Formal Languages, Automata and Codes

Oleg Gutik

Lecture 8

According to our definition, a language is regular if there exists a finite accepter for it. Therefore, every regular language can be described by some DFA or some NFA. Such a description can be very useful, for example, if we want to show the logic by which we decide if a given string is in a certain language. But in many instances, we need more concise ways of describing regular languages. In this series of lectures, we look at other ways of representing regular languages. These representations have important practical applications, a matter that is touched on in some of the examples and exercises.

One way of describing regular languages is via the notation of regular expressions. This notation involves a combination of strings of symbols from some alphabet Σ, parentheses, and the operators $+$, $\cdot$, and $*$. The simplest case is the language $\{a\}$, which will be denoted by the regular expression a. Slightly more complicated is the language $\{a, b, c\}$, for which, using the symbol $+$ to denote union, we have the regular expression $a + b + c$. We use the symbol $\cdot$ for concatenation and $*$ for star-closure in a similar way. The expression $(a + (b \cdot c))^*$ stands for the star-closure of $\{a\} \cup \{bc\}$, that is, the language $\{\lambda, a, bc, aa, abc, bca, bcbc, aaa, aabc, \ldots\}$.

According to our definition, a language is regular if there exists a finite accepter for it. Therefore, every regular language can be described by some DFA or some NFA. Such a description can be very useful, for example, if we want to show the logic by which we decide if a given string is in a certain language. But in many instances, we need more concise ways of describing regular languages. In this series of lectures, we look at other ways of representing regular languages. These representations have important practical applications, a matter that is touched on in some of the examples and exercises.

One way of describing regular languages is via the notation of regular expressions. This notation involves a combination of strings of symbols from some alphabet Σ, parentheses, and the operators $+$, $\cdot$, and $*$. The simplest case is the language $\{a\}$, which will be denoted by the regular expression a. Slightly more complicated is the language $\{a, b, c\}$, for which, using the symbol $+$ to denote union, we have the regular expression $a + b + c$. We use the symbol $\cdot$ for concatenation and $*$ for star-closure in a similar way. The expression $(a + (b \cdot c))^*$ stands for the star-closure of $\{a\} \cup \{bc\}$, that is, the language $\{\lambda, a, bc, aa, abc, bca, bcbc, aaa, aabc, \ldots\}$.

According to our definition, a language is regular if there exists a finite accepter for it. Therefore, every regular language can be described by some DFA or some NFA. Such a description can be very useful, for example, if we want to show the logic by which we decide if a given string is in a certain language. But in many instances, we need more concise ways of describing regular languages. In this series of lectures, we look at other ways of representing regular languages. These representations have important practical applications, a matter that is touched on in some of the examples and exercises.

One way of describing regular languages is via the notation of regular expressions. This notation involves a combination of strings of symbols from some alphabet Σ, parentheses, and the operators $+$, $\cdot$, and $*$. The simplest case is the language $\{a\}$, which will be denoted by the regular expression a. Slightly more complicated is the language $\{a, b, c\}$, for which, using the symbol $+$ to denote union, we have the regular expression $a + b + c$. We use the symbol $\cdot$ for concatenation and $*$ for star-closure in a similar way. The expression $(a + (b \cdot c))^*$ stands for the star-closure of $\{a\} \cup \{bc\}$, that is, the language $\{\lambda, a, bc, aa, abc, bca, bcbc, aaa, aabc, \ldots\}$.

According to our definition, a language is regular if there exists a finite accepter for it. Therefore, every regular language can be described by some DFA or some NFA. Such a description can be very useful, for example, if we want to show the logic by which we decide if a given string is in a certain language. But in many instances, we need more concise ways of describing regular languages. In this series of lectures, we look at other ways of representing regular languages. These representations have important practical applications, a matter that is touched on in some of the examples and exercises.

One way of describing regular languages is via the notation of regular expressions. This notation involves a combination of strings of symbols from some alphabet Σ, parentheses, and the operators $+$, $\cdot$, and $*$. The simplest case is the language $\{a\}$, which will be denoted by the regular expression a. Slightly more complicated is the language $\{a, b, c\}$, for which, using the symbol $+$ to denote union, we have the regular expression $a + b + c$. We use the symbol $\cdot$ for concatenation and $*$ for star-closure in a similar way. The expression $(a + (b \cdot c))^*$ stands for the star-closure of $\{a\} \cup \{bc\}$, that is, the language $\{\lambda, a, bc, aa, abc, bca, bcbc, aaa, aabc, \ldots\}$.

According to our definition, a language is regular if there exists a finite accepter for it. Therefore, every regular language can be described by some DFA or some NFA. Such a description can be very useful, for example, if we want to show the logic by which we decide if a given string is in a certain language. But in many instances, we need more concise ways of describing regular languages. In this series of lectures, we look at other ways of representing regular languages. These representations have important practical applications, a matter that is touched on in some of the examples and exercises.

One way of describing regular languages is via the notation of regular expressions. This notation involves a combination of strings of symbols from some alphabet Σ, parentheses, and the operators $+$, $\cdot$, and $*$. The simplest case is the language $\{a\}$, which will be denoted by the regular expression a. Slightly more complicated is the language $\{a, b, c\}$, for which, using the symbol $+$ to denote union, we have the regular expression $a + b + c$. We use the symbol $\cdot$ for concatenation and $*$ for star-closure in a similar way. The expression $(a + (b \cdot c))^*$ stands for the star-closure of $\{a\} \cup \{bc\}$, that is, the language $\{\lambda, a, bc, aa, abc, bca, bcbc, aaa, aabc, \ldots\}$.

According to our definition, a language is regular if there exists a finite accepter for it. Therefore, every regular language can be described by some DFA or some NFA. Such a description can be very useful, for example, if we want to show the logic by which we decide if a given string is in a certain language. But in many instances, we need more concise ways of describing regular languages. In this series of lectures, we look at other ways of representing regular languages. These representations have important practical applications, a matter that is touched on in some of the examples and exercises.

One way of describing regular languages is via the notation of regular expressions. This notation involves a combination of strings of symbols from some alphabet Σ, parentheses, and the operators $+$, $\cdot$, and $*$. The simplest case is the language $\{a\}$, which will be denoted by the regular expression a. Slightly more complicated is the language $\{a, b, c\}$, for which, using the symbol $+$ to denote union, we have the regular expression $a + b + c$. We use the symbol $\cdot$ for concatenation and $*$ for star-closure in a similar way. The expression $(a + (b \cdot c))^*$ stands for the star-closure of $\{a\} \cup \{bc\}$, that is, the language $\{\lambda, a, bc, aa, abc, bca, bcbc, aaa, aabc, \ldots\}$.

According to our definition, a language is regular if there exists a finite accepter for it. Therefore, every regular language can be described by some DFA or some NFA. Such a description can be very useful, for example, if we want to show the logic by which we decide if a given string is in a certain language. But in many instances, we need more concise ways of describing regular languages. In this series of lectures, we look at other ways of representing regular languages. These representations have important practical applications, a matter that is touched on in some of the examples and exercises.

One way of describing regular languages is via the notation of regular expressions. This notation involves a combination of strings of symbols from some alphabet Σ, parentheses, and the operators $+$, $\cdot$, and $*$. The simplest case is the language $\{a\}$, which will be denoted by the regular expression a. Slightly more complicated is the language $\{a, b, c\}$, for which, using the symbol $+$ to denote union, we have the regular expression $a + b + c$. We use the symbol $\cdot$ for concatenation and $*$ for star-closure in a similar way. The expression $(a + (b \cdot c))^*$ stands for the star-closure of $\{a\} \cup \{bc\}$, that is, the language $\{\lambda, a, bc, aa, abc, bca, bcbc, aaa, aabc, \ldots\}$.

According to our definition, a language is regular if there exists a finite accepter for it. Therefore, every regular language can be described by some DFA or some NFA. Such a description can be very useful, for example, if we want to show the logic by which we decide if a given string is in a certain language. But in many instances, we need more concise ways of describing regular languages. In this series of lectures, we look at other ways of representing regular languages. These representations have important practical applications, a matter that is touched on in some of the examples and exercises.

One way of describing regular languages is via the notation of regular expressions. This notation involves a combination of strings of symbols from some alphabet Σ, parentheses, and the operators $+$, $\cdot$, and $*$. The simplest case is the language $\{a\}$, which will be denoted by the regular expression a. Slightly more complicated is the language $\{a, b, c\}$, for which, using the symbol $+$ to denote union, we have the regular expression $a + b + c$. We use the symbol $\cdot$ for concatenation and $*$ for star-closure in a similar way. The expression $(a + (b \cdot c))^*$ stands for the star-closure of $\{a\} \cup \{bc\}$, that is, the language $\{\lambda, a, bc, aa, abc, bca, bcbc, aaa, aabc, \dots\}$.

According to our definition, a language is regular if there exists a finite accepter for it. Therefore, every regular language can be described by some DFA or some NFA. Such a description can be very useful, for example, if we want to show the logic by which we decide if a given string is in a certain language. But in many instances, we need more concise ways of describing regular languages. In this series of lectures, we look at other ways of representing regular languages. These representations have important practical applications, a matter that is touched on in some of the examples and exercises.

One way of describing regular languages is via the notation of regular expressions. This notation involves a combination of strings of symbols from some alphabet Σ, parentheses, and the operators $+$, $\cdot$, and $*$. The simplest case is the language $\{a\}$, which will be denoted by the regular expression a. Slightly more complicated is the language $\{a, b, c\}$, for which, using the symbol $+$ to denote union, we have the regular expression $a + b + c$. We use the symbol $\cdot$ for concatenation and $*$ for star-closure in a similar way. The expression $(a + (b \cdot c))^*$ stands for the star-closure of $\{a\} \cup \{bc\}$, that is, the language $\{\lambda, a, bc, aa, abc, bca, bcbc, aaa, aabc, \ldots\}$.

According to our definition, a language is regular if there exists a finite accepter for it. Therefore, every regular language can be described by some DFA or some NFA. Such a description can be very useful, for example, if we want to show the logic by which we decide if a given string is in a certain language. But in many instances, we need more concise ways of describing regular languages. In this series of lectures, we look at other ways of representing regular languages. These representations have important practical applications, a matter that is touched on in some of the examples and exercises.

One way of describing regular languages is via the notation of regular expressions. This notation involves a combination of strings of symbols from some alphabet Σ, parentheses, and the operators $+$, $\cdot$, and $*$. The simplest case is the language $\{a\}$, which will be denoted by the regular expression a. Slightly more complicated is the language $\{a, b, c\}$, for which, using the symbol $+$ to denote union, we have the regular expression $a + b + c$. We use the symbol $\cdot$ for concatenation and $*$ for star-closure in a similar way. The expression $(a + (b \cdot c))^*$ stands for the star-closure of $\{a\} \cup \{bc\}$, that is, the language $\{\lambda, a, bc, aa, abc, bca, bcbc, aaa, aabc, \ldots\}$.

According to our definition, a language is regular if there exists a finite accepter for it. Therefore, every regular language can be described by some DFA or some NFA. Such a description can be very useful, for example, if we want to show the logic by which we decide if a given string is in a certain language. But in many instances, we need more concise ways of describing regular languages. In this series of lectures, we look at other ways of representing regular languages. These representations have important practical applications, a matter that is touched on in some of the examples and exercises.

One way of describing regular languages is via the notation of regular expressions. This notation involves a combination of strings of symbols from some alphabet Σ, parentheses, and the operators $+$, $\cdot$, and $*$. The simplest case is the language $\{a\}$, which will be denoted by the regular expression a. Slightly more complicated is the language $\{a, b, c\}$, for which, using the symbol $+$ to denote union, we have the regular expression $a + b + c$. We use the symbol $\cdot$ for concatenation and $*$ for star-closure in a similar way. The expression $(a + (b \cdot c))^*$ stands for the star-closure of $\{a\} \cup \{bc\}$, that is, the language $\{\lambda, a, bc, aa, abc, bca, bcbc, aaa, aabc, \ldots\}$.

According to our definition, a language is regular if there exists a finite accepter for it. Therefore, every regular language can be described by some DFA or some NFA. Such a description can be very useful, for example, if we want to show the logic by which we decide if a given string is in a certain language. But in many instances, we need more concise ways of describing regular languages. In this series of lectures, we look at other ways of representing regular languages. These representations have important practical applications, a matter that is touched on in some of the examples and exercises.

One way of describing regular languages is via the notation of regular expressions. This notation involves a combination of strings of symbols from some alphabet Σ, parentheses, and the operators $+$, $\cdot$, and $*$. The simplest case is the language $\{a\}$, which will be denoted by the regular expression a. Slightly more complicated is the language $\{a, b, c\}$, for which, using the symbol $+$ to denote union, we have the regular expression $a + b + c$. We use the symbol $\cdot$ for concatenation and $*$ for star-closure in a similar way. The expression $(a + (b \cdot c))^*$ stands for the star-closure of $\{a\} \cup \{bc\}$, that is, the language $\{\lambda, a, bc, aa, abc, bca, bcbc, aaa, aabc, \ldots\}$.

According to our definition, a language is regular if there exists a finite accepter for it. Therefore, every regular language can be described by some DFA or some NFA. Such a description can be very useful, for example, if we want to show the logic by which we decide if a given string is in a certain language. But in many instances, we need more concise ways of describing regular languages. In this series of lectures, we look at other ways of representing regular languages. These representations have important practical applications, a matter that is touched on in some of the examples and exercises.

One way of describing regular languages is via the notation of regular expressions. This notation involves a combination of strings of symbols from some alphabet Σ, parentheses, and the operators $+$, $\cdot$, and $*$. The simplest case is the language $\{a\}$, which will be denoted by the regular expression a. Slightly more complicated is the language $\{a, b, c\}$, for which, using the symbol $+$ to denote union, we have the regular expression $a + b + c$. We use the symbol $\cdot$ for concatenation and $*$ for star-closure in a similar way. The expression $(a + (b \cdot c))^*$ stands for the star-closure of $\{a\} \cup \{bc\}$, that is, the language $\{\lambda, a, bc, aa, abc, bca, bcbc, aaa, aabc, \ldots\}$.

According to our definition, a language is regular if there exists a finite accepter for it. Therefore, every regular language can be described by some DFA or some NFA. Such a description can be very useful, for example, if we want to show the logic by which we decide if a given string is in a certain language. But in many instances, we need more concise ways of describing regular languages. In this series of lectures, we look at other ways of representing regular languages. These representations have important practical applications, a matter that is touched on in some of the examples and exercises.

One way of describing regular languages is via the notation of regular expressions. This notation involves a combination of strings of symbols from some alphabet Σ, parentheses, and the operators $+$, $\cdot$, and $*$. The simplest case is the language $\{a\}$, which will be denoted by the regular expression a. Slightly more complicated is the language $\{a, b, c\}$, for which, using the symbol $+$ to denote union, we have the regular expression $a + b + c$. We use the symbol $\cdot$ for concatenation and $*$ for star-closure in a similar way. The expression $(a + (b \cdot c))^*$ stands for the star-closure of $\{a\} \cup \{bc\}$, that is, the language $\{\lambda, a, bc, aa, abc, bca, bcbc, aaa, aabc, \ldots\}$.

According to our definition, a language is regular if there exists a finite accepter for it. Therefore, every regular language can be described by some DFA or some NFA. Such a description can be very useful, for example, if we want to show the logic by which we decide if a given string is in a certain language. But in many instances, we need more concise ways of describing regular languages. In this series of lectures, we look at other ways of representing regular languages. These representations have important practical applications, a matter that is touched on in some of the examples and exercises.

One way of describing regular languages is via the notation of regular expressions. This notation involves a combination of strings of symbols from some alphabet Σ, parentheses, and the operators $+$, $\cdot$, and $*$. The simplest case is the language $\{a\}$, which will be denoted by the regular expression a. Slightly more complicated is the language $\{a, b, c\}$, for which, using the symbol $+$ to denote union, we have the regular expression $a + b + c$. We use the symbol $\cdot$ for concatenation and $*$ for star-closure in a similar way. The expression $(a + (b \cdot c))^*$ stands for the star-closure of $\{a\} \cup \{bc\}$, that is, the language $\{\lambda, a, bc, aa, abc, bca, bcbc, aaa, aabc, \ldots\}$.

3.1 Regular Expressions

According to our definition, a language is regular if there exists a finite accepter for it. Therefore, every regular language can be described by some DFA or some NFA. Such a description can be very useful, for example, if we want to show the logic by which we decide if a given string is in a certain language. But in many instances, we need more concise ways of describing regular languages. In this series of lectures, we look at other ways of representing regular languages. These representations have important practical applications, a matter that is touched on in some of the examples and exercises.

One way of describing regular languages is via the notation of regular expressions. This notation involves a combination of strings of symbols from some alphabet Σ, parentheses, and the operators $+$, $\cdot$, and $*$. The simplest case is the language $\{a\}$, which will be denoted by the regular expression a. Slightly more complicated is the language $\{a, b, c\}$, for which, using the symbol $+$ to denote union, we have the regular expression $a + b + c$. We use the symbol $\cdot$ for concatenation and $*$ for star-closure in a similar way. The expression $(a + (b \cdot c))^*$ stands for the star-closure of $\{a\} \cup \{bc\}$, that is, the language $\{\lambda, a, bc, aa, abc, bca, bcbc, aaa, aabc, \ldots\}$.

