that w *derives* z or that z is *derived* from w. Successive strings are derived by applying the productions of the grammar in arbitrary order. A production can be used whenever it is applicable, and it can be applied as often as desired. If

$$w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n,$$

we say that w_1 *derives* w_n and write

$$w_1 \stackrel{*}{\Rightarrow} w_n.$$

The star $*$ indicates that an unspecified number of steps (including zero) can be taken to derive w_n from w_1.

The production rules are the heart of a grammar; they specify how the grammar transforms one string into another, and through this they define a language associated with the grammar. In our discussion we will assume that all production rules are of the form

$$x \to y,$$

where x is an element of $(V \cup T)^+$ and y is in $(V \cup T)^*$. The productions are applied in the following manner: Given a string w of the form

$$w = uxv,$$

we say the production $x \to y$ is applicable to this string, and we may use it to replace x with y, thereby obtaining a new string

$$z = uyv.$$

This is written as

$$w \Rightarrow z.$$

We say that w *derives* z or that z is *derived* from w. Successive strings are derived by applying the productions of the grammar in arbitrary order. A production can be used whenever it is applicable, and it can be applied as often as desired. If

$$w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n,$$

we say that w_1 *derives* w_n and write

$$w_1 \stackrel{*}{\Rightarrow} w_n.$$

The star $*$ indicates that an unspecified number of steps (including zero) can be taken to derive w_n from w_1.

The production rules are the heart of a grammar; they specify how the grammar transforms one string into another, and through this they define a language associated with the grammar. In our discussion we will assume that all production rules are of the form

$$x \to y,$$

where x is an element of $(V \cup T)^+$ and y is in $(V \cup T)^*$. The productions are applied in the following manner: Given a string w of the form

$$w = uxv,$$

we say the production $x \to y$ is applicable to this string, and we may use it to replace x with y, thereby obtaining a new string

$$z = uyv.$$

This is written as

$$w \Rightarrow z.$$

We say that w *derives* z or that z is *derived* from w. Successive strings are derived by applying the productions of the grammar in arbitrary order. A production can be used whenever it is applicable, and it can be applied as often as desired. If

$$w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n,$$

we say that w_1 *derives* w_n and write

$$w_1 \stackrel{*}{\Rightarrow} w_n.$$

The star $*$ indicates that an unspecified number of steps (including zero) can be taken to derive w_n from w_1.

The production rules are the heart of a grammar; they specify how the grammar transforms one string into another, and through this they define a language associated with the grammar. In our discussion we will assume that all production rules are of the form

$$x \to y,$$

where x is an element of $(V \cup T)^+$ and y is in $(V \cup T)^*$. The productions are applied in the following manner: Given a string w of the form

$$w = uxv,$$

we say the production $x \to y$ is applicable to this string, and we may use it to replace x with y, thereby obtaining a new string

$$z = uyv.$$

This is written as

$$w \Rightarrow z.$$

We say that w *derives* z or that z is *derived* from w. Successive strings are derived by applying the productions of the grammar in arbitrary order. A production can be used whenever it is applicable, and it can be applied as often as desired. If

$$w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n,$$

we say that w_1 *derives* w_n and write

$$w_1 \overset{*}{\Rightarrow} w_n.$$

The star $*$ indicates that an unspecified number of steps (including zero) can be taken to derive w_n from w_1.

The production rules are the heart of a grammar; they specify how the grammar transforms one string into another, and through this they define a language associated with the grammar. In our discussion we will assume that all production rules are of the form

$$x \to y,$$

where x is an element of $(V \cup T)^+$ and y is in $(V \cup T)^*$. The productions are applied in the following manner: Given a string w of the form

$$w = uxv,$$

we say the production $x \to y$ is applicable to this string, and we may use it to replace x with y, thereby obtaining a new string

$$z = uyv.$$

This is written as

$$w \Rightarrow z.$$

We say that w *derives* z or that z is *derived* from w. Successive strings are derived by applying the productions of the grammar in arbitrary order. A production can be used whenever it is applicable, and it can be applied as often as desired. If

$$w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n,$$

we say that w_1 *derives* w_n and write

$$w_1 \overset{*}{\Rightarrow} w_n.$$

The star $*$ indicates that an unspecified number of steps (including zero) can be taken to derive w_n from w_1.

By applying the production rules in a different order, a given grammar can normally generate many strings. The set of all such terminal strings is the language defined or generated by the grammar.

Definition 1.2

Let $G = (V, T, S, P)$ be a grammar. Then the set

$$L(G) = \left\{ w \in T^* : S \stackrel{*}{\Rightarrow} w \right\}$$

is the *language generated* by G.

If $w \in L(G)$ then the sequence

$$S \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n \Rightarrow w$$

is a *derivation* of the sentence w. The strings $S, w_1, w_2, \ldots, w_n$, which contain variables as well as terminals, are called *sentential forms* of the derivation.

By applying the production rules in a different order, a given grammar can normally generate many strings. The set of all such terminal strings is the language defined or generated by the grammar.

Definition 1.2

Let $G = (V, T, S, P)$ be a grammar. Then the set

$$L(G) = \left\{ w \in T^* : S \stackrel{*}{\Rightarrow} w \right\}$$

is the *language generated* by G.

If $w \in L(G)$ then the sequence

$$S \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n \Rightarrow w$$

is a *derivation* of the sentence w. The strings $S, w_1, w_2, \ldots, w_n$, which contain variables as well as terminals, are called *sentential forms* of the derivation.

By applying the production rules in a different order, a given grammar can normally generate many strings. The set of all such terminal strings is the language defined or generated by the grammar.

Definition 1.2

Let $G = (V, T, S, P)$ be a grammar. Then the set

$$L(G) = \left\{ w \in T^* : S \overset{*}{\Rightarrow} w \right\}$$

is the *language generated* by G.

If $w \in L(G)$ then the sequence

$$S \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n \Rightarrow w$$

is a *derivation* of the sentence w. The strings $S, w_1, w_2, \ldots, w_n$, which contain variables as well as terminals, are called *sentential forms* of the derivation.

By applying the production rules in a different order, a given grammar can normally generate many strings. The set of all such terminal strings is the language defined or generated by the grammar.

Definition 1.2

Let $G = (V, T, S, P)$ be a grammar. Then the set

$$L(G) = \left\{ w \in T^* : S \overset{*}{\Rightarrow} w \right\}$$

is the *language generated* by G.

If $w \in L(G)$ then the sequence

$$S \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n \Rightarrow w$$

is a *derivation* of the sentence w. The strings $S, w_1, w_2, \ldots, w_n$, which contain variables as well as terminals, are called *sentential forms* of the derivation.

By applying the production rules in a different order, a given grammar can normally generate many strings. The set of all such terminal strings is the language defined or generated by the grammar.

Definition 1.2

Let $G = (V, T, S, P)$ be a grammar. Then the set

$$L(G) = \left\{ w \in T^* : S \overset{*}{\Rightarrow} w \right\}$$

is the *language generated* by G.

If $w \in L(G)$ then the sequence

$$S \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n \Rightarrow w$$

is a *derivation* of the sentence w. The strings $S, w_1, w_2, \ldots, w_n$, which contain variables as well as terminals, are called *sentential forms* of the derivation.

By applying the production rules in a different order, a given grammar can normally generate many strings. The set of all such terminal strings is the language defined or generated by the grammar.

Definition 1.2

Let $G = (V, T, S, P)$ be a grammar. Then the set

$$L(G) = \left\{ w \in T^* : S \overset{*}{\Rightarrow} w \right\}$$

is the *language generated* by G.

If $w \in L(G)$ then the sequence

$$S \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n \Rightarrow w$$

is a *derivation* of the sentence w. The strings $S, w_1, w_2, \ldots, w_n$, which contain variables as well as terminals, are called *sentential forms* of the derivation.

By applying the production rules in a different order, a given grammar can normally generate many strings. The set of all such terminal strings is the language defined or generated by the grammar.

Definition 1.2

Let $G = (V, T, S, P)$ be a grammar. Then the set

$$L(G) = \left\{ w \in T^* : S \overset{*}{\Rightarrow} w \right\}$$

is the *language generated* by G.

If $w \in L(G)$ then the sequence

$$S \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n \Rightarrow w$$

is a *derivation* of the sentence w. The strings $S, w_1, w_2, \ldots, w_n$, which contain variables as well as terminals, are called *sentential forms* of the derivation.

By applying the production rules in a different order, a given grammar can normally generate many strings. The set of all such terminal strings is the language defined or generated by the grammar.

Definition 1.2

Let $G = (V, T, S, P)$ be a grammar. Then the set

$$L(G) = \left\{ w \in T^* : S \overset{*}{\Rightarrow} w \right\}$$

is the *language generated* by G.

If $w \in L(G)$ then the sequence

$$S \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n \Rightarrow w$$

is a *derivation* of the sentence w. The strings $S, w_1, w_2, \ldots, w_n$, which contain variables as well as terminals, are called *sentential forms* of the derivation.

By applying the production rules in a different order, a given grammar can normally generate many strings. The set of all such terminal strings is the language defined or generated by the grammar.

Definition 1.2

Let $G = (V, T, S, P)$ be a grammar. Then the set

$$L(G) = \left\{ w \in T^* : S \overset{*}{\Rightarrow} w \right\}$$

is the *language generated* by G.

If $w \in L(G)$ then the sequence

$$S \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n \Rightarrow w$$

is a *derivation* of the sentence w. The strings $S, w_1, w_2, \ldots, w_n$, which contain variables as well as terminals, are called *sentential forms* of the derivation.

By applying the production rules in a different order, a given grammar can normally generate many strings. The set of all such terminal strings is the language defined or generated by the grammar.

Definition 1.2

Let $G = (V, T, S, P)$ be a grammar. Then the set

$$L(G) = \left\{ w \in T^* : S \stackrel{*}{\Rightarrow} w \right\}$$

is the *language generated* by G.

If $w \in L(G)$ then the sequence

$$S \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n \Rightarrow w$$

is a *derivation* of the sentence w. The strings $S, w_1, w_2, \ldots, w_n$, which contain variables as well as terminals, are called *sentential forms* of the derivation.

By applying the production rules in a different order, a given grammar can normally generate many strings. The set of all such terminal strings is the language defined or generated by the grammar.

Definition 1.2

Let $G = (V, T, S, P)$ be a grammar. Then the set

$$L(G) = \left\{ w \in T^* : S \stackrel{*}{\Rightarrow} w \right\}$$

is the *language generated* by G.

If $w \in L(G)$ then the sequence

$$S \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n \Rightarrow w$$

is a *derivation* of the sentence w. The strings $S, w_1, w_2, \ldots, w_n$, which contain variables as well as terminals, are called *sentential forms* of the derivation.

By applying the production rules in a different order, a given grammar can normally generate many strings. The set of all such terminal strings is the language defined or generated by the grammar.

Definition 1.2

Let $G = (V, T, S, P)$ be a grammar. Then the set

$$L(G) = \left\{ w \in T^* : S \overset{*}{\Rightarrow} w \right\}$$

is the *language generated* by G.

If $w \in L(G)$ then the sequence

$$S \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n \Rightarrow w$$

is a *derivation* of the sentence w. The strings $S, w_1, w_2, \ldots, w_n$, which contain variables as well as terminals, are called *sentential forms* of the derivation.

By applying the production rules in a different order, a given grammar can normally generate many strings. The set of all such terminal strings is the language defined or generated by the grammar.

Definition 1.2

Let $G = (V, T, S, P)$ be a grammar. Then the set

$$L(G) = \left\{ w \in T^* : S \overset{*}{\Rightarrow} w \right\}$$

is the *language generated* by G.

If $w \in L(G)$ then the sequence

$$S \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n \Rightarrow w$$

is a *derivation* of the sentence w. The strings $S, w_1, w_2, \ldots, w_n$, which contain variables as well as terminals, are called *sentential forms* of the derivation.

By applying the production rules in a different order, a given grammar can normally generate many strings. The set of all such terminal strings is the language defined or generated by the grammar.

Definition 1.2

Let $G = (V, T, S, P)$ be a grammar. Then the set

$$L(G) = \left\{ w \in T^* : S \overset{*}{\Rightarrow} w \right\}$$

is the *language generated* by G.

If $w \in L(G)$ then the sequence

$$S \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n \Rightarrow w$$

is a *derivation* of the sentence w. The strings $S, w_1, w_2, \ldots, w_n$, which contain variables as well as terminals, are called *sentential forms* of the derivation.

Example 1.11

Consider the grammar

$$G = (\{S\}, \{a, b\}, S, P),$$

with P given by

$$S \to aSb,$$
$$S \to \lambda.$$

Then

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb,$$

so we can write

$$S \overset{*}{\Rightarrow} aabb.$$

The string $aabb$ is a sentence in the language generated by G, while the string $aaSbb$ is a sentential form.

Example 1.11

Consider the grammar

$$G = (\{S\}, \{a, b\}, S, P),$$

with P given by

$$S \to aSb,$$
$$S \to \lambda.$$

Then

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb,$$

so we can write

$$S \stackrel{*}{\Rightarrow} aabb.$$

The string $aabb$ is a sentence in the language generated by G, while the string $aaSbb$ is a sentential form.

Example 1.11

Consider the grammar

$$G = (\{S\}, \{a, b\}, S, P),$$

with P given by

$$S \to aSb,$$
$$S \to \lambda.$$

Then

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb,$$

so we can write

$$S \stackrel{*}{\Rightarrow} aabb.$$

The string $aabb$ is a sentence in the language generated by G, while the string $aaSbb$ is a sentential form.

Example 1.11

Consider the grammar

$$G = (\{S\}, \{a, b\}, S, P),$$

with P given by

$$S \to aSb,$$
$$S \to \lambda.$$

Then

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb,$$

so we can write

$$S \stackrel{*}{\Rightarrow} aabb.$$

The string $aabb$ is a sentence in the language generated by G, while the string $aaSbb$ is a sentential form.

Example 1.11

Consider the grammar

$$G = (\{S\}, \{a, b\}, S, P),$$

with P given by

$$S \to aSb,$$
$$S \to \lambda.$$

Then

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb,$$

so we can write

$$S \overset{*}{\Rightarrow} aabb.$$

The string $aabb$ is a sentence in the language generated by G, while the string $aaSbb$ is a sentential form.

Example 1.11

Consider the grammar

$$G = (\{S\}, \{a, b\}, S, P),$$

with P given by

$$S \to aSb,$$
$$S \to \lambda.$$

Then

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb,$$

so we can write

$$S \overset{*}{\Rightarrow} aabb.$$

The string $aabb$ is a sentence in the language generated by G, while the string $aaSbb$ is a sentential form.

Example 1.11

Consider the grammar

$$G = (\{S\}, \{a, b\}, S, P),$$

with P given by

$$S \to aSb,$$
$$S \to \lambda.$$

Then

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb,$$

so we can write

$$S \overset{*}{\Rightarrow} aabb.$$

The string $aabb$ is a sentence in the language generated by G, while the string $aaSbb$ is a sentential form.

Example 1.11

Consider the grammar

$$G = (\{S\}, \{a, b\}, S, P),$$

with P given by

$$S \to aSb,$$
$$S \to \lambda.$$

Then

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb,$$

so we can write

$$S \overset{*}{\Rightarrow} aabb.$$

The string $aabb$ is a sentence in the language generated by G, while the string $aaSbb$ is a sentential form.

Example 1.11

Consider the grammar

$$G = (\{S\}, \{a, b\}, S, P),$$

with P given by

$$S \to aSb,$$
$$S \to \lambda.$$

Then

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb,$$

so we can write

$$S \overset{*}{\Rightarrow} aabb.$$

The string $aabb$ is a sentence in the language generated by G, while the string $aaSbb$ is a sentential form.

Example 1.11

Consider the grammar

$$G = (\{S\}, \{a, b\}, S, P),$$

with P given by

$$S \to aSb,$$
$$S \to \lambda.$$

Then

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb,$$

so we can write

$$S \overset{*}{\Rightarrow} aabb.$$

The string $aabb$ is a sentence in the language generated by G, while the string $aaSbb$ is a sentential form.

Example 1.11

Consider the grammar

$$G = (\{S\}, \{a, b\}, S, P),$$

with P given by

$$S \to aSb,$$
$$S \to \lambda.$$

Then

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb,$$

so we can write

$$S \stackrel{*}{\Rightarrow} aabb.$$

The string $aabb$ is a sentence in the language generated by G, while the string $aaSbb$ is a sentential form.

Example 1.11

Consider the grammar

$$G = (\{S\}, \{a, b\}, S, P),$$

with P given by

$$S \to aSb,$$
$$S \to \lambda.$$

Then

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb,$$

so we can write

$$S \stackrel{*}{\Rightarrow} aabb.$$

The string $aabb$ is a sentence in the language generated by G, while the string $aaSbb$ is a sentential form.

Example 1.11 (continuation)

A grammar G completely defines $L(G)$, but it may not be easy to get a very explicit description of the language from the grammar. Here, however, the answer is fairly clear. It is not hard to conjecture that

$$L(G) = \{a^n b^n : n \geq 0\},$$

and it is easy to prove it. If we notice that the rule $S \to aSb$ is recursive, a proof by induction readily suggests itself. We first show that all sentential forms must have the form

$$w_i = a^i S b^i. \tag{2}$$

Suppose that condition (2) holds for all sentential forms w_i of length $2i+1$ or less. To get another sentential form (which is not a sentence), we can only apply the production $S \to aSb$. This gets us

$$a^i S b^i \Rightarrow a^{i+1} S b^{i+1},$$

so that every sentential form of length $2i+3$ is also of the form (2). Since (2) is obviously true for $i = 1$, it holds by induction for all i. Finally, to get a sentence, we must apply the production $S \to \lambda$, and we see that

$$S \overset{*}{\Rightarrow} a^n S b^n \Rightarrow a^n b^n$$

represents all possible derivations. Thus, G can derive only strings of the form $a^n b^n$.

We also have to show that all strings of this form can be derived. This is easy; we simply apply $S \to aSb$ as many times as needed, followed by $S \to \lambda$. ■

Example 1.11 (continuation)

A grammar G completely defines $L(G)$, but it may not be easy to get a very explicit description of the language from the grammar. Here, however, the answer is fairly clear. It is not hard to conjecture that

$$L(G) = \{a^n b^n : n \geq 0\},$$

and it is easy to prove it. If we notice that the rule $S \to aSb$ is recursive, a proof by induction readily suggests itself. We first show that all sentential forms must have the form

$$w_i = a^i S b^i. \tag{2}$$

Suppose that condition (2) holds for all sentential forms w_i of length $2i + 1$ or less. To get another sentential form (which is not a sentence), we can only apply the production $S \to aSb$. This gets us

$$a^i S b^i \Rightarrow a^{i+1} S b^{i+1},$$

so that every sentential form of length $2i + 3$ is also of the form (2). Since (2) is obviously true for $i = 1$, it holds by induction for all i. Finally, to get a sentence, we must apply the production $S \to \lambda$, and we see that

$$S \stackrel{*}{\Rightarrow} a^n S b^n \Rightarrow a^n b^n$$

represents all possible derivations. Thus, G can derive only strings of the form $a^n b^n$.

We also have to show that all strings of this form can be derived. This is easy; we simply apply $S \to aSb$ as many times as needed, followed by $S \to \lambda$. ■

Example 1.11 (continuation)

A grammar G completely defines $L(G)$, but it may not be easy to get a very explicit description of the language from the grammar. Here, however, the answer is fairly clear. It is not hard to conjecture that

$$L(G) = \{a^n b^n : n \geq 0\},$$

and it is easy to prove it. If we notice that the rule $S \to aSb$ is recursive, a proof by induction readily suggests itself. We first show that all sentential forms must have the form

$$w_i = a^i S b^i. \tag{2}$$

Suppose that condition (2) holds for all sentential forms w_i of length $2i + 1$ or less. To get another sentential form (which is not a sentence), we can only apply the production $S \to aSb$. This gets us

$$a^i S b^i \Rightarrow a^{i+1} S b^{i+1},$$

so that every sentential form of length $2i + 3$ is also of the form (2). Since (2) is obviously true for $i = 1$, it holds by induction for all i. Finally, to get a sentence, we must apply the production $S \to \lambda$, and we see that

$$S \stackrel{*}{\Rightarrow} a^n S b^n \Rightarrow a^n b^n$$

represents all possible derivations. Thus, G can derive only strings of the form $a^n b^n$.

We also have to show that all strings of this form can be derived. This is easy; we simply apply $S \to aSb$ as many times as needed, followed by $S \to \lambda$. ■

Example 1.11 (continuation)

A grammar G completely defines $L(G)$, but it may not be easy to get a very explicit description of the language from the grammar. Here, however, the answer is fairly clear. It is not hard to conjecture that

$$L(G) = \{a^n b^n : n \geq 0\},$$

and it is easy to prove it. If we notice that the rule $S \to aSb$ is recursive, a proof by induction readily suggests itself. We first show that all sentential forms must have the form

$$w_i = a^i S b^i. \tag{2}$$

Suppose that condition (2) holds for all sentential forms w_i of length $2i + 1$ or less. To get another sentential form (which is not a sentence), we can only apply the production $S \to aSb$. This gets us

$$a^i S b^i \Rightarrow a^{i+1} S b^{i+1},$$

so that every sentential form of length $2i + 3$ is also of the form (2). Since (2) is obviously true for $i = 1$, it holds by induction for all i. Finally, to get a sentence, we must apply the production $S \to \lambda$, and we see that

$$S \overset{*}{\Rightarrow} a^n S b^n \Rightarrow a^n b^n$$

represents all possible derivations. Thus, G can derive only strings of the form $a^n b^n$.

We also have to show that all strings of this form can be derived. This is easy; we simply apply $S \to aSb$ as many times as needed, followed by $S \to \lambda$. ■

Example 1.11 (continuation)

A grammar G completely defines $L(G)$, but it may not be easy to get a very explicit description of the language from the grammar. Here, however, the answer is fairly clear. It is not hard to conjecture that

$$L(G) = \{a^n b^n : n \geq 0\},$$

and it is easy to prove it. If we notice that the rule $S \to aSb$ is recursive, a proof by induction readily suggests itself. We first show that all sentential forms must have the form

$$w_i = a^i S b^i. \quad (2)$$

Suppose that condition (2) holds for all sentential forms w_i of length $2i + 1$ or less. To get another sentential form (which is not a sentence), we can only apply the production $S \to aSb$. This gets us

$$a^i S b^i \Rightarrow a^{i+1} S b^{i+1},$$

so that every sentential form of length $2i + 3$ is also of the form (2). Since (2) is obviously true for $i = 1$, it holds by induction for all i. Finally, to get a sentence, we must apply the production $S \to \lambda$, and we see that

$$S \overset{*}{\Rightarrow} a^n S b^n \Rightarrow a^n b^n$$

represents all possible derivations. Thus, G can derive only strings of the form $a^n b^n$.

We also have to show that all strings of this form can be derived. This is easy; we simply apply $S \to aSb$ as many times as needed, followed by $S \to \lambda$. ■

Example 1.11 (continuation)

A grammar G completely defines $L(G)$, but it may not be easy to get a very explicit description of the language from the grammar. Here, however, the answer is fairly clear. It is not hard to conjecture that

$$L(G) = \{a^n b^n : n \geq 0\},$$

and it is easy to prove it. If we notice that the rule $S \to aSb$ is recursive, a proof by induction readily suggests itself. We first show that all sentential forms must have the form

$$w_i = a^i S b^i. \quad (2)$$

Suppose that condition (2) holds for all sentential forms w_i of length $2i + 1$ or less. To get another sentential form (which is not a sentence), we can only apply the production $S \to aSb$. This gets us

$$a^i S b^i \Rightarrow a^{i+1} S b^{i+1},$$

so that every sentential form of length $2i + 3$ is also of the form (2). Since (2) is obviously true for $i = 1$, it holds by induction for all i. Finally, to get a sentence, we must apply the production $S \to \lambda$, and we see that

$$S \overset{*}{\Rightarrow} a^n S b^n \Rightarrow a^n b^n$$

represents all possible derivations. Thus, G can derive only strings of the form $a^n b^n$.

We also have to show that all strings of this form can be derived. This is easy; we simply apply $S \to aSb$ as many times as needed, followed by $S \to \lambda$. ■

Example 1.11 (continuation)

A grammar G completely defines $L(G)$, but it may not be easy to get a very explicit description of the language from the grammar. Here, however, the answer is fairly clear. It is not hard to conjecture that

$$L(G) = \{a^n b^n : n \geq 0\},$$

and it is easy to prove it. If we notice that the rule $S \to aSb$ is recursive, a proof by induction readily suggests itself. We first show that all sentential forms must have the form

$$w_i = a^i S b^i. \tag{2}$$

Suppose that condition (2) holds for all sentential forms w_i of length $2i + 1$ or less. To get another sentential form (which is not a sentence), we can only apply the production $S \to aSb$. This gets us

$$a^i S b^i \Rightarrow a^{i+1} S b^{i+1},$$

so that every sentential form of length $2i + 3$ is also of the form (2). Since (2) is obviously true for $i = 1$, it holds by induction for all i. Finally, to get a sentence, we must apply the production $S \to \lambda$, and we see that

$$S \overset{*}{\Rightarrow} a^n S b^n \Rightarrow a^n b^n$$

represents all possible derivations. Thus, G can derive only strings of the form $a^n b^n$.

We also have to show that all strings of this form can be derived. This is easy; we simply apply $S \to aSb$ as many times as needed, followed by $S \to \lambda$. ■

Example 1.11 (continuation)

A grammar G completely defines $L(G)$, but it may not be easy to get a very explicit description of the language from the grammar. Here, however, the answer is fairly clear. It is not hard to conjecture that

$$L(G) = \{a^n b^n : n \geq 0\},$$

and it is easy to prove it. If we notice that the rule $S \to aSb$ is recursive, a proof by induction readily suggests itself. We first show that all sentential forms must have the form

$$w_i = a^i S b^i. \qquad (2)$$

Suppose that condition (2) holds for all sentential forms w_i of length $2i + 1$ or less. To get another sentential form (which is not a sentence), we can only apply the production $S \to aSb$. This gets us

$$a^i S b^i \Rightarrow a^{i+1} S b^{i+1},$$

so that every sentential form of length $2i + 3$ is also of the form (2). Since (2) is obviously true for $i = 1$, it holds by induction for all i. Finally, to get a sentence, we must apply the production $S \to \lambda$, and we see that

$$S \overset{*}{\Rightarrow} a^n S b^n \Rightarrow a^n b^n$$

represents all possible derivations. Thus, G can derive only strings of the form $a^n b^n$.

We also have to show that all strings of this form can be derived. This is easy; we simply apply $S \to aSb$ as many times as needed, followed by $S \to \lambda$. ■

Example 1.11 (continuation)

A grammar G completely defines $L(G)$, but it may not be easy to get a very explicit description of the language from the grammar. Here, however, the answer is fairly clear. It is not hard to conjecture that

$$L(G) = \{a^n b^n : n \geq 0\},$$

and it is easy to prove it. If we notice that the rule $S \to aSb$ is recursive, a proof by induction readily suggests itself. We first show that all sentential forms must have the form

$$w_i = a^i S b^i. \tag{2}$$

Suppose that condition (2) holds for all sentential forms w_i of length $2i + 1$ or less. To get another sentential form (which is not a sentence), we can only apply the production $S \to aSb$. This gets us

$$a^i S b^i \Rightarrow a^{i+1} S b^{i+1},$$

so that every sentential form of length $2i + 3$ is also of the form (2). Since (2) is obviously true for $i = 1$, it holds by induction for all i. Finally, to get a sentence, we must apply the production $S \to \lambda$, and we see that

$$S \overset{*}{\Rightarrow} a^n S b^n \Rightarrow a^n b^n$$

represents all possible derivations. Thus, G can derive only strings of the form $a^n b^n$.

We also have to show that all strings of this form can be derived. This is easy; we simply apply $S \to aSb$ as many times as needed, followed by $S \to \lambda$. ■

Example 1.11 (continuation)

A grammar G completely defines $L(G)$, but it may not be easy to get a very explicit description of the language from the grammar. Here, however, the answer is fairly clear. It is not hard to conjecture that

$$L(G) = \{a^n b^n : n \geq 0\},$$

and it is easy to prove it. If we notice that the rule $S \to aSb$ is recursive, a proof by induction readily suggests itself. We first show that all sentential forms must have the form

$$w_i = a^i S b^i. \tag{2}$$

Suppose that condition (2) holds for all sentential forms w_i of length $2i+1$ or less. To get another sentential form (which is not a sentence), we can only apply the production $S \to aSb$. This gets us

$$a^i S b^i \Rightarrow a^{i+1} S b^{i+1},$$

so that every sentential form of length $2i+3$ is also of the form (2). Since (2) is obviously true for $i = 1$, it holds by induction for all i. Finally, to get a sentence, we must apply the production $S \to \lambda$, and we see that

$$S \overset{*}{\Rightarrow} a^n S b^n \Rightarrow a^n b^n$$

represents all possible derivations. Thus, G can derive only strings of the form $a^n b^n$.

We also have to show that all strings of this form can be derived. This is easy; we simply apply $S \to aSb$ as many times as needed, followed by $S \to \lambda$. ■

Example 1.11 (continuation)

A grammar G completely defines $L(G)$, but it may not be easy to get a very explicit description of the language from the grammar. Here, however, the answer is fairly clear. It is not hard to conjecture that

$$L(G) = \{a^n b^n : n \geq 0\},$$

and it is easy to prove it. If we notice that the rule $S \to aSb$ is recursive, a proof by induction readily suggests itself. We first show that all sentential forms must have the form

$$w_i = a^i S b^i. \tag{2}$$

Suppose that condition (2) holds for all sentential forms w_i of length $2i + 1$ or less. To get another sentential form (which is not a sentence), we can only apply the production $S \to aSb$. This gets us

$$a^i S b^i \Rightarrow a^{i+1} S b^{i+1},$$

so that every sentential form of length $2i + 3$ is also of the form (2). Since (2) is obviously true for $i = 1$, it holds by induction for all i. Finally, to get a sentence, we must apply the production $S \to \lambda$, and we see that

$$S \overset{*}{\Rightarrow} a^n S b^n \Rightarrow a^n b^n$$

represents all possible derivations. Thus, G can derive only strings of the form $a^n b^n$.

We also have to show that all strings of this form can be derived. This is easy; we simply apply $S \to aSb$ as many times as needed, followed by $S \to \lambda$. ■

Example 1.11 (continuation)

A grammar G completely defines $L(G)$, but it may not be easy to get a very explicit description of the language from the grammar. Here, however, the answer is fairly clear. It is not hard to conjecture that

$$L(G) = \{a^n b^n : n \geq 0\},$$

and it is easy to prove it. If we notice that the rule $S \to aSb$ is recursive, a proof by induction readily suggests itself. We first show that all sentential forms must have the form

$$w_i = a^i S b^i. \quad (2)$$

Suppose that condition (2) holds for all sentential forms w_i of length $2i + 1$ or less. To get another sentential form (which is not a sentence), we can only apply the production $S \to aSb$. This gets us

$$a^i S b^i \Rightarrow a^{i+1} S b^{i+1},$$

so that every sentential form of length $2i + 3$ is also of the form (2). Since (2) is obviously true for $i = 1$, it holds by induction for all i. Finally, to get a sentence, we must apply the production $S \to \lambda$, and we see that

$$S \stackrel{*}{\Rightarrow} a^n S b^n \Rightarrow a^n b^n$$

represents all possible derivations. Thus, G can derive only strings of the form $a^n b^n$.

We also have to show that all strings of this form can be derived. This is easy; we simply apply $S \to aSb$ as many times as needed, followed by $S \to \lambda$. ■

Example 1.11 (continuation)

A grammar G completely defines $L(G)$, but it may not be easy to get a very explicit description of the language from the grammar. Here, however, the answer is fairly clear. It is not hard to conjecture that

$$L(G) = \{a^n b^n : n \geq 0\},$$

and it is easy to prove it. If we notice that the rule $S \to aSb$ is recursive, a proof by induction readily suggests itself. We first show that all sentential forms must have the form

$$w_i = a^i S b^i. \tag{2}$$

Suppose that condition (2) holds for all sentential forms w_i of length $2i+1$ or less. To get another sentential form (which is not a sentence), we can only apply the production $S \to aSb$. This gets us

$$a^i S b^i \Rightarrow a^{i+1} S b^{i+1},$$

so that every sentential form of length $2i+3$ is also of the form (2). Since (2) is obviously true for $i = 1$, it holds by induction for all i. Finally, to get a sentence, we must apply the production $S \to \lambda$, and we see that

$$S \overset{*}{\Rightarrow} a^n S b^n \Rightarrow a^n b^n$$

represents all possible derivations. Thus, G can derive only strings of the form $a^n b^n$.

We also have to show that all strings of this form can be derived. This is easy; we simply apply $S \to aSb$ as many times as needed, followed by $S \to \lambda$. ■

Example 1.11 (continuation)

A grammar G completely defines $L(G)$, but it may not be easy to get a very explicit description of the language from the grammar. Here, however, the answer is fairly clear. It is not hard to conjecture that

$$L(G) = \{a^n b^n : n \geq 0\},$$

and it is easy to prove it. If we notice that the rule $S \to aSb$ is recursive, a proof by induction readily suggests itself. We first show that all sentential forms must have the form

$$w_i = a^i S b^i. \qquad (2)$$

Suppose that condition (2) holds for all sentential forms w_i of length $2i + 1$ or less. To get another sentential form (which is not a sentence), we can only apply the production $S \to aSb$. This gets us

$$a^i S b^i \Rightarrow a^{i+1} S b^{i+1},$$

so that every sentential form of length $2i + 3$ is also of the form (2). Since (2) is obviously true for $i = 1$, it holds by induction for all i. Finally, to get a sentence, we must apply the production $S \to \lambda$, and we see that

$$S \stackrel{*}{\Rightarrow} a^n S b^n \Rightarrow a^n b^n$$

represents all possible derivations. Thus, G can derive only strings of the form $a^n b^n$.

We also have to show that all strings of this form can be derived. This is easy; we simply apply $S \to aSb$ as many times as needed, followed by $S \to \lambda$. ■

Example 1.11 (continuation)

A grammar G completely defines $L(G)$, but it may not be easy to get a very explicit description of the language from the grammar. Here, however, the answer is fairly clear. It is not hard to conjecture that

$$L(G) = \{a^n b^n : n \geq 0\},$$

and it is easy to prove it. If we notice that the rule $S \to aSb$ is recursive, a proof by induction readily suggests itself. We first show that all sentential forms must have the form

$$w_i = a^i S b^i. \tag{2}$$

Suppose that condition (2) holds for all sentential forms w_i of length $2i + 1$ or less. To get another sentential form (which is not a sentence), we can only apply the production $S \to aSb$. This gets us

$$a^i S b^i \Rightarrow a^{i+1} S b^{i+1},$$

so that every sentential form of length $2i + 3$ is also of the form (2). Since (2) is obviously true for $i = 1$, it holds by induction for all i. Finally, to get a sentence, we must apply the production $S \to \lambda$, and we see that

$$S \overset{*}{\Rightarrow} a^n S b^n \Rightarrow a^n b^n$$

represents all possible derivations. Thus, G can derive only strings of the form $a^n b^n$.

We also have to show that all strings of this form can be derived. This is easy; we simply apply $S \to aSb$ as many times as needed, followed by $S \to \lambda$. ■

Example 1.11 (continuation)

A grammar G completely defines $L(G)$, but it may not be easy to get a very explicit description of the language from the grammar. Here, however, the answer is fairly clear. It is not hard to conjecture that

$$L(G) = \{a^n b^n : n \geq 0\},$$

and it is easy to prove it. If we notice that the rule $S \to aSb$ is recursive, a proof by induction readily suggests itself. We first show that all sentential forms must have the form

$$w_i = a^i S b^i. \quad (2)$$

Suppose that condition (2) holds for all sentential forms w_i of length $2i + 1$ or less. To get another sentential form (which is not a sentence), we can only apply the production $S \to aSb$. This gets us

$$a^i S b^i \Rightarrow a^{i+1} S b^{i+1},$$

so that every sentential form of length $2i + 3$ is also of the form (2). Since (2) is obviously true for $i = 1$, it holds by induction for all i. Finally, to get a sentence, we must apply the production $S \to \lambda$, and we see that

$$S \overset{*}{\Rightarrow} a^n S b^n \Rightarrow a^n b^n$$

represents all possible derivations. Thus, G can derive only strings of the form $a^n b^n$.

We also have to show that all strings of this form can be derived. This is easy; we simply apply $S \to aSb$ as many times as needed, followed by $S \to \lambda$. ∎

Example 1.11 (continuation)

A grammar G completely defines $L(G)$, but it may not be easy to get a very explicit description of the language from the grammar. Here, however, the answer is fairly clear. It is not hard to conjecture that

$$L(G) = \{a^n b^n : n \geq 0\},$$

and it is easy to prove it. If we notice that the rule $S \to aSb$ is recursive, a proof by induction readily suggests itself. We first show that all sentential forms must have the form

$$w_i = a^i S b^i. \tag{2}$$

Suppose that condition (2) holds for all sentential forms w_i of length $2i + 1$ or less. To get another sentential form (which is not a sentence), we can only apply the production $S \to aSb$. This gets us

$$a^i S b^i \Rightarrow a^{i+1} S b^{i+1},$$

so that every sentential form of length $2i + 3$ is also of the form (2). Since (2) is obviously true for $i = 1$, it holds by induction for all i. Finally, to get a sentence, we must apply the production $S \to \lambda$, and we see that

$$S \overset{*}{\Rightarrow} a^n S b^n \Rightarrow a^n b^n$$

represents all possible derivations. Thus, G can derive only strings of the form $a^n b^n$.

We also have to show that all strings of this form can be derived. This is easy; we simply apply $S \to aSb$ as many times as needed, followed by $S \to \lambda$. ■

Example 1.11 (continuation)

A grammar G completely defines $L(G)$, but it may not be easy to get a very explicit description of the language from the grammar. Here, however, the answer is fairly clear. It is not hard to conjecture that

$$L(G) = \{a^n b^n : n \geq 0\},$$

and it is easy to prove it. If we notice that the rule $S \to aSb$ is recursive, a proof by induction readily suggests itself. We first show that all sentential forms must have the form

$$w_i = a^i S b^i. \qquad (2)$$

Suppose that condition (2) holds for all sentential forms w_i of length $2i+1$ or less. To get another sentential form (which is not a sentence), we can only apply the production $S \to aSb$. This gets us

$$a^i S b^i \Rightarrow a^{i+1} S b^{i+1},$$

so that every sentential form of length $2i+3$ is also of the form (2). Since (2) is obviously true for $i = 1$, it holds by induction for all i. Finally, to get a sentence, we must apply the production $S \to \lambda$, and we see that

$$S \overset{*}{\Rightarrow} a^n S b^n \Rightarrow a^n b^n$$

represents all possible derivations. Thus, G can derive only strings of the form $a^n b^n$.

We also have to show that all strings of this form can be derived. This is easy; we simply apply $S \to aSb$ as many times as needed, followed by $S \to \lambda$. ■

Example 1.11 (continuation)

A grammar G completely defines $L(G)$, but it may not be easy to get a very explicit description of the language from the grammar. Here, however, the answer is fairly clear. It is not hard to conjecture that

$$L(G) = \{a^n b^n : n \geq 0\},$$

and it is easy to prove it. If we notice that the rule $S \to aSb$ is recursive, a proof by induction readily suggests itself. We first show that all sentential forms must have the form

$$w_i = a^i S b^i. \tag{2}$$

Suppose that condition (2) holds for all sentential forms w_i of length $2i + 1$ or less. To get another sentential form (which is not a sentence), we can only apply the production $S \to aSb$. This gets us

$$a^i S b^i \Rightarrow a^{i+1} S b^{i+1},$$

so that every sentential form of length $2i + 3$ is also of the form (2). Since (2) is obviously true for $i = 1$, it holds by induction for all i. Finally, to get a sentence, we must apply the production $S \to \lambda$, and we see that

$$S \overset{*}{\Rightarrow} a^n S b^n \Rightarrow a^n b^n$$

represents all possible derivations. Thus, G can derive only strings of the form $a^n b^n$.

We also have to show that all strings of this form can be derived. This is easy; we simply apply $S \to aSb$ as many times as needed, followed by $S \to \lambda$. ■

Example 1.11 (continuation)

A grammar G completely defines $L(G)$, but it may not be easy to get a very explicit description of the language from the grammar. Here, however, the answer is fairly clear. It is not hard to conjecture that

$$L(G) = \{a^n b^n : n \geq 0\},$$

and it is easy to prove it. If we notice that the rule $S \to aSb$ is recursive, a proof by induction readily suggests itself. We first show that all sentential forms must have the form

$$w_i = a^i S b^i. \tag{2}$$

Suppose that condition (2) holds for all sentential forms w_i of length $2i+1$ or less. To get another sentential form (which is not a sentence), we can only apply the production $S \to aSb$. This gets us

$$a^i S b^i \Rightarrow a^{i+1} S b^{i+1},$$

so that every sentential form of length $2i+3$ is also of the form (2). Since (2) is obviously true for $i = 1$, it holds by induction for all i. Finally, to get a sentence, we must apply the production $S \to \lambda$, and we see that

$$S \overset{*}{\Rightarrow} a^n S b^n \Rightarrow a^n b^n$$

represents all possible derivations. Thus, G can derive only strings of the form $a^n b^n$.

We also have to show that all strings of this form can be derived. This is easy; we simply apply $S \to aSb$ as many times as needed, followed by $S \to \lambda$. ■

Example 1.11 (continuation)

A grammar G completely defines $L(G)$, but it may not be easy to get a very explicit description of the language from the grammar. Here, however, the answer is fairly clear. It is not hard to conjecture that

$$L(G) = \{a^n b^n : n \geq 0\},$$

and it is easy to prove it. If we notice that the rule $S \to aSb$ is recursive, a proof by induction readily suggests itself. We first show that all sentential forms must have the form

$$w_i = a^i S b^i. \quad (2)$$

Suppose that condition (2) holds for all sentential forms w_i of length $2i + 1$ or less. To get another sentential form (which is not a sentence), we can only apply the production $S \to aSb$. This gets us

$$a^i S b^i \Rightarrow a^{i+1} S b^{i+1},$$

so that every sentential form of length $2i + 3$ is also of the form (2). Since (2) is obviously true for $i = 1$, it holds by induction for all i. Finally, to get a sentence, we must apply the production $S \to \lambda$, and we see that

$$S \stackrel{*}{\Rightarrow} a^n S b^n \Rightarrow a^n b^n$$

represents all possible derivations. Thus, G can derive only strings of the form $a^n b^n$.

We also have to show that all strings of this form can be derived. This is easy; we simply apply $S \to aSb$ as many times as needed, followed by $S \to \lambda$. ■

Example 1.11 (continuation)

A grammar G completely defines $L(G)$, but it may not be easy to get a very explicit description of the language from the grammar. Here, however, the answer is fairly clear. It is not hard to conjecture that

$$L(G) = \{a^n b^n : n \geq 0\},$$

and it is easy to prove it. If we notice that the rule $S \to aSb$ is recursive, a proof by induction readily suggests itself. We first show that all sentential forms must have the form

$$w_i = a^i S b^i. \tag{2}$$

Suppose that condition (2) holds for all sentential forms w_i of length $2i+1$ or less. To get another sentential form (which is not a sentence), we can only apply the production $S \to aSb$. This gets us

$$a^i S b^i \Rightarrow a^{i+1} S b^{i+1},$$

so that every sentential form of length $2i+3$ is also of the form (2). Since (2) is obviously true for $i = 1$, it holds by induction for all i. Finally, to get a sentence, we must apply the production $S \to \lambda$, and we see that

$$S \overset{*}{\Rightarrow} a^n S b^n \Rightarrow a^n b^n$$

represents all possible derivations. Thus, G can derive only strings of the form $a^n b^n$.

We also have to show that all strings of this form can be derived. This is easy; we simply apply $S \to aSb$ as many times as needed, followed by $S \to \lambda$. ■

Example 1.11 (continuation)

A grammar G completely defines $L(G)$, but it may not be easy to get a very explicit description of the language from the grammar. Here, however, the answer is fairly clear. It is not hard to conjecture that

$$L(G) = \{a^n b^n : n \geq 0\},$$

and it is easy to prove it. If we notice that the rule $S \to aSb$ is recursive, a proof by induction readily suggests itself. We first show that all sentential forms must have the form

$$w_i = a^i S b^i. \tag{2}$$

Suppose that condition (2) holds for all sentential forms w_i of length $2i + 1$ or less. To get another sentential form (which is not a sentence), we can only apply the production $S \to aSb$. This gets us

$$a^i S b^i \Rightarrow a^{i+1} S b^{i+1},$$

so that every sentential form of length $2i + 3$ is also of the form (2). Since (2) is obviously true for $i = 1$, it holds by induction for all i. Finally, to get a sentence, we must apply the production $S \to \lambda$, and we see that

$$S \stackrel{*}{\Rightarrow} a^n S b^n \Rightarrow a^n b^n$$

represents all possible derivations. Thus, G can derive only strings of the form $a^n b^n$.

We also have to show that all strings of this form can be derived. This is easy; we simply apply $S \to aSb$ as many times as needed, followed by $S \to \lambda$. ■

Example 1.11 (continuation)

A grammar G completely defines $L(G)$, but it may not be easy to get a very explicit description of the language from the grammar. Here, however, the answer is fairly clear. It is not hard to conjecture that

$$L(G) = \{a^n b^n : n \geq 0\},$$

and it is easy to prove it. If we notice that the rule $S \to aSb$ is recursive, a proof by induction readily suggests itself. We first show that all sentential forms must have the form

$$w_i = a^i S b^i. \tag{2}$$

Suppose that condition (2) holds for all sentential forms w_i of length $2i + 1$ or less. To get another sentential form (which is not a sentence), we can only apply the production $S \to aSb$. This gets us

$$a^i S b^i \Rightarrow a^{i+1} S b^{i+1},$$

so that every sentential form of length $2i + 3$ is also of the form (2). Since (2) is obviously true for $i = 1$, it holds by induction for all i. Finally, to get a sentence, we must apply the production $S \to \lambda$, and we see that

$$S \stackrel{*}{\Rightarrow} a^n S b^n \Rightarrow a^n b^n$$

represents all possible derivations. Thus, G can derive only strings of the form $a^n b^n$.

We also have to show that all strings of this form can be derived. This is easy; we simply apply $S \to aSb$ as many times as needed, followed by $S \to \lambda$. ■

Example 1.11 (continuation)

A grammar G completely defines $L(G)$, but it may not be easy to get a very explicit description of the language from the grammar. Here, however, the answer is fairly clear. It is not hard to conjecture that

$$L(G) = \{a^n b^n : n \geq 0\},$$

and it is easy to prove it. If we notice that the rule $S \to aSb$ is recursive, a proof by induction readily suggests itself. We first show that all sentential forms must have the form

$$w_i = a^i S b^i. \tag{2}$$

Suppose that condition (2) holds for all sentential forms w_i of length $2i + 1$ or less. To get another sentential form (which is not a sentence), we can only apply the production $S \to aSb$. This gets us

$$a^i S b^i \Rightarrow a^{i+1} S b^{i+1},$$

so that every sentential form of length $2i + 3$ is also of the form (2). Since (2) is obviously true for $i = 1$, it holds by induction for all i. Finally, to get a sentence, we must apply the production $S \to \lambda$, and we see that

$$S \overset{*}{\Rightarrow} a^n S b^n \Rightarrow a^n b^n$$

represents all possible derivations. Thus, G can derive only strings of the form $a^n b^n$.

We also have to show that all strings of this form can be derived. This is easy; we simply apply $S \to aSb$ as many times as needed, followed by $S \to \lambda$. ■

Example 1.11 (continuation)

A grammar G completely defines $L(G)$, but it may not be easy to get a very explicit description of the language from the grammar. Here, however, the answer is fairly clear. It is not hard to conjecture that

$$L(G) = \{a^n b^n : n \geq 0\},$$

and it is easy to prove it. If we notice that the rule $S \to aSb$ is recursive, a proof by induction readily suggests itself. We first show that all sentential forms must have the form

$$w_i = a^i S b^i. \qquad (2)$$

Suppose that condition (2) holds for all sentential forms w_i of length $2i+1$ or less. To get another sentential form (which is not a sentence), we can only apply the production $S \to aSb$. This gets us

$$a^i S b^i \Rightarrow a^{i+1} S b^{i+1},$$

so that every sentential form of length $2i+3$ is also of the form (2). Since (2) is obviously true for $i = 1$, it holds by induction for all i. Finally, to get a sentence, we must apply the production $S \to \lambda$, and we see that

$$S \overset{*}{\Rightarrow} a^n S b^n \Rightarrow a^n b^n$$

represents all possible derivations. Thus, G can derive only strings of the form $a^n b^n$.

We also have to show that all strings of this form can be derived. This is easy; we simply apply $S \to aSb$ as many times as needed, followed by $S \to \lambda$. ■

Example 1.11 (continuation)

A grammar G completely defines $L(G)$, but it may not be easy to get a very explicit description of the language from the grammar. Here, however, the answer is fairly clear. It is not hard to conjecture that

$$L(G) = \{a^n b^n : n \geq 0\},$$

and it is easy to prove it. If we notice that the rule $S \to aSb$ is recursive, a proof by induction readily suggests itself. We first show that all sentential forms must have the form

$$w_i = a^i S b^i. \tag{2}$$

Suppose that condition (2) holds for all sentential forms w_i of length $2i + 1$ or less. To get another sentential form (which is not a sentence), we can only apply the production $S \to aSb$. This gets us

$$a^i S b^i \Rightarrow a^{i+1} S b^{i+1},$$

so that every sentential form of length $2i + 3$ is also of the form (2). Since (2) is obviously true for $i = 1$, it holds by induction for all i. Finally, to get a sentence, we must apply the production $S \to \lambda$, and we see that

$$S \overset{*}{\Rightarrow} a^n S b^n \Rightarrow a^n b^n$$

represents all possible derivations. Thus, G can derive only strings of the form $a^n b^n$.

We also have to show that all strings of this form can be derived. This is easy; we simply apply $S \to aSb$ as many times as needed, followed by $S \to \lambda$. ∎

Example 1.11 (continuation)

A grammar G completely defines $L(G)$, but it may not be easy to get a very explicit description of the language from the grammar. Here, however, the answer is fairly clear. It is not hard to conjecture that

$$L(G) = \{a^n b^n : n \geq 0\},$$

and it is easy to prove it. If we notice that the rule $S \to aSb$ is recursive, a proof by induction readily suggests itself. We first show that all sentential forms must have the form

$$w_i = a^i S b^i. \tag{2}$$

Suppose that condition (2) holds for all sentential forms w_i of length $2i + 1$ or less. To get another sentential form (which is not a sentence), we can only apply the production $S \to aSb$. This gets us

$$a^i S b^i \Rightarrow a^{i+1} S b^{i+1},$$

so that every sentential form of length $2i + 3$ is also of the form (2). Since (2) is obviously true for $i = 1$, it holds by induction for all i. Finally, to get a sentence, we must apply the production $S \to \lambda$, and we see that

$$S \stackrel{*}{\Rightarrow} a^n S b^n \Rightarrow a^n b^n$$

represents all possible derivations. Thus, G can derive only strings of the form $a^n b^n$.