According to our definition, a language is regular if there exists a finite accepter for it. Therefore, every regular language can be described by some DFA or some NFA. Such a description can be very useful, for example, if we want to show the logic by which we decide if a given string is in a certain language. But in many instances, we need more concise ways of describing regular languages. In this series of lectures, we look at other ways of representing regular languages. These representations have important practical applications, a matter that is touched on in some of the examples and exercises.

One way of describing regular languages is via the notation of regular expressions. This notation involves a combination of strings of symbols from some alphabet Σ, parentheses, and the operators $+$, $\cdot$, and $*$. The simplest case is the language $\{a\}$, which will be denoted by the regular expression a. Slightly more complicated is the language $\{a, b, c\}$, for which, using the symbol $+$ to denote union, we have the regular expression $a + b + c$. We use the symbol $\cdot$ for concatenation and $*$ for star-closure in a similar way. The expression $(a + (b \cdot c))^*$ stands for the star-closure of $\{a\} \cup \{bc\}$, that is, the language $\{\lambda, a, bc, aa, abc, bca, bcbc, aaa, aabc, \ldots\}$.

According to our definition, a language is regular if there exists a finite accepter for it. Therefore, every regular language can be described by some DFA or some NFA. Such a description can be very useful, for example, if we want to show the logic by which we decide if a given string is in a certain language. But in many instances, we need more concise ways of describing regular languages. In this series of lectures, we look at other ways of representing regular languages. These representations have important practical applications, a matter that is touched on in some of the examples and exercises.

One way of describing regular languages is via the notation of regular expressions. This notation involves a combination of strings of symbols from some alphabet Σ, parentheses, and the operators $+$, $\cdot$, and $*$. The simplest case is the language $\{a\}$, which will be denoted by the regular expression a. Slightly more complicated is the language $\{a, b, c\}$, for which, using the symbol $+$ to denote union, we have the regular expression $a + b + c$. We use the symbol $\cdot$ for concatenation and $*$ for star-closure in a similar way. The expression $(a + (b \cdot c))^*$ stands for the star-closure of $\{a\} \cup \{bc\}$, that is, the language $\{\lambda, a, bc, aa, abc, bca, bcbc, aaa, aabc, \ldots\}$.

According to our definition, a language is regular if there exists a finite accepter for it. Therefore, every regular language can be described by some DFA or some NFA. Such a description can be very useful, for example, if we want to show the logic by which we decide if a given string is in a certain language. But in many instances, we need more concise ways of describing regular languages. In this series of lectures, we look at other ways of representing regular languages. These representations have important practical applications, a matter that is touched on in some of the examples and exercises.

One way of describing regular languages is via the notation of regular expressions. This notation involves a combination of strings of symbols from some alphabet Σ, parentheses, and the operators $+$, $\cdot$, and $*$. The simplest case is the language $\{a\}$, which will be denoted by the regular expression a. Slightly more complicated is the language $\{a, b, c\}$, for which, using the symbol $+$ to denote union, we have the regular expression $a + b + c$. We use the symbol $\cdot$ for concatenation and $*$ for star-closure in a similar way. The expression $(a + (b \cdot c))^*$ stands for the star-closure of $\{a\} \cup \{bc\}$, that is, the language $\{\lambda, a, bc, aa, abc, bca, bcbc, aaa, aabc, \ldots\}$.

Formal Definition of a Regular Expression

We construct regular expressions from primitive constituents by repeatedly applying certain recursive rules. This is similar to the way we construct familiar arithmetic expressions.

Definition 3.1

Let Σ be a given alphabet. Then

1. $\varnothing$, λ, and $a \in \Sigma$ are all regular expressions.

2. If r_1 and r_2 are regular expressions, so are $r_1 + r_2$, $r_1 \cdot r_2$, r_1^*, and (r_1).
3. A string is a regular expression if and only if it can be derived from the primitive regular expressions by a finite number of applications of the rules in item 2.

Example 3.1

For $\Sigma = \{a, b, c\}$, the string

$$(a + b \cdot c)^* \cdot (c + \varnothing)$$

is a regular expression, since it is constructed by application of the above rules. For example, if we take $r_1 = c$ and $r_2 = \varnothing$, we find that $c + \varnothing$ and $(c + \varnothing)$ are also regular expressions. Repeating this, we eventually generate the whole string. On the other hand, $(a + b+)$ is not a regular expression, because there is no way it can be constructed from the primitive regular expressions.

Formal Definition of a Regular Expression

We construct regular expressions from primitive constituents by repeatedly applying certain recursive rules. This is similar to the way we construct familiar arithmetic expressions.

Definition 3.1

Let Σ be a given alphabet. Then

1. $\varnothing$, λ, and $a \in \Sigma$ are all regular expressions.
2. If r_1 and r_2 are regular expressions, so are $r_1 + r_2$, $r_1 \cdot r_2$, r_1^*, and (r_1).
3. A string is a regular expression if and only if it can be derived from the primitive regular expressions by a finite number of applications of the rules in item 2.

Example 3.1

For $\Sigma = \{a, b, c\}$, the string

$$(a + b \cdot c)^* \cdot (c + \varnothing)$$

is a regular expression, since it is constructed by application of the above rules. For example, if we take $r_1 = c$ and $r_2 = \varnothing$, we find that $c + \varnothing$ and $(c + \varnothing)$ are also regular expressions. Repeating this, we eventually generate the whole string. On the other hand, $(a + b+)$ is not a regular expression, because there is no way it can be constructed from the primitive regular expressions.

Formal Definition of a Regular Expression

We construct regular expressions from primitive constituents by repeatedly applying certain recursive rules. This is similar to the way we construct familiar arithmetic expressions.

Definition 3.1

Let Σ be a given alphabet. Then

1. $\varnothing$, λ, and $a \in \Sigma$ are all regular expressions.
2. If r_1 and r_2 are regular expressions, so are $r_1 + r_2$, $r_1 \cdot r_2$, r_1^*, and (r_1).
3. A string is a regular expression if and only if it can be derived from the primitive regular expressions by a finite number of applications of the rules in item 2.

Example 3.1

For $\Sigma = \{a, b, c\}$, the string

$$(a + b \cdot c)^* \cdot (c + \varnothing)$$

is a regular expression, since it is constructed by application of the above rules. For example, if we take $r_1 = c$ and $r_2 = \varnothing$, we find that $c + \varnothing$ and $(c + \varnothing)$ are also regular expressions. Repeating this, we eventually generate the whole string. On the other hand, $(a + b+)$ is not a regular expression, because there is no way it can be constructed from the primitive regular expressions.

Formal Definition of a Regular Expression

We construct regular expressions from primitive constituents by repeatedly applying certain recursive rules. This is similar to the way we construct familiar arithmetic expressions.

Definition 3.1

Let Σ be a given alphabet. Then

1. $\varnothing$, λ, and $a \in \Sigma$ are all regular expressions.
2. If r_1 and r_2 are regular expressions, so are $r_1 + r_2$, $r_1 \cdot r_2$, r_1^*, and (r_1).
3. A string is a regular expression if and only if it can be derived from the primitive regular expressions by a finite number of applications of the rules in item 2.

Example 3.1

For $\Sigma = \{a, b, c\}$, the string

$$(a + b \cdot c)^* \cdot (c + \varnothing)$$

is a regular expression, since it is constructed by application of the above rules. For example, if we take $r_1 = c$ and $r_2 = \varnothing$, we find that $c + \varnothing$ and $(c + \varnothing)$ are also regular expressions. Repeating this, we eventually generate the whole string. On the other hand, $(a + b+)$ is not a regular expression, because there is no way it can be constructed from the primitive regular expressions.

Formal Definition of a Regular Expression

We construct regular expressions from primitive constituents by repeatedly applying certain recursive rules. This is similar to the way we construct familiar arithmetic expressions.

Definition 3.1

Let Σ be a given alphabet. Then

1. $\varnothing$, λ, and $a \in \Sigma$ are all regular expressions.
2. If r_1 and r_2 are regular expressions, so are $r_1 + r_2$, $r_1 \cdot r_2$, r_1^*, and (r_1).
3. A string is a regular expression if and only if it can be derived from the primitive regular expressions by a finite number of applications of the rules in item 2.

Example 3.1

For $\Sigma = \{a, b, c\}$, the string

$$(a + b \cdot c)^* \cdot (c + \varnothing)$$

is a regular expression, since it is constructed by application of the above rules. For example, if we take $r_1 = c$ and $r_2 = \varnothing$, we find that $c + \varnothing$ and $(c + \varnothing)$ are also regular expressions. Repeating this, we eventually generate the whole string. On the other hand, $(a + b+)$ is not a regular expression, because there is no way it can be constructed from the primitive regular expressions.

Formal Definition of a Regular Expression

We construct regular expressions from primitive constituents by repeatedly applying certain recursive rules. This is similar to the way we construct familiar arithmetic expressions.

Definition 3.1

Let Σ be a given alphabet. Then

1. $\varnothing$, λ, and $a \in \Sigma$ are all regular expressions.
2. If r_1 and r_2 are regular expressions, so are $r_1 + r_2$, $r_1 \cdot r_2$, r_1^*, and (r_1).
3. A string is a regular expression if and only if it can be derived from the primitive regular expressions by a finite number of applications of the rules in item 2.

Example 3.1

For $\Sigma = \{a, b, c\}$, the string

$$(a + b \cdot c)^* \cdot (c + \varnothing)$$

is a regular expression, since it is constructed by application of the above rules. For example, if we take $r_1 = c$ and $r_2 = \varnothing$, we find that $c + \varnothing$ and $(c + \varnothing)$ are also regular expressions. Repeating this, we eventually generate the whole string. On the other hand, $(a + b+)$ is not a regular expression, because there is no way it can be constructed from the primitive regular expressions.

3.1 Regular Expressions

Formal Definition of a Regular Expression

We construct regular expressions from primitive constituents by repeatedly applying certain recursive rules. This is similar to the way we construct familiar arithmetic expressions.

Definition 3.1

Let Σ be a given alphabet. Then

1. $\varnothing$, λ, and $a \in \Sigma$ are all regular expressions.
2. If r_1 and r_2 are regular expressions, so are $r_1 + r_2$, $r_1 \cdot r_2$, r_1^*, and (r_1).
3. A string is a regular expression if and only if it can be derived from the primitive regular expressions by a finite number of applications of the rules in item 2.

Example 3.1

For $\Sigma = \{a, b, c\}$, the string

$$(a + b \cdot c)^* \cdot (c + \varnothing)$$

is a regular expression, since it is constructed by application of the above rules. For example, if we take $r_1 = c$ and $r_2 = \varnothing$, we find that $c + \varnothing$ and $(c + \varnothing)$ are also regular expressions. Repeating this, we eventually generate the whole string. On the other hand, $(a + b+)$ is not a regular expression, because there is no way it can be constructed from the primitive regular expressions.

3.1 Regular Expressions

Formal Definition of a Regular Expression

We construct regular expressions from primitive constituents by repeatedly applying certain recursive rules. This is similar to the way we construct familiar arithmetic expressions.

Definition 3.1

Let Σ be a given alphabet. Then

1. $\varnothing$, λ, and $a \in \Sigma$ are all regular expressions. These are called *primitive regular expressions*.
2. If r_1 and r_2 are regular expressions, so are $r_1 + r_2$, $r_1 \cdot r_2$, r_1^*, and (r_1).
3. A string is a regular expression if and only if it can be derived from the primitive regular expressions by a finite number of applications of the rules in item 2.

Example 3.1

For $\Sigma = \{a, b, c\}$, the string

$$(a + b \cdot c)^* \cdot (c + \varnothing)$$

is a regular expression, since it is constructed by application of the above rules. For example, if we take $r_1 = c$ and $r_2 = \varnothing$, we find that $c + \varnothing$ and $(c + \varnothing)$ are also regular expressions. Repeating this, we eventually generate the whole string. On the other hand, $(a + b+)$ is not a regular expression, because there is no way it can be constructed from the primitive regular expressions.

Formal Definition of a Regular Expression

We construct regular expressions from primitive constituents by repeatedly applying certain recursive rules. This is similar to the way we construct familiar arithmetic expressions.

Definition 3.1

Let Σ be a given alphabet. Then

1. $\varnothing$, λ, and $a \in \Sigma$ are all regular expressions. These are called *primitive regular expressions*.
2. If r_1 and r_2 are regular expressions, so are $r_1 + r_2$, $r_1 \cdot r_2$, r_1^*, and (r_1).
3. A string is a regular expression if and only if it can be derived from the primitive regular expressions by a finite number of applications of the rules in item 2.

Example 3.1

For $\Sigma = \{a, b, c\}$, the string

$$(a + b \cdot c)^* \cdot (c + \varnothing)$$

is a regular expression, since it is constructed by application of the above rules. For example, if we take $r_1 = c$ and $r_2 = \varnothing$, we find that $c + \varnothing$ and $(c + \varnothing)$ are also regular expressions. Repeating this, we eventually generate the whole string. On the other hand, $(a + b+)$ is not a regular expression, because there is no way it can be constructed from the primitive regular expressions.

Formal Definition of a Regular Expression

We construct regular expressions from primitive constituents by repeatedly applying certain recursive rules. This is similar to the way we construct familiar arithmetic expressions.

Definition 3.1

Let Σ be a given alphabet. Then

1. $\varnothing$, λ, and $a \in \Sigma$ are all regular expressions. These are called *primitive regular expressions*.
2. If r_1 and r_2 are regular expressions, so are $r_1 + r_2$, $r_1 \cdot r_2$, r_1^*, and (r_1).
3. A string is a regular expression if and only if it can be derived from the primitive regular expressions by a finite number of applications of the rules in item 2.

Example 3.1

For $\Sigma = \{a, b, c\}$, the string

$$(a + b \cdot c)^* \cdot (c + \varnothing)$$

is a regular expression, since it is constructed by application of the above rules. For example, if we take $r_1 = c$ and $r_2 = \varnothing$, we find that $c + \varnothing$ and $(c + \varnothing)$ are also regular expressions. Repeating this, we eventually generate the whole string. On the other hand, $(a + b+)$ is not a regular expression, because there is no way it can be constructed from the primitive regular expressions.

3.1 Regular Expressions

Formal Definition of a Regular Expression

We construct regular expressions from primitive constituents by repeatedly applying certain recursive rules. This is similar to the way we construct familiar arithmetic expressions.

Definition 3.1

Let Σ be a given alphabet. Then

1. $\varnothing$, λ, and $a \in \Sigma$ are all regular expressions. These are called *primitive regular expressions*.
2. If r_1 and r_2 are regular expressions, so are $r_1 + r_2$, $r_1 \cdot r_2$, r_1^*, and (r_1).
3. A string is a regular expression if and only if it can be derived from the primitive regular expressions by a finite number of applications of the rules in item 2.

Example 3.1

For $\Sigma = \{a, b, c\}$, the string

$$(a + b \cdot c)^* \cdot (c + \varnothing)$$

is a regular expression, since it is constructed by application of the above rules. For example, if we take $r_1 = c$ and $r_2 = \varnothing$, we find that $c + \varnothing$ and $(c + \varnothing)$ are also regular expressions. Repeating this, we eventually generate the whole string. On the other hand, $(a + b+)$ is not a regular expression, because there is no way it can be constructed from the primitive regular expressions.

Formal Definition of a Regular Expression

We construct regular expressions from primitive constituents by repeatedly applying certain recursive rules. This is similar to the way we construct familiar arithmetic expressions.

Definition 3.1

Let Σ be a given alphabet. Then

1. $\varnothing$, λ, and $a \in \Sigma$ are all regular expressions. These are called *primitive regular expressions*.
2. If r_1 and r_2 are regular expressions, so are $r_1 + r_2$, $r_1 \cdot r_2$, r_1^*, and (r_1).
3. A string is a regular expression if and only if it can be derived from the primitive regular expressions by a finite number of applications of the rules in item 2.

Example 3.1

For $\Sigma = \{a, b, c\}$, the string

$$(a + b \cdot c)^* \cdot (c + \varnothing)$$

is a regular expression, since it is constructed by application of the above rules. For example, if we take $r_1 = c$ and $r_2 = \varnothing$, we find that $c + \varnothing$ and $(c + \varnothing)$ are also regular expressions. Repeating this, we eventually generate the whole string. On the other hand, $(a + b+)$ is not a regular expression, because there is no way it can be constructed from the primitive regular expressions.

Formal Definition of a Regular Expression

We construct regular expressions from primitive constituents by repeatedly applying certain recursive rules. This is similar to the way we construct familiar arithmetic expressions.

Definition 3.1

Let Σ be a given alphabet. Then

1. $\varnothing$, λ, and $a \in \Sigma$ are all regular expressions. These are called *primitive regular expressions*.
2. If r_1 and r_2 are regular expressions, so are $r_1 + r_2$, $r_1 \cdot r_2$, r_1^*, and (r_1).
3. A string is a regular expression if and only if it can be derived from the primitive regular expressions by a finite number of applications of the rules in item 2.

Example 3.1

For $\Sigma = \{a, b, c\}$, the string

$$(a + b \cdot c)^* \cdot (c + \varnothing)$$

is a regular expression, since it is constructed by application of the above rules. For example, if we take $r_1 = c$ and $r_2 = \varnothing$, we find that $c + \varnothing$ and $(c + \varnothing)$ are also regular expressions. Repeating this, we eventually generate the whole string. On the other hand, $(a + b+)$ is not a regular expression, because there is no way it can be constructed from the primitive regular expressions.

Formal Definition of a Regular Expression

We construct regular expressions from primitive constituents by repeatedly applying certain recursive rules. This is similar to the way we construct familiar arithmetic expressions.