We also have to show that all strings of this form can be derived. This is easy; we simply apply $S \to aSb$ as many times as needed, followed by $S \to \lambda$. ■

Example 1.12

Find a grammar that generates

$$L = \{a^n b^{n+1} : n \geq 0\}.$$

The idea behind the previous example can be extended to this case. All we need to do is generate an extra b. This can be done with a production $S \to Ab$, with other productions chosen so that A can derive the language in the previous example. Reasoning in this fashion, we get the grammar $G = (\{S, A\}, \{a, b\}, S, P)$, with productions

$$S \to Ab,$$
$$A \to aAb,$$
$$A \to \lambda.$$

Derive a few specific sentences to convince yourself that this works. ■

The previous examples are fairly easy ones, so rigorous arguments may seem superfluous. But often it is not so easy to find a grammar for a language described in an informal way or to give an intuitive characterization of the language defined by a grammar. To show that a given language is indeed generated by a certain grammar G, we must be able to show (a) that every $w \in L$ can be derived from S using G and (b) that every string so derived is in L.

Example 1.12

Find a grammar that generates

$$L = \{a^n b^{n+1} : n \geq 0\}.$$

The idea behind the previous example can be extended to this case. All we need to do is generate an extra b. This can be done with a production $S \to Ab$, with other productions chosen so that A can derive the language in the previous example. Reasoning in this fashion, we get the grammar $G = (\{S, A\}, \{a, b\}, S, P)$, with productions

$$S \to Ab,$$
$$A \to aAb,$$
$$A \to \lambda.$$

Derive a few specific sentences to convince yourself that this works. ■

The previous examples are fairly easy ones, so rigorous arguments may seem superfluous. But often it is not so easy to find a grammar for a language described in an informal way or to give an intuitive characterization of the language defined by a grammar. To show that a given language is indeed generated by a certain grammar G, we must be able to show (a) that every $w \in L$ can be derived from S using G and (b) that every string so derived is in L.

Example 1.12

Find a grammar that generates

$$L = \{a^n b^{n+1} : n \geq 0\}.$$

The idea behind the previous example can be extended to this case. All we need to do is generate an extra b. This can be done with a production $S \to Ab$, with other productions chosen so that A can derive the language in the previous example. Reasoning in this fashion, we get the grammar $G = (\{S, A\}, \{a, b\}, S, P)$, with productions

$$S \to Ab,$$
$$A \to aAb,$$
$$A \to \lambda.$$

Derive a few specific sentences to convince yourself that this works. ■

The previous examples are fairly easy ones, so rigorous arguments may seem superfluous. But often it is not so easy to find a grammar for a language described in an informal way or to give an intuitive characterization of the language defined by a grammar. To show that a given language is indeed generated by a certain grammar G, we must be able to show (a) that every $w \in L$ can be derived from S using G and (b) that every string so derived is in L.

Example 1.12

Find a grammar that generates

$$L = \{a^n b^{n+1} : n \geq 0\}.$$

The idea behind the previous example can be extended to this case. All we need to do is generate an extra b. This can be done with a production $S \to Ab$, with other productions chosen so that A can derive the language in the previous example. Reasoning in this fashion, we get the grammar $G = (\{S, A\}, \{a, b\}, S, P)$, with productions

$$S \to Ab,$$
$$A \to aAb,$$
$$A \to \lambda.$$

Derive a few specific sentences to convince yourself that this works. ■

The previous examples are fairly easy ones, so rigorous arguments may seem superfluous. But often it is not so easy to find a grammar for a language described in an informal way or to give an intuitive characterization of the language defined by a grammar. To show that a given language is indeed generated by a certain grammar G, we must be able to show (a) that every $w \in L$ can be derived from S using G and (b) that every string so derived is in L.

Example 1.12

Find a grammar that generates

$$L = \left\{a^n b^{n+1} : n \geq 0\right\}.$$

The idea behind the previous example can be extended to this case. All we need to do is generate an extra b. This can be done with a production $S \to Ab$, with other productions chosen so that A can derive the language in the previous example. Reasoning in this fashion, we get the grammar $G = (\{S, A\}, \{a, b\}, S, P)$, with productions

$$S \to Ab,$$
$$A \to aAb,$$
$$A \to \lambda.$$

Derive a few specific sentences to convince yourself that this works. ■

The previous examples are fairly easy ones, so rigorous arguments may seem superfluous. But often it is not so easy to find a grammar for a language described in an informal way or to give an intuitive characterization of the language defined by a grammar. To show that a given language is indeed generated by a certain grammar G, we must be able to show (a) that every $w \in L$ can be derived from S using G and (b) that every string so derived is in L.

Example 1.12

Find a grammar that generates

$$L = \{a^n b^{n+1} : n \geq 0\}.$$

The idea behind the previous example can be extended to this case. All we need to do is generate an extra b. This can be done with a production $S \to Ab$, with other productions chosen so that A can derive the language in the previous example. Reasoning in this fashion, we get the grammar $G = (\{S, A\}, \{a, b\}, S, P)$, with productions

$$S \to Ab,$$
$$A \to aAb,$$
$$A \to \lambda.$$

Derive a few specific sentences to convince yourself that this works. ■

The previous examples are fairly easy ones, so rigorous arguments may seem superfluous. But often it is not so easy to find a grammar for a language described in an informal way or to give an intuitive characterization of the language defined by a grammar. To show that a given language is indeed generated by a certain grammar G, we must be able to show (a) that every $w \in L$ can be derived from S using G and (b) that every string so derived is in L.

Example 1.12

Find a grammar that generates

$$L = \left\{a^n b^{n+1} : n \geq 0\right\}.$$

The idea behind the previous example can be extended to this case. All we need to do is generate an extra b. This can be done with a production $S \to Ab$, with other productions chosen so that A can derive the language in the previous example. Reasoning in this fashion, we get the grammar $G = (\{S, A\}, \{a, b\}, S, P)$, with productions

$$S \to Ab,$$
$$A \to aAb,$$
$$A \to \lambda.$$

Derive a few specific sentences to convince yourself that this works. ■

The previous examples are fairly easy ones, so rigorous arguments may seem superfluous. But often it is not so easy to find a grammar for a language described in an informal way or to give an intuitive characterization of the language defined by a grammar. To show that a given language is indeed generated by a certain grammar G, we must be able to show (a) that every $w \in L$ can be derived from S using G and (b) that every string so derived is in L.

Example 1.12

Find a grammar that generates

$$L = \{a^n b^{n+1} : n \geq 0\}.$$

The idea behind the previous example can be extended to this case. All we need to do is generate an extra b. This can be done with a production $S \to Ab$, with other productions chosen so that A can derive the language in the previous example. Reasoning in this fashion, we get the grammar $G = (\{S, A\}, \{a, b\}, S, P)$, with productions

$$S \to Ab,$$
$$A \to aAb,$$
$$A \to \lambda.$$

Derive a few specific sentences to convince yourself that this works. ■

The previous examples are fairly easy ones, so rigorous arguments may seem superfluous. But often it is not so easy to find a grammar for a language described in an informal way or to give an intuitive characterization of the language defined by a grammar. To show that a given language is indeed generated by a certain grammar G, we must be able to show (a) that every $w \in L$ can be derived from S using G and (b) that every string so derived is in L.

Example 1.12

Find a grammar that generates
$$L = \left\{a^n b^{n+1} : n \geq 0\right\}.$$
The idea behind the previous example can be extended to this case. All we need to do is generate an extra b. This can be done with a production $S \to Ab$, with other productions chosen so that A can derive the language in the previous example. Reasoning in this fashion, we get the grammar $G = (\{S, A\}, \{a, b\}, S, P)$, with productions

$$S \to Ab,$$
$$A \to aAb,$$
$$A \to \lambda.$$

Derive a few specific sentences to convince yourself that this works. ■

The previous examples are fairly easy ones, so rigorous arguments may seem superfluous. But often it is not so easy to find a grammar for a language described in an informal way or to give an intuitive characterization of the language defined by a grammar. To show that a given language is indeed generated by a certain grammar G, we must be able to show (a) that every $w \in L$ can be derived from S using G and (b) that every string so derived is in L.

Example 1.12

Find a grammar that generates

$$L = \left\{a^n b^{n+1} : n \geq 0\right\}.$$

The idea behind the previous example can be extended to this case. All we need to do is generate an extra b. This can be done with a production $S \to Ab$, with other productions chosen so that A can derive the language in the previous example. Reasoning in this fashion, we get the grammar $G = (\{S, A\}, \{a, b\}, S, P)$, with productions

$$S \to Ab,$$
$$A \to aAb,$$
$$A \to \lambda.$$

Derive a few specific sentences to convince yourself that this works. ■

The previous examples are fairly easy ones, so rigorous arguments may seem superfluous. But often it is not so easy to find a grammar for a language described in an informal way or to give an intuitive characterization of the language defined by a grammar. To show that a given language is indeed generated by a certain grammar G, we must be able to show (a) that every $w \in L$ can be derived from S using G and (b) that every string so derived is in L.

Example 1.12

Find a grammar that generates
$$L = \left\{a^n b^{n+1} : n \geq 0\right\}.$$
The idea behind the previous example can be extended to this case. All we need to do is generate an extra b. This can be done with a production $S \to Ab$, with other productions chosen so that A can derive the language in the previous example. Reasoning in this fashion, we get the grammar $G = (\{S, A\}, \{a, b\}, S, P)$, with productions

$$S \to Ab,$$
$$A \to aAb,$$
$$A \to \lambda.$$

Derive a few specific sentences to convince yourself that this works.

The previous examples are fairly easy ones, so rigorous arguments may seem superfluous. But often it is not so easy to find a grammar for a language described in an informal way or to give an intuitive characterization of the language defined by a grammar. To show that a given language is indeed generated by a certain grammar G, we must be able to show (a) that every $w \in L$ can be derived from S using G and (b) that every string so derived is in L.

Example 1.12

Find a grammar that generates

$$L = \left\{a^n b^{n+1} : n \geq 0\right\}.$$

The idea behind the previous example can be extended to this case. All we need to do is generate an extra b. This can be done with a production $S \to Ab$, with other productions chosen so that A can derive the language in the previous example. Reasoning in this fashion, we get the grammar $G = (\{S, A\}, \{a, b\}, S, P)$, with productions

$$S \to Ab,$$
$$A \to aAb,$$
$$A \to \lambda.$$

Derive a few specific sentences to convince yourself that this works. ■

The previous examples are fairly easy ones, so rigorous arguments may seem superfluous. But often it is not so easy to find a grammar for a language described in an informal way or to give an intuitive characterization of the language defined by a grammar. To show that a given language is indeed generated by a certain grammar G, we must be able to show (a) that every $w \in L$ can be derived from S using G and (b) that every string so derived is in L.

Example 1.12

Find a grammar that generates

$$L = \left\{ a^n b^{n+1} : n \geq 0 \right\}.$$

The idea behind the previous example can be extended to this case. All we need to do is generate an extra b. This can be done with a production $S \to Ab$, with other productions chosen so that A can derive the language in the previous example. Reasoning in this fashion, we get the grammar $G = (\{S, A\}, \{a, b\}, S, P)$, with productions

$$S \to Ab,$$
$$A \to aAb,$$
$$A \to \lambda.$$

Derive a few specific sentences to convince yourself that this works. ∎

The previous examples are fairly easy ones, so rigorous arguments may seem superfluous. But often it is not so easy to find a grammar for a language described in an informal way or to give an intuitive characterization of the language defined by a grammar. To show that a given language is indeed generated by a certain grammar G, we must be able to show (a) that every $w \in L$ can be derived from S using G and (b) that every string so derived is in L.

Example 1.12

Find a grammar that generates

$$L = \left\{a^n b^{n+1} : n \geq 0\right\}.$$

The idea behind the previous example can be extended to this case. All we need to do is generate an extra b. This can be done with a production $S \to Ab$, with other productions chosen so that A can derive the language in the previous example. Reasoning in this fashion, we get the grammar $G = (\{S, A\}, \{a, b\}, S, P)$, with productions

$$S \to Ab,$$
$$A \to aAb,$$
$$A \to \lambda.$$

Derive a few specific sentences to convince yourself that this works. ■

The previous examples are fairly easy ones, so rigorous arguments may seem superfluous. But often it is not so easy to find a grammar for a language described in an informal way or to give an intuitive characterization of the language defined by a grammar. To show that a given language is indeed generated by a certain grammar G, we must be able to show (a) that every $w \in L$ can be derived from S using G and (b) that every string so derived is in L.

Example 1.12

Find a grammar that generates

$$L = \left\{a^n b^{n+1} : n \geq 0\right\}.$$

The idea behind the previous example can be extended to this case. All we need to do is generate an extra b. This can be done with a production $S \to Ab$, with other productions chosen so that A can derive the language in the previous example. Reasoning in this fashion, we get the grammar $G = (\{S, A\}, \{a, b\}, S, P)$, with productions

$$S \to Ab,$$
$$A \to aAb,$$
$$A \to \lambda.$$

Derive a few specific sentences to convince yourself that this works. ■

The previous examples are fairly easy ones, so rigorous arguments may seem superfluous. But often it is not so easy to find a grammar for a language described in an informal way or to give an intuitive characterization of the language defined by a grammar. To show that a given language is indeed generated by a certain grammar G, we must be able to show (a) that every $w \in L$ can be derived from S using G and (b) that every string so derived is in L.

Example 1.12

Find a grammar that generates
$$L = \left\{a^n b^{n+1} : n \geq 0\right\}.$$
The idea behind the previous example can be extended to this case. All we need to do is generate an extra b. This can be done with a production $S \to Ab$, with other productions chosen so that A can derive the language in the previous example. Reasoning in this fashion, we get the grammar $G = (\{S, A\}, \{a, b\}, S, P)$, with productions

$$S \to Ab,$$
$$A \to aAb,$$
$$A \to \lambda.$$

Derive a few specific sentences to convince yourself that this works. ■

The previous examples are fairly easy ones, so rigorous arguments may seem superfluous. But often it is not so easy to find a grammar for a language described in an informal way or to give an intuitive characterization of the language defined by a grammar. To show that a given language is indeed generated by a certain grammar G, we must be able to show (a) that every $w \in L$ can be derived from S using G and (b) that every string so derived is in L.

Example 1.12

Find a grammar that generates

$$L = \{a^n b^{n+1} : n \geq 0\}.$$

The idea behind the previous example can be extended to this case. All we need to do is generate an extra b. This can be done with a production $S \to Ab$, with other productions chosen so that A can derive the language in the previous example. Reasoning in this fashion, we get the grammar $G = (\{S, A\}, \{a, b\}, S, P)$, with productions

$$S \to Ab,$$
$$A \to aAb,$$
$$A \to \lambda.$$

Derive a few specific sentences to convince yourself that this works. ■

The previous examples are fairly easy ones, so rigorous arguments may seem superfluous. But often it is not so easy to find a grammar for a language described in an informal way or to give an intuitive characterization of the language defined by a grammar. To show that a given language is indeed generated by a certain grammar G, we must be able to show (a) that every $w \in L$ can be derived from S using G and (b) that every string so derived is in L.

Example 1.12

Find a grammar that generates

$$L = \left\{a^n b^{n+1} : n \geq 0\right\}.$$

The idea behind the previous example can be extended to this case. All we need to do is generate an extra b. This can be done with a production $S \to Ab$, with other productions chosen so that A can derive the language in the previous example. Reasoning in this fashion, we get the grammar $G = (\{S, A\}, \{a, b\}, S, P)$, with productions

$$S \to Ab,$$
$$A \to aAb,$$
$$A \to \lambda.$$

Derive a few specific sentences to convince yourself that this works. ■

The previous examples are fairly easy ones, so rigorous arguments may seem superfluous. But often it is not so easy to find a grammar for a language described in an informal way or to give an intuitive characterization of the language defined by a grammar. To show that a given language is indeed generated by a certain grammar G, we must be able to show (a) that every $w \in L$ can be derived from S using G and (b) that every string so derived is in L.

Example 1.13

Take $\Sigma = \{a, b\}$, and let $n_a(w)$ and $n_b(w)$ denote the number of a's and b's in the string w, respectively. Then the grammar G with productions

$$S \to SS,$$
$$S \to \lambda,$$
$$S \to aSb,$$
$$S \to bSa$$

generates the language

$$L = \{w : n_a(w) = n_b(w)\}.$$

This claim is not so obvious, and we need to provide convincing arguments.

First, it is clear that every sentential form of G has an equal number of a's and b's, because the only productions that generate the string a, namely $S \to aSb$ and $S \to bSa$, simultaneously generate the string b. Therefore, every element of $L(G)$ is in L. It is a little harder to see that every string in L can be derived with G.

Example 1.13

Take $\Sigma = \{a, b\}$, and let $n_a(w)$ and $n_b(w)$ denote the number of a's and b's in the string w, respectively. Then the grammar G with productions

$$S \to SS,$$
$$S \to \lambda,$$
$$S \to aSb,$$
$$S \to bSa$$

generates the language

$$L = \{w\colon n_a(w) = n_b(w)\}.$$

This claim is not so obvious, and we need to provide convincing arguments.

First, it is clear that every sentential form of G has an equal number of a's and b's, because the only productions that generate the string a, namely $S \to aSb$ and $S \to bSa$, simultaneously generate the string b. Therefore, every element of $L(G)$ is in L. It is a little harder to see that every string in L can be derived with G.

Example 1.13

Take $\Sigma = \{a, b\}$, and let $n_a(w)$ and $n_b(w)$ denote the number of a's and b's in the string w, respectively. Then the grammar G with productions

$$S \to SS,$$
$$S \to \lambda,$$
$$S \to aSb,$$
$$S \to bSa$$

generates the language

$$L = \{w \colon n_a(w) = n_b(w)\}.$$

This claim is not so obvious, and we need to provide convincing arguments.

First, it is clear that every sentential form of G has an equal number of a's and b's, because the only productions that generate the string a, namely $S \to aSb$ and $S \to bSa$, simultaneously generate the string b. Therefore, every element of $L(G)$ is in L. It is a little harder to see that every string in L can be derived with G.

Example 1.13

Take $\Sigma = \{a, b\}$, and let $n_a(w)$ and $n_b(w)$ denote the number of a's and b's in the string w, respectively. Then the grammar G with productions

$$S \to SS,$$
$$S \to \lambda,$$
$$S \to aSb,$$
$$S \to bSa$$

generates the language

$$L = \{w : n_a(w) = n_b(w)\}.$$

This claim is not so obvious, and we need to provide convincing arguments.

First, it is clear that every sentential form of G has an equal number of a's and b's, because the only productions that generate the string a, namely $S \to aSb$ and $S \to bSa$, simultaneously generate the string b. Therefore, every element of $L(G)$ is in L. It is a little harder to see that every string in L can be derived with G.

Example 1.13

Take $\Sigma = \{a, b\}$, and let $n_a(w)$ and $n_b(w)$ denote the number of a's and b's in the string w, respectively. Then the grammar G with productions

$$S \to SS,$$
$$S \to \lambda,$$
$$S \to aSb,$$
$$S \to bSa$$

generates the language

$$L = \{w \colon n_a(w) = n_b(w)\}.$$

This claim is not so obvious, and we need to provide convincing arguments.

First, it is clear that every sentential form of G has an equal number of a's and b's, because the only productions that generate the string a, namely $S \to aSb$ and $S \to bSa$, simultaneously generate the string b. Therefore, every element of $L(G)$ is in L. It is a little harder to see that every string in L can be derived with G.

Example 1.13

Take $\Sigma = \{a, b\}$, and let $n_a(w)$ and $n_b(w)$ denote the number of a's and b's in the string w, respectively. Then the grammar G with productions

$$S \to SS,$$
$$S \to \lambda,$$
$$S \to aSb,$$
$$S \to bSa$$

generates the language

$$L = \{w \colon n_a(w) = n_b(w)\}.$$

This claim is not so obvious, and we need to provide convincing arguments.

First, it is clear that every sentential form of G has an equal number of a's and b's, because the only productions that generate the string a, namely $S \to aSb$ and $S \to bSa$, simultaneously generate the string b. Therefore, every element of $L(G)$ is in L. It is a little harder to see that every string in L can be derived with G.

Example 1.13

Take $\Sigma = \{a, b\}$, and let $n_a(w)$ and $n_b(w)$ denote the number of a's and b's in the string w, respectively. Then the grammar G with productions

$$S \to SS,$$
$$S \to \lambda,$$
$$S \to aSb,$$
$$S \to bSa$$

generates the language

$$L = \{w\colon n_a(w) = n_b(w)\}.$$

This claim is not so obvious, and we need to provide convincing arguments.

First, it is clear that every sentential form of G has an equal number of a's and b's, because the only productions that generate the string a, namely $S \to aSb$ and $S \to bSa$, simultaneously generate the string b. Therefore, every element of $L(G)$ is in L. It is a little harder to see that every string in L can be derived with G.

Example 1.13

Take $\Sigma = \{a, b\}$, and let $n_a(w)$ and $n_b(w)$ denote the number of a's and b's in the string w, respectively. Then the grammar G with productions

$$S \to SS,$$
$$S \to \lambda,$$
$$S \to aSb,$$
$$S \to bSa$$

generates the language

$$L = \{w : n_a(w) = n_b(w)\}.$$

This claim is not so obvious, and we need to provide convincing arguments.

First, it is clear that every sentential form of G has an equal number of a's and b's, because the only productions that generate the string a, namely $S \to aSb$ and $S \to bSa$, simultaneously generate the string b. Therefore, every element of $L(G)$ is in L. It is a little harder to see that every string in L can be derived with G.

Example 1.13

Take $\Sigma = \{a, b\}$, and let $n_a(w)$ and $n_b(w)$ denote the number of a's and b's in the string w, respectively. Then the grammar G with productions

$$S \to SS,$$
$$S \to \lambda,$$
$$S \to aSb,$$
$$S \to bSa$$

generates the language

$$L = \{w\colon n_a(w) = n_b(w)\}.$$

This claim is not so obvious, and we need to provide convincing arguments.

First, it is clear that every sentential form of G has an equal number of a's and b's, because the only productions that generate the string a, namely $S \to aSb$ and $S \to bSa$, simultaneously generate the string b. Therefore, every element of $L(G)$ is in L. It is a little harder to see that every string in L can be derived with G.

Example 1.13

Take $\Sigma = \{a, b\}$, and let $n_a(w)$ and $n_b(w)$ denote the number of a's and b's in the string w, respectively. Then the grammar G with productions

$$S \to SS,$$
$$S \to \lambda,$$
$$S \to aSb,$$
$$S \to bSa$$

generates the language

$$L = \{w\colon n_a(w) = n_b(w)\}.$$

This claim is not so obvious, and we need to provide convincing arguments.

First, it is clear that every sentential form of G has an equal number of a's and b's, because the only productions that generate the string a, namely $S \to aSb$ and $S \to bSa$, simultaneously generate the string b. Therefore, every element of $L(G)$ is in L. It is a little harder to see that every string in L can be derived with G.

Example 1.13

Take $\Sigma = \{a, b\}$, and let $n_a(w)$ and $n_b(w)$ denote the number of a's and b's in the string w, respectively. Then the grammar G with productions

$$S \to SS,$$
$$S \to \lambda,$$
$$S \to aSb,$$
$$S \to bSa$$

generates the language

$$L = \{w\colon n_a(w) = n_b(w)\}.$$

This claim is not so obvious, and we need to provide convincing arguments.

First, it is clear that every sentential form of G has an equal number of a's and b's, because the only productions that generate the string a, namely $S \to aSb$ and $S \to bSa$, simultaneously generate the string b. Therefore, every element of $L(G)$ is in L. It is a little harder to see that every string in L can be derived with G.

Example 1.13

Take $\Sigma = \{a, b\}$, and let $n_a(w)$ and $n_b(w)$ denote the number of a's and b's in the string w, respectively. Then the grammar G with productions

$$S \to SS,$$
$$S \to \lambda,$$
$$S \to aSb,$$
$$S \to bSa$$

generates the language

$$L = \{w \colon n_a(w) = n_b(w)\}.$$

This claim is not so obvious, and we need to provide convincing arguments.

First, it is clear that every sentential form of G has an equal number of a's and b's, because the only productions that generate the string a, namely $S \to aSb$ and $S \to bSa$, simultaneously generate the string b. Therefore, every element of $L(G)$ is in L. It is a little harder to see that every string in L can be derived with G.

Example 1.13

Take $\Sigma = \{a, b\}$, and let $n_a(w)$ and $n_b(w)$ denote the number of a's and b's in the string w, respectively. Then the grammar G with productions

$$S \to SS,$$
$$S \to \lambda,$$
$$S \to aSb,$$
$$S \to bSa$$

generates the language

$$L = \{w \colon n_a(w) = n_b(w)\}.$$

This claim is not so obvious, and we need to provide convincing arguments.

First, it is clear that every sentential form of G has an equal number of a's and b's, because the only productions that generate the string a, namely $S \to aSb$ and $S \to bSa$, simultaneously generate the string b. Therefore, every element of $L(G)$ is in L. It is a little harder to see that every string in L can be derived with G.

Example 1.13

Take $\Sigma = \{a, b\}$, and let $n_a(w)$ and $n_b(w)$ denote the number of a's and b's in the string w, respectively. Then the grammar G with productions

$$S \to SS,$$
$$S \to \lambda,$$
$$S \to aSb,$$
$$S \to bSa$$

generates the language

$$L = \{w \colon n_a(w) = n_b(w)\}.$$

This claim is not so obvious, and we need to provide convincing arguments.

First, it is clear that every sentential form of G has an equal number of a's and b's, because the only productions that generate the string a, namely $S \to aSb$ and $S \to bSa$, simultaneously generate the string b. Therefore, every element of $L(G)$ is in L. It is a little harder to see that every string in L can be derived with G.

Example 1.13

Take $\Sigma = \{a, b\}$, and let $n_a(w)$ and $n_b(w)$ denote the number of a's and b's in the string w, respectively. Then the grammar G with productions

$$S \to SS,$$
$$S \to \lambda,$$
$$S \to aSb,$$
$$S \to bSa$$

generates the language

$$L = \{w\colon n_a(w) = n_b(w)\}.$$

This claim is not so obvious, and we need to provide convincing arguments.

First, it is clear that every sentential form of G has an equal number of a's and b's, because the only productions that generate the string a, namely $S \to aSb$ and $S \to bSa$, simultaneously generate the string b. Therefore, every element of $L(G)$ is in L. It is a little harder to see that every string in L can be derived with G.

Example 1.13 (continuation)

Let us begin by looking at the problem in outline, considering the various forms $w \in L$ can have. Suppose w starts with a and ends with b. Then it has the form

$$w = aw_1b,$$

where w_1 is also in L. We can think of this case as being derived starting with

$$S \Rightarrow aSb$$

if S does indeed derive any string in L. A similar argument can be made if w starts with b and ends with a. But this does not take care of all cases, because a string in L can begin and end with the same symbol. If we write down a string of this type, say $aabbba$, we see that it can be considered as the concatenation of two shorter strings $aabb$ and ba, both of which are in L. Is this true in general? To show that this is indeed so, we can use the following argument: Suppose that, starting at the left end of the string, we count $+1$ for the string a and -1 for the string b. If a string w starts and ends with a, then the count will be $+1$ after the leftmost symbol and -1 immediately before the rightmost one. Therefore, the count has to go through zero somewhere in the middle of the string, indicating that such a string must have the form

$$w = w_1w_2,$$

where both w_1 and w_2 are in L. This case can be taken care of by the production $S \rightarrow SS$.

Example 1.13 (continuation)

Let us begin by looking at the problem in outline, considering the various forms $w \in L$ can have. Suppose w starts with a and ends with b. Then it has the form

$$w = aw_1b,$$

where w_1 is also in L. We can think of this case as being derived starting with

$$S \Rightarrow aSb$$

if S does indeed derive any string in L. A similar argument can be made if w starts with b and ends with a. But this does not take care of all cases, because a string in L can begin and end with the same symbol. If we write down a string of this type, say $aabbba$, we see that it can be considered as the concatenation of two shorter strings $aabb$ and ba, both of which are in L. Is this true in general? To show that this is indeed so, we can use the following argument: Suppose that, starting at the left end of the string, we count $+1$ for the string a and -1 for the string b. If a string w starts and ends with a, then the count will be $+1$ after the leftmost symbol and -1 immediately before the rightmost one. Therefore, the count has to go through zero somewhere in the middle of the string, indicating that such a string must have the form

$$w = w_1w_2,$$

where both w_1 and w_2 are in L. This case can be taken care of by the production $S \to SS$.

Example 1.13 (continuation)

Let us begin by looking at the problem in outline, considering the various forms $w \in L$ can have. Suppose w starts with a and ends with b. Then it has the form

$$w = aw_1b,$$

where w_1 is also in L. We can think of this case as being derived starting with

$$S \Rightarrow aSb$$

if S does indeed derive any string in L. A similar argument can be made if w starts with b and ends with a. But this does not take care of all cases, because a string in L can begin and end with the same symbol. If we write down a string of this type, say $aabbba$, we see that it can be considered as the concatenation of two shorter strings $aabb$ and ba, both of which are in L. Is this true in general? To show that this is indeed so, we can use the following argument: Suppose that, starting at the left end of the string, we count $+1$ for the string a and -1 for the string b. If a string w starts and ends with a, then the count will be $+1$ after the leftmost symbol and -1 immediately before the rightmost one. Therefore, the count has to go through zero somewhere in the middle of the string, indicating that such a string must have the form

$$w = w_1w_2,$$

where both w_1 and w_2 are in L. This case can be taken care of by the production $S \to SS$.

Example 1.13 (continuation)

Let us begin by looking at the problem in outline, considering the various forms $w \in L$ can have. Suppose w starts with a and ends with b. Then it has the form

$$w = aw_1b,$$

where w_1 is also in L. We can think of this case as being derived starting with

$$S \Rightarrow aSb$$

if S does indeed derive any string in L. A similar argument can be made if w starts with b and ends with a. But this does not take care of all cases, because a string in L can begin and end with the same symbol. If we write down a string of this type, say $aabbba$, we see that it can be considered as the concatenation of two shorter strings $aabb$ and ba, both of which are in L. Is this true in general? To show that this is indeed so, we can use the following argument: Suppose that, starting at the left end of the string, we count $+1$ for the string a and -1 for the string b. If a string w starts and ends with a, then the count will be $+1$ after the leftmost symbol and -1 immediately before the rightmost one. Therefore, the count has to go through zero somewhere in the middle of the string, indicating that such a string must have the form

$$w = w_1w_2,$$

where both w_1 and w_2 are in L. This case can be taken care of by the production $S \rightarrow SS$.

Example 1.13 (continuation)

Let us begin by looking at the problem in outline, considering the various forms $w \in L$ can have. Suppose w starts with a and ends with b. Then it has the form

$$w = aw_1b,$$

where w_1 is also in L. We can think of this case as being derived starting with

$$S \Rightarrow aSb$$

if S does indeed derive any string in L. A similar argument can be made if w starts with b and ends with a. But this does not take care of all cases, because a string in L can begin and end with the same symbol. If we write down a string of this type, say $aabbba$, we see that it can be considered as the concatenation of two shorter strings $aabb$ and ba, both of which are in L. Is this true in general? To show that this is indeed so, we can use the following argument: Suppose that, starting at the left end of the string, we count $+1$ for the string a and -1 for the string b. If a string w starts and ends with a, then the count will be $+1$ after the leftmost symbol and -1 immediately before the rightmost one. Therefore, the count has to go through zero somewhere in the middle of the string, indicating that such a string must have the form

$$w = w_1w_2,$$

where both w_1 and w_2 are in L. This case can be taken care of by the production $S \to SS$.

Example 1.13 (continuation)

Let us begin by looking at the problem in outline, considering the various forms $w \in L$ can have. Suppose w starts with a and ends with b. Then it has the form

$$w = aw_1b,$$

where w_1 is also in L. We can think of this case as being derived starting with

$$S \Rightarrow aSb$$

if S does indeed derive any string in L. A similar argument can be made if w starts with b and ends with a. But this does not take care of all cases, because a string in L can begin and end with the same symbol. If we write down a string of this type, say $aabbba$, we see that it can be considered as the concatenation of two shorter strings $aabb$ and ba, both of which are in L. Is this true in general? To show that this is indeed so, we can use the following argument: Suppose that, starting at the left end of the string, we count $+1$ for the string a and -1 for the string b. If a string w starts and ends with a, then the count will be $+1$ after the leftmost symbol and -1 immediately before the rightmost one. Therefore, the count has to go through zero somewhere in the middle of the string, indicating that such a string must have the form

$$w = w_1w_2,$$

where both w_1 and w_2 are in L. This case can be taken care of by the production $S \to SS$.

Example 1.13 (continuation)

Let us begin by looking at the problem in outline, considering the various forms $w \in L$ can have. Suppose w starts with a and ends with b. Then it has the form

$$w = aw_1b,$$

where w_1 is also in L. We can think of this case as being derived starting with

$$S \Rightarrow aSb$$

if S does indeed derive any string in L. A similar argument can be made if w starts with b and ends with a. But this does not take care of all cases, because a string in L can begin and end with the same symbol. If we write down a string of this type, say $aabbba$, we see that it can be considered as the concatenation of two shorter strings $aabb$ and ba, both of which are in L. Is this true in general? To show that this is indeed so, we can use the following argument: Suppose that, starting at the left end of the string, we count $+1$ for the string a and -1 for the string b. If a string w starts and ends with a, then the count will be $+1$ after the leftmost symbol and -1 immediately before the rightmost one. Therefore, the count has to go through zero somewhere in the middle of the string, indicating that such a string must have the form

$$w = w_1w_2,$$

where both w_1 and w_2 are in L. This case can be taken care of by the production $S \to SS$.

Example 1.13 (continuation)

Let us begin by looking at the problem in outline, considering the various forms $w \in L$ can have. Suppose w starts with a and ends with b. Then it has the form

$$w = aw_1b,$$

where w_1 is also in L. We can think of this case as being derived starting with

$$S \Rightarrow aSb$$

if S does indeed derive any string in L. A similar argument can be made if w starts with b and ends with a. But this does not take care of all cases, because a string in L can begin and end with the same symbol. If we write down a string of this type, say $aabbba$, we see that it can be considered as the concatenation of two shorter strings $aabb$ and ba, both of which are in L. Is this true in general? To show that this is indeed so, we can use the following argument: Suppose that, starting at the left end of the string, we count $+1$ for the string a and -1 for the string b. If a string w starts and ends with a, then the count will be $+1$ after the leftmost symbol and -1 immediately before the rightmost one. Therefore, the count has to go through zero somewhere in the middle of the string, indicating that such a string must have the form

$$w = w_1w_2,$$

where both w_1 and w_2 are in L. This case can be taken care of by the production $S \to SS$.

Example 1.13 (continuation)

Let us begin by looking at the problem in outline, considering the various forms $w \in L$ can have. Suppose w starts with a and ends with b. Then it has the form

$$w = aw_1b,$$

where w_1 is also in L. We can think of this case as being derived starting with

$$S \Rightarrow aSb$$

if S does indeed derive any string in L. A similar argument can be made if w starts with b and ends with a. But this does not take care of all cases, because a string in L can begin and end with the same symbol. If we write down a string of this type, say $aabbba$, we see that it can be considered as the concatenation of two shorter strings $aabb$ and ba, both of which are in L. Is this true in general? To show that this is indeed so, we can use the following argument: Suppose that, starting at the left end of the string, we count $+1$ for the string a and -1 for the string b. If a string w starts and ends with a, then the count will be $+1$ after the leftmost symbol and -1 immediately before the rightmost one. Therefore, the count has to go through zero somewhere in the middle of the string, indicating that such a string must have the form

$$w = w_1w_2,$$

where both w_1 and w_2 are in L. This case can be taken care of by the production $S \to SS$.

Example 1.13 (continuation)

Let us begin by looking at the problem in outline, considering the various forms $w \in L$ can have. Suppose w starts with a and ends with b. Then it has the form

$$w = aw_1b,$$

where w_1 is also in L. We can think of this case as being derived starting with

$$S \Rightarrow aSb$$

if S does indeed derive any string in L. A similar argument can be made if w starts with b and ends with a. But this does not take care of all cases, because a string in L can begin and end with the same symbol. If we write down a string of this type, say $aabbba$, we see that it can be considered as the concatenation of two shorter strings $aabb$ and ba, both of which are in L. Is this true in general? To show that this is indeed so, we can use the following argument: Suppose that, starting at the left end of the string, we count $+1$ for the string a and -1 for the string b. If a string w starts and ends with a, then the count will be $+1$ after the leftmost symbol and -1 immediately before the rightmost one. Therefore, the count has to go through zero somewhere in the middle of the string, indicating that such a string must have the form

$$w = w_1w_2,$$

where both w_1 and w_2 are in L. This case can be taken care of by the production $S \to SS$.

Example 1.13 (continuation)

Let us begin by looking at the problem in outline, considering the various forms $w \in L$ can have. Suppose w starts with a and ends with b. Then it has the form

$$w = aw_1b,$$

where w_1 is also in L. We can think of this case as being derived starting with

$$S \Rightarrow aSb$$

if S does indeed derive any string in L. A similar argument can be made if w starts with b and ends with a. But this does not take care of all cases, because a string in L can begin and end with the same symbol. If we write down a string of this type, say $aabbba$, we see that it can be considered as the concatenation of two shorter strings $aabb$ and ba, both of which are in L. Is this true in general? To show that this is indeed so, we can use the following argument: Suppose that, starting at the left end of the string, we count $+1$ for the string a and -1 for the string b. If a string w starts and ends with a, then the count will be $+1$ after the leftmost symbol and -1 immediately before the rightmost one. Therefore, the count has to go through zero somewhere in the middle of the string, indicating that such a string must have the form

$$w = w_1w_2,$$

where both w_1 and w_2 are in L. This case can be taken care of by the production $S \rightarrow SS$.

Example 1.13 (continuation)

Let us begin by looking at the problem in outline, considering the various forms $w \in L$ can have. Suppose w starts with a and ends with b. Then it has the form

$$w = aw_1b,$$

where w_1 is also in L. We can think of this case as being derived starting with

$$S \Rightarrow aSb$$

if S does indeed derive any string in L. A similar argument can be made if w starts with b and ends with a. But this does not take care of all cases, because a string in L can begin and end with the same symbol. If we write down a string of this type, say $aabbba$, we see that it can be considered as the concatenation of two shorter strings $aabb$ and ba, both of which are in L. Is this true in general? To show that this is indeed so, we can use the following argument: Suppose that, starting at the left end of the string, we count $+1$ for the string a and -1 for the string b. If a string w starts and ends with a, then the count will be $+1$ after the leftmost symbol and -1 immediately before the rightmost one. Therefore, the count has to go through zero somewhere in the middle of the string, indicating that such a string must have the form

$$w = w_1w_2,$$

where both w_1 and w_2 are in L. This case can be taken care of by the production $S \to SS$.

Example 1.13 (continuation)

Let us begin by looking at the problem in outline, considering the various forms $w \in L$ can have. Suppose w starts with a and ends with b. Then it has the form

$$w = aw_1b,$$

where w_1 is also in L. We can think of this case as being derived starting with

$$S \Rightarrow aSb$$

if S does indeed derive any string in L. A similar argument can be made if w starts with b and ends with a. But this does not take care of all cases, because a string in L can begin and end with the same symbol. If we write down a string of this type, say $aabbba$, we see that it can be considered as the concatenation of two shorter strings $aabb$ and ba, both of which are in L. Is this true in general? To show that this is indeed so, we can use the following argument: Suppose that, starting at the left end of the string, we count $+1$ for the string a and -1 for the string b. If a string w starts and ends with a, then the count will be $+1$ after the leftmost symbol and -1 immediately before the rightmost one. Therefore, the count has to go through zero somewhere in the middle of the string, indicating that such a string must have the form

$$w = w_1w_2,$$

where both w_1 and w_2 are in L. This case can be taken care of by the production $S \to SS$.

Example 1.13 (continuation)

Let us begin by looking at the problem in outline, considering the various forms $w \in L$ can have. Suppose w starts with a and ends with b. Then it has the form

$$w = aw_1b,$$

where w_1 is also in L. We can think of this case as being derived starting with

$$S \Rightarrow aSb$$

if S does indeed derive any string in L. A similar argument can be made if w starts with b and ends with a. But this does not take care of all cases, because a string in L can begin and end with the same symbol. If we write down a string of this type, say $aabbba$, we see that it can be considered as the concatenation of two shorter strings $aabb$ and ba, both of which are in L. Is this true in general? To show that this is indeed so, we can use the following argument: Suppose that, starting at the left end of the string, we count $+1$ for the string a and -1 for the string b. If a string w starts and ends with a, then the count will be $+1$ after the leftmost symbol and -1 immediately before the rightmost one. Therefore, the count has to go through zero somewhere in the middle of the string, indicating that such a string must have the form

$$w = w_1w_2,$$

where both w_1 and w_2 are in L. This case can be taken care of by the production $S \to SS$.

Example 1.13 (continuation)

Let us begin by looking at the problem in outline, considering the various forms $w \in L$ can have. Suppose w starts with a and ends with b. Then it has the form

$$w = aw_1b,$$

where w_1 is also in L. We can think of this case as being derived starting with

$$S \Rightarrow aSb$$

if S does indeed derive any string in L. A similar argument can be made if w starts with b and ends with a. But this does not take care of all cases, because a string in L can begin and end with the same symbol. If we write down a string of this type, say $aabbba$, we see that it can be considered as the concatenation of two shorter strings $aabb$ and ba, both of which are in L. Is this true in general? To show that this is indeed so, we can use the following argument: Suppose that, starting at the left end of the string, we count $+1$ for the string a and -1 for the string b. If a string w starts and ends with a, then the count will be $+1$ after the leftmost symbol and -1 immediately before the rightmost one. Therefore, the count has to go through zero somewhere in the middle of the string, indicating that such a string must have the form

$$w = w_1w_2,$$

where both w_1 and w_2 are in L. This case can be taken care of by the production $S \to SS$.

Example 1.13 (continuation)

Let us begin by looking at the problem in outline, considering the various forms $w \in L$ can have. Suppose w starts with a and ends with b. Then it has the form

$$w = aw_1b,$$

where w_1 is also in L. We can think of this case as being derived starting with

$$S \Rightarrow aSb$$

if S does indeed derive any string in L. A similar argument can be made if w starts with b and ends with a. But this does not take care of all cases, because a string in L can begin and end with the same symbol. If we write down a string of this type, say $aabbba$, we see that it can be considered as the concatenation of two shorter strings $aabb$ and ba, both of which are in L. Is this true in general? To show that this is indeed so, we can use the following argument: Suppose that, starting at the left end of the string, we count $+1$ for the string a and -1 for the string b. If a string w starts and ends with a, then the count will be $+1$ after the leftmost symbol and -1 immediately before the rightmost one. Therefore, the count has to go through zero somewhere in the middle of the string, indicating that such a string must have the form

$$w = w_1w_2,$$

where both w_1 and w_2 are in L. This case can be taken care of by the production $S \to SS$.

Example 1.13 (continuation)

Let us begin by looking at the problem in outline, considering the various forms $w \in L$ can have. Suppose w starts with a and ends with b. Then it has the form

$$w = aw_1b,$$

where w_1 is also in L. We can think of this case as being derived starting with

$$S \Rightarrow aSb$$

if S does indeed derive any string in L. A similar argument can be made if w starts with b and ends with a. But this does not take care of all cases, because a string in L can begin and end with the same symbol. If we write down a string of this type, say $aabbba$, we see that it can be considered as the concatenation of two shorter strings $aabb$ and ba, both of which are in L. Is this true in general? To show that this is indeed so, we can use the following argument: Suppose that, starting at the left end of the string, we count $+1$ for the string a and -1 for the string b. If a string w starts and ends with a, then the count will be $+1$ after the leftmost symbol and -1 immediately before the rightmost one. Therefore, the count has to go through zero somewhere in the middle of the string, indicating that such a string must have the form

$$w = w_1w_2,$$

where both w_1 and w_2 are in L. This case can be taken care of by the production $S \to SS$.

Example 1.13 (continuation)

Let us begin by looking at the problem in outline, considering the various forms $w \in L$ can have. Suppose w starts with a and ends with b. Then it has the form

$$w = aw_1b,$$

where w_1 is also in L. We can think of this case as being derived starting with

$$S \Rightarrow aSb$$

if S does indeed derive any string in L. A similar argument can be made if w starts with b and ends with a. But this does not take care of all cases, because a string in L can begin and end with the same symbol. If we write down a string of this type, say $aabbba$, we see that it can be considered as the concatenation of two shorter strings $aabb$ and ba, both of which are in L. Is this true in general? To show that this is indeed so, we can use the following argument: Suppose that, starting at the left end of the string, we count $+1$ for the string a and -1 for the string b. If a string w starts and ends with a, then the count will be $+1$ after the leftmost symbol and -1 immediately before the rightmost one. Therefore, the count has to go through zero somewhere in the middle of the string, indicating that such a string must have the form

$$w = w_1w_2,$$

where both w_1 and w_2 are in L. This case can be taken care of by the production $S \to SS$.

Example 1.13 (continuation)

Let us begin by looking at the problem in outline, considering the various forms $w \in L$ can have. Suppose w starts with a and ends with b. Then it has the form

$$w = aw_1b,$$

where w_1 is also in L. We can think of this case as being derived starting with

$$S \Rightarrow aSb$$

if S does indeed derive any string in L. A similar argument can be made if w starts with b and ends with a. But this does not take care of all cases, because a string in L can begin and end with the same symbol. If we write down a string of this type, say $aabbba$, we see that it can be considered as the concatenation of two shorter strings $aabb$ and ba, both of which are in L. Is this true in general? To show that this is indeed so, we can use the following argument: Suppose that, starting at the left end of the string, we count $+1$ for the string a and -1 for the string b. If a string w starts and ends with a, then the count will be $+1$ after the leftmost symbol and -1 immediately before the rightmost one. Therefore, the count has to go through zero somewhere in the middle of the string, indicating that such a string must have the form

$$w = w_1w_2,$$

where both w_1 and w_2 are in L. This case can be taken care of by the production $S \to SS$.

Example 1.13 (continuation)

Let us begin by looking at the problem in outline, considering the various forms $w \in L$ can have. Suppose w starts with a and ends with b. Then it has the form

$$w = aw_1b,$$

where w_1 is also in L. We can think of this case as being derived starting with

$$S \Rightarrow aSb$$

if S does indeed derive any string in L. A similar argument can be made if w starts with b and ends with a. But this does not take care of all cases, because a string in L can begin and end with the same symbol. If we write down a string of this type, say $aabbba$, we see that it can be considered as the concatenation of two shorter strings $aabb$ and ba, both of which are in L. Is this true in general? To show that this is indeed so, we can use the following argument: Suppose that, starting at the left end of the string, we count $+1$ for the string a and -1 for the string b. If a string w starts and ends with a, then the count will be $+1$ after the leftmost symbol and -1 immediately before the rightmost one. Therefore, the count has to go through zero somewhere in the middle of the string, indicating that such a string must have the form

$$w = w_1w_2,$$

where both w_1 and w_2 are in L. This case can be taken care of by the production $S \to SS$.

Example 1.13 (continuation)

Let us begin by looking at the problem in outline, considering the various forms $w \in L$ can have. Suppose w starts with a and ends with b. Then it has the form

$$w = aw_1b,$$

where w_1 is also in L. We can think of this case as being derived starting with

$$S \Rightarrow aSb$$

if S does indeed derive any string in L. A similar argument can be made if w starts with b and ends with a. But this does not take care of all cases, because a string in L can begin and end with the same symbol. If we write down a string of this type, say $aabbba$, we see that it can be considered as the concatenation of two shorter strings $aabb$ and ba, both of which are in L. Is this true in general? To show that this is indeed so, we can use the following argument: Suppose that, starting at the left end of the string, we count $+1$ for the string a and -1 for the string b. If a string w starts and ends with a, then the count will be $+1$ after the leftmost symbol and -1 immediately before the rightmost one. Therefore, the count has to go through zero somewhere in the middle of the string, indicating that such a string must have the form

$$w = w_1w_2,$$

where both w_1 and w_2 are in L. This case can be taken care of by the production $S \to SS$.

Example 1.13 (continuation)

Let us begin by looking at the problem in outline, considering the various forms $w \in L$ can have. Suppose w starts with a and ends with b. Then it has the form

$$w = aw_1b,$$

where w_1 is also in L. We can think of this case as being derived starting with

$$S \Rightarrow aSb$$

if S does indeed derive any string in L. A similar argument can be made if w starts with b and ends with a. But this does not take care of all cases, because a string in L can begin and end with the same symbol. If we write down a string of this type, say $aabbba$, we see that it can be considered as the concatenation of two shorter strings $aabb$ and ba, both of which are in L. Is this true in general? To show that this is indeed so, we can use the following argument: Suppose that, starting at the left end of the string, we count $+1$ for the string a and -1 for the string b. If a string w starts and ends with a, then the count will be $+1$ after the leftmost symbol and -1 immediately before the rightmost one. Therefore, the count has to go through zero somewhere in the middle of the string, indicating that such a string must have the form

$$w = w_1w_2,$$

where both w_1 and w_2 are in L. This case can be taken care of by the production $S \rightarrow SS$.

Example 1.13 (continuation)

Let us begin by looking at the problem in outline, considering the various forms $w \in L$ can have. Suppose w starts with a and ends with b. Then it has the form

$$w = aw_1b,$$

where w_1 is also in L. We can think of this case as being derived starting with

$$S \Rightarrow aSb$$

if S does indeed derive any string in L. A similar argument can be made if w starts with b and ends with a. But this does not take care of all cases, because a string in L can begin and end with the same symbol. If we write down a string of this type, say $aabbba$, we see that it can be considered as the concatenation of two shorter strings $aabb$ and ba, both of which are in L. Is this true in general? To show that this is indeed so, we can use the following argument: Suppose that, starting at the left end of the string, we count $+1$ for the string a and -1 for the string b. If a string w starts and ends with a, then the count will be $+1$ after the leftmost symbol and -1 immediately before the rightmost one. Therefore, the count has to go through zero somewhere in the middle of the string, indicating that such a string must have the form

$$w = w_1w_2,$$

where both w_1 and w_2 are in L. This case can be taken care of by the production $S \rightarrow SS$.