Definition 3.1

Let Σ be a given alphabet. Then

1. $\varnothing$, λ, and $a \in \Sigma$ are all regular expressions. These are called *primitive regular expressions*.
2. If r_1 and r_2 are regular expressions, so are $r_1 + r_2$, $r_1 \cdot r_2$, r_1^*, and (r_1).
3. A string is a regular expression if and only if it can be derived from the primitive regular expressions by a finite number of applications of the rules in item 2.

Example 3.1

For $\Sigma = \{a, b, c\}$, the string

$$(a + b \cdot c)^* \cdot (c + \varnothing)$$

is a regular expression, since it is constructed by application of the above rules. For example, if we take $r_1 = c$ and $r_2 = \varnothing$, we find that $c + \varnothing$ and $(c + \varnothing)$ are also regular expressions. Repeating this, we eventually generate the whole string. On the other hand, $(a + b+)$ is not a regular expression, because there is no way it can be constructed from the primitive regular expressions.

Formal Definition of a Regular Expression

We construct regular expressions from primitive constituents by repeatedly applying certain recursive rules. This is similar to the way we construct familiar arithmetic expressions.

Definition 3.1

Let Σ be a given alphabet. Then

1. $\varnothing$, λ, and $a \in \Sigma$ are all regular expressions. These are called *primitive regular expressions*.
2. If r_1 and r_2 are regular expressions, so are $r_1 + r_2$, $r_1 \cdot r_2$, r_1^*, and (r_1).
3. A string is a regular expression if and only if it can be derived from the primitive regular expressions by a finite number of applications of the rules in item 2.

Example 3.1

For $\Sigma = \{a, b, c\}$, the string

$$(a + b \cdot c)^* \cdot (c + \varnothing)$$

is a regular expression, since it is constructed by application of the above rules. For example, if we take $r_1 = c$ and $r_2 = \varnothing$, we find that $c + \varnothing$ and $(c + \varnothing)$ are also regular expressions. Repeating this, we eventually generate the whole string. On the other hand, $(a + b+)$ is not a regular expression, because there is no way it can be constructed from the primitive regular expressions.

Formal Definition of a Regular Expression

We construct regular expressions from primitive constituents by repeatedly applying certain recursive rules. This is similar to the way we construct familiar arithmetic expressions.

Definition 3.1

Let Σ be a given alphabet. Then

1. $\varnothing$, λ, and $a \in \Sigma$ are all regular expressions. These are called *primitive regular expressions*.
2. If r_1 and r_2 are regular expressions, so are $r_1 + r_2$, $r_1 \cdot r_2$, r_1^*, and (r_1).
3. A string is a regular expression if and only if it can be derived from the primitive regular expressions by a finite number of applications of the rules in item 2.

Example 3.1

For $\Sigma = \{a, b, c\}$, the string

$$(a + b \cdot c)^* \cdot (c + \varnothing)$$

is a regular expression, since it is constructed by application of the above rules. For example, if we take $r_1 = c$ and $r_2 = \varnothing$, we find that $c + \varnothing$ and $(c + \varnothing)$ are also regular expressions. Repeating this, we eventually generate the whole string. On the other hand, $(a + b+)$ is not a regular expression, because there is no way it can be constructed from the primitive regular expressions.

Formal Definition of a Regular Expression

We construct regular expressions from primitive constituents by repeatedly applying certain recursive rules. This is similar to the way we construct familiar arithmetic expressions.

Definition 3.1

Let Σ be a given alphabet. Then

1. $\varnothing$, λ, and $a \in \Sigma$ are all regular expressions. These are called *primitive regular expressions*.
2. If r_1 and r_2 are regular expressions, so are $r_1 + r_2$, $r_1 \cdot r_2$, r_1^*, and (r_1).
3. A string is a regular expression if and only if it can be derived from the primitive regular expressions by a finite number of applications of the rules in item 2.

Example 3.1

For $\Sigma = \{a, b, c\}$, the string

$$(a + b \cdot c)^* \cdot (c + \varnothing)$$

is a regular expression, since it is constructed by application of the above rules. For example, if we take $r_1 = c$ and $r_2 = \varnothing$, we find that $c + \varnothing$ and $(c + \varnothing)$ are also regular expressions. Repeating this, we eventually generate the whole string. On the other hand, $(a + b+)$ is not a regular expression, because there is no way it can be constructed from the primitive regular expressions.

Formal Definition of a Regular Expression

We construct regular expressions from primitive constituents by repeatedly applying certain recursive rules. This is similar to the way we construct familiar arithmetic expressions.

Definition 3.1

Let Σ be a given alphabet. Then

1. $\varnothing$, λ, and $a \in \Sigma$ are all regular expressions. These are called *primitive regular expressions*.
2. If r_1 and r_2 are regular expressions, so are $r_1 + r_2$, $r_1 \cdot r_2$, r_1^*, and (r_1).
3. A string is a regular expression if and only if it can be derived from the primitive regular expressions by a finite number of applications of the rules in item 2.

Example 3.1

For $\Sigma = \{a, b, c\}$, the string

$$(a + b \cdot c)^* \cdot (c + \varnothing)$$

is a regular expression, since it is constructed by application of the above rules. For example, if we take $r_1 = c$ and $r_2 = \varnothing$, we find that $c + \varnothing$ and $(c + \varnothing)$ are also regular expressions. Repeating this, we eventually generate the whole string. On the other hand, $(a + b+)$ is not a regular expression, because there is no way it can be constructed from the primitive regular expressions.

Formal Definition of a Regular Expression

We construct regular expressions from primitive constituents by repeatedly applying certain recursive rules. This is similar to the way we construct familiar arithmetic expressions.

Definition 3.1

Let Σ be a given alphabet. Then

1. $\varnothing$, λ, and $a \in \Sigma$ are all regular expressions. These are called *primitive regular expressions*.
2. If r_1 and r_2 are regular expressions, so are $r_1 + r_2$, $r_1 \cdot r_2$, r_1^*, and (r_1).
3. A string is a regular expression if and only if it can be derived from the primitive regular expressions by a finite number of applications of the rules in item 2.

Example 3.1

For $\Sigma = \{a, b, c\}$, the string

$$(a + b \cdot c)^* \cdot (c + \varnothing)$$

is a regular expression, since it is constructed by application of the above rules. For example, if we take $r_1 = c$ and $r_2 = \varnothing$, we find that $c + \varnothing$ and $(c + \varnothing)$ are also regular expressions. Repeating this, we eventually generate the whole string. On the other hand, $(a + b+)$ is not a regular expression, because there is no way it can be constructed from the primitive regular expressions.

Formal Definition of a Regular Expression

We construct regular expressions from primitive constituents by repeatedly applying certain recursive rules. This is similar to the way we construct familiar arithmetic expressions.

Definition 3.1

Let Σ be a given alphabet. Then

1. $\varnothing$, λ, and $a \in \Sigma$ are all regular expressions. These are called *primitive regular expressions*.
2. If r_1 and r_2 are regular expressions, so are $r_1 + r_2$, $r_1 \cdot r_2$, r_1^*, and (r_1).
3. A string is a regular expression if and only if it can be derived from the primitive regular expressions by a finite number of applications of the rules in item 2.

Example 3.1

For $\Sigma = \{a, b, c\}$, the string

$$(a + b \cdot c)^* \cdot (c + \varnothing)$$

is a regular expression, since it is constructed by application of the above rules. For example, if we take $r_1 = c$ and $r_2 = \varnothing$, we find that $c + \varnothing$ and $(c + \varnothing)$ are also regular expressions. Repeating this, we eventually generate the whole string. On the other hand, $(a + b+)$ is not a regular expression, because there is no way it can be constructed from the primitive regular expressions.

Formal Definition of a Regular Expression

We construct regular expressions from primitive constituents by repeatedly applying certain recursive rules. This is similar to the way we construct familiar arithmetic expressions.

Definition 3.1

Let Σ be a given alphabet. Then

1. $\varnothing$, λ, and $a \in \Sigma$ are all regular expressions. These are called *primitive regular expressions*.
2. If r_1 and r_2 are regular expressions, so are $r_1 + r_2$, $r_1 \cdot r_2$, r_1^*, and (r_1).
3. A string is a regular expression if and only if it can be derived from the primitive regular expressions by a finite number of applications of the rules in item 2.

Example 3.1

For $\Sigma = \{a, b, c\}$, the string

$$(a + b \cdot c)^* \cdot (c + \varnothing)$$

is a regular expression, since it is constructed by application of the above rules. For example, if we take $r_1 = c$ and $r_2 = \varnothing$, we find that $c + \varnothing$ and $(c + \varnothing)$ are also regular expressions. Repeating this, we eventually generate the whole string. On the other hand, $(a + b+)$ is not a regular expression, because there is no way it can be constructed from the primitive regular expressions.

Languages Associated with Regular Expressions

Regular expressions can be used to describe some simple languages. If r is a regular expression, we shall let $L(r)$ denote the language associated with r.

Definition 3.2

The language $L(r)$ denoted by any regular expression r is defined by the following rules.

1. $\varnothing$ is a regular expression denoting the empty set.
2. λ is a regular expression denoting $\{\lambda\}$.
3. For every $a \in \Sigma$, a is a regular expression denoting $\{a\}$.

If r_1 and r_2 are regular expressions, then

4. $L(r_1 + r_2) = L(r_1) \cup L(r_2)$.
5. $L(r_1 \cdot r_2) = L(r_1)L(r_2)$.
6. $L((r_1)) = L(r_1)$.
7. $L(r_1^*) = (L(r_1))^*$.

The last four rules of this definition are used to reduce $L(r)$ to simpler components recursively; the first three are the termination conditions for this recursion. To see what language a given expression denotes, we apply these rules repeatedly.

Languages Associated with Regular Expressions

Regular expressions can be used to describe some simple languages. If r is a regular expression, we shall let $L(r)$ denote the language associated with r.

Definition 3.2

The language $L(r)$ denoted by any regular expression r is defined by the following rules.

1. $\varnothing$ is a regular expression denoting the empty set.
2. λ is a regular expression denoting $\{\lambda\}$.
3. For every $a \in \Sigma$, a is a regular expression denoting $\{a\}$.

   If r_1 and r_2 are regular expressions, then

4. $L(r_1 + r_2) = L(r_1) \cup L(r_2)$.
5. $L(r_1 \cdot r_2) = L(r_1)L(r_2)$.
6. $L((r_1)) = L(r_1)$.
7. $L(r_1^*) = (L(r_1))^*$.

The last four rules of this definition are used to reduce $L(r)$ to simpler components recursively; the first three are the termination conditions for this recursion. To see what language a given expression denotes, we apply these rules repeatedly.

Languages Associated with Regular Expressions

Regular expressions can be used to describe some simple languages. If r is a regular expression, we shall let $L(r)$ denote the language associated with r.

Definition 3.2

The language $L(r)$ denoted by any regular expression r is defined by the following rules.

1. $\emptyset$ is a regular expression denoting the empty set.
2. λ is a regular expression denoting $\{\lambda\}$.
3. For every $a \in \Sigma$, a is a regular expression denoting $\{a\}$.

If r_1 and r_2 are regular expressions, then

4. $L(r_1 + r_2) = L(r_1) \cup L(r_2)$.
5. $L(r_1 \cdot r_2) = L(r_1)L(r_2)$.
6. $L((r_1)) = L(r_1)$.
7. $L(r_1^*) = (L(r_1))^*$.

The last four rules of this definition are used to reduce $L(r)$ to simpler components recursively; the first three are the termination conditions for this recursion. To see what language a given expression denotes, we apply these rules repeatedly.

Languages Associated with Regular Expressions

Regular expressions can be used to describe some simple languages. If r is a regular expression, we shall let $L(r)$ denote the language associated with r.

Definition 3.2

The language $L(r)$ denoted by any regular expression r is defined by the following rules.

1. $\varnothing$ is a regular expression denoting the empty set.
2. λ is a regular expression denoting $\{\lambda\}$.
3. For every $a \in \Sigma$, a is a regular expression denoting $\{a\}$.

If r_1 and r_2 are regular expressions, then

4. $L(r_1 + r_2) = L(r_1) \cup L(r_2)$.
5. $L(r_1 \cdot r_2) = L(r_1)L(r_2)$.
6. $L((r_1)) = L(r_1)$.
7. $L(r_1^*) = (L(r_1))^*$.

The last four rules of this definition are used to reduce $L(r)$ to simpler components recursively; the first three are the termination conditions for this recursion. To see what language a given expression denotes, we apply these rules repeatedly.

Languages Associated with Regular Expressions

Regular expressions can be used to describe some simple languages. If r is a regular expression, we shall let $L(r)$ denote the language associated with r.

Definition 3.2

The language $L(r)$ denoted by any regular expression r is defined by the following rules.

1. $\varnothing$ is a regular expression denoting the empty set,
2. λ is a regular expression denoting $\{\lambda\}$,
3. For every $a \in \Sigma$, a is a regular expression denoting $\{a\}$.

   If r_1 and r_2 are regular expressions, then

4. $L(r_1 + r_2) = L(r_1) \cup L(r_2)$,
5. $L(r_1 \cdot r_2) = L(r_1)L(r_2)$,
6. $L((r_1)) = L(r_1)$,
7. $L(r_1^*) = (L(r_1))^*$.

The last four rules of this definition are used to reduce $L(r)$ to simpler components recursively; the first three are the termination conditions for this recursion. To see what language a given expression denotes, we apply these rules repeatedly.

3.1 Regular Expressions

Languages Associated with Regular Expressions

Regular expressions can be used to describe some simple languages. If r is a regular expression, we shall let $L(r)$ denote the language associated with r.

Definition 3.2

The language $L(r)$ denoted by any regular expression r is defined by the following rules.

1. $\varnothing$ is a regular expression denoting the empty set,
2. λ is a regular expression denoting $\{\lambda\}$,
3. For every $a \in \Sigma$, a is a regular expression denoting $\{a\}$.

   If r_1 and r_2 are regular expressions, then

4. $L(r_1 + r_2) = L(r_1) \cup L(r_2)$,
5. $L(r_1 \cdot r_2) = L(r_1)L(r_2)$,
6. $L((r_1)) = L(r_1)$,
7. $L(r_1^*) = (L(r_1))^*$.

The last four rules of this definition are used to reduce $L(r)$ to simpler components recursively; the first three are the termination conditions for this recursion. To see what language a given expression denotes, we apply these rules repeatedly.

Languages Associated with Regular Expressions

Regular expressions can be used to describe some simple languages. If r is a regular expression, we shall let $L(r)$ denote the language associated with r.

Definition 3.2

The language $L(r)$ denoted by any regular expression r is defined by the following rules.

1. $\varnothing$ is a regular expression denoting the empty set,
2. λ is a regular expression denoting $\{\lambda\}$,
3. For every $a \in \Sigma$, a is a regular expression denoting $\{a\}$.

   If r_1 and r_2 are regular expressions, then
4. $L(r_1 + r_2) = L(r_1) \cup L(r_2)$,
5. $L(r_1 \cdot r_2) = L(r_1)L(r_2)$,
6. $L((r_1)) = L(r_1)$,
7. $L(r_1^*) = (L(r_1))^*$.

The last four rules of this definition are used to reduce $L(r)$ to simpler components recursively; the first three are the termination conditions for this recursion. To see what language a given expression denotes, we apply these rules repeatedly.

Languages Associated with Regular Expressions

Regular expressions can be used to describe some simple languages. If r is a regular expression, we shall let $L(r)$ denote the language associated with r.

Definition 3.2

The language $L(r)$ denoted by any regular expression r is defined by the following rules.

1. $\varnothing$ is a regular expression denoting the empty set,
2. λ is a regular expression denoting $\{\lambda\}$,
3. For every $a \in \Sigma$, a is a regular expression denoting $\{a\}$.

   If r_1 and r_2 are regular expressions, then
4. $L(r_1 + r_2) = L(r_1) \cup L(r_2)$,
5. $L(r_1 \cdot r_2) = L(r_1)L(r_2)$,
6. $L((r_1)) = L(r_1)$,
7. $L(r_1^*) = (L(r_1))^*$.

The last four rules of this definition are used to reduce $L(r)$ to simpler components recursively; the first three are the termination conditions for this recursion. To see what language a given expression denotes, we apply these rules repeatedly.

Languages Associated with Regular Expressions

Regular expressions can be used to describe some simple languages. If r is a regular expression, we shall let $L(r)$ denote the language associated with r.

Definition 3.2

The language $L(r)$ denoted by any regular expression r is defined by the following rules.

1. $\varnothing$ is a regular expression denoting the empty set,
2. λ is a regular expression denoting $\{\lambda\}$,
3. For every $a \in \Sigma$, a is a regular expression denoting $\{a\}$.

   If r_1 and r_2 are regular expressions, then
4. $L(r_1 + r_2) = L(r_1) \cup L(r_2)$,
5. $L(r_1 \cdot r_2) = L(r_1)L(r_2)$,
6. $L((r_1)) = L(r_1)$,
7. $L(r_1^*) = (L(r_1))^*$.

The last four rules of this definition are used to reduce $L(r)$ to simpler components recursively; the first three are the termination conditions for this recursion. To see what language a given expression denotes, we apply these rules repeatedly.

Languages Associated with Regular Expressions

Regular expressions can be used to describe some simple languages. If r is a regular expression, we shall let $L(r)$ denote the language associated with r.

Definition 3.2

The language $L(r)$ denoted by any regular expression r is defined by the following rules.

1. $\varnothing$ is a regular expression denoting the empty set,
2. λ is a regular expression denoting $\{\lambda\}$,
3. For every $a \in \Sigma$, a is a regular expression denoting $\{a\}$.

   If r_1 and r_2 are regular expressions, then

4. $L(r_1 + r_2) = L(r_1) \cup L(r_2)$,
5. $L(r_1 \cdot r_2) = L(r_1)L(r_2)$,
6. $L((r_1)) = L(r_1)$,
7. $L(r_1^*) = (L(r_1))^*$.

The last four rules of this definition are used to reduce $L(r)$ to simpler components recursively; the first three are the termination conditions for this recursion. To see what language a given expression denotes, we apply these rules repeatedly.

Languages Associated with Regular Expressions

Regular expressions can be used to describe some simple languages. If r is a regular expression, we shall let $L(r)$ denote the language associated with r.

Definition 3.2

The language $L(r)$ denoted by any regular expression r is defined by the following rules.

1. $\varnothing$ is a regular expression denoting the empty set,
2. λ is a regular expression denoting $\{\lambda\}$,
3. For every $a \in \Sigma$, a is a regular expression denoting $\{a\}$.

   If r_1 and r_2 are regular expressions, then
4. $L(r_1 + r_2) = L(r_1) \cup L(r_2)$,
5. $L(r_1 \cdot r_2) = L(r_1)L(r_2)$,
6. $L((r_1)) = L(r_1)$,
7. $L(r_1^*) = (L(r_1))^*$.

The last four rules of this definition are used to reduce $L(r)$ to simpler components recursively; the first three are the termination conditions for this recursion. To see what language a given expression denotes, we apply these rules repeatedly.

Languages Associated with Regular Expressions

Regular expressions can be used to describe some simple languages. If r is a regular expression, we shall let $L(r)$ denote the language associated with r.

Definition 3.2

The language $L(r)$ denoted by any regular expression r is defined by the following rules.

1. $\varnothing$ is a regular expression denoting the empty set,
2. λ is a regular expression denoting $\{\lambda\}$,
3. For every $a \in \Sigma$, a is a regular expression denoting $\{a\}$.

   If r_1 and r_2 are regular expressions, then
4. $L(r_1 + r_2) = L(r_1) \cup L(r_2)$,
5. $L(r_1 \cdot r_2) = L(r_1)L(r_2)$,
6. $L((r_1)) = L(r_1)$,
7. $L(r_1^*) = (L(r_1))^*$.

The last four rules of this definition are used to reduce $L(r)$ to simpler components recursively; the first three are the termination conditions for this recursion. To see what language a given expression denotes, we apply these rules repeatedly.

Languages Associated with Regular Expressions

Regular expressions can be used to describe some simple languages. If r is a regular expression, we shall let $L(r)$ denote the language associated with r.

Definition 3.2

The language $L(r)$ denoted by any regular expression r is defined by the following rules.

1. $\varnothing$ is a regular expression denoting the empty set,
2. λ is a regular expression denoting $\{\lambda\}$,
3. For every $a \in \Sigma$, a is a regular expression denoting $\{a\}$.

   If r_1 and r_2 are regular expressions, then
4. $L(r_1 + r_2) = L(r_1) \cup L(r_2)$,
5. $L(r_1 \cdot r_2) = L(r_1)L(r_2)$,
6. $L((r_1)) = L(r_1)$,
7. $L(r_1^*) = (L(r_1))^*$.

The last four rules of this definition are used to reduce $L(r)$ to simpler components recursively; the first three are the termination conditions for this recursion. To see what language a given expression denotes, we apply these rules repeatedly.

Languages Associated with Regular Expressions

Regular expressions can be used to describe some simple languages. If r is a regular expression, we shall let $L(r)$ denote the language associated with r.

Definition 3.2

The language $L(r)$ denoted by any regular expression r is defined by the following rules.

1. $\varnothing$ is a regular expression denoting the empty set,
2. λ is a regular expression denoting $\{\lambda\}$,
3. For every $a \in \Sigma$, a is a regular expression denoting $\{a\}$.

   If r_1 and r_2 are regular expressions, then
4. $L(r_1 + r_2) = L(r_1) \cup L(r_2)$,
5. $L(r_1 \cdot r_2) = L(r_1)L(r_2)$,
6. $L((r_1)) = L(r_1)$,
7. $L(r_1^*) = (L(r_1))^*$.

The last four rules of this definition are used to reduce $L(r)$ to simpler components recursively; the first three are the termination conditions for this recursion. To see what language a given expression denotes, we apply these rules repeatedly.

Languages Associated with Regular Expressions

Regular expressions can be used to describe some simple languages. If r is a regular expression, we shall let $L(r)$ denote the language associated with r.

Definition 3.2

The language $L(r)$ denoted by any regular expression r is defined by the following rules.

1. $\varnothing$ is a regular expression denoting the empty set,
2. λ is a regular expression denoting $\{\lambda\}$,
3. For every $a \in \Sigma$, a is a regular expression denoting $\{a\}$.

   If r_1 and r_2 are regular expressions, then

4. $L(r_1 + r_2) = L(r_1) \cup L(r_2)$,
5. $L(r_1 \cdot r_2) = L(r_1)L(r_2)$,
6. $L((r_1)) = L(r_1)$,
7. $L(r_1^*) = (L(r_1))^*$.

The last four rules of this definition are used to reduce $L(r)$ to simpler components recursively; the first three are the termination conditions for this recursion. To see what language a given expression denotes, we apply these rules repeatedly.

Languages Associated with Regular Expressions

Regular expressions can be used to describe some simple languages. If r is a regular expression, we shall let $L(r)$ denote the language associated with r.

Definition 3.2

The language $L(r)$ denoted by any regular expression r is defined by the following rules.

1. $\varnothing$ is a regular expression denoting the empty set,
2. λ is a regular expression denoting $\{\lambda\}$,
3. For every $a \in \Sigma$, a is a regular expression denoting $\{a\}$.

   If r_1 and r_2 are regular expressions, then
4. $L(r_1 + r_2) = L(r_1) \cup L(r_2)$,
5. $L(r_1 \cdot r_2) = L(r_1)L(r_2)$,
6. $L((r_1)) = L(r_1)$,
7. $L(r_1^*) = (L(r_1))^*$.

The last four rules of this definition are used to reduce $L(r)$ to simpler components recursively; the first three are the termination conditions for this recursion. To see what language a given expression denotes, we apply these rules repeatedly.

Languages Associated with Regular Expressions

Regular expressions can be used to describe some simple languages. If r is a regular expression, we shall let $L(r)$ denote the language associated with r.

Definition 3.2

The language $L(r)$ denoted by any regular expression r is defined by the following rules.

1. $\varnothing$ is a regular expression denoting the empty set,
2. λ is a regular expression denoting $\{\lambda\}$,
3. For every $a \in \Sigma$, a is a regular expression denoting $\{a\}$.

   If r_1 and r_2 are regular expressions, then
4. $L(r_1 + r_2) = L(r_1) \cup L(r_2)$,
5. $L(r_1 \cdot r_2) = L(r_1)L(r_2)$,
6. $L((r_1)) = L(r_1)$,
7. $L(r_1^*) = (L(r_1))^*$.

The last four rules of this definition are used to reduce $L(r)$ to simpler components recursively; the first three are the termination conditions for this recursion. To see what language a given expression denotes, we apply these rules repeatedly.

Languages Associated with Regular Expressions

Regular expressions can be used to describe some simple languages. If r is a regular expression, we shall let $L(r)$ denote the language associated with r.

Definition 3.2

The language $L(r)$ denoted by any regular expression r is defined by the following rules.

1. $\varnothing$ is a regular expression denoting the empty set,
2. λ is a regular expression denoting $\{\lambda\}$,
3. For every $a \in \Sigma$, a is a regular expression denoting $\{a\}$.

   If r_1 and r_2 are regular expressions, then
4. $L(r_1 + r_2) = L(r_1) \cup L(r_2)$,
5. $L(r_1 \cdot r_2) = L(r_1)L(r_2)$,
6. $L((r_1)) = L(r_1)$,
7. $L(r_1^*) = (L(r_1))^*$.

The last four rules of this definition are used to reduce $L(r)$ to simpler components recursively; the first three are the termination conditions for this recursion. To see what language a given expression denotes, we apply these rules repeatedly.

Languages Associated with Regular Expressions

Regular expressions can be used to describe some simple languages. If r is a regular expression, we shall let $L(r)$ denote the language associated with r.

Definition 3.2

The language $L(r)$ denoted by any regular expression r is defined by the following rules.

1. $\varnothing$ is a regular expression denoting the empty set,
2. λ is a regular expression denoting $\{\lambda\}$,
3. For every $a \in \Sigma$, a is a regular expression denoting $\{a\}$.

   If r_1 and r_2 are regular expressions, then
4. $L(r_1 + r_2) = L(r_1) \cup L(r_2)$,
5. $L(r_1 \cdot r_2) = L(r_1)L(r_2)$,
6. $L((r_1)) = L(r_1)$,
7. $L(r_1^*) = (L(r_1))^*$.

The last four rules of this definition are used to reduce $L(r)$ to simpler components recursively; the first three are the termination conditions for this recursion. To see what language a given expression denotes, we apply these rules repeatedly.

Example 3.2

Exhibit the language $L(a^* \cdot (a+b))$ in set notation.

$$\begin{aligned} L(a^* \cdot (a+b)) &= L(a^*)L(a+b) = \\ &= (L(a))^*(L(a) \cup L(b)) = \\ &= \{\lambda, a, aa, aaa, \ldots\}\{a, b\} = \\ &= \{a, aa, aaa, \ldots, b, ab, aab, \ldots\}. \end{aligned}$$

There is one problem with rules (4) to (7) in Definition 3.2. They define a language precisely if r_1 and r_2 are given, but there may be some ambiguity in breaking a complicated expression into parts. Consider, for example, the regular expression $a \cdot b + c$. We can consider this as being made up of $r_1 = a \cdot b$ and $r_2 = c$. In this case, we find $L(a \cdot b + c) = \{ab, c\}$. But there is nothing in Definition 3.2 to stop us from taking $r_1 = a$ and $r_2 = b + c$. We now get a different result, $L(a \cdot b + c) = \{ab, ac\}$. To overcome this, we could require that all expressions be fully parenthesized, but this gives cumbersome results. Instead, we use a convention familiar from mathematics and programming languages. We establish a set of precedence rules for evaluation in which star-closure precedes concatenation and concatenation precedes union. Also, the symbol for concatenation may be omitted, so we can write r_1r_2 for $r_1 \cdot r_2$.

Example 3.2

Exhibit the language $L(a^* \cdot (a+b))$ in set notation.

$$\begin{aligned} L(a^* \cdot (a+b)) &= L(a^*)L(a+b) = \\ &= (L(a))^*(L(a) \cup L(b)) = \\ &= \{\lambda, a, aa, aaa, \ldots\}\{a, b\} = \\ &= \{a, aa, aaa, \ldots, b, ab, aab, \ldots\}. \end{aligned}$$

There is one problem with rules (4) to (7) in Definition 3.2. They define a language precisely if r_1 and r_2 are given, but there may be some ambiguity in breaking a complicated expression into parts. Consider, for example, the regular expression $a \cdot b + c$. We can consider this as being made up of $r_1 = a \cdot b$ and $r_2 = c$. In this case, we find $L(a \cdot b + c) = \{ab, c\}$. But there is nothing in Definition 3.2 to stop us from taking $r_1 = a$ and $r_2 = b + c$. We now get a different result, $L(a \cdot b + c) = \{ab, ac\}$. To overcome this, we could require that all expressions be fully parenthesized, but this gives cumbersome results. Instead, we use a convention familiar from mathematics and programming languages. We establish a set of precedence rules for evaluation in which star-closure precedes concatenation and concatenation precedes union. Also, the symbol for concatenation may be omitted, so we can write $r_1 r_2$ for $r_1 \cdot r_2$.

Example 3.2

Exhibit the language $L(a^* \cdot (a+b))$ in set notation.

$$\begin{aligned} L(a^* \cdot (a+b)) &= L(a^*)L(a+b) = \\ &= (L(a))^*(L(a) \cup L(b)) = \\ &= \{\lambda, a, aa, aaa, \ldots\}\{a, b\} = \\ &= \{a, aa, aaa, \ldots, b, ab, aab, \ldots\}. \end{aligned}$$

There is one problem with rules (4) to (7) in Definition 3.2. They define a language precisely if r_1 and r_2 are given, but there may be some ambiguity in breaking a complicated expression into parts. Consider, for example, the regular expression $a \cdot b + c$. We can consider this as being made up of $r_1 = a \cdot b$ and $r_2 = c$. In this case, we find $L(a \cdot b + c) = \{ab, c\}$. But there is nothing in Definition 3.2 to stop us from taking $r_1 = a$ and $r_2 = b + c$. We now get a different result, $L(a \cdot b + c) = \{ab, ac\}$. To overcome this, we could require that all expressions be fully parenthesized, but this gives cumbersome results. Instead, we use a convention familiar from mathematics and programming languages. We establish a set of precedence rules for evaluation in which star-closure precedes concatenation and concatenation precedes union. Also, the symbol for concatenation may be omitted, so we can write $r_1 r_2$ for $r_1 \cdot r_2$.

Example 3.2

Exhibit the language $L(a^* \cdot (a+b))$ in set notation.

$$\begin{aligned} L(a^* \cdot (a+b)) &= L(a^*)L(a+b) = \\ &= (L(a))^*(L(a) \cup L(b)) = \\ &= \{\lambda, a, aa, aaa, \ldots\}\{a, b\} = \\ &= \{a, aa, aaa, \ldots, b, ab, aab, \ldots\}. \end{aligned}$$

There is one problem with rules (4) to (7) in Definition 3.2. They define a language precisely if r_1 and r_2 are given, but there may be some ambiguity in breaking a complicated expression into parts. Consider, for example, the regular expression $a \cdot b + c$. We can consider this as being made up of $r_1 = a \cdot b$ and $r_2 = c$. In this case, we find $L(a \cdot b + c) = \{ab, c\}$. But there is nothing in Definition 3.2 to stop us from taking $r_1 = a$ and $r_2 = b + c$. We now get a different result, $L(a \cdot b + c) = \{ab, ac\}$. To overcome this, we could require that all expressions be fully parenthesized, but this gives cumbersome results. Instead, we use a convention familiar from mathematics and programming languages. We establish a set of precedence rules for evaluation in which star-closure precedes concatenation and concatenation precedes union. Also, the symbol for concatenation may be omitted, so we can write $r_1 r_2$ for $r_1 \cdot r_2$.

Example 3.2

Exhibit the language $L(a^* \cdot (a+b))$ in set notation.

$$\begin{aligned} L(a^* \cdot (a+b)) &= L(a^*)L(a+b) = \\ &= (L(a))^*(L(a) \cup L(b)) = \\ &= \{\lambda, a, aa, aaa, \ldots\}\{a, b\} = \\ &= \{a, aa, aaa, \ldots, b, ab, aab, \ldots\}. \end{aligned}$$

There is one problem with rules (4) to (7) in Definition 3.2. They define a language precisely if r_1 and r_2 are given, but there may be some ambiguity in breaking a complicated expression into parts. Consider, for example, the regular expression $a \cdot b + c$. We can consider this as being made up of $r_1 = a \cdot b$ and $r_2 = c$. In this case, we find $L(a \cdot b + c) = \{ab, c\}$. But there is nothing in Definition 3.2 to stop us from taking $r_1 = a$ and $r_2 = b + c$. We now get a different result, $L(a \cdot b + c) = \{ab, ac\}$. To overcome this, we could require that all expressions be fully parenthesized, but this gives cumbersome results. Instead, we use a convention familiar from mathematics and programming languages. We establish a set of precedence rules for evaluation in which star-closure precedes concatenation and concatenation precedes union. Also, the symbol for concatenation may be omitted, so we can write $r_1 r_2$ for $r_1 \cdot r_2$.

Example 3.2

Exhibit the language $L(a^* \cdot (a+b))$ in set notation.

$$\begin{aligned} L(a^* \cdot (a+b)) &= L(a^*)L(a+b) = \\ &= (L(a))^*(L(a) \cup L(b)) = \\ &= \{\lambda, a, aa, aaa, \ldots\}\{a, b\} = \\ &= \{a, aa, aaa, \ldots, b, ab, aab, \ldots\}. \end{aligned}$$

There is one problem with rules (4) to (7) in Definition 3.2. They define a language precisely if r_1 and r_2 are given, but there may be some ambiguity in breaking a complicated expression into parts. Consider, for example, the regular expression $a \cdot b + c$. We can consider this as being made up of $r_1 = a \cdot b$ and $r_2 = c$. In this case, we find $L(a \cdot b + c) = \{ab, c\}$. But there is nothing in Definition 3.2 to stop us from taking $r_1 = a$ and $r_2 = b + c$. We now get a different result, $L(a \cdot b + c) = \{ab, ac\}$. To overcome this, we could require that all expressions be fully parenthesized, but this gives cumbersome results. Instead, we use a convention familiar from mathematics and programming languages. We establish a set of precedence rules for evaluation in which star-closure precedes concatenation and concatenation precedes union. Also, the symbol for concatenation may be omitted, so we can write $r_1 r_2$ for $r_1 \cdot r_2$.