Example 1.13 (continuation)

Let us begin by looking at the problem in outline, considering the various forms $w \in L$ can have. Suppose w starts with a and ends with b. Then it has the form

$$w = aw_1b,$$

where w_1 is also in L. We can think of this case as being derived starting with

$$S \Rightarrow aSb$$

if S does indeed derive any string in L. A similar argument can be made if w starts with b and ends with a. But this does not take care of all cases, because a string in L can begin and end with the same symbol. If we write down a string of this type, say $aabbba$, we see that it can be considered as the concatenation of two shorter strings $aabb$ and ba, both of which are in L. Is this true in general? To show that this is indeed so, we can use the following argument: Suppose that, starting at the left end of the string, we count $+1$ for the string a and -1 for the string b. If a string w starts and ends with a, then the count will be $+1$ after the leftmost symbol and -1 immediately before the rightmost one. Therefore, the count has to go through zero somewhere in the middle of the string, indicating that such a string must have the form

$$w = w_1w_2,$$

where both w_1 and w_2 are in L. This case can be taken care of by the production $S \to SS$.

Example 1.13 (continuation)

Let us begin by looking at the problem in outline, considering the various forms $w \in L$ can have. Suppose w starts with a and ends with b. Then it has the form

$$w = aw_1b,$$

where w_1 is also in L. We can think of this case as being derived starting with

$$S \Rightarrow aSb$$

if S does indeed derive any string in L. A similar argument can be made if w starts with b and ends with a. But this does not take care of all cases, because a string in L can begin and end with the same symbol. If we write down a string of this type, say $aabbba$, we see that it can be considered as the concatenation of two shorter strings $aabb$ and ba, both of which are in L. Is this true in general? To show that this is indeed so, we can use the following argument: Suppose that, starting at the left end of the string, we count $+1$ for the string a and -1 for the string b. If a string w starts and ends with a, then the count will be $+1$ after the leftmost symbol and -1 immediately before the rightmost one. Therefore, the count has to go through zero somewhere in the middle of the string, indicating that such a string must have the form

$$w = w_1w_2,$$

where both w_1 and w_2 are in L. This case can be taken care of by the production $S \to SS$.

Example 1.13 (continuation)

Let us begin by looking at the problem in outline, considering the various forms $w \in L$ can have. Suppose w starts with a and ends with b. Then it has the form

$$w = aw_1b,$$

where w_1 is also in L. We can think of this case as being derived starting with

$$S \Rightarrow aSb$$

if S does indeed derive any string in L. A similar argument can be made if w starts with b and ends with a. But this does not take care of all cases, because a string in L can begin and end with the same symbol. If we write down a string of this type, say $aabbba$, we see that it can be considered as the concatenation of two shorter strings $aabb$ and ba, both of which are in L. Is this true in general? To show that this is indeed so, we can use the following argument: Suppose that, starting at the left end of the string, we count $+1$ for the string a and -1 for the string b. If a string w starts and ends with a, then the count will be $+1$ after the leftmost symbol and -1 immediately before the rightmost one. Therefore, the count has to go through zero somewhere in the middle of the string, indicating that such a string must have the form

$$w = w_1w_2,$$

where both w_1 and w_2 are in L. This case can be taken care of by the production $S \to SS$.

Example 1.13 (continuation)

Let us begin by looking at the problem in outline, considering the various forms $w \in L$ can have. Suppose w starts with a and ends with b. Then it has the form

$$w = aw_1b,$$

where w_1 is also in L. We can think of this case as being derived starting with

$$S \Rightarrow aSb$$

if S does indeed derive any string in L. A similar argument can be made if w starts with b and ends with a. But this does not take care of all cases, because a string in L can begin and end with the same symbol. If we write down a string of this type, say $aabbba$, we see that it can be considered as the concatenation of two shorter strings $aabb$ and ba, both of which are in L. Is this true in general? To show that this is indeed so, we can use the following argument: Suppose that, starting at the left end of the string, we count $+1$ for the string a and -1 for the string b. If a string w starts and ends with a, then the count will be $+1$ after the leftmost symbol and -1 immediately before the rightmost one. Therefore, the count has to go through zero somewhere in the middle of the string, indicating that such a string must have the form

$$w = w_1w_2,$$

where both w_1 and w_2 are in L. This case can be taken care of by the production $S \to SS$.

Example 1.13 (continuation)

Let us begin by looking at the problem in outline, considering the various forms $w \in L$ can have. Suppose w starts with a and ends with b. Then it has the form

$$w = aw_1b,$$

where w_1 is also in L. We can think of this case as being derived starting with

$$S \Rightarrow aSb$$

if S does indeed derive any string in L. A similar argument can be made if w starts with b and ends with a. But this does not take care of all cases, because a string in L can begin and end with the same symbol. If we write down a string of this type, say $aabbba$, we see that it can be considered as the concatenation of two shorter strings $aabb$ and ba, both of which are in L. Is this true in general? To show that this is indeed so, we can use the following argument: Suppose that, starting at the left end of the string, we count $+1$ for the string a and -1 for the string b. If a string w starts and ends with a, then the count will be $+1$ after the leftmost symbol and -1 immediately before the rightmost one. Therefore, the count has to go through zero somewhere in the middle of the string, indicating that such a string must have the form

$$w = w_1w_2,$$

where both w_1 and w_2 are in L. This case can be taken care of by the production $S \to SS$.

Example 1.13 (continuation)

Once we see the argument intuitively, we are ready to proceed more rigorously. Again we use induction. Assume that all $w \in L$ with $|w| \le 2n$ can be derived with G. Take any $w \in L$ of length $2n+2$. If $w = aw_1b$, then w_1 is in L, and $|w_1| = 2n$. Therefore, by assumption we have that

$$S \stackrel{*}{\Rightarrow} w_1.$$

Then

$$S \Rightarrow aSb \stackrel{*}{\Rightarrow} aw_1b = w$$

is possible, and w can be derived with G. Obviously, similar arguments can be made if $w = bw_1a$.

If w is not of this form, that is, if it starts and ends with the same symbol, then the counting argument tells us that it must have the form $w = w_1w_2$, with w_1 and w_2 both in L and of length less than or equal to $2n$. Hence again we see that

$$S \Rightarrow SS \stackrel{*}{\Rightarrow} w_1S \stackrel{*}{\Rightarrow} w_1w_2 = w$$

is possible.

Since the inductive assumption is clearly satisfied for $n = 1$, we have a basis, and the claim is true for all n, completing our argument. ∎

Example 1.13 (continuation)

Once we see the argument intuitively, we are ready to proceed more rigorously. Again we use induction. Assume that all $w \in L$ with $|w| \leq 2n$ can be derived with G. Take any $w \in L$ of length $2n + 2$. If $w = aw_1b$, then w_1 is in L, and $|w_1| = 2n$. Therefore, by assumption we have that

$$S \stackrel{*}{\Rightarrow} w_1.$$

Then

$$S \Rightarrow aSb \stackrel{*}{\Rightarrow} aw_1b = w$$

is possible, and w can be derived with G. Obviously, similar arguments can be made if $w = bw_1a$.

If w is not of this form, that is, if it starts and ends with the same symbol, then the counting argument tells us that it must have the form $w = w_1w_2$, with w_1 and w_2 both in L and of length less than or equal to $2n$. Hence again we see that

$$S \Rightarrow SS \stackrel{*}{\Rightarrow} w_1S \stackrel{*}{\Rightarrow} w_1w_2 = w$$

is possible.

Since the inductive assumption is clearly satisfied for $n = 1$, we have a basis, and the claim is true for all n, completing our argument. ■

Example 1.13 (continuation)

Once we see the argument intuitively, we are ready to proceed more rigorously. Again we use induction. Assume that all $w \in L$ with $|w| \leq 2n$ can be derived with G. Take any $w \in L$ of length $2n+2$. If $w = aw_1b$, then w_1 is in L, and $|w_1| = 2n$. Therefore, by assumption we have that

$$S \stackrel{*}{\Rightarrow} w_1.$$

Then

$$S \Rightarrow aSb \stackrel{*}{\Rightarrow} aw_1b = w$$

is possible, and w can be derived with G. Obviously, similar arguments can be made if $w = bw_1a$.

If w is not of this form, that is, if it starts and ends with the same symbol, then the counting argument tells us that it must have the form $w = w_1w_2$, with w_1 and w_2 both in L and of length less than or equal to $2n$. Hence again we see that

$$S \Rightarrow SS \stackrel{*}{\Rightarrow} w_1S \stackrel{*}{\Rightarrow} w_1w_2 = w$$

is possible.

Since the inductive assumption is clearly satisfied for $n = 1$, we have a basis, and the claim is true for all n, completing our argument. ∎

Example 1.13 (continuation)

Once we see the argument intuitively, we are ready to proceed more rigorously. Again we use induction. Assume that all $w \in L$ with $|w| \leq 2n$ can be derived with G. Take any $w \in L$ of length $2n + 2$. If $w = aw_1b$, then w_1 is in L, and $|w_1| = 2n$. Therefore, by assumption we have that

$$S \overset{*}{\Rightarrow} w_1.$$

Then

$$S \Rightarrow aSb \overset{*}{\Rightarrow} aw_1b = w$$

is possible, and w can be derived with G. Obviously, similar arguments can be made if $w = bw_1a$.

If w is not of this form, that is, if it starts and ends with the same symbol, then the counting argument tells us that it must have the form $w = w_1w_2$, with w_1 and w_2 both in L and of length less than or equal to $2n$. Hence again we see that

$$S \Rightarrow SS \overset{*}{\Rightarrow} w_1S \overset{*}{\Rightarrow} w_1w_2 = w$$

is possible.

Since the inductive assumption is clearly satisfied for $n = 1$, we have a basis, and the claim is true for all n, completing our argument. ∎

Example 1.13 (continuation)

Once we see the argument intuitively, we are ready to proceed more rigorously. Again we use induction. Assume that all $w \in L$ with $|w| \leq 2n$ can be derived with G. Take any $w \in L$ of length $2n+2$. If $w = aw_1b$, then w_1 is in L, and $|w_1| = 2n$. Therefore, by assumption we have that

$$S \stackrel{*}{\Rightarrow} w_1.$$

Then

$$S \Rightarrow aSb \stackrel{*}{\Rightarrow} aw_1b = w$$

is possible, and w can be derived with G. Obviously, similar arguments can be made if $w = bw_1a$.

If w is not of this form, that is, if it starts and ends with the same symbol, then the counting argument tells us that it must have the form $w = w_1w_2$, with w_1 and w_2 both in L and of length less than or equal to $2n$. Hence again we see that

$$S \Rightarrow SS \stackrel{*}{\Rightarrow} w_1S \stackrel{*}{\Rightarrow} w_1w_2 = w$$

is possible.

Since the inductive assumption is clearly satisfied for $n = 1$, we have a basis, and the claim is true for all n, completing our argument. ∎

Example 1.13 (continuation)

Once we see the argument intuitively, we are ready to proceed more rigorously. Again we use induction. Assume that all $w \in L$ with $|w| \leq 2n$ can be derived with G. Take any $w \in L$ of length $2n+2$. If $w = aw_1b$, then w_1 is in L, and $|w_1| = 2n$. Therefore, by assumption we have that

$$S \overset{*}{\Rightarrow} w_1.$$

Then

$$S \Rightarrow aSb \overset{*}{\Rightarrow} aw_1b = w$$

is possible, and w can be derived with G. Obviously, similar arguments can be made if $w = bw_1a$.

If w is not of this form, that is, if it starts and ends with the same symbol, then the counting argument tells us that it must have the form $w = w_1w_2$, with w_1 and w_2 both in L and of length less than or equal to $2n$. Hence again we see that

$$S \Rightarrow SS \overset{*}{\Rightarrow} w_1S \overset{*}{\Rightarrow} w_1w_2 = w$$

is possible.

Since the inductive assumption is clearly satisfied for $n = 1$, we have a basis, and the claim is true for all n, completing our argument. ■

Example 1.13 (continuation)

Once we see the argument intuitively, we are ready to proceed more rigorously. Again we use induction. Assume that all $w \in L$ with $|w| \leq 2n$ can be derived with G. Take any $w \in L$ of length $2n+2$. If $w = aw_1b$, then w_1 is in L, and $|w_1| = 2n$. Therefore, by assumption we have that

$$S \stackrel{*}{\Rightarrow} w_1.$$

Then

$$S \Rightarrow aSb \stackrel{*}{\Rightarrow} aw_1b = w$$

is possible, and w can be derived with G. Obviously, similar arguments can be made if $w = bw_1a$.

If w is not of this form, that is, if it starts and ends with the same symbol, then the counting argument tells us that it must have the form $w = w_1w_2$, with w_1 and w_2 both in L and of length less than or equal to $2n$. Hence again we see that

$$S \Rightarrow SS \stackrel{*}{\Rightarrow} w_1S \stackrel{*}{\Rightarrow} w_1w_2 = w$$

is possible.

Since the inductive assumption is clearly satisfied for $n = 1$, we have a basis, and the claim is true for all n, completing our argument. ■

Example 1.13 (continuation)

Once we see the argument intuitively, we are ready to proceed more rigorously. Again we use induction. Assume that all $w \in L$ with $|w| \leq 2n$ can be derived with G. Take any $w \in L$ of length $2n + 2$. If $w = aw_1b$, then w_1 is in L, and $|w_1| = 2n$. Therefore, by assumption we have that

$$S \stackrel{*}{\Rightarrow} w_1.$$

Then

$$S \Rightarrow aSb \stackrel{*}{\Rightarrow} aw_1b = w$$

is possible, and w can be derived with G. Obviously, similar arguments can be made if $w = bw_1a$.

If w is not of this form, that is, if it starts and ends with the same symbol, then the counting argument tells us that it must have the form $w = w_1w_2$, with w_1 and w_2 both in L and of length less than or equal to $2n$. Hence again we see that

$$S \Rightarrow SS \stackrel{*}{\Rightarrow} w_1S \stackrel{*}{\Rightarrow} w_1w_2 = w$$

is possible.

Since the inductive assumption is clearly satisfied for $n = 1$, we have a basis, and the claim is true for all n, completing our argument. ∎

Example 1.13 (continuation)

Once we see the argument intuitively, we are ready to proceed more rigorously. Again we use induction. Assume that all $w \in L$ with $|w| \leq 2n$ can be derived with G. Take any $w \in L$ of length $2n+2$. If $w = aw_1b$, then w_1 is in L, and $|w_1| = 2n$. Therefore, by assumption we have that

$$S \stackrel{*}{\Rightarrow} w_1.$$

Then

$$S \Rightarrow aSb \stackrel{*}{\Rightarrow} aw_1b = w$$

is possible, and w can be derived with G. Obviously, similar arguments can be made if $w = bw_1a$.

If w is not of this form, that is, if it starts and ends with the same symbol, then the counting argument tells us that it must have the form $w = w_1w_2$, with w_1 and w_2 both in L and of length less than or equal to $2n$. Hence again we see that

$$S \Rightarrow SS \stackrel{*}{\Rightarrow} w_1S \stackrel{*}{\Rightarrow} w_1w_2 = w$$

is possible.

Since the inductive assumption is clearly satisfied for $n = 1$, we have a basis, and the claim is true for all n, completing our argument. ∎

Example 1.13 (continuation)

Once we see the argument intuitively, we are ready to proceed more rigorously. Again we use induction. Assume that all $w \in L$ with $|w| \leq 2n$ can be derived with G. Take any $w \in L$ of length $2n+2$. If $w = aw_1b$, then w_1 is in L, and $|w_1| = 2n$. Therefore, by assumption we have that

$$S \stackrel{*}{\Rightarrow} w_1.$$

Then

$$S \Rightarrow aSb \stackrel{*}{\Rightarrow} aw_1b = w$$

is possible, and w can be derived with G. Obviously, similar arguments can be made if $w = bw_1a$.

If w is not of this form, that is, if it starts and ends with the same symbol, then the counting argument tells us that it must have the form $w = w_1w_2$, with w_1 and w_2 both in L and of length less than or equal to $2n$. Hence again we see that

$$S \Rightarrow SS \stackrel{*}{\Rightarrow} w_1S \stackrel{*}{\Rightarrow} w_1w_2 = w$$

is possible.

Since the inductive assumption is clearly satisfied for $n = 1$, we have a basis, and the claim is true for all n, completing our argument. ■

Example 1.13 (continuation)

Once we see the argument intuitively, we are ready to proceed more rigorously. Again we use induction. Assume that all $w \in L$ with $|w| \leq 2n$ can be derived with G. Take any $w \in L$ of length $2n+2$. If $w = aw_1b$, then w_1 is in L, and $|w_1| = 2n$. Therefore, by assumption we have that

$$S \stackrel{*}{\Rightarrow} w_1.$$

Then

$$S \Rightarrow aSb \stackrel{*}{\Rightarrow} aw_1b = w$$

is possible, and w can be derived with G. Obviously, similar arguments can be made if $w = bw_1a$.

If w is not of this form, that is, if it starts and ends with the same symbol, then the counting argument tells us that it must have the form $w = w_1w_2$, with w_1 and w_2 both in L and of length less than or equal to $2n$. Hence again we see that

$$S \Rightarrow SS \stackrel{*}{\Rightarrow} w_1S \stackrel{*}{\Rightarrow} w_1w_2 = w$$

is possible.

Since the inductive assumption is clearly satisfied for $n = 1$, we have a basis, and the claim is true for all n, completing our argument. ■

Example 1.13 (continuation)

Once we see the argument intuitively, we are ready to proceed more rigorously. Again we use induction. Assume that all $w \in L$ with $|w| \leq 2n$ can be derived with G. Take any $w \in L$ of length $2n + 2$. If $w = aw_1b$, then w_1 is in L, and $|w_1| = 2n$. Therefore, by assumption we have that

$$S \stackrel{*}{\Rightarrow} w_1.$$

Then

$$S \Rightarrow aSb \stackrel{*}{\Rightarrow} aw_1b = w$$

is possible, and w can be derived with G. Obviously, similar arguments can be made if $w = bw_1a$.

If w is not of this form, that is, if it starts and ends with the same symbol, then the counting argument tells us that it must have the form $w = w_1w_2$, with w_1 and w_2 both in L and of length less than or equal to $2n$. Hence again we see that

$$S \Rightarrow SS \stackrel{*}{\Rightarrow} w_1S \stackrel{*}{\Rightarrow} w_1w_2 = w$$

is possible.

Since the inductive assumption is clearly satisfied for $n = 1$, we have a basis, and the claim is true for all n, completing our argument. ∎

Example 1.13 (continuation)

Once we see the argument intuitively, we are ready to proceed more rigorously. Again we use induction. Assume that all $w \in L$ with $|w| \leq 2n$ can be derived with G. Take any $w \in L$ of length $2n+2$. If $w = aw_1b$, then w_1 is in L, and $|w_1| = 2n$. Therefore, by assumption we have that

$$S \stackrel{*}{\Rightarrow} w_1.$$

Then

$$S \Rightarrow aSb \stackrel{*}{\Rightarrow} aw_1b = w$$

is possible, and w can be derived with G. Obviously, similar arguments can be made if $w = bw_1a$.

If w is not of this form, that is, if it starts and ends with the same symbol, then the counting argument tells us that it must have the form $w = w_1w_2$, with w_1 and w_2 both in L and of length less than or equal to $2n$. Hence again we see that

$$S \Rightarrow SS \stackrel{*}{\Rightarrow} w_1S \stackrel{*}{\Rightarrow} w_1w_2 = w$$

is possible.

Since the inductive assumption is clearly satisfied for $n = 1$, we have a basis, and the claim is true for all n, completing our argument. ∎

Example 1.13 (continuation)

Once we see the argument intuitively, we are ready to proceed more rigorously. Again we use induction. Assume that all $w \in L$ with $|w| \leq 2n$ can be derived with G. Take any $w \in L$ of length $2n + 2$. If $w = aw_1b$, then w_1 is in L, and $|w_1| = 2n$. Therefore, by assumption we have that

$$S \stackrel{*}{\Rightarrow} w_1.$$

Then

$$S \Rightarrow aSb \stackrel{*}{\Rightarrow} aw_1b = w$$

is possible, and w can be derived with G. Obviously, similar arguments can be made if $w = bw_1a$.

If w is not of this form, that is, if it starts and ends with the same symbol, then the counting argument tells us that it must have the form $w = w_1w_2$, with w_1 and w_2 both in L and of length less than or equal to $2n$. Hence again we see that

$$S \Rightarrow SS \stackrel{*}{\Rightarrow} w_1S \stackrel{*}{\Rightarrow} w_1w_2 = w$$

is possible.

Since the inductive assumption is clearly satisfied for $n = 1$, we have a basis, and the claim is true for all n, completing our argument. ■

Example 1.13 (continuation)

Once we see the argument intuitively, we are ready to proceed more rigorously. Again we use induction. Assume that all $w \in L$ with $|w| \leq 2n$ can be derived with G. Take any $w \in L$ of length $2n + 2$. If $w = aw_1b$, then w_1 is in L, and $|w_1| = 2n$. Therefore, by assumption we have that

$$S \overset{*}{\Rightarrow} w_1.$$

Then

$$S \Rightarrow aSb \overset{*}{\Rightarrow} aw_1b = w$$

is possible, and w can be derived with G. Obviously, similar arguments can be made if $w = bw_1a$.

If w is not of this form, that is, if it starts and ends with the same symbol, then the counting argument tells us that it must have the form $w = w_1w_2$, with w_1 and w_2 both in L and of length less than or equal to $2n$. Hence again we see that

$$S \Rightarrow SS \overset{*}{\Rightarrow} w_1S \overset{*}{\Rightarrow} w_1w_2 = w$$

is possible.

Since the inductive assumption is clearly satisfied for $n = 1$, we have a basis, and the claim is true for all n, completing our argument. ■

Example 1.13 (continuation)

Once we see the argument intuitively, we are ready to proceed more rigorously. Again we use induction. Assume that all $w \in L$ with $|w| \leq 2n$ can be derived with G. Take any $w \in L$ of length $2n+2$. If $w = aw_1b$, then w_1 is in L, and $|w_1| = 2n$. Therefore, by assumption we have that

$$S \overset{*}{\Rightarrow} w_1.$$

Then

$$S \Rightarrow aSb \overset{*}{\Rightarrow} aw_1b = w$$

is possible, and w can be derived with G. Obviously, similar arguments can be made if $w = bw_1a$.

If w is not of this form, that is, if it starts and ends with the same symbol, then the counting argument tells us that it must have the form $w = w_1w_2$, with w_1 and w_2 both in L and of length less than or equal to $2n$. Hence again we see that

$$S \Rightarrow SS \overset{*}{\Rightarrow} w_1S \overset{*}{\Rightarrow} w_1w_2 = w$$

is possible.

Since the inductive assumption is clearly satisfied for $n = 1$, we have a basis, and the claim is true for all n, completing our argument. ■

Example 1.13 (continuation)

Once we see the argument intuitively, we are ready to proceed more rigorously. Again we use induction. Assume that all $w \in L$ with $|w| \leq 2n$ can be derived with G. Take any $w \in L$ of length $2n + 2$. If $w = aw_1b$, then w_1 is in L, and $|w_1| = 2n$. Therefore, by assumption we have that

$$S \stackrel{*}{\Rightarrow} w_1.$$

Then

$$S \Rightarrow aSb \stackrel{*}{\Rightarrow} aw_1b = w$$

is possible, and w can be derived with G. Obviously, similar arguments can be made if $w = bw_1a$.

If w is not of this form, that is, if it starts and ends with the same symbol, then the counting argument tells us that it must have the form $w = w_1w_2$, with w_1 and w_2 both in L and of length less than or equal to $2n$. Hence again we see that

$$S \Rightarrow SS \stackrel{*}{\Rightarrow} w_1S \stackrel{*}{\Rightarrow} w_1w_2 = w$$

is possible.

Since the inductive assumption is clearly satisfied for $n = 1$, we have a basis, and the claim is true for all n, completing our argument. ■

Example 1.13 (continuation)

Once we see the argument intuitively, we are ready to proceed more rigorously. Again we use induction. Assume that all $w \in L$ with $|w| \leq 2n$ can be derived with G. Take any $w \in L$ of length $2n+2$. If $w = aw_1b$, then w_1 is in L, and $|w_1| = 2n$. Therefore, by assumption we have that

$$S \stackrel{*}{\Rightarrow} w_1.$$

Then

$$S \Rightarrow aSb \stackrel{*}{\Rightarrow} aw_1b = w$$

is possible, and w can be derived with G. Obviously, similar arguments can be made if $w = bw_1a$.

If w is not of this form, that is, if it starts and ends with the same symbol, then the counting argument tells us that it must have the form $w = w_1w_2$, with w_1 and w_2 both in L and of length less than or equal to $2n$. Hence again we see that

$$S \Rightarrow SS \stackrel{*}{\Rightarrow} w_1S \stackrel{*}{\Rightarrow} w_1w_2 = w$$

is possible.

Since the inductive assumption is clearly satisfied for $n = 1$, we have a basis, and the claim is true for all n, completing our argument. ■

Example 1.13 (continuation)

Once we see the argument intuitively, we are ready to proceed more rigorously. Again we use induction. Assume that all $w \in L$ with $|w| \leq 2n$ can be derived with G. Take any $w \in L$ of length $2n + 2$. If $w = aw_1b$, then w_1 is in L, and $|w_1| = 2n$. Therefore, by assumption we have that

$$S \overset{*}{\Rightarrow} w_1.$$

Then

$$S \Rightarrow aSb \overset{*}{\Rightarrow} aw_1b = w$$

is possible, and w can be derived with G. Obviously, similar arguments can be made if $w = bw_1a$.

If w is not of this form, that is, if it starts and ends with the same symbol, then the counting argument tells us that it must have the form $w = w_1w_2$, with w_1 and w_2 both in L and of length less than or equal to $2n$. Hence again we see that

$$S \Rightarrow SS \overset{*}{\Rightarrow} w_1S \overset{*}{\Rightarrow} w_1w_2 = w$$

is possible.

Since the inductive assumption is clearly satisfied for $n = 1$, we have a basis, and the claim is true for all n, completing our argument. ■

Example 1.13 (continuation)

Once we see the argument intuitively, we are ready to proceed more rigorously. Again we use induction. Assume that all $w \in L$ with $|w| \leq 2n$ can be derived with G. Take any $w \in L$ of length $2n+2$. If $w = aw_1b$, then w_1 is in L, and $|w_1| = 2n$. Therefore, by assumption we have that

$$S \stackrel{*}{\Rightarrow} w_1.$$

Then

$$S \Rightarrow aSb \stackrel{*}{\Rightarrow} aw_1b = w$$

is possible, and w can be derived with G. Obviously, similar arguments can be made if $w = bw_1a$.

If w is not of this form, that is, if it starts and ends with the same symbol, then the counting argument tells us that it must have the form $w = w_1w_2$, with w_1 and w_2 both in L and of length less than or equal to $2n$. Hence again we see that

$$S \Rightarrow SS \stackrel{*}{\Rightarrow} w_1S \stackrel{*}{\Rightarrow} w_1w_2 = w$$

is possible.

Since the inductive assumption is clearly satisfied for $n = 1$, we have a basis, and the claim is true for all n, completing our argument. ∎

Example 1.13 (continuation)

Once we see the argument intuitively, we are ready to proceed more rigorously. Again we use induction. Assume that all $w \in L$ with $|w| \leq 2n$ can be derived with G. Take any $w \in L$ of length $2n+2$. If $w = aw_1b$, then w_1 is in L, and $|w_1| = 2n$. Therefore, by assumption we have that

$$S \stackrel{*}{\Rightarrow} w_1.$$

Then

$$S \Rightarrow aSb \stackrel{*}{\Rightarrow} aw_1b = w$$

is possible, and w can be derived with G. Obviously, similar arguments can be made if $w = bw_1a$.
If w is not of this form, that is, if it starts and ends with the same symbol, then the counting argument tells us that it must have the form $w = w_1w_2$, with w_1 and w_2 both in L and of length less than or equal to $2n$. Hence again we see that

$$S \Rightarrow SS \stackrel{*}{\Rightarrow} w_1S \stackrel{*}{\Rightarrow} w_1w_2 = w$$

is possible.
Since the inductive assumption is clearly satisfied for $n = 1$, we have a basis, and the claim is true for all n, completing our argument. ■

Example 1.13 (continuation)

Once we see the argument intuitively, we are ready to proceed more rigorously. Again we use induction. Assume that all $w \in L$ with $|w| \leq 2n$ can be derived with G. Take any $w \in L$ of length $2n + 2$. If $w = aw_1b$, then w_1 is in L, and $|w_1| = 2n$. Therefore, by assumption we have that

$$S \overset{*}{\Rightarrow} w_1.$$

Then

$$S \Rightarrow aSb \overset{*}{\Rightarrow} aw_1b = w$$

is possible, and w can be derived with G. Obviously, similar arguments can be made if $w = bw_1a$.

If w is not of this form, that is, if it starts and ends with the same symbol, then the counting argument tells us that it must have the form $w = w_1w_2$, with w_1 and w_2 both in L and of length less than or equal to $2n$. Hence again we see that

$$S \Rightarrow SS \overset{*}{\Rightarrow} w_1S \overset{*}{\Rightarrow} w_1w_2 = w$$

is possible.

Since the inductive assumption is clearly satisfied for $n = 1$, we have a basis, and the claim is true for all n, completing our argument. ■

Example 1.13 (continuation)

Once we see the argument intuitively, we are ready to proceed more rigorously. Again we use induction. Assume that all $w \in L$ with $|w| \leq 2n$ can be derived with G. Take any $w \in L$ of length $2n+2$. If $w = aw_1b$, then w_1 is in L, and $|w_1| = 2n$. Therefore, by assumption we have that

$$S \stackrel{*}{\Rightarrow} w_1.$$

Then

$$S \Rightarrow aSb \stackrel{*}{\Rightarrow} aw_1b = w$$

is possible, and w can be derived with G. Obviously, similar arguments can be made if $w = bw_1a$.

If w is not of this form, that is, if it starts and ends with the same symbol, then the counting argument tells us that it must have the form $w = w_1w_2$, with w_1 and w_2 both in L and of length less than or equal to $2n$. Hence again we see that

$$S \Rightarrow SS \stackrel{*}{\Rightarrow} w_1S \stackrel{*}{\Rightarrow} w_1w_2 = w$$

is possible.

Since the inductive assumption is clearly satisfied for $n = 1$, we have a basis, and the claim is true for all n, completing our argument. ∎

Example 1.13 (continuation)

Once we see the argument intuitively, we are ready to proceed more rigorously. Again we use induction. Assume that all $w \in L$ with $|w| \leq 2n$ can be derived with G. Take any $w \in L$ of length $2n+2$. If $w = aw_1b$, then w_1 is in L, and $|w_1| = 2n$. Therefore, by assumption we have that

$$S \overset{*}{\Rightarrow} w_1.$$

Then

$$S \Rightarrow aSb \overset{*}{\Rightarrow} aw_1b = w$$

is possible, and w can be derived with G. Obviously, similar arguments can be made if $w = bw_1a$.

If w is not of this form, that is, if it starts and ends with the same symbol, then the counting argument tells us that it must have the form $w = w_1w_2$, with w_1 and w_2 both in L and of length less than or equal to $2n$. Hence again we see that

$$S \Rightarrow SS \overset{*}{\Rightarrow} w_1S \overset{*}{\Rightarrow} w_1w_2 = w$$

is possible.

Since the inductive assumption is clearly satisfied for $n = 1$, we have a basis, and the claim is true for all n, completing our argument. ■

Example 1.13 (continuation)

Once we see the argument intuitively, we are ready to proceed more rigorously. Again we use induction. Assume that all $w \in L$ with $|w| \leq 2n$ can be derived with G. Take any $w \in L$ of length $2n + 2$. If $w = aw_1b$, then w_1 is in L, and $|w_1| = 2n$. Therefore, by assumption we have that

$$S \stackrel{*}{\Rightarrow} w_1.$$

Then

$$S \Rightarrow aSb \stackrel{*}{\Rightarrow} aw_1b = w$$

is possible, and w can be derived with G. Obviously, similar arguments can be made if $w = bw_1a$.

If w is not of this form, that is, if it starts and ends with the same symbol, then the counting argument tells us that it must have the form $w = w_1w_2$, with w_1 and w_2 both in L and of length less than or equal to $2n$. Hence again we see that

$$S \Rightarrow SS \stackrel{*}{\Rightarrow} w_1S \stackrel{*}{\Rightarrow} w_1w_2 = w$$

is possible.

Since the inductive assumption is clearly satisfied for $n = 1$, we have a basis, and the claim is true for all n, completing our argument.

Example 1.13 (continuation)

Once we see the argument intuitively, we are ready to proceed more rigorously. Again we use induction. Assume that all $w \in L$ with $|w| \leq 2n$ can be derived with G. Take any $w \in L$ of length $2n + 2$. If $w = aw_1b$, then w_1 is in L, and $|w_1| = 2n$. Therefore, by assumption we have that

$$S \stackrel{*}{\Rightarrow} w_1.$$

Then

$$S \Rightarrow aSb \stackrel{*}{\Rightarrow} aw_1b = w$$

is possible, and w can be derived with G. Obviously, similar arguments can be made if $w = bw_1a$.

If w is not of this form, that is, if it starts and ends with the same symbol, then the counting argument tells us that it must have the form $w = w_1w_2$, with w_1 and w_2 both in L and of length less than or equal to $2n$. Hence again we see that

$$S \Rightarrow SS \stackrel{*}{\Rightarrow} w_1S \stackrel{*}{\Rightarrow} w_1w_2 = w$$

is possible.

Since the inductive assumption is clearly satisfied for $n = 1$, we have a basis, and the claim is true for all n, completing our argument. ■

Example 1.13 (continuation)

Once we see the argument intuitively, we are ready to proceed more rigorously. Again we use induction. Assume that all $w \in L$ with $|w| \leq 2n$ can be derived with G. Take any $w \in L$ of length $2n+2$. If $w = aw_1b$, then w_1 is in L, and $|w_1| = 2n$. Therefore, by assumption we have that

$$S \overset{*}{\Rightarrow} w_1.$$

Then

$$S \Rightarrow aSb \overset{*}{\Rightarrow} aw_1b = w$$

is possible, and w can be derived with G. Obviously, similar arguments can be made if $w = bw_1a$.

If w is not of this form, that is, if it starts and ends with the same symbol, then the counting argument tells us that it must have the form $w = w_1w_2$, with w_1 and w_2 both in L and of length less than or equal to $2n$. Hence again we see that

$$S \Rightarrow SS \overset{*}{\Rightarrow} w_1S \overset{*}{\Rightarrow} w_1w_2 = w$$

is possible.

Since the inductive assumption is clearly satisfied for $n = 1$, we have a basis, and the claim is true for all n, completing our argument. ■

Example 1.13 (continuation)

Once we see the argument intuitively, we are ready to proceed more rigorously. Again we use induction. Assume that all $w \in L$ with $|w| \leq 2n$ can be derived with G. Take any $w \in L$ of length $2n+2$. If $w = aw_1b$, then w_1 is in L, and $|w_1| = 2n$. Therefore, by assumption we have that

$$S \stackrel{*}{\Rightarrow} w_1.$$

Then

$$S \Rightarrow aSb \stackrel{*}{\Rightarrow} aw_1b = w$$

is possible, and w can be derived with G. Obviously, similar arguments can be made if $w = bw_1a$.

If w is not of this form, that is, if it starts and ends with the same symbol, then the counting argument tells us that it must have the form $w = w_1w_2$, with w_1 and w_2 both in L and of length less than or equal to $2n$. Hence again we see that

$$S \Rightarrow SS \stackrel{*}{\Rightarrow} w_1S \stackrel{*}{\Rightarrow} w_1w_2 = w$$

is possible.

Since the inductive assumption is clearly satisfied for $n = 1$, we have a basis, and the claim is true for all n, completing our argument. ■

Normally, a given language has many grammars that generate it. Even though these grammars are different, they are equivalent in some sense. We say that two grammars G_1 and G_2 are *equivalent* if they generate the same language, that is, if

$$L(G_1) = L(G_2).$$

As we will see later, it is not always easy to see if two grammars are equivalent.

Example 1.14

Consider the grammar $G_1 = (\{A, S\}, \{a, b\}, S, P_1)$, with P_1 consisting of the productions

$$S \to aAb|\lambda,$$

$$A \to aAb|\lambda.$$

Here we introduce a convenient shorthand notation in which several production rules with the same left-hand sides are written on the same line, with alternative right-hand sides separated by the symbol $|$. In this notation $S \to aAb|\lambda$ stands for the two productions $S \to aAb$ and $S \to \lambda$. This grammar is equivalent to the grammar G in Example 1.11:

$$G = (\{S\}, \{a, b\}, S, P),$$

with P given by

$$S \to aSb,$$

$$S \to \lambda.$$

The equivalence is easy to prove by showing that

$$L(G_1) = \{a^n b^n : n \geq 0\}.$$

We leave this as an exercise. ∎

2 THREE BASIC CONCEPTS: Grammars

Normally, a given language has many grammars that generate it. Even though these grammars are different, they are equivalent in some sense. We say that two grammars G_1 and G_2 are *equivalent* if they generate the same language, that is, if

$$L(G_1) = L(G_2).$$

As we will see later, it is not always easy to see if two grammars are equivalent.

Example 1.14

Consider the grammar $G_1 = (\{A, S\}, \{a, b\}, S, P_1)$, with P_1 consisting of the productions

$$S \to aAb|\lambda,$$

$$A \to aAb|\lambda.$$

Here we introduce a convenient shorthand notation in which several production rules with the same left-hand sides are written on the same line, with alternative right-hand sides separated by the symbol $|$. In this notation $S \to aAb|\lambda$ stands for the two productions $S \to aAb$ and $S \to \lambda$. This grammar is equivalent to the grammar G in Example 1.11:

$$G = (\{S\}, \{a, b\}, S, P),$$

with P given by

$$S \to aSb,$$

$$S \to \lambda.$$

The equivalence is easy to prove by showing that

$$L(G_1) = \{a^n b^n : n \geq 0\}.$$

We leave this as an exercise. ■

Normally, a given language has many grammars that generate it. Even though these grammars are different, they are equivalent in some sense. We say that two grammars G_1 and G_2 are *equivalent* if they generate the same language, that is, if

$$L(G_1) = L(G_2).$$

As we will see later, it is not always easy to see if two grammars are equivalent.

Example 1.14

Consider the grammar $G_1 = (\{A, S\}, \{a, b\}, S, P_1)$, with P_1 consisting of the productions

$$S \to aAb|\lambda,$$
$$A \to aAb|\lambda.$$

Here we introduce a convenient shorthand notation in which several production rules with the same left-hand sides are written on the same line, with alternative right-hand sides separated by the symbol $|$. In this notation $S \to aAb|\lambda$ stands for the two productions $S \to aAb$ and $S \to \lambda$. This grammar is equivalent to the grammar G in Example 1.11:

$$G = (\{S\}, \{a, b\}, S, P),$$

with P given by

$$S \to aSb,$$
$$S \to \lambda.$$

The equivalence is easy to prove by showing that

$$L(G_1) = \{a^n b^n : n \geq 0\}.$$

We leave this as an exercise. ■

2 THREE BASIC CONCEPTS: Grammars

Normally, a given language has many grammars that generate it. Even though these grammars are different, they are equivalent in some sense. We say that two grammars G_1 and G_2 are *equivalent* if they generate the same language, that is, if

$$L(G_1) = L(G_2).$$

As we will see later, it is not always easy to see if two grammars are equivalent.

Example 1.14

Consider the grammar $G_1 = (\{A, S\}, \{a, b\}, S, P_1)$, with P_1 consisting of the productions

$$S \to aAb|\lambda,$$

$$A \to aAb|\lambda.$$

Here we introduce a convenient shorthand notation in which several production rules with the same left-hand sides are written on the same line, with alternative right-hand sides separated by the symbol $|$. In this notation $S \to aAb|\lambda$ stands for the two productions $S \to aAb$ and $S \to \lambda$. This grammar is equivalent to the grammar G in Example 1.11:

$$G = (\{S\}, \{a, b\}, S, P),$$

with P given by

$$S \to aSb,$$

$$S \to \lambda.$$

The equivalence is easy to prove by showing that

$$L(G_1) = \{a^n b^n : n \geq 0\}.$$

We leave this as an exercise. ∎

Normally, a given language has many grammars that generate it. Even though these grammars are different, they are equivalent in some sense. We say that two grammars G_1 and G_2 are *equivalent* if they generate the same language, that is, if

$$L(G_1) = L(G_2).$$

As we will see later, it is not always easy to see if two grammars are equivalent.

Example 1.14

Consider the grammar $G_1 = (\{A, S\}, \{a, b\}, S, P_1)$, with P_1 consisting of the productions

$$S \to aAb|\lambda,$$

$$A \to aAb|\lambda.$$

Here we introduce a convenient shorthand notation in which several production rules with the same left-hand sides are written on the same line, with alternative right-hand sides separated by the symbol $|$. In this notation $S \to aAb|\lambda$ stands for the two productions $S \to aAb$ and $S \to \lambda$. This grammar is equivalent to the grammar G in Example 1.11:

$$G = (\{S\}, \{a, b\}, S, P),$$

with P given by

$$S \to aSb,$$

$$S \to \lambda.$$

The equivalence is easy to prove by showing that

$$L(G_1) = \{a^n b^n : n \geq 0\}.$$

We leave this as an exercise. ∎

Normally, a given language has many grammars that generate it. Even though these grammars are different, they are equivalent in some sense. We say that two grammars G_1 and G_2 are *equivalent* if they generate the same language, that is, if

$$L(G_1) = L(G_2).$$

As we will see later, it is not always easy to see if two grammars are equivalent.

Example 1.14

Consider the grammar $G_1 = (\{A, S\}, \{a, b\}, S, P_1)$, with P_1 consisting of the productions

$$S \to aAb|\lambda,$$

$$A \to aAb|\lambda.$$

Here we introduce a convenient shorthand notation in which several production rules with the same left-hand sides are written on the same line, with alternative right-hand sides separated by the symbol $|$. In this notation $S \to aAb|\lambda$ stands for the two productions $S \to aAb$ and $S \to \lambda$. This grammar is equivalent to the grammar G in Example 1.11:

$$G = (\{S\}, \{a, b\}, S, P),$$

with P given by

$$S \to aSb,$$

$$S \to \lambda.$$

The equivalence is easy to prove by showing that

$$L(G_1) = \{a^n b^n : n \geq 0\}.$$

We leave this as an exercise.

Normally, a given language has many grammars that generate it. Even though these grammars are different, they are equivalent in some sense. We say that two grammars G_1 and G_2 are *equivalent* if they generate the same language, that is, if

$$L(G_1) = L(G_2).$$

As we will see later, it is not always easy to see if two grammars are equivalent.

Example 1.14

Consider the grammar $G_1 = (\{A, S\}, \{a, b\}, S, P_1)$, with P_1 consisting of the productions

$$S \to aAb|\lambda,$$
$$A \to aAb|\lambda.$$

Here we introduce a convenient shorthand notation in which several production rules with the same left-hand sides are written on the same line, with alternative right-hand sides separated by the symbol $|$. In this notation $S \to aAb|\lambda$ stands for the two productions $S \to aAb$ and $S \to \lambda$. This grammar is equivalent to the grammar G in Example 1.11:

$$G = (\{S\}, \{a, b\}, S, P),$$

with P given by

$$S \to aSb,$$
$$S \to \lambda.$$

The equivalence is easy to prove by showing that

$$L(G_1) = \{a^n b^n : n \geq 0\}.$$

We leave this as an exercise. ■

2 THREE BASIC CONCEPTS: Grammars

Normally, a given language has many grammars that generate it. Even though these grammars are different, they are equivalent in some sense. We say that two grammars G_1 and G_2 are *equivalent* if they generate the same language, that is, if

$$L(G_1) = L(G_2).$$

As we will see later, it is not always easy to see if two grammars are equivalent.

Example 1.14

Consider the grammar $G_1 = (\{A, S\}, \{a, b\}, S, P_1)$, with P_1 consisting of the productions

$$S \to aAb|\lambda,$$

$$A \to aAb|\lambda.$$

Here we introduce a convenient shorthand notation in which several production rules with the same left-hand sides are written on the same line, with alternative right-hand sides separated by the symbol $|$. In this notation $S \to aAb|\lambda$ stands for the two productions $S \to aAb$ and $S \to \lambda$. This grammar is equivalent to the grammar G in Example 1.11:

$$G = (\{S\}, \{a, b\}, S, P),$$

with P given by

$$S \to aSb,$$

$$S \to \lambda.$$

The equivalence is easy to prove by showing that

$$L(G_1) = \{a^n b^n : n \geq 0\}.$$

We leave this as an exercise. ∎

Normally, a given language has many grammars that generate it. Even though these grammars are different, they are equivalent in some sense. We say that two grammars G_1 and G_2 are *equivalent* if they generate the same language, that is, if

$$L(G_1) = L(G_2).$$

As we will see later, it is not always easy to see if two grammars are equivalent.

Example 1.14

Consider the grammar $G_1 = (\{A, S\}, \{a, b\}, S, P_1)$, with P_1 consisting of the productions

$$S \to aAb|\lambda,$$
$$A \to aAb|\lambda.$$

Here we introduce a convenient shorthand notation in which several production rules with the same left-hand sides are written on the same line, with alternative right-hand sides separated by the symbol $|$. In this notation $S \to aAb|\lambda$ stands for the two productions $S \to aAb$ and $S \to \lambda$. This grammar is equivalent to the grammar G in Example 1.11:

$$G = (\{S\}, \{a, b\}, S, P),$$

with P given by

$$S \to aSb,$$
$$S \to \lambda.$$

The equivalence is easy to prove by showing that

$$L(G_1) = \{a^n b^n : n \geq 0\}.$$

We leave this as an exercise. ■

Normally, a given language has many grammars that generate it. Even though these grammars are different, they are equivalent in some sense. We say that two grammars G_1 and G_2 are *equivalent* if they generate the same language, that is, if

$$L(G_1) = L(G_2).$$

As we will see later, it is not always easy to see if two grammars are equivalent.

Example 1.14

Consider the grammar $G_1 = (\{A, S\}, \{a, b\}, S, P_1)$, with P_1 consisting of the productions

$$S \to aAb|\lambda,$$
$$A \to aAb|\lambda.$$

Here we introduce a convenient shorthand notation in which several production rules with the same left-hand sides are written on the same line, with alternative right-hand sides separated by the symbol $|$. In this notation $S \to aAb|\lambda$ stands for the two productions $S \to aAb$ and $S \to \lambda$. This grammar is equivalent to the grammar G in Example 1.11:

$$G = (\{S\}, \{a, b\}, S, P),$$

with P given by

$$S \to aSb,$$
$$S \to \lambda.$$

The equivalence is easy to prove by showing that

$$L(G_1) = \{a^n b^n : n \geq 0\}.$$

We leave this as an exercise. ∎

Normally, a given language has many grammars that generate it. Even though these grammars are different, they are equivalent in some sense. We say that two grammars G_1 and G_2 are *equivalent* if they generate the same language, that is, if

$$L(G_1) = L(G_2).$$

As we will see later, it is not always easy to see if two grammars are equivalent.

Example 1.14

Consider the grammar $G_1 = (\{A, S\}, \{a, b\}, S, P_1)$, with P_1 consisting of the productions

$$S \to aAb|\lambda,$$
$$A \to aAb|\lambda.$$

Here we introduce a convenient shorthand notation in which several production rules with the same left-hand sides are written on the same line, with alternative right-hand sides separated by the symbol $|$. In this notation $S \to aAb|\lambda$ stands for the two productions $S \to aAb$ and $S \to \lambda$. This grammar is equivalent to the grammar G in Example 1.11:

$$G = (\{S\}, \{a, b\}, S, P),$$

with P given by

$$S \to aSb,$$
$$S \to \lambda.$$

The equivalence is easy to prove by showing that

$$L(G_1) = \{a^n b^n : n \geq 0\}.$$

We leave this as an exercise. ■

Normally, a given language has many grammars that generate it. Even though these grammars are different, they are equivalent in some sense. We say that two grammars G_1 and G_2 are *equivalent* if they generate the same language, that is, if

$$L(G_1) = L(G_2).$$

As we will see later, it is not always easy to see if two grammars are equivalent.

Example 1.14

Consider the grammar $G_1 = (\{A, S\}, \{a, b\}, S, P_1)$, with P_1 consisting of the productions

$$S \to aAb|\lambda,$$
$$A \to aAb|\lambda.$$

Here we introduce a convenient shorthand notation in which several production rules with the same left-hand sides are written on the same line, with alternative right-hand sides separated by the symbol $|$. In this notation $S \to aAb|\lambda$ stands for the two productions $S \to aAb$ and $S \to \lambda$. This grammar is equivalent to the grammar G in Example 1.11:

$$G = (\{S\}, \{a, b\}, S, P),$$

with P given by

$$S \to aSb,$$
$$S \to \lambda.$$

The equivalence is easy to prove by showing that

$$L(G_1) = \{a^n b^n : n \geq 0\}.$$

We leave this as an exercise. ■

Normally, a given language has many grammars that generate it. Even though these grammars are different, they are equivalent in some sense. We say that two grammars G_1 and G_2 are *equivalent* if they generate the same language, that is, if

$$L(G_1) = L(G_2).$$

As we will see later, it is not always easy to see if two grammars are equivalent.

Example 1.14

Consider the grammar $G_1 = (\{A, S\}, \{a, b\}, S, P_1)$, with P_1 consisting of the productions

$$S \to aAb|\lambda,$$
$$A \to aAb|\lambda.$$

Here we introduce a convenient shorthand notation in which several production rules with the same left-hand sides are written on the same line, with alternative right-hand sides separated by the symbol $|$. In this notation $S \to aAb|\lambda$ stands for the two productions $S \to aAb$ and $S \to \lambda$. This grammar is equivalent to the grammar G in Example 1.11:

$$G = (\{S\}, \{a, b\}, S, P),$$

with P given by

$$S \to aSb,$$
$$S \to \lambda.$$

The equivalence is easy to prove by showing that

$$L(G_1) = \{a^n b^n : n \geq 0\}.$$

We leave this as an exercise. ■

Normally, a given language has many grammars that generate it. Even though these grammars are different, they are equivalent in some sense. We say that two grammars G_1 and G_2 are *equivalent* if they generate the same language, that is, if

$$L(G_1) = L(G_2).$$

As we will see later, it is not always easy to see if two grammars are equivalent.