Example 3.2

Exhibit the language $L(a^* \cdot (a+b))$ in set notation.

$$\begin{aligned} L(a^* \cdot (a+b)) &= L(a^*)L(a+b) = \\ &= (L(a))^*(L(a) \cup L(b)) = \\ &= \{\lambda, a, aa, aaa, \ldots\}\{a, b\} = \\ &= \{a, aa, aaa, \ldots, b, ab, aab, \ldots\}. \end{aligned}$$

There is one problem with rules (4) to (7) in Definition 3.2. They define a language precisely if r_1 and r_2 are given, but there may be some ambiguity in breaking a complicated expression into parts. Consider, for example, the regular expression $a \cdot b + c$. We can consider this as being made up of $r_1 = a \cdot b$ and $r_2 = c$. In this case, we find $L(a \cdot b + c) = \{ab, c\}$. But there is nothing in Definition 3.2 to stop us from taking $r_1 = a$ and $r_2 = b + c$. We now get a different result, $L(a \cdot b + c) = \{ab, ac\}$. To overcome this, we could require that all expressions be fully parenthesized, but this gives cumbersome results. Instead, we use a convention familiar from mathematics and programming languages. We establish a set of precedence rules for evaluation in which star-closure precedes concatenation and concatenation precedes union. Also, the symbol for concatenation may be omitted, so we can write $r_1 r_2$ for $r_1 \cdot r_2$.

Example 3.2

Exhibit the language $L(a^* \cdot (a+b))$ in set notation.

$$\begin{aligned} L(a^* \cdot (a+b)) &= L(a^*)L(a+b) = \\ &= (L(a))^*(L(a) \cup L(b)) = \\ &= \{\lambda, a, aa, aaa, \ldots\}\{a, b\} = \\ &= \{a, aa, aaa, \ldots, b, ab, aab, \ldots\}. \end{aligned}$$

There is one problem with rules (4) to (7) in Definition 3.2. They define a language precisely if r_1 and r_2 are given, but there may be some ambiguity in breaking a complicated expression into parts. Consider, for example, the regular expression $a \cdot b + c$. We can consider this as being made up of $r_1 = a \cdot b$ and $r_2 = c$. In this case, we find $L(a \cdot b + c) = \{ab, c\}$. But there is nothing in Definition 3.2 to stop us from taking $r_1 = a$ and $r_2 = b + c$. We now get a different result, $L(a \cdot b + c) = \{ab, ac\}$. To overcome this, we could require that all expressions be fully parenthesized, but this gives cumbersome results. Instead, we use a convention familiar from mathematics and programming languages. We establish a set of precedence rules for evaluation in which star-closure precedes concatenation and concatenation precedes union. Also, the symbol for concatenation may be omitted, so we can write $r_1 r_2$ for $r_1 \cdot r_2$.

Example 3.2

Exhibit the language $L(a^* \cdot (a+b))$ in set notation.

$$\begin{aligned} L(a^* \cdot (a+b)) &= L(a^*)L(a+b) = \\ &= (L(a))^*(L(a) \cup L(b)) = \\ &= \{\lambda, a, aa, aaa, \ldots\}\{a, b\} = \\ &= \{a, aa, aaa, \ldots, b, ab, aab, \ldots\}. \end{aligned}$$

There is one problem with rules (4) to (7) in Definition 3.2. They define a language precisely if r_1 and r_2 are given, but there may be some ambiguity in breaking a complicated expression into parts. Consider, for example, the regular expression $a \cdot b + c$. We can consider this as being made up of $r_1 = a \cdot b$ and $r_2 = c$. In this case, we find $L(a \cdot b + c) = \{ab, c\}$. But there is nothing in Definition 3.2 to stop us from taking $r_1 = a$ and $r_2 = b + c$. We now get a different result, $L(a \cdot b + c) = \{ab, ac\}$. To overcome this, we could require that all expressions be fully parenthesized, but this gives cumbersome results. Instead, we use a convention familiar from mathematics and programming languages. We establish a set of precedence rules for evaluation in which star-closure precedes concatenation and concatenation precedes union. Also, the symbol for concatenation may be omitted, so we can write $r_1 r_2$ for $r_1 \cdot r_2$.

Example 3.2

Exhibit the language $L(a^* \cdot (a+b))$ in set notation.

$$\begin{aligned} L(a^* \cdot (a+b)) &= L(a^*)L(a+b) = \\ &= (L(a))^*(L(a) \cup L(b)) = \\ &= \{\lambda, a, aa, aaa, \ldots\}\{a, b\} = \\ &= \{a, aa, aaa, \ldots, b, ab, aab, \ldots\}. \end{aligned}$$

There is one problem with rules (4) to (7) in Definition 3.2. They define a language precisely if r_1 and r_2 are given, but there may be some ambiguity in breaking a complicated expression into parts. Consider, for example, the regular expression $a \cdot b + c$. We can consider this as being made up of $r_1 = a \cdot b$ and $r_2 = c$. In this case, we find $L(a \cdot b + c) = \{ab, c\}$. But there is nothing in Definition 3.2 to stop us from taking $r_1 = a$ and $r_2 = b + c$. We now get a different result, $L(a \cdot b + c) = \{ab, ac\}$. To overcome this, we could require that all expressions be fully parenthesized, but this gives cumbersome results. Instead, we use a convention familiar from mathematics and programming languages. We establish a set of precedence rules for evaluation in which star-closure precedes concatenation and concatenation precedes union. Also, the symbol for concatenation may be omitted, so we can write r_1r_2 for $r_1 \cdot r_2$.

Example 3.2

Exhibit the language $L(a^* \cdot (a+b))$ in set notation.

$$\begin{aligned} L(a^* \cdot (a+b)) &= L(a^*)L(a+b) = \\ &= (L(a))^*(L(a) \cup L(b)) = \\ &= \{\lambda, a, aa, aaa, \ldots\}\{a, b\} = \\ &= \{a, aa, aaa, \ldots, b, ab, aab, \ldots\}. \end{aligned}$$

There is one problem with rules (4) to (7) in Definition 3.2. They define a language precisely if r_1 and r_2 are given, but there may be some ambiguity in breaking a complicated expression into parts. Consider, for example, the regular expression $a \cdot b + c$. We can consider this as being made up of $r_1 = a \cdot b$ and $r_2 = c$. In this case, we find $L(a \cdot b + c) = \{ab, c\}$. But there is nothing in Definition 3.2 to stop us from taking $r_1 = a$ and $r_2 = b + c$. We now get a different result, $L(a \cdot b + c) = \{ab, ac\}$. To overcome this, we could require that all expressions be fully parenthesized, but this gives cumbersome results. Instead, we use a convention familiar from mathematics and programming languages. We establish a set of precedence rules for evaluation in which star-closure precedes concatenation and concatenation precedes union. Also, the symbol for concatenation may be omitted, so we can write $r_1 r_2$ for $r_1 \cdot r_2$.

Example 3.2

Exhibit the language $L(a^* \cdot (a+b))$ in set notation.

$$\begin{aligned} L(a^* \cdot (a+b)) &= L(a^*)L(a+b) = \\ &= (L(a))^*(L(a) \cup L(b)) = \\ &= \{\lambda, a, aa, aaa, \ldots\}\{a, b\} = \\ &= \{a, aa, aaa, \ldots, b, ab, aab, \ldots\}. \end{aligned}$$

There is one problem with rules (4) to (7) in Definition 3.2. They define a language precisely if r_1 and r_2 are given, but there may be some ambiguity in breaking a complicated expression into parts. Consider, for example, the regular expression $a \cdot b + c$. We can consider this as being made up of $r_1 = a \cdot b$ and $r_2 = c$. In this case, we find $L(a \cdot b + c) = \{ab, c\}$. But there is nothing in Definition 3.2 to stop us from taking $r_1 = a$ and $r_2 = b + c$. We now get a different result, $L(a \cdot b + c) = \{ab, ac\}$. To overcome this, we could require that all expressions be fully parenthesized, but this gives cumbersome results. Instead, we use a convention familiar from mathematics and programming languages. We establish a set of precedence rules for evaluation in which star-closure precedes concatenation and concatenation precedes union. Also, the symbol for concatenation may be omitted, so we can write $r_1 r_2$ for $r_1 \cdot r_2$.

Example 3.2

Exhibit the language $L(a^* \cdot (a+b))$ in set notation.

$$\begin{aligned} L(a^* \cdot (a+b)) &= L(a^*)L(a+b) = \\ &= (L(a))^*(L(a) \cup L(b)) = \\ &= \{\lambda, a, aa, aaa, \ldots\}\{a, b\} = \\ &= \{a, aa, aaa, \ldots, b, ab, aab, \ldots\}. \end{aligned}$$

There is one problem with rules (4) to (7) in Definition 3.2. They define a language precisely if r_1 and r_2 are given, but there may be some ambiguity in breaking a complicated expression into parts. Consider, for example, the regular expression $a \cdot b + c$. We can consider this as being made up of $r_1 = a \cdot b$ and $r_2 = c$. In this case, we find $L(a \cdot b + c) = \{ab, c\}$. But there is nothing in Definition 3.2 to stop us from taking $r_1 = a$ and $r_2 = b + c$. We now get a different result, $L(a \cdot b + c) = \{ab, ac\}$. To overcome this, we could require that all expressions be fully parenthesized, but this gives cumbersome results. Instead, we use a convention familiar from mathematics and programming languages. We establish a set of precedence rules for evaluation in which star-closure precedes concatenation and concatenation precedes union. Also, the symbol for concatenation may be omitted, so we can write r_1r_2 for $r_1 \cdot r_2$.

Example 3.2

Exhibit the language $L(a^* \cdot (a+b))$ in set notation.

$$\begin{aligned} L(a^* \cdot (a+b)) &= L(a^*)L(a+b) = \\ &= (L(a))^*(L(a) \cup L(b)) = \\ &= \{\lambda, a, aa, aaa, \ldots\}\{a, b\} = \\ &= \{a, aa, aaa, \ldots, b, ab, aab, \ldots\}. \end{aligned}$$

There is one problem with rules (4) to (7) in Definition 3.2. They define a language precisely if r_1 and r_2 are given, but there may be some ambiguity in breaking a complicated expression into parts. Consider, for example, the regular expression $a \cdot b + c$. We can consider this as being made up of $r_1 = a \cdot b$ and $r_2 = c$. In this case, we find $L(a \cdot b + c) = \{ab, c\}$. But there is nothing in Definition 3.2 to stop us from taking $r_1 = a$ and $r_2 = b + c$. We now get a different result, $L(a \cdot b + c) = \{ab, ac\}$. To overcome this, we could require that all expressions be fully parenthesized, but this gives cumbersome results. Instead, we use a convention familiar from mathematics and programming languages. We establish a set of precedence rules for evaluation in which star-closure precedes concatenation and concatenation precedes union. Also, the symbol for concatenation may be omitted, so we can write $r_1 r_2$ for $r_1 \cdot r_2$.

Example 3.2

Exhibit the language $L(a^* \cdot (a+b))$ in set notation.

$$\begin{aligned} L(a^* \cdot (a+b)) &= L(a^*)L(a+b) = \\ &= (L(a))^*(L(a) \cup L(b)) = \\ &= \{\lambda, a, aa, aaa, \ldots\}\{a, b\} = \\ &= \{a, aa, aaa, \ldots, b, ab, aab, \ldots\}. \end{aligned}$$

There is one problem with rules (4) to (7) in Definition 3.2. They define a language precisely if r_1 and r_2 are given, but there may be some ambiguity in breaking a complicated expression into parts. Consider, for example, the regular expression $a \cdot b + c$. We can consider this as being made up of $r_1 = a \cdot b$ and $r_2 = c$. In this case, we find $L(a \cdot b + c) = \{ab, c\}$. But there is nothing in Definition 3.2 to stop us from taking $r_1 = a$ and $r_2 = b + c$. We now get a different result, $L(a \cdot b + c) = \{ab, ac\}$. To overcome this, we could require that all expressions be fully parenthesized, but this gives cumbersome results. Instead, we use a convention familiar from mathematics and programming languages. We establish a set of precedence rules for evaluation in which star-closure precedes concatenation and concatenation precedes union. Also, the symbol for concatenation may be omitted, so we can write r_1r_2 for $r_1 \cdot r_2$.

Example 3.2

Exhibit the language $L(a^* \cdot (a+b))$ in set notation.

$$\begin{aligned} L(a^* \cdot (a+b)) &= L(a^*)L(a+b) = \\ &= (L(a))^*(L(a) \cup L(b)) = \\ &= \{\lambda, a, aa, aaa, \ldots\}\{a, b\} = \\ &= \{a, aa, aaa, \ldots, b, ab, aab, \ldots\}. \end{aligned}$$

There is one problem with rules (4) to (7) in Definition 3.2. They define a language precisely if r_1 and r_2 are given, but there may be some ambiguity in breaking a complicated expression into parts. Consider, for example, the regular expression $a \cdot b + c$. We can consider this as being made up of $r_1 = a \cdot b$ and $r_2 = c$. In this case, we find $L(a \cdot b + c) = \{ab, c\}$. But there is nothing in Definition 3.2 to stop us from taking $r_1 = a$ and $r_2 = b + c$. We now get a different result, $L(a \cdot b + c) = \{ab, ac\}$. To overcome this, we could require that all expressions be fully parenthesized, but this gives cumbersome results. Instead, we use a convention familiar from mathematics and programming languages. We establish a set of precedence rules for evaluation in which star-closure precedes concatenation and concatenation precedes union. Also, the symbol for concatenation may be omitted, so we can write $r_1 r_2$ for $r_1 \cdot r_2$.

Example 3.2

Exhibit the language $L(a^* \cdot (a+b))$ in set notation.

$$\begin{aligned} L(a^* \cdot (a+b)) &= L(a^*)L(a+b) = \\ &= (L(a))^*(L(a) \cup L(b)) = \\ &= \{\lambda, a, aa, aaa, \ldots\}\{a, b\} = \\ &= \{a, aa, aaa, \ldots, b, ab, aab, \ldots\}. \end{aligned}$$

There is one problem with rules (4) to (7) in Definition 3.2. They define a language precisely if r_1 and r_2 are given, but there may be some ambiguity in breaking a complicated expression into parts. Consider, for example, the regular expression $a \cdot b + c$. We can consider this as being made up of $r_1 = a \cdot b$ and $r_2 = c$. In this case, we find $L(a \cdot b + c) = \{ab, c\}$. But there is nothing in Definition 3.2 to stop us from taking $r_1 = a$ and $r_2 = b + c$. We now get a different result, $L(a \cdot b + c) = \{ab, ac\}$. To overcome this, we could require that all expressions be fully parenthesized, but this gives cumbersome results. Instead, we use a convention familiar from mathematics and programming languages. We establish a set of precedence rules for evaluation in which star-closure precedes concatenation and concatenation precedes union. Also, the symbol for concatenation may be omitted, so we can write $r_1 r_2$ for $r_1 \cdot r_2$.

Example 3.2

Exhibit the language $L(a^* \cdot (a+b))$ in set notation.

$$\begin{aligned} L(a^* \cdot (a+b)) &= L(a^*)L(a+b) = \\ &= (L(a))^*(L(a) \cup L(b)) = \\ &= \{\lambda, a, aa, aaa, \ldots\}\{a, b\} = \\ &= \{a, aa, aaa, \ldots, b, ab, aab, \ldots\}. \end{aligned}$$

There is one problem with rules (4) to (7) in Definition 3.2. They define a language precisely if r_1 and r_2 are given, but there may be some ambiguity in breaking a complicated expression into parts. Consider, for example, the regular expression $a \cdot b + c$. We can consider this as being made up of $r_1 = a \cdot b$ and $r_2 = c$. In this case, we find $L(a \cdot b + c) = \{ab, c\}$. But there is nothing in Definition 3.2 to stop us from taking $r_1 = a$ and $r_2 = b + c$. We now get a different result, $L(a \cdot b + c) = \{ab, ac\}$. To overcome this, we could require that all expressions be fully parenthesized, but this gives cumbersome results. Instead, we use a convention familiar from mathematics and programming languages. We establish a set of precedence rules for evaluation in which star-closure precedes concatenation and concatenation precedes union. Also, the symbol for concatenation may be omitted, so we can write $r_1 r_2$ for $r_1 \cdot r_2$.

Example 3.2

Exhibit the language $L(a^* \cdot (a+b))$ in set notation.

$$\begin{aligned} L(a^* \cdot (a+b)) &= L(a^*)L(a+b) = \\ &= (L(a))^*(L(a) \cup L(b)) = \\ &= \{\lambda, a, aa, aaa, \ldots\}\{a, b\} = \\ &= \{a, aa, aaa, \ldots, b, ab, aab, \ldots\}. \end{aligned}$$

There is one problem with rules (4) to (7) in Definition 3.2. They define a language precisely if r_1 and r_2 are given, but there may be some ambiguity in breaking a complicated expression into parts. Consider, for example, the regular expression $a \cdot b + c$. We can consider this as being made up of $r_1 = a \cdot b$ and $r_2 = c$. In this case, we find $L(a \cdot b + c) = \{ab, c\}$. But there is nothing in Definition 3.2 to stop us from taking $r_1 = a$ and $r_2 = b + c$. We now get a different result, $L(a \cdot b + c) = \{ab, ac\}$. To overcome this, we could require that all expressions be fully parenthesized, but this gives cumbersome results. Instead, we use a convention familiar from mathematics and programming languages. We establish a set of precedence rules for evaluation in which star-closure precedes concatenation and concatenation precedes union. Also, the symbol for concatenation may be omitted, so we can write $r_1 r_2$ for $r_1 \cdot r_2$.

With a little practice, we can see quickly what language a particular regular expression denotes.