Example 1.14

Consider the grammar $G_1 = (\{A, S\}, \{a, b\}, S, P_1)$, with P_1 consisting of the productions

$$S \to aAb|\lambda,$$
$$A \to aAb|\lambda.$$

Here we introduce a convenient shorthand notation in which several production rules with the same left-hand sides are written on the same line, with alternative right-hand sides separated by the symbol |. In this notation $S \to aAb|\lambda$ stands for the two productions $S \to aAb$ and $S \to \lambda$. This grammar is equivalent to the grammar G in Example 1.11:

$$G = (\{S\}, \{a, b\}, S, P),$$

with P given by

$$S \to aSb,$$
$$S \to \lambda.$$

The equivalence is easy to prove by showing that

$$L(G_1) = \{a^n b^n : n \geq 0\}.$$

We leave this as an exercise. ■

Normally, a given language has many grammars that generate it. Even though these grammars are different, they are equivalent in some sense. We say that two grammars G_1 and G_2 are *equivalent* if they generate the same language, that is, if

$$L(G_1) = L(G_2).$$

As we will see later, it is not always easy to see if two grammars are equivalent.

Example 1.14

Consider the grammar $G_1 = (\{A, S\}, \{a, b\}, S, P_1)$, with P_1 consisting of the productions

$$S \to aAb|\lambda,$$
$$A \to aAb|\lambda.$$

Here we introduce a convenient shorthand notation in which several production rules with the same left-hand sides are written on the same line, with alternative right-hand sides separated by the symbol |. In this notation $S \to aAb|\lambda$ stands for the two productions $S \to aAb$ and $S \to \lambda$. This grammar is equivalent to the grammar G in Example 1.11:

$$G = (\{S\}, \{a, b\}, S, P),$$

with P given by

$$S \to aSb,$$
$$S \to \lambda.$$

The equivalence is easy to prove by showing that

$$L(G_1) = \{a^n b^n : n \geq 0\}.$$

We leave this as an exercise. ■

Normally, a given language has many grammars that generate it. Even though these grammars are different, they are equivalent in some sense. We say that two grammars G_1 and G_2 are *equivalent* if they generate the same language, that is, if

$$L(G_1) = L(G_2).$$

As we will see later, it is not always easy to see if two grammars are equivalent.

Example 1.14

Consider the grammar $G_1 = (\{A, S\}, \{a, b\}, S, P_1)$, with P_1 consisting of the productions

$$S \to aAb|\lambda,$$
$$A \to aAb|\lambda.$$

Here we introduce a convenient shorthand notation in which several production rules with the same left-hand sides are written on the same line, with alternative right-hand sides separated by the symbol $|$. In this notation $S \to aAb|\lambda$ stands for the two productions $S \to aAb$ and $S \to \lambda$. This grammar is equivalent to the grammar G in Example 1.11:

$$G = (\{S\}, \{a, b\}, S, P),$$

with P given by

$$S \to aSb,$$
$$S \to \lambda.$$

The equivalence is easy to prove by showing that

$$L(G_1) = \{a^n b^n : n \geq 0\}.$$

We leave this as an exercise. ■

Normally, a given language has many grammars that generate it. Even though these grammars are different, they are equivalent in some sense. We say that two grammars G_1 and G_2 are *equivalent* if they generate the same language, that is, if

$$L(G_1) = L(G_2).$$

As we will see later, it is not always easy to see if two grammars are equivalent.

Example 1.14

Consider the grammar $G_1 = (\{A, S\}, \{a, b\}, S, P_1)$, with P_1 consisting of the productions

$$S \to aAb|\lambda,$$
$$A \to aAb|\lambda.$$

Here we introduce a convenient shorthand notation in which several production rules with the same left-hand sides are written on the same line, with alternative right-hand sides separated by the symbol $|$. In this notation $S \to aAb|\lambda$ stands for the two productions $S \to aAb$ and $S \to \lambda$. This grammar is equivalent to the grammar G in Example 1.11:

$$G = (\{S\}, \{a, b\}, S, P),$$

with P given by

$$S \to aSb,$$
$$S \to \lambda.$$

The equivalence is easy to prove by showing that

$$L(G_1) = \{a^n b^n : n \geq 0\}.$$

We leave this as an exercise. ■

Normally, a given language has many grammars that generate it. Even though these grammars are different, they are equivalent in some sense. We say that two grammars G_1 and G_2 are *equivalent* if they generate the same language, that is, if

$$L(G_1) = L(G_2).$$

As we will see later, it is not always easy to see if two grammars are equivalent.

Example 1.14

Consider the grammar $G_1 = (\{A, S\}, \{a, b\}, S, P_1)$, with P_1 consisting of the productions

$$S \to aAb|\lambda,$$
$$A \to aAb|\lambda.$$

Here we introduce a convenient shorthand notation in which several production rules with the same left-hand sides are written on the same line, with alternative right-hand sides separated by the symbol $|$. In this notation $S \to aAb|\lambda$ stands for the two productions $S \to aAb$ and $S \to \lambda$. This grammar is equivalent to the grammar G in Example 1.11:

$$G = (\{S\}, \{a, b\}, S, P),$$

with P given by

$$S \to aSb,$$
$$S \to \lambda.$$

The equivalence is easy to prove by showing that

$$L(G_1) = \{a^n b^n : n \geq 0\}.$$

We leave this as an exercise.

Normally, a given language has many grammars that generate it. Even though these grammars are different, they are equivalent in some sense. We say that two grammars G_1 and G_2 are *equivalent* if they generate the same language, that is, if

$$L(G_1) = L(G_2).$$

As we will see later, it is not always easy to see if two grammars are equivalent.

Example 1.14

Consider the grammar $G_1 = (\{A, S\}, \{a, b\}, S, P_1)$, with P_1 consisting of the productions

$$S \to aAb|\lambda,$$
$$A \to aAb|\lambda.$$

Here we introduce a convenient shorthand notation in which several production rules with the same left-hand sides are written on the same line, with alternative right-hand sides separated by the symbol $|$. In this notation $S \to aAb|\lambda$ stands for the two productions $S \to aAb$ and $S \to \lambda$. This grammar is equivalent to the grammar G in Example 1.11:

$$G = (\{S\}, \{a, b\}, S, P),$$

with P given by

$$S \to aSb,$$
$$S \to \lambda.$$

The equivalence is easy to prove by showing that

$$L(G_1) = \{a^n b^n : n \geq 0\}.$$

We leave this as an exercise. ∎

Normally, a given language has many grammars that generate it. Even though these grammars are different, they are equivalent in some sense. We say that two grammars G_1 and G_2 are *equivalent* if they generate the same language, that is, if

$$L(G_1) = L(G_2).$$

As we will see later, it is not always easy to see if two grammars are equivalent.

Example 1.14

Consider the grammar $G_1 = (\{A, S\}, \{a, b\}, S, P_1)$, with P_1 consisting of the productions

$$S \to aAb|\lambda,$$
$$A \to aAb|\lambda.$$

Here we introduce a convenient shorthand notation in which several production rules with the same left-hand sides are written on the same line, with alternative right-hand sides separated by the symbol $|$. In this notation $S \to aAb|\lambda$ stands for the two productions $S \to aAb$ and $S \to \lambda$. This grammar is equivalent to the grammar G in Example 1.11:

$$G = (\{S\}, \{a, b\}, S, P),$$

with P given by

$$S \to aSb,$$
$$S \to \lambda.$$

The equivalence is easy to prove by showing that

$$L(G_1) = \{a^n b^n : n \geq 0\}.$$

We leave this as an exercise. ■

Normally, a given language has many grammars that generate it. Even though these grammars are different, they are equivalent in some sense. We say that two grammars G_1 and G_2 are *equivalent* if they generate the same language, that is, if

$$L(G_1) = L(G_2).$$

As we will see later, it is not always easy to see if two grammars are equivalent.

Example 1.14

Consider the grammar $G_1 = (\{A, S\}, \{a, b\}, S, P_1)$, with P_1 consisting of the productions

$$S \to aAb|\lambda,$$

$$A \to aAb|\lambda.$$

Here we introduce a convenient shorthand notation in which several production rules with the same left-hand sides are written on the same line, with alternative right-hand sides separated by the symbol $|$. In this notation $S \to aAb|\lambda$ stands for the two productions $S \to aAb$ and $S \to \lambda$. This grammar is equivalent to the grammar G in Example 1.11:

$$G = (\{S\}, \{a, b\}, S, P),$$

with P given by

$$S \to aSb,$$

$$S \to \lambda.$$

The equivalence is easy to prove by showing that

$$L(G_1) = \{a^n b^n : n \geq 0\}.$$

We leave this as an exercise. ∎

Normally, a given language has many grammars that generate it. Even though these grammars are different, they are equivalent in some sense. We say that two grammars G_1 and G_2 are *equivalent* if they generate the same language, that is, if

$$L(G_1) = L(G_2).$$

As we will see later, it is not always easy to see if two grammars are equivalent.

Example 1.14

Consider the grammar $G_1 = (\{A, S\}, \{a, b\}, S, P_1)$, with P_1 consisting of the productions

$$S \to aAb|\lambda,$$
$$A \to aAb|\lambda.$$

Here we introduce a convenient shorthand notation in which several production rules with the same left-hand sides are written on the same line, with alternative right-hand sides separated by the symbol $|$. In this notation $S \to aAb|\lambda$ stands for the two productions $S \to aAb$ and $S \to \lambda$. This grammar is equivalent to the grammar G in Example 1.11:

$$G = (\{S\}, \{a, b\}, S, P),$$

with P given by

$$S \to aSb,$$
$$S \to \lambda.$$

The equivalence is easy to prove by showing that

$$L(G_1) = \{a^n b^n : n \geq 0\}.$$

We leave this as an exercise. ■

An automaton is an abstract model of a digital computer. As such, every automaton includes some essential features. It has a mechanism for reading input. It will be assumed that the input is a string over a given alphabet, written on an *input file*, which the automaton can read but not change. The input file is divided into cells, each of which can hold one symbol. The input mechanism can read the input file from left to right, one symbol at a time.

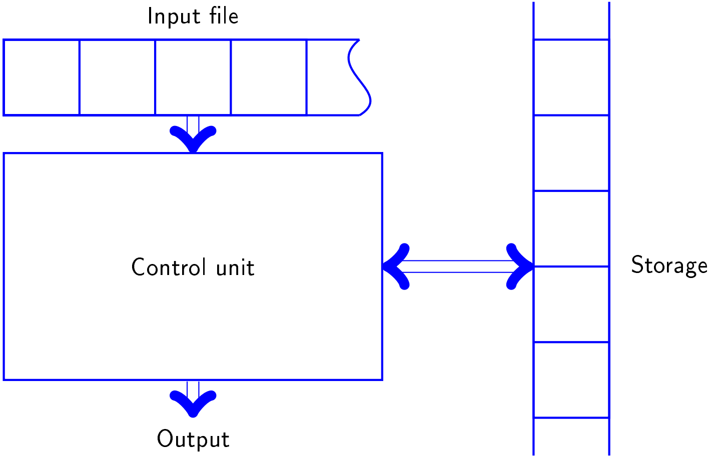
An automaton is an abstract model of a digital computer. As such, every automaton includes some essential features. It has a mechanism for reading input. It will be assumed that the input is a string over a given alphabet, written on an *input file*, which the automaton can read but not change. The input file is divided into cells, each of which can hold one symbol. The input mechanism can read the input file from left to right, one symbol at a time.

An automaton is an abstract model of a digital computer. As such, every automaton includes some essential features. It has a mechanism for reading input. It will be assumed that the input is a string over a given alphabet, written on an *input file*, which the automaton can read but not change. The input file is divided into cells, each of which can hold one symbol. The input mechanism can read the input file from left to right, one symbol at a time.

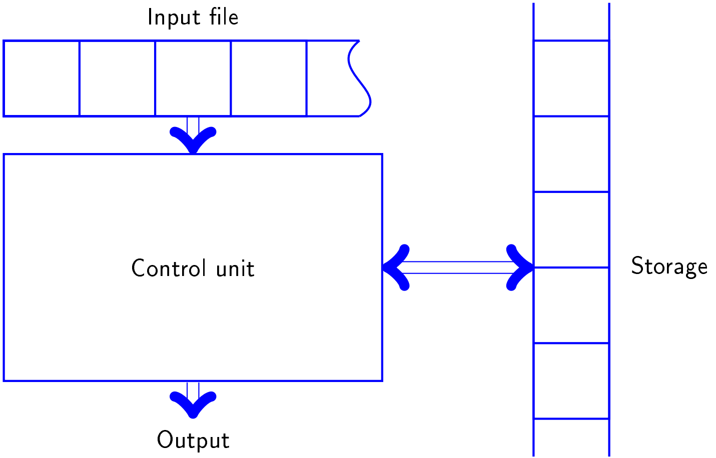
An automaton is an abstract model of a digital computer. As such, every automaton includes some essential features. It has a mechanism for reading input. It will be assumed that the input is a string over a given alphabet, written on an *input file*, which the automaton can read but not change. The input file is divided into cells, each of which can hold one symbol. The input mechanism can read the input file from left to right, one symbol at a time.

An automaton is an abstract model of a digital computer. As such, every automaton includes some essential features. It has a mechanism for reading input. It will be assumed that the input is a string over a given alphabet, written on an *input file*, which the automaton can read but not change. The input file is divided into cells, each of which can hold one symbol. The input mechanism can read the input file from left to right, one symbol at a time.

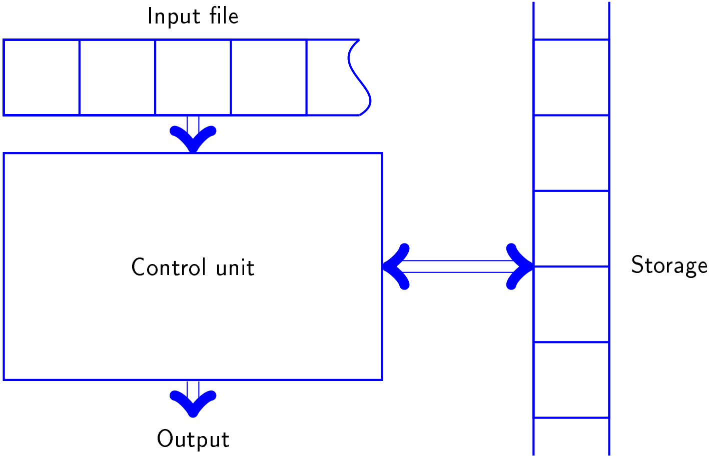
An automaton is an abstract model of a digital computer. As such, every automaton includes some essential features. It has a mechanism for reading input. It will be assumed that the input is a string over a given alphabet, written on an *input file*, which the automaton can read but not change. The input file is divided into cells, each of which can hold one symbol. The input mechanism can read the input file from left to right, one symbol at a time.



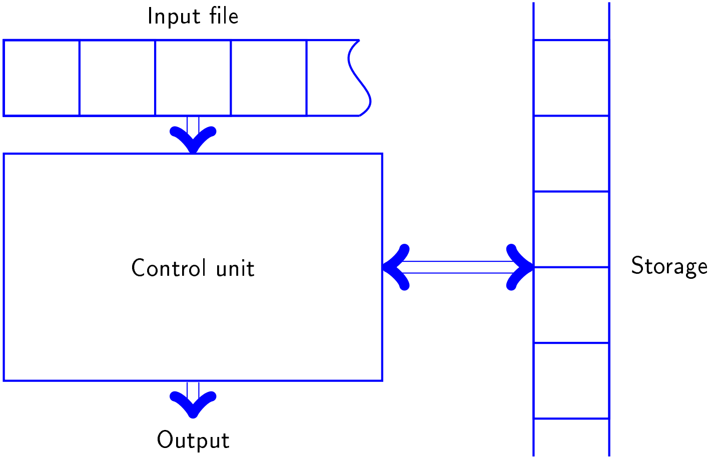
An automaton is an abstract model of a digital computer. As such, every automaton includes some essential features. It has a mechanism for reading input. It will be assumed that the input is a string over a given alphabet, written on an *input file*, which the automaton can read but not change. The input file is divided into cells, each of which can hold one symbol. The input mechanism can read the input file from left to right, one symbol at a time.



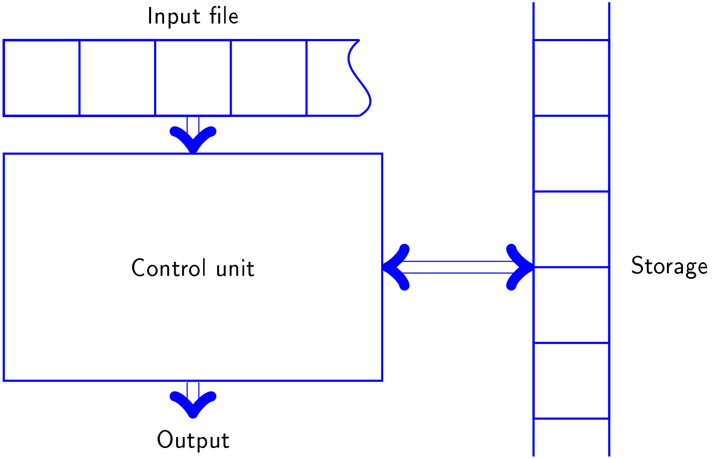
An automaton is an abstract model of a digital computer. As such, every automaton includes some essential features. It has a mechanism for reading input. It will be assumed that the input is a string over a given alphabet, written on an *input file*, which the automaton can read but not change. The input file is divided into cells, each of which can hold one symbol. The input mechanism can read the input file from left to right, one symbol at a time.



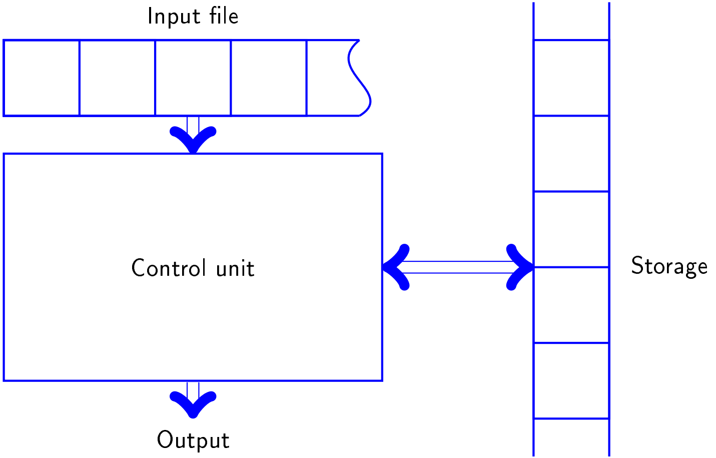
The input mechanism can also detect the end of the input string (by sensing an end-of-file condition). The automaton can produce output of some form. It may have a temporary *storage* device, consisting of an unlimited number of cells, each capable of holding a single symbol from an alphabet (not necessarily the same one as the input alphabet).



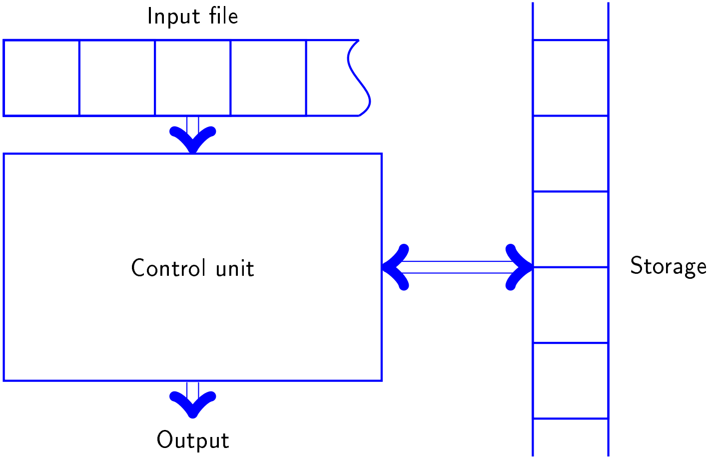
The input mechanism can also detect the end of the input string (by sensing an end-of-file condition). The automaton can produce output of some form. It may have a temporary *storage* device, consisting of an unlimited number of cells, each capable of holding a single symbol from an alphabet (not necessarily the same one as the input alphabet).



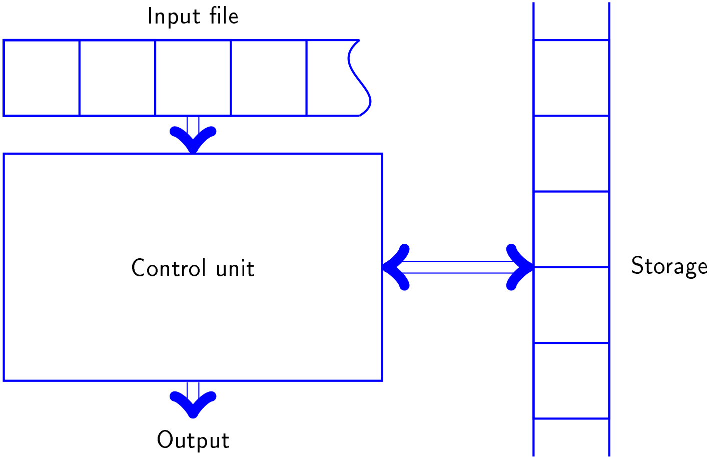
The input mechanism can also detect the end of the input string (by sensing an end-of-file condition). The automaton can produce output of some form. It may have a temporary *storage* device, consisting of an unlimited number of cells, each capable of holding a single symbol from an alphabet (not necessarily the same one as the input alphabet).



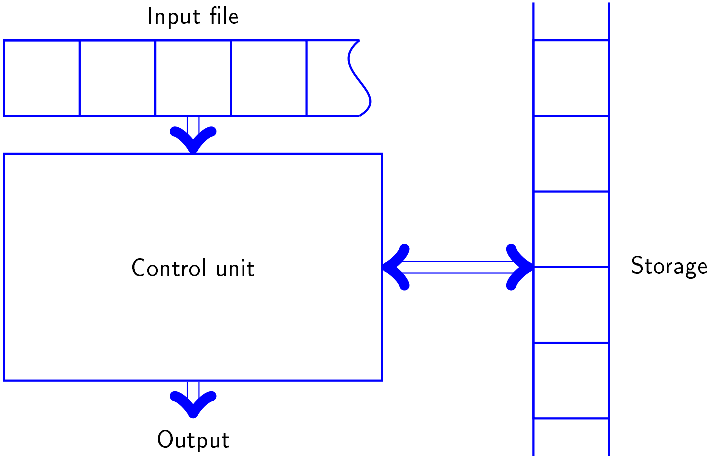
The input mechanism can also detect the end of the input string (by sensing an end-of-file condition). The automaton can produce output of some form. It may have a temporary *storage* device, consisting of an unlimited number of cells, each capable of holding a single symbol from an alphabet (not necessarily the same one as the input alphabet).



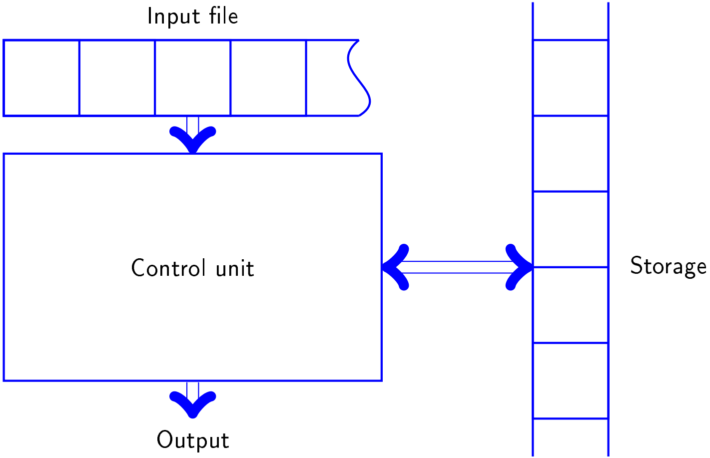
The input mechanism can also detect the end of the input string (by sensing an end-of-file condition). The automaton can produce output of some form. It may have a temporary *storage* device, consisting of an unlimited number of cells, each capable of holding a single symbol from an alphabet (not necessarily the same one as the input alphabet).



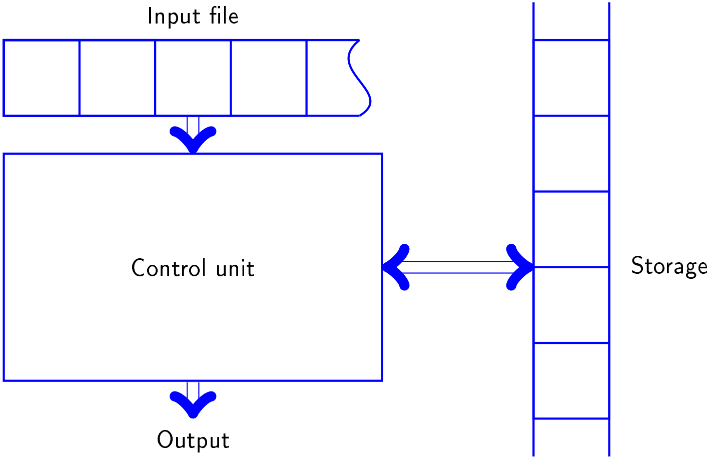
The input mechanism can also detect the end of the input string (by sensing an end-of-file condition). The automaton can produce output of some form. It may have a temporary *storage* device, consisting of an unlimited number of cells, each capable of holding a single symbol from an alphabet (not necessarily the same one as the input alphabet).



The input mechanism can also detect the end of the input string (by sensing an end-of-file condition). The automaton can produce output of some form. It may have a temporary *storage* device, consisting of an unlimited number of cells, each capable of holding a single symbol from an alphabet (not necessarily the same one as the input alphabet).