Example 3.3

For $\Sigma = \{a, b\}$, the expression

$$r = (a + b)^* (a + bb)$$

is regular. It denotes the language

$$L(r) = \{a, bb, aa, abb, ba, bbb, \ldots\}.$$

We can see this by considering the various parts of r. The first part, $(a + b)^*$, stands for any string of a's and b's. The second part, $(a + bb)$ represents either symbol a or the double of b. Consequently, $L(r)$ is the set of all strings on $\{a, b\}$, terminated by either the symbol a or bb.

Example 3.4

The expression

$$r = (aa)^* (bb)^* b$$

denotes the set of all strings with an even number of a's followed by an odd number of b's; that is,

$$L(r) = \left\{a^{2n} b^{2m+1} : n \geqslant 0, m \geqslant 0\right\}.$$

Going from an informal description or set notation to a regular expression tends to be a little harder.

With a little practice, we can see quickly what language a particular regular expression denotes.

Example 3.3

For $\Sigma = \{a, b\}$, the expression

$$r = (a + b)^* (a + bb)$$

is regular. It denotes the language

$$L(r) = \{a, bb, aa, abb, ba, bbb, \ldots\}.$$

We can see this by considering the various parts of r. The first part, $(a + b)^*$, stands for any string of a's and b's. The second part, $(a + bb)$ represents either symbol a or the double of b. Consequently, $L(r)$ is the set of all strings on $\{a, b\}$, terminated by either the symbol a or bb.

Example 3.4

The expression

$$r = (aa)^* (bb)^* b$$

denotes the set of all strings with an even number of a's followed by an odd number of b's; that is,

$$L(r) = \{a^{2n} b^{2m+1} : n \geqslant 0, m \geqslant 0\}.$$

Going from an informal description or set notation to a regular expression tends to be a little harder.

With a little practice, we can see quickly what language a particular regular expression denotes.

Example 3.3

For $\Sigma = \{a, b\}$, the expression

$$r = (a + b)^* (a + bb)$$

is regular. It denotes the language

$$L(r) = \{a, bb, aa, abb, ba, bbb, \ldots\}.$$

We can see this by considering the various parts of r. The first part, $(a + b)^*$, stands for any string of a's and b's. The second part, $(a + bb)$ represents either symbol a or the double of b. Consequently, $L(r)$ is the set of all strings on $\{a, b\}$, terminated by either the symbol a or bb.

Example 3.4

The expression

$$r = (aa)^* (bb)^* b$$

denotes the set of all strings with an even number of a's followed by an odd number of b's; that is,

$$L(r) = \{a^{2n} b^{2m+1} : n \geqslant 0, m \geqslant 0\}.$$

Going from an informal description or set notation to a regular expression tends to be a little harder.

With a little practice, we can see quickly what language a particular regular expression denotes.

Example 3.3

For $\Sigma = \{a, b\}$, the expression

$$r = (a + b)^* (a + bb)$$

is regular. It denotes the language

$$L(r) = \{a, bb, aa, abb, ba, bbb, \ldots\}.$$

We can see this by considering the various parts of r. The first part, $(a + b)^*$, stands for any string of a's and b's. The second part, $(a + bb)$ represents either symbol a or the double of b. Consequently, $L(r)$ is the set of all strings on $\{a, b\}$, terminated by either the symbol a or bb.

Example 3.4

The expression

$$r = (aa)^* (bb)^* b$$

denotes the set of all strings with an even number of a's followed by an odd number of b's; that is,

$$L(r) = \{a^{2n} b^{2m+1} : n \geqslant 0, m \geqslant 0\}.$$

Going from an informal description or set notation to a regular expression tends to be a little harder.

With a little practice, we can see quickly what language a particular regular expression denotes.

Example 3.3

For $\Sigma = \{a, b\}$, the expression

$$r = (a + b)^*(a + bb)$$

is regular. It denotes the language

$$L(r) = \{a, bb, aa, abb, ba, bbb, \ldots\}.$$

We can see this by considering the various parts of r. The first part, $(a + b)^*$, stands for any string of a's and b's. The second part, $(a + bb)$ represents either symbol a or the double of b. Consequently, $L(r)$ is the set of all strings on $\{a, b\}$, terminated by either the symbol a or bb.

Example 3.4

The expression

$$r = (aa)^*(bb)^*b$$

denotes the set of all strings with an even number of a's followed by an odd number of b's; that is,

$$L(r) = \{a^{2n}b^{2m+1} : n \geqslant 0, m \geqslant 0\}.$$

Going from an informal description or set notation to a regular expression tends to be a little harder.

With a little practice, we can see quickly what language a particular regular expression denotes.

Example 3.3

For $\Sigma = \{a, b\}$, the expression

$$r = (a + b)^*(a + bb)$$

is regular. It denotes the language

$$L(r) = \{a, bb, aa, abb, ba, bbb, \ldots\}.$$

We can see this by considering the various parts of r. The first part, $(a + b)^*$, stands for any string of a's and b's. The second part, $(a + bb)$ represents either symbol a or the double of b. Consequently, $L(r)$ is the set of all strings on $\{a, b\}$, terminated by either the symbol a or bb.

Example 3.4

The expression

$$r = (aa)^*(bb)^*b$$

denotes the set of all strings with an even number of a's followed by an odd number of b's; that is,

$$L(r) = \{a^{2n}b^{2m+1} : n \geqslant 0, m \geqslant 0\}.$$

Going from an informal description or set notation to a regular expression tends to be a little harder.

3.1 Regular Expressions

With a little practice, we can see quickly what language a particular regular expression denotes.

Example 3.3

For $\Sigma = \{a, b\}$, the expression

$$r = (a+b)^*(a+bb)$$

is regular. It denotes the language

$$L(r) = \{a, bb, aa, abb, ba, bbb, \ldots\}.$$

We can see this by considering the various parts of r. The first part, $(a+b)^*$, stands for any string of a's and b's. The second part, $(a+bb)$ represents either symbol a or the double of b. Consequently, $L(r)$ is the set of all strings on $\{a, b\}$, terminated by either the symbol a or bb.

Example 3.4

The expression

$$r = (aa)^*(bb)^*b$$

denotes the set of all strings with an even number of a's followed by an odd number of b's; that is,

$$L(r) = \{a^{2n}b^{2m+1} : n \geqslant 0, m \geqslant 0\}.$$

Going from an informal description or set notation to a regular expression tends to be a little harder.

With a little practice, we can see quickly what language a particular regular expression denotes.

Example 3.3

For $\Sigma = \{a, b\}$, the expression

$$r = (a + b)^*(a + bb)$$

is regular. It denotes the language

$$L(r) = \{a, bb, aa, abb, ba, bbb, \ldots\}.$$

We can see this by considering the various parts of r. The first part, $(a + b)^*$, stands for any string of a's and b's. The second part, $(a + bb)$ represents either symbol a or the double of b. Consequently, $L(r)$ is the set of all strings on $\{a, b\}$, terminated by either the symbol a or bb.

Example 3.4

The expression

$$r = (aa)^*(bb)^*b$$

denotes the set of all strings with an even number of a's followed by an odd number of b's; that is,

$$L(r) = \{a^{2n}b^{2m+1} : n \geqslant 0, m \geqslant 0\}.$$

Going from an informal description or set notation to a regular expression tends to be a little harder.

3.1 Regular Expressions

With a little practice, we can see quickly what language a particular regular expression denotes.

Example 3.3

For $\Sigma = \{a, b\}$, the expression

$$r = (a+b)^*(a+bb)$$

is regular. It denotes the language

$$L(r) = \{a, bb, aa, abb, ba, bbb, \ldots\}.$$

We can see this by considering the various parts of r. The first part, $(a+b)^*$, stands for any string of a's and b's. The second part, $(a+bb)$ represents either symbol a or the double of b. Consequently, $L(r)$ is the set of all strings on $\{a, b\}$, terminated by either the symbol a or bb.

Example 3.4

The expression

$$r = (aa)^*(bb)^*b$$

denotes the set of all strings with an even number of a's followed by an odd number of b's; that is,

$$L(r) = \{a^{2n}b^{2m+1} : n \geqslant 0, m \geqslant 0\}.$$

Going from an informal description or set notation to a regular expression tends to be a little harder.

With a little practice, we can see quickly what language a particular regular expression denotes.

Example 3.3

For $\Sigma = \{a, b\}$, the expression

$$r = (a + b)^*(a + bb)$$

is regular. It denotes the language

$$L(r) = \{a, bb, aa, abb, ba, bbb, \ldots\}.$$

We can see this by considering the various parts of r. The first part, $(a + b)^*$, stands for any string of a's and b's. The second part, $(a + bb)$ represents either symbol a or the double of b. Consequently, $L(r)$ is the set of all strings on $\{a, b\}$, terminated by either the symbol a or bb.

Example 3.4

The expression

$$r = (aa)^*(bb)^*b$$

denotes the set of all strings with an even number of a's followed by an odd number of b's; that is,

$$L(r) = \{a^{2n}b^{2m+1} : n \geqslant 0, m \geqslant 0\}.$$

Going from an informal description or set notation to a regular expression tends to be a little harder.

3.1 Regular Expressions

With a little practice, we can see quickly what language a particular regular expression denotes.

Example 3.3

For $\Sigma = \{a, b\}$, the expression

$$r = (a + b)^*(a + bb)$$

is regular. It denotes the language

$$L(r) = \{a, bb, aa, abb, ba, bbb, \ldots\}.$$

We can see this by considering the various parts of r. The first part, $(a + b)^*$, stands for any string of a's and b's. The second part, $(a + bb)$ represents either symbol a or the double of b. Consequently, $L(r)$ is the set of all strings on $\{a, b\}$, terminated by either the symbol a or bb.

Example 3.4

The expression

$$r = (aa)^*(bb)^*b$$

denotes the set of all strings with an even number of a's followed by an odd number of b's; that is,

$$L(r) = \{a^{2n}b^{2m+1} : n \geqslant 0, m \geqslant 0\}.$$

Going from an informal description or set notation to a regular expression tends to be a little harder.

With a little practice, we can see quickly what language a particular regular expression denotes.

Example 3.3

For $\Sigma = \{a, b\}$, the expression

$$r = (a + b)^*(a + bb)$$

is regular. It denotes the language

$$L(r) = \{a, bb, aa, abb, ba, bbb, \ldots\}.$$

We can see this by considering the various parts of r. The first part, $(a + b)^*$, stands for any string of a's and b's. The second part, $(a + bb)$ represents either symbol a or the double of b. Consequently, $L(r)$ is the set of all strings on $\{a, b\}$, terminated by either the symbol a or bb.

Example 3.4

The expression

$$r = (aa)^*(bb)^*b$$

denotes the set of all strings with an even number of a's followed by an odd number of b's; that is,

$$L(r) = \{a^{2n}b^{2m+1} : n \geqslant 0, m \geqslant 0\}.$$

Going from an informal description or set notation to a regular expression tends to be a little harder.

3.1 Regular Expressions

With a little practice, we can see quickly what language a particular regular expression denotes.

Example 3.3

For $\Sigma = \{a, b\}$, the expression

$$r = (a + b)^*(a + bb)$$

is regular. It denotes the language

$$L(r) = \{a, bb, aa, abb, ba, bbb, \ldots\}.$$

We can see this by considering the various parts of r. The first part, $(a + b)^*$, stands for any string of a's and b's. The second part, $(a + bb)$ represents either symbol a or the double of b. Consequently, $L(r)$ is the set of all strings on $\{a, b\}$, terminated by either the symbol a or bb.

Example 3.4

The expression

$$r = (aa)^*(bb)^*b$$

denotes the set of all strings with an even number of a's followed by an odd number of b's; that is,

$$L(r) = \{a^{2n}b^{2m+1} : n \geqslant 0, m \geqslant 0\}.$$

Going from an informal description or set notation to a regular expression tends to be a little harder.

With a little practice, we can see quickly what language a particular regular expression denotes.

Example 3.3

For $\Sigma = \{a, b\}$, the expression

$$r = (a + b)^*(a + bb)$$

is regular. It denotes the language

$$L(r) = \{a, bb, aa, abb, ba, bbb, \ldots\}.$$

We can see this by considering the various parts of r. The first part, $(a + b)^*$, stands for any string of a's and b's. The second part, $(a + bb)$ represents either symbol a or the double of b. Consequently, $L(r)$ is the set of all strings on $\{a, b\}$, terminated by either the symbol a or bb.

Example 3.4

The expression

$$r = (aa)^*(bb)^*b$$

denotes the set of all strings with an even number of a's followed by an odd number of b's; that is,

$$L(r) = \left\{a^{2n}b^{2m+1} : n \geqslant 0, m \geqslant 0\right\}.$$

Going from an informal description or set notation to a regular expression tends to be a little harder.

With a little practice, we can see quickly what language a particular regular expression denotes.

Example 3.3

For $\Sigma = \{a, b\}$, the expression

$$r = (a + b)^*(a + bb)$$

is regular. It denotes the language

$$L(r) = \{a, bb, aa, abb, ba, bbb, \ldots\}.$$

We can see this by considering the various parts of r. The first part, $(a + b)^*$, stands for any string of a's and b's. The second part, $(a + bb)$ represents either symbol a or the double of b. Consequently, $L(r)$ is the set of all strings on $\{a, b\}$, terminated by either the symbol a or bb.

Example 3.4

The expression

$$r = (aa)^*(bb)^*b$$

denotes the set of all strings with an even number of a's followed by an odd number of b's; that is,

$$L(r) = \left\{a^{2n}b^{2m+1} : n \geqslant 0, m \geqslant 0\right\}.$$

Going from an informal description or set notation to a regular expression tends to be a little harder.

With a little practice, we can see quickly what language a particular regular expression denotes.

Example 3.3

For $\Sigma = \{a, b\}$, the expression
$$r = (a + b)^*(a + bb)$$
is regular. It denotes the language
$$L(r) = \{a, bb, aa, abb, ba, bbb, \dots\}.$$
We can see this by considering the various parts of r. The first part, $(a + b)^*$, stands for any string of a's and b's. The second part, $(a + bb)$ represents either symbol a or the double of b. Consequently, $L(r)$ is the set of all strings on $\{a, b\}$, terminated by either the symbol a or bb.

Example 3.4

The expression
$$r = (aa)^*(bb)^*b$$
denotes the set of all strings with an even number of a's followed by an odd number of b's; that is,
$$L(r) = \left\{a^{2n}b^{2m+1} : n \geqslant 0, m \geqslant 0\right\}.$$

Going from an informal description or set notation to a regular expression tends to be a little harder.

With a little practice, we can see quickly what language a particular regular expression denotes.

Example 3.3

For $\Sigma = \{a, b\}$, the expression

$$r = (a + b)^* (a + bb)$$

is regular. It denotes the language

$$L(r) = \{a, bb, aa, abb, ba, bbb, \ldots\}.$$

We can see this by considering the various parts of r. The first part, $(a + b)^*$, stands for any string of a's and b's. The second part, $(a + bb)$ represents either symbol a or the double of b. Consequently, $L(r)$ is the set of all strings on $\{a, b\}$, terminated by either the symbol a or bb.

Example 3.4

The expression

$$r = (aa)^* (bb)^* b$$

denotes the set of all strings with an even number of a's followed by an odd number of b's; that is,

$$L(r) = \left\{a^{2n} b^{2m+1} : n \geqslant 0, m \geqslant 0\right\}.$$

Going from an informal description or set notation to a regular expression tends to be a little harder.

With a little practice, we can see quickly what language a particular regular expression denotes.

Example 3.3

For $\Sigma = \{a, b\}$, the expression
$$r = (a + b)^*(a + bb)$$
is regular. It denotes the language
$$L(r) = \{a, bb, aa, abb, ba, bbb, \ldots\}.$$
We can see this by considering the various parts of r. The first part, $(a + b)^*$, stands for any string of a's and b's. The second part, $(a + bb)$ represents either symbol a or the double of b. Consequently, $L(r)$ is the set of all strings on $\{a, b\}$, terminated by either the symbol a or bb.

Example 3.4

The expression
$$r = (aa)^*(bb)^*b$$
denotes the set of all strings with an even number of a's followed by an odd number of b's; that is,
$$L(r) = \left\{a^{2n}b^{2m+1} : n \geqslant 0, m \geqslant 0\right\}.$$

Going from an informal description or set notation to a regular expression tends to be a little harder.

With a little practice, we can see quickly what language a particular regular expression denotes.

Example 3.3

For $\Sigma = \{a, b\}$, the expression

$$r = (a+b)^*(a+bb)$$

is regular. It denotes the language

$$L(r) = \{a, bb, aa, abb, ba, bbb, \ldots\}.$$

We can see this by considering the various parts of r. The first part, $(a+b)^*$, stands for any string of a's and b's. The second part, $(a+bb)$ represents either symbol a or the double of b. Consequently, $L(r)$ is the set of all strings on $\{a, b\}$, terminated by either the symbol a or bb.

Example 3.4

The expression

$$r = (aa)^*(bb)^*b$$

denotes the set of all strings with an even number of a's followed by an odd number of b's; that is,

$$L(r) = \left\{a^{2n}b^{2m+1} : n \geqslant 0, m \geqslant 0\right\}.$$

Going from an informal description or set notation to a regular expression tends to be a little harder.

With a little practice, we can see quickly what language a particular regular expression denotes.

Example 3.3

For $\Sigma = \{a, b\}$, the expression
$$r = (a+b)^*(a+bb)$$
is regular. It denotes the language
$$L(r) = \{a, bb, aa, abb, ba, bbb, \ldots\}.$$
We can see this by considering the various parts of r. The first part, $(a+b)^*$, stands for any string of a's and b's. The second part, $(a+bb)$ represents either symbol a or the double of b. Consequently, $L(r)$ is the set of all strings on $\{a, b\}$, terminated by either the symbol a or bb.

Example 3.4

The expression
$$r = (aa)^*(bb)^*b$$
denotes the set of all strings with an even number of a's followed by an odd number of b's; that is,
$$L(r) = \left\{a^{2n}b^{2m+1} : n \geqslant 0, m \geqslant 0\right\}.$$

Going from an informal description or set notation to a regular expression tends to be a little harder.

Example 3.5

For $\Sigma = \{0, 1\}$, give a regular expression r such that

$$L(r) = \{w \in \Sigma^* : w \text{ has at least one pair of consecutive zeros}\}.$$

One can arrive at an answer by reasoning something like this: Every string in $L(r)$ must contain 00 somewhere, but what comes before and what goes after is completely arbitrary. An arbitrary string on $\{0, 1\}$ can be denoted by $(0 + 1)^*$. Putting these observations together, we arrive at the solution

$$r = (0 + 1)^* 00(0 + 1)^*.$$

Example 3.6

Find a regular expression for the language

$$L = \{w \in \{0, 1\}^* : w \text{ has no pair of consecutive zeros}\}.$$

Even though this looks similar to Example 3.5, the answer is harder to construct. One helpful observation is that whenever the symbol 0 occurs, it must be followed immediately by the symbol 1. Such a substring may be preceded and followed by an arbitrary number of 1's. This suggests that the answer involves the repetition of strings of the form $1 \ldots 101 \ldots 1$, that is, the language denoted by the regular expression $(1^* 011^*)^*$. However, the answer is still incomplete, because the strings ending in 0 or consisting of all 1's are unaccounted for. After taking care of these special cases we arrive at the answer

$$r = (1^* 011^*)^* (0 + \lambda) + 1^* (0 + \lambda).$$

Example 3.5

For $\Sigma = \{0, 1\}$, give a regular expression r such that

$$L(r) = \{w \in \Sigma^* : w \text{ has at least one pair of consecutive zeros}\}.$$

One can arrive at an answer by reasoning something like this: Every string in $L(r)$ must contain 00 somewhere, but what comes before and what goes after is completely arbitrary. An arbitrary string on $\{0, 1\}$ can be denoted by $(0 + 1)^*$. Putting these observations together, we arrive at the solution

$$r = (0 + 1)^* 00 (0 + 1)^*.$$

Example 3.6

Find a regular expression for the language

$$L = \{w \in \{0, 1\}^* : w \text{ has no pair of consecutive zeros}\}.$$

Even though this looks similar to Example 3.5, the answer is harder to construct. One helpful observation is that whenever the symbol 0 occurs, it must be followed immediately by the symbol 1. Such a substring may be preceded and followed by an arbitrary number of 1's. This suggests that the answer involves the repetition of strings of the form $1 \ldots 101 \ldots 1$, that is, the language denoted by the regular expression $(1^* 011^*)^*$. However, the answer is still incomplete, because the strings ending in 0 or consisting of all 1's are unaccounted for. After taking care of these special cases we arrive at the answer

$$r = (1^* 011^*)^* (0 + \lambda) + 1^* (0 + \lambda).$$

Example 3.5

For $\Sigma = \{0, 1\}$, give a regular expression r such that

$$L(r) = \{w \in \Sigma^* : w \text{ has at least one pair of consecutive zeros}\}.$$

One can arrive at an answer by reasoning something like this: Every string in $L(r)$ must contain 00 somewhere, but what comes before and what goes after is completely arbitrary. An arbitrary string on $\{0, 1\}$ can be denoted by $(0 + 1)^*$. Putting these observations together, we arrive at the solution

$$r = (0 + 1)^* 00 (0 + 1)^*.$$

Example 3.6

Find a regular expression for the language

$$L = \{w \in \{0, 1\}^* : w \text{ has no pair of consecutive zeros}\}.$$

Even though this looks similar to Example 3.5, the answer is harder to construct. One helpful observation is that whenever the symbol 0 occurs, it must be followed immediately by the symbol 1. Such a substring may be preceded and followed by an arbitrary number of 1's. This suggests that the answer involves the repetition of strings of the form $1 \ldots 101 \ldots 1$, that is, the language denoted by the regular expression $(1^* 011^*)^*$. However, the answer is still incomplete, because the strings ending in 0 or consisting of all 1's are unaccounted for. After taking care of these special cases we arrive at the answer

$$r = (1^* 011^*)^* (0 + \lambda) + 1^* (0 + \lambda).$$

Example 3.5

For $\Sigma = \{0, 1\}$, give a regular expression r such that

$$L(r) = \{w \in \Sigma^* : w \text{ has at least one pair of consecutive zeros}\}.$$

One can arrive at an answer by reasoning something like this: Every string in $L(r)$ must contain 00 somewhere, but what comes before and what goes after is completely arbitrary. An arbitrary string on $\{0, 1\}$ can be denoted by $(0+1)^*$. Putting these observations together, we arrive at the solution

$$r = (0+1)^* 00 (0+1)^*.$$

Example 3.6

Find a regular expression for the language

$$L = \{w \in \{0, 1\}^* : w \text{ has no pair of consecutive zeros}\}.$$

Even though this looks similar to Example 3.5, the answer is harder to construct. One helpful observation is that whenever the symbol 0 occurs, it must be followed immediately by the symbol 1. Such a substring may be preceded and followed by an arbitrary number of 1's. This suggests that the answer involves the repetition of strings of the form $1 \ldots 101 \ldots 1$, that is, the language denoted by the regular expression $(1^*011^*)^*$. However, the answer is still incomplete, because the strings ending in 0 or consisting of all 1's are unaccounted for. After taking care of these special cases we arrive at the answer

$$r = (1^*011^*)^*(0+\lambda) + 1^*(0+\lambda).$$

Example 3.5

For $\Sigma = \{0, 1\}$, give a regular expression r such that

$$L(r) = \{w \in \Sigma^* : w \text{ has at least one pair of consecutive zeros}\}.$$

One can arrive at an answer by reasoning something like this: Every string in $L(r)$ must contain 00 somewhere, but what comes before and what goes after is completely arbitrary. An arbitrary string on $\{0, 1\}$ can be denoted by $(0 + 1)^*$. Putting these observations together, we arrive at the solution

$$r = (0 + 1)^* 00 (0 + 1)^*.$$

Example 3.6

Find a regular expression for the language

$$L = \{w \in \{0, 1\}^* : w \text{ has no pair of consecutive zeros}\}.$$

Even though this looks similar to Example 3.5, the answer is harder to construct. One helpful observation is that whenever the symbol 0 occurs, it must be followed immediately by the symbol 1. Such a substring may be preceded and followed by an arbitrary number of 1's. This suggests that the answer involves the repetition of strings of the form $1 \ldots 101 \ldots 1$, that is, the language denoted by the regular expression $(1^* 011^*)^*$. However, the answer is still incomplete, because the strings ending in 0 or consisting of all 1's are unaccounted for. After taking care of these special cases we arrive at the answer

$$r = (1^* 011^*)^* (0 + \lambda) + 1^* (0 + \lambda).$$

3.1 Regular Expressions

Example 3.5

For $\Sigma = \{0, 1\}$, give a regular expression r such that

$$L(r) = \{w \in \Sigma^* : w \text{ has at least one pair of consecutive zeros}\}.$$

One can arrive at an answer by reasoning something like this: Every string in $L(r)$ must contain 00 somewhere, but what comes before and what goes after is completely arbitrary. An arbitrary string on $\{0, 1\}$ can be denoted by $(0 + 1)^*$. Putting these observations together, we arrive at the solution

$$r = (0 + 1)^* 00 (0 + 1)^*.$$

Example 3.6

Find a regular expression for the language

$$L = \{w \in \{0, 1\}^* : w \text{ has no pair of consecutive zeros}\}.$$

Even though this looks similar to Example 3.5, the answer is harder to construct. One helpful observation is that whenever the symbol 0 occurs, it must be followed immediately by the symbol 1. Such a substring may be preceded and followed by an arbitrary number of 1's. This suggests that the answer involves the repetition of strings of the form $1 \dots 101 \dots 1$, that is, the language denoted by the regular expression $(1^* 011^*)^*$. However, the answer is still incomplete, because the strings ending in 0 or consisting of all 1's are unaccounted for. After taking care of these special cases we arrive at the answer

$$r = (1^* 011^*)^* (0 + \lambda) + 1^* (0 + \lambda).$$

Example 3.5

For $\Sigma = \{0, 1\}$, give a regular expression r such that
$$L(r) = \{w \in \Sigma^* : w \text{ has at least one pair of consecutive zeros}\}.$$
One can arrive at an answer by reasoning something like this: Every string in $L(r)$ must contain 00 somewhere, but what comes before and what goes after is completely arbitrary. An arbitrary string on $\{0, 1\}$ can be denoted by $(0 + 1)^*$. Putting these observations together, we arrive at the solution
$$r = (0 + 1)^* 00 (0 + 1)^*.$$

Example 3.6

Find a regular expression for the language
$$L = \{w \in \{0, 1\}^* : w \text{ has no pair of consecutive zeros}\}.$$
Even though this looks similar to Example 3.5, the answer is harder to construct. One helpful observation is that whenever the symbol 0 occurs, it must be followed immediately by the symbol 1. Such a substring may be preceded and followed by an arbitrary number of 1's. This suggests that the answer involves the repetition of strings of the form $1 \ldots 101 \ldots 1$, that is, the language denoted by the regular expression $(1^* 011^*)^*$. However, the answer is still incomplete, because the strings ending in 0 or consisting of all 1's are unaccounted for. After taking care of these special cases we arrive at the answer
$$r = (1^* 011^*)^* (0 + \lambda) + 1^* (0 + \lambda).$$

3.1 Regular Expressions

Example 3.5

For $\Sigma = \{0, 1\}$, give a regular expression r such that

$$L(r) = \{w \in \Sigma^* : w \text{ has at least one pair of consecutive zeros}\}.$$

One can arrive at an answer by reasoning something like this: Every string in $L(r)$ must contain 00 somewhere, but what comes before and what goes after is completely arbitrary. An arbitrary string on $\{0, 1\}$ can be denoted by $(0 + 1)^*$. Putting these observations together, we arrive at the solution

$$r = (0 + 1)^* 00 (0 + 1)^*.$$

Example 3.6

Find a regular expression for the language

$$L = \{w \in \{0, 1\}^* : w \text{ has no pair of consecutive zeros}\}.$$

Even though this looks similar to Example 3.5, the answer is harder to construct. One helpful observation is that whenever the symbol 0 occurs, it must be followed immediately by the symbol 1. Such a substring may be preceded and followed by an arbitrary number of 1's. This suggests that the answer involves the repetition of strings of the form $1 \ldots 101 \ldots 1$, that is, the language denoted by the regular expression $(1^* 011^*)^*$. However, the answer is still incomplete, because the strings ending in 0 or consisting of all 1's are unaccounted for. After taking care of these special cases we arrive at the answer

$$r = (1^* 011^*)^* (0 + \lambda) + 1^* (0 + \lambda).$$

Example 3.5

For $\Sigma = \{0, 1\}$, give a regular expression r such that

$$L(r) = \{w \in \Sigma^* : w \text{ has at least one pair of consecutive zeros}\}.$$

One can arrive at an answer by reasoning something like this: Every string in $L(r)$ must contain 00 somewhere, but what comes before and what goes after is completely arbitrary. An arbitrary string on $\{0, 1\}$ can be denoted by $(0 + 1)^*$. Putting these observations together, we arrive at the solution

$$r = (0 + 1)^* 00 (0 + 1)^*.$$

Example 3.6

Find a regular expression for the language

$$L = \{w \in \{0, 1\}^* : w \text{ has no pair of consecutive zeros}\}.$$

Even though this looks similar to Example 3.5, the answer is harder to construct. One helpful observation is that whenever the symbol 0 occurs, it must be followed immediately by the symbol 1. Such a substring may be preceded and followed by an arbitrary number of 1's. This suggests that the answer involves the repetition of strings of the form $1 \dots 101 \dots 1$, that is, the language denoted by the regular expression $(1^* 011^*)^*$. However, the answer is still incomplete, because the strings ending in 0 or consisting of all 1's are unaccounted for. After taking care of these special cases we arrive at the answer

$$r = (1^* 011^*)^* (0 + \lambda) + 1^* (0 + \lambda).$$

Example 3.5

For $\Sigma = \{0, 1\}$, give a regular expression r such that

$$L(r) = \{w \in \Sigma^* : w \text{ has at least one pair of consecutive zeros}\}.$$

One can arrive at an answer by reasoning something like this: Every string in $L(r)$ must contain 00 somewhere, but what comes before and what goes after is completely arbitrary. An arbitrary string on $\{0, 1\}$ can be denoted by $(0 + 1)^*$. Putting these observations together, we arrive at the solution

$$r = (0 + 1)^* 00 (0 + 1)^*.$$

Example 3.6

Find a regular expression for the language

$$L = \{w \in \{0, 1\}^* : w \text{ has no pair of consecutive zeros}\}.$$

Even though this looks similar to Example 3.5, the answer is harder to construct. One helpful observation is that whenever the symbol 0 occurs, it must be followed immediately by the symbol 1. Such a substring may be preceded and followed by an arbitrary number of 1's. This suggests that the answer involves the repetition of strings of the form $1 \ldots 101 \ldots 1$, that is, the language denoted by the regular expression $(1^* 011^*)^*$. However, the answer is still incomplete, because the strings ending in 0 or consisting of all 1's are unaccounted for. After taking care of these special cases we arrive at the answer

$$r = (1^* 011^*)^* (0 + \lambda) + 1^* (0 + \lambda).$$

Example 3.5

For $\Sigma = \{0, 1\}$, give a regular expression r such that

$$L(r) = \{w \in \Sigma^* : w \text{ has at least one pair of consecutive zeros}\}.$$

One can arrive at an answer by reasoning something like this: Every string in $L(r)$ must contain 00 somewhere, but what comes before and what goes after is completely arbitrary. An arbitrary string on $\{0, 1\}$ can be denoted by $(0 + 1)^*$. Putting these observations together, we arrive at the solution

$$r = (0 + 1)^* 00 (0 + 1)^*.$$

Example 3.6

Find a regular expression for the language

$$L = \{w \in \{0, 1\}^* : w \text{ has no pair of consecutive zeros}\}.$$

Even though this looks similar to Example 3.5, the answer is harder to construct. One helpful observation is that whenever the symbol 0 occurs, it must be followed immediately by the symbol 1. Such a substring may be preceded and followed by an arbitrary number of 1's. This suggests that the answer involves the repetition of strings of the form $1 \ldots 101 \ldots 1$, that is, the language denoted by the regular expression $(1^* 011^*)^*$. However, the answer is still incomplete, because the strings ending in 0 or consisting of all 1's are unaccounted for. After taking care of these special cases we arrive at the answer

$$r = (1^* 011^*)^* (0 + \lambda) + 1^* (0 + \lambda).$$

Example 3.5

For $\Sigma = \{0, 1\}$, give a regular expression r such that

$$L(r) = \{w \in \Sigma^* : w \text{ has at least one pair of consecutive zeros}\}.$$

One can arrive at an answer by reasoning something like this: Every string in $L(r)$ must contain 00 somewhere, but what comes before and what goes after is completely arbitrary. An arbitrary string on $\{0, 1\}$ can be denoted by $(0+1)^*$. Putting these observations together, we arrive at the solution

$$r = (0+1)^* 00 (0+1)^*.$$

Example 3.6

Find a regular expression for the language

$$L = \{w \in \{0,1\}^* : w \text{ has no pair of consecutive zeros}\}.$$

Even though this looks similar to Example 3.5, the answer is harder to construct. One helpful observation is that whenever the symbol 0 occurs, it must be followed immediately by the symbol 1. Such a substring may be preceded and followed by an arbitrary number of 1's. This suggests that the answer involves the repetition of strings of the form $1 \ldots 101 \ldots 1$, that is, the language denoted by the regular expression $(1^*011^*)^*$. However, the answer is still incomplete, because the strings ending in 0 or consisting of all 1's are unaccounted for. After taking care of these special cases we arrive at the answer

$$r = (1^*011^*)^*(0+\lambda) + 1^*(0+\lambda).$$

Example 3.5

For $\Sigma = \{0,1\}$, give a regular expression r such that

$$L(r) = \{w \in \Sigma^* : w \text{ has at least one pair of consecutive zeros}\}.$$

One can arrive at an answer by reasoning something like this: Every string in $L(r)$ must contain 00 somewhere, but what comes before and what goes after is completely arbitrary. An arbitrary string on $\{0,1\}$ can be denoted by $(0+1)^*$. Putting these observations together, we arrive at the solution

$$r = (0+1)^* 00 (0+1)^*.$$

Example 3.6

Find a regular expression for the language

$$L = \{w \in \{0,1\}^* : w \text{ has no pair of consecutive zeros}\}.$$

Even though this looks similar to Example 3.5, the answer is harder to construct. One helpful observation is that whenever the symbol 0 occurs, it must be followed immediately by the symbol 1. Such a substring may be preceded and followed by an arbitrary number of 1's. This suggests that the answer involves the repetition of strings of the form $1 \dots 101 \dots 1$, that is, the language denoted by the regular expression $(1^* 011^*)^*$. However, the answer is still incomplete, because the strings ending in 0 or consisting of all 1's are unaccounted for. After taking care of these special cases we arrive at the answer

$$r = (1^* 011^*)^* (0 + \lambda) + 1^* (0 + \lambda).$$

Example 3.5

For $\Sigma = \{0, 1\}$, give a regular expression r such that

$$L(r) = \{w \in \Sigma^* : w \text{ has at least one pair of consecutive zeros}\}.$$