The input mechanism can also detect the end of the input string (by sensing an end-of-file condition). The automaton can produce output of some form. It may have a temporary *storage* device, consisting of an unlimited number of cells, each capable of holding a single symbol from an alphabet (not necessarily the same one as the input alphabet).



The automaton can read and change the contents of the storage cells. Finally, the automaton has a *control unit*, which can be in any one of a finite number of *internal states*, and which can change state in some defined manner. The following Figure shows a schematic representation of a general automaton.

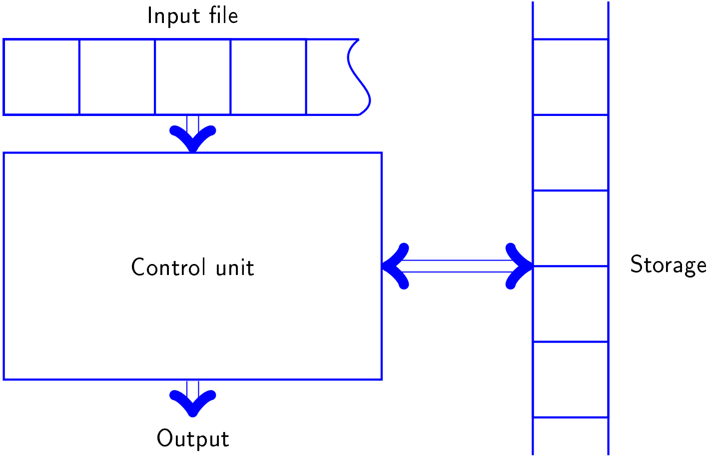
The automaton can read and change the contents of the storage cells. Finally, the automaton has a *control unit*, which can be in any one of a finite number of *internal states*, and which can change state in some defined manner. The following Figure shows a schematic representation of a general automaton.

The automaton can read and change the contents of the storage cells. Finally, the automaton has a *control unit*, which can be in any one of a finite number of *internal states*, and which can change state in some defined manner. The following Figure shows a schematic representation of a general automaton.

The automaton can read and change the contents of the storage cells. Finally, the automaton has a *control unit*, which can be in any one of a finite number of *internal states*, and which can change state in some defined manner. The following Figure shows a schematic representation of a general automaton.

The automaton can read and change the contents of the storage cells. Finally, the automaton has a *control unit*, which can be in any one of a finite number of *internal states*, and which can change state in some defined manner. The following Figure shows a schematic representation of a general automaton.

The automaton can read and change the contents of the storage cells. Finally, the automaton has a *control unit*, which can be in any one of a finite number of *internal states*, and which can change state in some defined manner. The following Figure shows a schematic representation of a general automaton.



An automaton is assumed to operate in a discrete timeframe. At any given time, the control unit is in some internal state, and the input mechanism is scanning a particular symbol on the input file. The internal state of the control unit at the next time step is determined by the *next-state* or *transition function*. This transition function gives the next state in terms of the current state, the current input symbol, and the information currently in the temporary storage. During the transition from one time interval to the next, output may be produced or the information in the temporary storage changed. The term *configuration* will be used to refer to a particular state of the control unit, input file, and temporary storage. The transition of the automaton from one configuration to the next will be called a *move*.

This general model covers all the automata we will discuss in this lectures. A finite-state control will be common to all specific cases, but differences will arise from the way in which the output can be produced and the nature of the temporary storage. As we will see, the nature of the temporary storage governs the power of different types of automata.

An automaton is assumed to operate in a discrete timeframe. At any given time, the control unit is in some internal state, and the input mechanism is scanning a particular symbol on the input file. The internal state of the control unit at the next time step is determined by the *next-state* or *transition function*. This transition function gives the next state in terms of the current state, the current input symbol, and the information currently in the temporary storage. During the transition from one time interval to the next, output may be produced or the information in the temporary storage changed. The term *configuration* will be used to refer to a particular state of the control unit, input file, and temporary storage. The transition of the automaton from one configuration to the next will be called a *move*.

This general model covers all the automata we will discuss in this lectures. A finite-state control will be common to all specific cases, but differences will arise from the way in which the output can be produced and the nature of the temporary storage. As we will see, the nature of the temporary storage governs the power of different types of automata.

An automaton is assumed to operate in a discrete timeframe. At any given time, the control unit is in some internal state, and the input mechanism is scanning a particular symbol on the input file. The internal state of the control unit at the next time step is determined by the *next-state* or *transition function*. This transition function gives the next state in terms of the current state, the current input symbol, and the information currently in the temporary storage. During the transition from one time interval to the next, output may be produced or the information in the temporary storage changed. The term *configuration* will be used to refer to a particular state of the control unit, input file, and temporary storage. The transition of the automaton from one configuration to the next will be called a *move*.

This general model covers all the automata we will discuss in this lectures. A finite-state control will be common to all specific cases, but differences will arise from the way in which the output can be produced and the nature of the temporary storage. As we will see, the nature of the temporary storage governs the power of different types of automata.

An automaton is assumed to operate in a discrete timeframe. At any given time, the control unit is in some internal state, and the input mechanism is scanning a particular symbol on the input file. The internal state of the control unit at the next time step is determined by the *next-state* or *transition function*. This transition function gives the next state in terms of the current state, the current input symbol, and the information currently in the temporary storage. During the transition from one time interval to the next, output may be produced or the information in the temporary storage changed. The term *configuration* will be used to refer to a particular state of the control unit, input file, and temporary storage. The transition of the automaton from one configuration to the next will be called a *move*.

This general model covers all the automata we will discuss in this lectures. A finite-state control will be common to all specific cases, but differences will arise from the way in which the output can be produced and the nature of the temporary storage. As we will see, the nature of the temporary storage governs the power of different types of automata.

An automaton is assumed to operate in a discrete timeframe. At any given time, the control unit is in some internal state, and the input mechanism is scanning a particular symbol on the input file. The internal state of the control unit at the next time step is determined by the *next-state* or *transition function*. This transition function gives the next state in terms of the current state, the current input symbol, and the information currently in the temporary storage. During the transition from one time interval to the next, output may be produced or the information in the temporary storage changed. The term *configuration* will be used to refer to a particular state of the control unit, input file, and temporary storage. The transition of the automaton from one configuration to the next will be called a *move*.

This general model covers all the automata we will discuss in this lectures. A finite-state control will be common to all specific cases, but differences will arise from the way in which the output can be produced and the nature of the temporary storage. As we will see, the nature of the temporary storage governs the power of different types of automata.

An automaton is assumed to operate in a discrete timeframe. At any given time, the control unit is in some internal state, and the input mechanism is scanning a particular symbol on the input file. The internal state of the control unit at the next time step is determined by the *next-state* or *transition function*. This transition function gives the next state in terms of the current state, the current input symbol, and the information currently in the temporary storage. During the transition from one time interval to the next, output may be produced or the information in the temporary storage changed. The term *configuration* will be used to refer to a particular state of the control unit, input file, and temporary storage. The transition of the automaton from one configuration to the next will be called a *move*.

This general model covers all the automata we will discuss in this lectures. A finite-state control will be common to all specific cases, but differences will arise from the way in which the output can be produced and the nature of the temporary storage. As we will see, the nature of the temporary storage governs the power of different types of automata.

An automaton is assumed to operate in a discrete timeframe. At any given time, the control unit is in some internal state, and the input mechanism is scanning a particular symbol on the input file. The internal state of the control unit at the next time step is determined by the *next-state* or *transition function*. This transition function gives the next state in terms of the current state, the current input symbol, and the information currently in the temporary storage. During the transition from one time interval to the next, output may be produced or the information in the temporary storage changed. The term *configuration* will be used to refer to a particular state of the control unit, input file, and temporary storage. The transition of the automaton from one configuration to the next will be called a *move*.

This general model covers all the automata we will discuss in this lectures. A finite-state control will be common to all specific cases, but differences will arise from the way in which the output can be produced and the nature of the temporary storage. As we will see, the nature of the temporary storage governs the power of different types of automata.

An automaton is assumed to operate in a discrete timeframe. At any given time, the control unit is in some internal state, and the input mechanism is scanning a particular symbol on the input file. The internal state of the control unit at the next time step is determined by the *next-state* or *transition function*. This transition function gives the next state in terms of the current state, the current input symbol, and the information currently in the temporary storage. During the transition from one time interval to the next, output may be produced or the information in the temporary storage changed. The term *configuration* will be used to refer to a particular state of the control unit, input file, and temporary storage. The transition of the automaton from one configuration to the next will be called a *move*.

This general model covers all the automata we will discuss in this lectures. A finite-state control will be common to all specific cases, but differences will arise from the way in which the output can be produced and the nature of the temporary storage. As we will see, the nature of the temporary storage governs the power of different types of automata.

An automaton is assumed to operate in a discrete timeframe. At any given time, the control unit is in some internal state, and the input mechanism is scanning a particular symbol on the input file. The internal state of the control unit at the next time step is determined by the *next-state* or *transition function*. This transition function gives the next state in terms of the current state, the current input symbol, and the information currently in the temporary storage. During the transition from one time interval to the next, output may be produced or the information in the temporary storage changed. The term *configuration* will be used to refer to a particular state of the control unit, input file, and temporary storage. The transition of the automaton from one configuration to the next will be called a *move*.

This general model covers all the automata we will discuss in this lectures. A finite-state control will be common to all specific cases, but differences will arise from the way in which the output can be produced and the nature of the temporary storage. As we will see, the nature of the temporary storage governs the power of different types of automata.

An automaton is assumed to operate in a discrete timeframe. At any given time, the control unit is in some internal state, and the input mechanism is scanning a particular symbol on the input file. The internal state of the control unit at the next time step is determined by the *next-state* or *transition function*. This transition function gives the next state in terms of the current state, the current input symbol, and the information currently in the temporary storage. During the transition from one time interval to the next, output may be produced or the information in the temporary storage changed. The term *configuration* will be used to refer to a particular state of the control unit, input file, and temporary storage. The transition of the automaton from one configuration to the next will be called a *move*.

This general model covers all the automata we will discuss in this lectures. A finite-state control will be common to all specific cases, but differences will arise from the way in which the output can be produced and the nature of the temporary storage. As we will see, the nature of the temporary storage governs the power of different types of automata.

An automaton is assumed to operate in a discrete timeframe. At any given time, the control unit is in some internal state, and the input mechanism is scanning a particular symbol on the input file. The internal state of the control unit at the next time step is determined by the *next-state* or *transition function*. This transition function gives the next state in terms of the current state, the current input symbol, and the information currently in the temporary storage. During the transition from one time interval to the next, output may be produced or the information in the temporary storage changed. The term *configuration* will be used to refer to a particular state of the control unit, input file, and temporary storage. The transition of the automaton from one configuration to the next will be called a *move*.

This general model covers all the automata we will discuss in this lectures. A finite-state control will be common to all specific cases, but differences will arise from the way in which the output can be produced and the nature of the temporary storage. As we will see, the nature of the temporary storage governs the power of different types of automata.

An automaton is assumed to operate in a discrete timeframe. At any given time, the control unit is in some internal state, and the input mechanism is scanning a particular symbol on the input file. The internal state of the control unit at the next time step is determined by the *next-state* or *transition function*. This transition function gives the next state in terms of the current state, the current input symbol, and the information currently in the temporary storage. During the transition from one time interval to the next, output may be produced or the information in the temporary storage changed. The term *configuration* will be used to refer to a particular state of the control unit, input file, and temporary storage. The transition of the automaton from one configuration to the next will be called a *move*.

This general model covers all the automata we will discuss in this lectures. A finite-state control will be common to all specific cases, but differences will arise from the way in which the output can be produced and the nature of the temporary storage. As we will see, the nature of the temporary storage governs the power of different types of automata.

An automaton is assumed to operate in a discrete timeframe. At any given time, the control unit is in some internal state, and the input mechanism is scanning a particular symbol on the input file. The internal state of the control unit at the next time step is determined by the *next-state* or *transition function*. This transition function gives the next state in terms of the current state, the current input symbol, and the information currently in the temporary storage. During the transition from one time interval to the next, output may be produced or the information in the temporary storage changed. The term *configuration* will be used to refer to a particular state of the control unit, input file, and temporary storage. The transition of the automaton from one configuration to the next will be called a *move*.

This general model covers all the automata we will discuss in this lectures. A finite-state control will be common to all specific cases, but differences will arise from the way in which the output can be produced and the nature of the temporary storage. As we will see, the nature of the temporary storage governs the power of different types of automata.

An automaton is assumed to operate in a discrete timeframe. At any given time, the control unit is in some internal state, and the input mechanism is scanning a particular symbol on the input file. The internal state of the control unit at the next time step is determined by the *next-state* or *transition function*. This transition function gives the next state in terms of the current state, the current input symbol, and the information currently in the temporary storage. During the transition from one time interval to the next, output may be produced or the information in the temporary storage changed. The term *configuration* will be used to refer to a particular state of the control unit, input file, and temporary storage. The transition of the automaton from one configuration to the next will be called a *move*.

This general model covers all the automata we will discuss in this lectures. A finite-state control will be common to all specific cases, but differences will arise from the way in which the output can be produced and the nature of the temporary storage. As we will see, the nature of the temporary storage governs the power of different types of automata.

An automaton is assumed to operate in a discrete timeframe. At any given time, the control unit is in some internal state, and the input mechanism is scanning a particular symbol on the input file. The internal state of the control unit at the next time step is determined by the *next-state* or *transition function*. This transition function gives the next state in terms of the current state, the current input symbol, and the information currently in the temporary storage. During the transition from one time interval to the next, output may be produced or the information in the temporary storage changed. The term *configuration* will be used to refer to a particular state of the control unit, input file, and temporary storage. The transition of the automaton from one configuration to the next will be called a *move*.

This general model covers all the automata we will discuss in this lectures. A finite-state control will be common to all specific cases, but differences will arise from the way in which the output can be produced and the nature of the temporary storage. As we will see, the nature of the temporary storage governs the power of different types of automata.

An automaton is assumed to operate in a discrete timeframe. At any given time, the control unit is in some internal state, and the input mechanism is scanning a particular symbol on the input file. The internal state of the control unit at the next time step is determined by the *next-state* or *transition function*. This transition function gives the next state in terms of the current state, the current input symbol, and the information currently in the temporary storage. During the transition from one time interval to the next, output may be produced or the information in the temporary storage changed. The term *configuration* will be used to refer to a particular state of the control unit, input file, and temporary storage. The transition of the automaton from one configuration to the next will be called a *move*.

This general model covers all the automata we will discuss in this lectures. A finite-state control will be common to all specific cases, but differences will arise from the way in which the output can be produced and the nature of the temporary storage. As we will see, the nature of the temporary storage governs the power of different types of automata.

An automaton is assumed to operate in a discrete timeframe. At any given time, the control unit is in some internal state, and the input mechanism is scanning a particular symbol on the input file. The internal state of the control unit at the next time step is determined by the *next-state* or *transition function*. This transition function gives the next state in terms of the current state, the current input symbol, and the information currently in the temporary storage. During the transition from one time interval to the next, output may be produced or the information in the temporary storage changed. The term *configuration* will be used to refer to a particular state of the control unit, input file, and temporary storage. The transition of the automaton from one configuration to the next will be called a *move*.

This general model covers all the automata we will discuss in this lectures. A finite-state control will be common to all specific cases, but differences will arise from the way in which the output can be produced and the nature of the temporary storage. As we will see, the nature of the temporary storage governs the power of different types of automata.

An automaton is assumed to operate in a discrete timeframe. At any given time, the control unit is in some internal state, and the input mechanism is scanning a particular symbol on the input file. The internal state of the control unit at the next time step is determined by the *next-state* or *transition function*. This transition function gives the next state in terms of the current state, the current input symbol, and the information currently in the temporary storage. During the transition from one time interval to the next, output may be produced or the information in the temporary storage changed. The term *configuration* will be used to refer to a particular state of the control unit, input file, and temporary storage. The transition of the automaton from one configuration to the next will be called a *move*.

This general model covers all the automata we will discuss in this lectures. A finite-state control will be common to all specific cases, but differences will arise from the way in which the output can be produced and the nature of the temporary storage. As we will see, the nature of the temporary storage governs the power of different types of automata.

An automaton is assumed to operate in a discrete timeframe. At any given time, the control unit is in some internal state, and the input mechanism is scanning a particular symbol on the input file. The internal state of the control unit at the next time step is determined by the *next-state* or *transition function*. This transition function gives the next state in terms of the current state, the current input symbol, and the information currently in the temporary storage. During the transition from one time interval to the next, output may be produced or the information in the temporary storage changed. The term *configuration* will be used to refer to a particular state of the control unit, input file, and temporary storage. The transition of the automaton from one configuration to the next will be called a *move*.

This general model covers all the automata we will discuss in this lectures. A finite-state control will be common to all specific cases, but differences will arise from the way in which the output can be produced and the nature of the temporary storage. As we will see, the nature of the temporary storage governs the power of different types of automata.

For subsequent discussions, it will be necessary to distinguish between deterministic automata and nondeterministic automata. A deterministic automaton is one in which each move is uniquely determined by the current configuration. If we know the internal state, the input, and the contents of the temporary storage, we can predict the future behavior of the automaton exactly. In a nondeterministic automaton, this is not so. At each point, a nondeterministic automaton may have several possible moves, so we can only predict a set of possible actions. The relation between deterministic and nondeterministic automata of various types will play a significant role in our study.

An automaton whose output response is limited to a simple "yes" or "no" is called an *accepter*. Presented with an input string, an accepter either accepts the string or rejects it. A more general automaton, capable of producing strings of symbols as output, is called a transducer.

For subsequent discussions, it will be necessary to distinguish between deterministic automata and nondeterministic automata. A deterministic automaton is one in which each move is uniquely determined by the current configuration. If we know the internal state, the input, and the contents of the temporary storage, we can predict the future behavior of the automaton exactly. In a nondeterministic automaton, this is not so. At each point, a nondeterministic automaton may have several possible moves, so we can only predict a set of possible actions. The relation between deterministic and nondeterministic automata of various types will play a significant role in our study.

An automaton whose output response is limited to a simple "yes" or "no" is called an *accepter*. Presented with an input string, an accepter either accepts the string or rejects it. A more general automaton, capable of producing strings of symbols as output, is called a transducer.

For subsequent discussions, it will be necessary to distinguish between deterministic automata and nondeterministic automata. A deterministic automaton is one in which each move is uniquely determined by the current configuration. If we know the internal state, the input, and the contents of the temporary storage, we can predict the future behavior of the automaton exactly. In a nondeterministic automaton, this is not so. At each point, a nondeterministic automaton may have several possible moves, so we can only predict a set of possible actions. The relation between deterministic and nondeterministic automata of various types will play a significant role in our study.

An automaton whose output response is limited to a simple "yes" or "no" is called an *accepter*. Presented with an input string, an accepter either accepts the string or rejects it. A more general automaton, capable of producing strings of symbols as output, is called a transducer.

For subsequent discussions, it will be necessary to distinguish between deterministic automata and nondeterministic automata. A deterministic automaton is one in which each move is uniquely determined by the current configuration. If we know the internal state, the input, and the contents of the temporary storage, we can predict the future behavior of the automaton exactly. In a nondeterministic automaton, this is not so. At each point, a nondeterministic automaton may have several possible moves, so we can only predict a set of possible actions. The relation between deterministic and nondeterministic automata of various types will play a significant role in our study.

An automaton whose output response is limited to a simple "yes" or "no" is called an *accepter*. Presented with an input string, an accepter either accepts the string or rejects it. A more general automaton, capable of producing strings of symbols as output, is called a transducer.

For subsequent discussions, it will be necessary to distinguish between deterministic automata and nondeterministic automata. A deterministic automaton is one in which each move is uniquely determined by the current configuration. If we know the internal state, the input, and the contents of the temporary storage, we can predict the future behavior of the automaton exactly. In a nondeterministic automaton, this is not so. At each point, a nondeterministic automaton may have several possible moves, so we can only predict a set of possible actions. The relation between deterministic and nondeterministic automata of various types will play a significant role in our study.

An automaton whose output response is limited to a simple "yes" or "no" is called an *accepter*. Presented with an input string, an accepter either accepts the string or rejects it. A more general automaton, capable of producing strings of symbols as output, is called a transducer.

For subsequent discussions, it will be necessary to distinguish between deterministic automata and nondeterministic automata. A deterministic automaton is one in which each move is uniquely determined by the current configuration. If we know the internal state, the input, and the contents of the temporary storage, we can predict the future behavior of the automaton exactly. In a nondeterministic automaton, this is not so. At each point, a nondeterministic automaton may have several possible moves, so we can only predict a set of possible actions. The relation between deterministic and nondeterministic automata of various types will play a significant role in our study.

An automaton whose output response is limited to a simple "yes" or "no" is called an *accepter*. Presented with an input string, an accepter either accepts the string or rejects it. A more general automaton, capable of producing strings of symbols as output, is called a transducer.

For subsequent discussions, it will be necessary to distinguish between deterministic automata and nondeterministic automata. A deterministic automaton is one in which each move is uniquely determined by the current configuration. If we know the internal state, the input, and the contents of the temporary storage, we can predict the future behavior of the automaton exactly. In a nondeterministic automaton, this is not so. At each point, a nondeterministic automaton may have several possible moves, so we can only predict a set of possible actions. The relation between deterministic and nondeterministic automata of various types will play a significant role in our study.

An automaton whose output response is limited to a simple "yes" or "no" is called an *accepter*. Presented with an input string, an accepter either accepts the string or rejects it. A more general automaton, capable of producing strings of symbols as output, is called a transducer.

For subsequent discussions, it will be necessary to distinguish between deterministic automata and nondeterministic automata. A deterministic automaton is one in which each move is uniquely determined by the current configuration. If we know the internal state, the input, and the contents of the temporary storage, we can predict the future behavior of the automaton exactly. In a nondeterministic automaton, this is not so. At each point, a nondeterministic automaton may have several possible moves, so we can only predict a set of possible actions. The relation between deterministic and nondeterministic automata of various types will play a significant role in our study.

An automaton whose output response is limited to a simple “yes” or “no” is called an *accepter*. Presented with an input string, an accepter either accepts the string or rejects it. A more general automaton, capable of producing strings of symbols as output, is called a transducer.

For subsequent discussions, it will be necessary to distinguish between deterministic automata and nondeterministic automata. A deterministic automaton is one in which each move is uniquely determined by the current configuration. If we know the internal state, the input, and the contents of the temporary storage, we can predict the future behavior of the automaton exactly. In a nondeterministic automaton, this is not so. At each point, a nondeterministic automaton may have several possible moves, so we can only predict a set of possible actions. The relation between deterministic and nondeterministic automata of various types will play a significant role in our study.

An automaton whose output response is limited to a simple "yes" or "no" is called an *accepter*. Presented with an input string, an accepter either accepts the string or rejects it. A more general automaton, capable of producing strings of symbols as output, is called a transducer.

For subsequent discussions, it will be necessary to distinguish between deterministic automata and nondeterministic automata. A deterministic automaton is one in which each move is uniquely determined by the current configuration. If we know the internal state, the input, and the contents of the temporary storage, we can predict the future behavior of the automaton exactly. In a nondeterministic automaton, this is not so. At each point, a nondeterministic automaton may have several possible moves, so we can only predict a set of possible actions. The relation between deterministic and nondeterministic automata of various types will play a significant role in our study.

An automaton whose output response is limited to a simple "yes" or "no" is called an *accepter*. Presented with an input string, an accepter either accepts the string or rejects it. A more general automaton, capable of producing strings of symbols as output, is called a transducer.

For subsequent discussions, it will be necessary to distinguish between deterministic automata and nondeterministic automata. A deterministic automaton is one in which each move is uniquely determined by the current configuration. If we know the internal state, the input, and the contents of the temporary storage, we can predict the future behavior of the automaton exactly. In a nondeterministic automaton, this is not so. At each point, a nondeterministic automaton may have several possible moves, so we can only predict a set of possible actions. The relation between deterministic and nondeterministic automata of various types will play a significant role in our study.

An automaton whose output response is limited to a simple "yes" or "no" is called an *accepter*. Presented with an input string, an accepter either accepts the string or rejects it. A more general automaton, capable of producing strings of symbols as output, is called a transducer.

For subsequent discussions, it will be necessary to distinguish between deterministic automata and nondeterministic automata. A deterministic automaton is one in which each move is uniquely determined by the current configuration. If we know the internal state, the input, and the contents of the temporary storage, we can predict the future behavior of the automaton exactly. In a nondeterministic automaton, this is not so. At each point, a nondeterministic automaton may have several possible moves, so we can only predict a set of possible actions. The relation between deterministic and nondeterministic automata of various types will play a significant role in our study.

An automaton whose output response is limited to a simple "yes" or "no" is called an *accepter*. Presented with an input string, an accepter either accepts the string or rejects it. A more general automaton, capable of producing strings of symbols as output, is called a transducer.

For subsequent discussions, it will be necessary to distinguish between deterministic automata and nondeterministic automata. A deterministic automaton is one in which each move is uniquely determined by the current configuration. If we know the internal state, the input, and the contents of the temporary storage, we can predict the future behavior of the automaton exactly. In a nondeterministic automaton, this is not so. At each point, a nondeterministic automaton may have several possible moves, so we can only predict a set of possible actions. The relation between deterministic and nondeterministic automata of various types will play a significant role in our study.

An automaton whose output response is limited to a simple "yes" or "no" is called an *accepter*. Presented with an input string, an accepter either accepts the string or rejects it. A more general automaton, capable of producing strings of symbols as output, is called a transducer.

Thank You for attention!