One can arrive at an answer by reasoning something like this: Every string in $L(r)$ must contain 00 somewhere, but what comes before and what goes after is completely arbitrary. An arbitrary string on $\{0, 1\}$ can be denoted by $(0 + 1)^*$. Putting these observations together, we arrive at the solution

$$r = (0 + 1)^* 00 (0 + 1)^*.$$

Example 3.6

Find a regular expression for the language

$$L = \{w \in \{0, 1\}^* : w \text{ has no pair of consecutive zeros}\}.$$

Even though this looks similar to Example 3.5, the answer is harder to construct. One helpful observation is that whenever the symbol 0 occurs, it must be followed immediately by the symbol 1. Such a substring may be preceded and followed by an arbitrary number of 1's. This suggests that the answer involves the repetition of strings of the form $1 \ldots 101 \ldots 1$, that is, the language denoted by the regular expression $(1^* 011^*)^*$. However, the answer is still incomplete, because the strings ending in 0 or consisting of all 1's are unaccounted for. After taking care of these special cases we arrive at the answer

$$r = (1^* 011^*)^* (0 + \lambda) + 1^* (0 + \lambda).$$

Example 3.5

For $\Sigma = \{0, 1\}$, give a regular expression r such that

$$L(r) = \{w \in \Sigma^* : w \text{ has at least one pair of consecutive zeros}\}.$$

One can arrive at an answer by reasoning something like this: Every string in $L(r)$ must contain 00 somewhere, but what comes before and what goes after is completely arbitrary. An arbitrary string on $\{0, 1\}$ can be denoted by $(0 + 1)^*$. Putting these observations together, we arrive at the solution

$$r = (0 + 1)^* 00 (0 + 1)^*.$$

Example 3.6

Find a regular expression for the language

$$L = \{w \in \{0, 1\}^* : w \text{ has no pair of consecutive zeros}\}.$$

Even though this looks similar to Example 3.5, the answer is harder to construct. One helpful observation is that whenever the symbol 0 occurs, it must be followed immediately by the symbol 1. Such a substring may be preceded and followed by an arbitrary number of 1's. This suggests that the answer involves the repetition of strings of the form $1 \ldots 101 \ldots 1$, that is, the language denoted by the regular expression $(1^*011^*)^*$. However, the answer is still incomplete, because the strings ending in 0 or consisting of all 1's are unaccounted for. After taking care of these special cases we arrive at the answer

$$r = (1^*011^*)^*(0 + \lambda) + 1^*(0 + \lambda).$$

Example 3.5

For $\Sigma = \{0, 1\}$, give a regular expression r such that

$$L(r) = \{w \in \Sigma^* : w \text{ has at least one pair of consecutive zeros}\}.$$

One can arrive at an answer by reasoning something like this: Every string in $L(r)$ must contain 00 somewhere, but what comes before and what goes after is completely arbitrary. An arbitrary string on $\{0, 1\}$ can be denoted by $(0 + 1)^*$. Putting these observations together, we arrive at the solution

$$r = (0 + 1)^* 00 (0 + 1)^*.$$

Example 3.6

Find a regular expression for the language

$$L = \{w \in \{0, 1\}^* : w \text{ has no pair of consecutive zeros}\}.$$

Even though this looks similar to Example 3.5, the answer is harder to construct. One helpful observation is that whenever the symbol 0 occurs, it must be followed immediately by the symbol 1. Such a substring may be preceded and followed by an arbitrary number of 1's. This suggests that the answer involves the repetition of strings of the form $1 \ldots 101 \ldots 1$, that is, the language denoted by the regular expression $(1^*011^*)^*$. However, the answer is still incomplete, because the strings ending in 0 or consisting of all 1's are unaccounted for. After taking care of these special cases we arrive at the answer

$$r = (1^*011^*)^*(0 + \lambda) + 1^*(0 + \lambda).$$

Example 3.5

For $\Sigma = \{0, 1\}$, give a regular expression r such that

$$L(r) = \{w \in \Sigma^* : w \text{ has at least one pair of consecutive zeros}\}.$$

One can arrive at an answer by reasoning something like this: Every string in $L(r)$ must contain 00 somewhere, but what comes before and what goes after is completely arbitrary. An arbitrary string on $\{0, 1\}$ can be denoted by $(0 + 1)^*$. Putting these observations together, we arrive at the solution

$$r = (0 + 1)^* 00 (0 + 1)^*.$$

Example 3.6

Find a regular expression for the language

$$L = \{w \in \{0, 1\}^* : w \text{ has no pair of consecutive zeros}\}.$$

Even though this looks similar to Example 3.5, the answer is harder to construct. One helpful observation is that whenever the symbol 0 occurs, it must be followed immediately by the symbol 1. Such a substring may be preceded and followed by an arbitrary number of 1's. This suggests that the answer involves the repetition of strings of the form $1 \ldots 101 \ldots 1$, that is, the language denoted by the regular expression $(1^* 011^*)^*$. However, the answer is still incomplete, because the strings ending in 0 or consisting of all 1's are unaccounted for. After taking care of these special cases we arrive at the answer

$$r = (1^* 011^*)^* (0 + \lambda) + 1^* (0 + \lambda).$$

Example 3.5

For $\Sigma = \{0, 1\}$, give a regular expression r such that

$$L(r) = \{w \in \Sigma^* : w \text{ has at least one pair of consecutive zeros}\}.$$

One can arrive at an answer by reasoning something like this: Every string in $L(r)$ must contain 00 somewhere, but what comes before and what goes after is completely arbitrary. An arbitrary string on $\{0, 1\}$ can be denoted by $(0 + 1)^*$. Putting these observations together, we arrive at the solution

$$r = (0 + 1)^* 00 (0 + 1)^*.$$

Example 3.6

Find a regular expression for the language

$$L = \{w \in \{0, 1\}^* : w \text{ has no pair of consecutive zeros}\}.$$

Even though this looks similar to Example 3.5, the answer is harder to construct. One helpful observation is that whenever the symbol 0 occurs, it must be followed immediately by the symbol 1. Such a substring may be preceded and followed by an arbitrary number of 1's. This suggests that the answer involves the repetition of strings of the form $1 \dots 101 \dots 1$, that is, the language denoted by the regular expression $(1^* 011^*)^*$. However, the answer is still incomplete, because the strings ending in 0 or consisting of all 1's are unaccounted for. After taking care of these special cases we arrive at the answer

$$r = (1^* 011^*)^* (0 + \lambda) + 1^* (0 + \lambda).$$

Example 3.5

For $\Sigma = \{0, 1\}$, give a regular expression r such that

$$L(r) = \{w \in \Sigma^* : w \text{ has at least one pair of consecutive zeros}\}.$$

One can arrive at an answer by reasoning something like this: Every string in $L(r)$ must contain 00 somewhere, but what comes before and what goes after is completely arbitrary. An arbitrary string on $\{0, 1\}$ can be denoted by $(0 + 1)^*$. Putting these observations together, we arrive at the solution

$$r = (0 + 1)^* 00 (0 + 1)^*.$$

Example 3.6

Find a regular expression for the language

$$L = \{w \in \{0, 1\}^* : w \text{ has no pair of consecutive zeros}\}.$$

Even though this looks similar to Example 3.5, the answer is harder to construct. One helpful observation is that whenever the symbol 0 occurs, it must be followed immediately by the symbol 1. Such a substring may be preceded and followed by an arbitrary number of 1's. This suggests that the answer involves the repetition of strings of the form $1 \ldots 101 \ldots 1$, that is, the language denoted by the regular expression $(1^*011^*)^*$. However, the answer is still incomplete, because the strings ending in 0 or consisting of all 1's are unaccounted for. After taking care of these special cases we arrive at the answer

$$r = (1^*011^*)^*(0 + \lambda) + 1^*(0 + \lambda).$$

Example 3.5

For $\Sigma = \{0, 1\}$, give a regular expression r such that

$$L(r) = \{w \in \Sigma^* : w \text{ has at least one pair of consecutive zeros}\}.$$

One can arrive at an answer by reasoning something like this: Every string in $L(r)$ must contain 00 somewhere, but what comes before and what goes after is completely arbitrary. An arbitrary string on $\{0, 1\}$ can be denoted by $(0 + 1)^*$. Putting these observations together, we arrive at the solution

$$r = (0 + 1)^* 00 (0 + 1)^*.$$

Example 3.6

Find a regular expression for the language

$$L = \{w \in \{0, 1\}^* : w \text{ has no pair of consecutive zeros}\}.$$

Even though this looks similar to Example 3.5, the answer is harder to construct. One helpful observation is that whenever the symbol 0 occurs, it must be followed immediately by the symbol 1. Such a substring may be preceded and followed by an arbitrary number of 1's. This suggests that the answer involves the repetition of strings of the form $1 \ldots 101 \ldots 1$, that is, the language denoted by the regular expression $(1^*011^*)^*$. However, the answer is still incomplete, because the strings ending in 0 or consisting of all 1's are unaccounted for. After taking care of these special cases we arrive at the answer

$$r = (1^*011^*)^*(0 + \lambda) + 1^*(0 + \lambda).$$

Example 3.5

For $\Sigma = \{0, 1\}$, give a regular expression r such that

$$L(r) = \{w \in \Sigma^* : w \text{ has at least one pair of consecutive zeros}\}.$$

One can arrive at an answer by reasoning something like this: Every string in $L(r)$ must contain 00 somewhere, but what comes before and what goes after is completely arbitrary. An arbitrary string on $\{0, 1\}$ can be denoted by $(0 + 1)^*$. Putting these observations together, we arrive at the solution

$$r = (0 + 1)^* 00 (0 + 1)^*.$$

Example 3.6

Find a regular expression for the language

$$L = \{w \in \{0, 1\}^* : w \text{ has no pair of consecutive zeros}\}.$$

Even though this looks similar to Example 3.5, the answer is harder to construct. One helpful observation is that whenever the symbol 0 occurs, it must be followed immediately by the symbol 1. Such a substring may be preceded and followed by an arbitrary number of 1's. This suggests that the answer involves the repetition of strings of the form $1 \ldots 101 \ldots 1$, that is, the language denoted by the regular expression $(1^* 011^*)^*$. However, the answer is still incomplete, because the strings ending in 0 or consisting of all 1's are unaccounted for. After taking care of these special cases we arrive at the answer

$$r = (1^* 011^*)^* (0 + \lambda) + 1^* (0 + \lambda).$$

Example 3.5

For $\Sigma = \{0, 1\}$, give a regular expression r such that

$$L(r) = \{w \in \Sigma^* : w \text{ has at least one pair of consecutive zeros}\}.$$

One can arrive at an answer by reasoning something like this: Every string in $L(r)$ must contain 00 somewhere, but what comes before and what goes after is completely arbitrary. An arbitrary string on $\{0, 1\}$ can be denoted by $(0 + 1)^*$. Putting these observations together, we arrive at the solution

$$r = (0 + 1)^* 00 (0 + 1)^*.$$

Example 3.6

Find a regular expression for the language

$$L = \{w \in \{0, 1\}^* : w \text{ has no pair of consecutive zeros}\}.$$

Even though this looks similar to Example 3.5, the answer is harder to construct. One helpful observation is that whenever the symbol 0 occurs, it must be followed immediately by the symbol 1. Such a substring may be preceded and followed by an arbitrary number of 1's. This suggests that the answer involves the repetition of strings of the form $1 \ldots 101 \ldots 1$, that is, the language denoted by the regular expression $(1^* 011^*)^*$. However, the answer is still incomplete, because the strings ending in 0 or consisting of all 1's are unaccounted for. After taking care of these special cases we arrive at the answer

$$r = (1^* 011^*)^* (0 + \lambda) + 1^* (0 + \lambda).$$

Example 3.5

For $\Sigma = \{0, 1\}$, give a regular expression r such that

$$L(r) = \{w \in \Sigma^* : w \text{ has at least one pair of consecutive zeros}\}.$$

One can arrive at an answer by reasoning something like this: Every string in $L(r)$ must contain 00 somewhere, but what comes before and what goes after is completely arbitrary. An arbitrary string on $\{0, 1\}$ can be denoted by $(0 + 1)^*$. Putting these observations together, we arrive at the solution

$$r = (0 + 1)^* 00 (0 + 1)^*.$$

Example 3.6

Find a regular expression for the language

$$L = \{w \in \{0, 1\}^* : w \text{ has no pair of consecutive zeros}\}.$$

Even though this looks similar to Example 3.5, the answer is harder to construct. One helpful observation is that whenever the symbol 0 occurs, it must be followed immediately by the symbol 1. Such a substring may be preceded and followed by an arbitrary number of 1's. This suggests that the answer involves the repetition of strings of the form $1 \ldots 101 \ldots 1$, that is, the language denoted by the regular expression $(1^* 011^*)^*$. However, the answer is still incomplete, because the strings ending in 0 or consisting of all 1's are unaccounted for. After taking care of these special cases we arrive at the answer

$$r = (1^* 011^*)^* (0 + \lambda) + 1^* (0 + \lambda).$$

Example 3.5

For $\Sigma = \{0, 1\}$, give a regular expression r such that

$$L(r) = \{w \in \Sigma^* \colon w \text{ has at least one pair of consecutive zeros}\}.$$

One can arrive at an answer by reasoning something like this: Every string in $L(r)$ must contain 00 somewhere, but what comes before and what goes after is completely arbitrary. An arbitrary string on $\{0, 1\}$ can be denoted by $(0 + 1)^*$. Putting these observations together, we arrive at the solution

$$r = (0 + 1)^* 00 (0 + 1)^*.$$

Example 3.6

Find a regular expression for the language

$$L = \{w \in \{0, 1\}^* \colon w \text{ has no pair of consecutive zeros}\}.$$

Even though this looks similar to Example 3.5, the answer is harder to construct. One helpful observation is that whenever the symbol 0 occurs, it must be followed immediately by the symbol 1. Such a substring may be preceded and followed by an arbitrary number of 1's. This suggests that the answer involves the repetition of strings of the form $1 \ldots 101 \ldots 1$, that is, the language denoted by the regular expression $(1^* 011^*)^*$. However, the answer is still incomplete, because the strings ending in 0 or consisting of all 1's are unaccounted for. After taking care of these special cases we arrive at the answer

$$r = (1^* 011^*)^* (0 + \lambda) + 1^* (0 + \lambda).$$

Example 3.6 (continuation)

If we reason slightly differently, we might come up with another answer. If we see L as the repetition of the strings 1 and 01, the shorter expression

$$r = (1 + 01)^*(0 + \lambda)$$

might be reached. Although the two expressions look different, both answers are correct, as they denote the same language. Generally, there are an unlimited number of regular expressions for any given language.

Note that this language is the complement of the language in Example 3.5. However, the regular expressions are not very similar and do not suggest clearly the close relationship between the languages.

The last example introduces the notion of equivalence of regular expressions. We say the two regular expressions are equivalent if they denote the same language. One can derive a variety of rules for simplifying regular expressions, but since we have little need for such manipulations we shall not pursue this.

Example 3.6 (continuation)

If we reason slightly differently, we might come up with another answer. If we see L as the repetition of the strings 1 and 01, the shorter expression

$$r = (1 + 01)^*(0 + \lambda)$$

might be reached. Although the two expressions look different, both answers are correct, as they denote the same language. Generally, there are an unlimited number of regular expressions for any given language.

Note that this language is the complement of the language in Example 3.5. However, the regular expressions are not very similar and do not suggest clearly the close relationship between the languages.

The last example introduces the notion of equivalence of regular expressions. We say the two regular expressions are equivalent if they denote the same language. One can derive a variety of rules for simplifying regular expressions, but since we have little need for such manipulations we shall not pursue this.

Example 3.6 (continuation)

If we reason slightly differently, we might come up with another answer. If we see L as the repetition of the strings 1 and 01, the shorter expression

$$r = (1 + 01)^*(0 + \lambda)$$

might be reached. Although the two expressions look different, both answers are correct, as they denote the same language. Generally, there are an unlimited number of regular expressions for any given language.

Note that this language is the complement of the language in Example 3.5. However, the regular expressions are not very similar and do not suggest clearly the close relationship between the languages.

The last example introduces the notion of equivalence of regular expressions. We say the two regular expressions are equivalent if they denote the same language. One can derive a variety of rules for simplifying regular expressions, but since we have little need for such manipulations we shall not pursue this.

Example 3.6 (continuation)

If we reason slightly differently, we might come up with another answer. If we see L as the repetition of the strings 1 and 01, the shorter expression

$$r = (1 + 01)^*(0 + \lambda)$$

might be reached. Although the two expressions look different, both answers are correct, as they denote the same language. Generally, there are an unlimited number of regular expressions for any given language.

Note that this language is the complement of the language in Example 3.5. However, the regular expressions are not very similar and do not suggest clearly the close relationship between the languages.

The last example introduces the notion of equivalence of regular expressions. We say the two regular expressions are equivalent if they denote the same language. One can derive a variety of rules for simplifying regular expressions, but since we have little need for such manipulations we shall not pursue this.

Example 3.6 (continuation)

If we reason slightly differently, we might come up with another answer. If we see L as the repetition of the strings 1 and 01, the shorter expression

$$r = (1 + 01)^*(0 + \lambda)$$

might be reached. Although the two expressions look different, both answers are correct, as they denote the same language. Generally, there are an unlimited number of regular expressions for any given language.

Note that this language is the complement of the language in Example 3.5. However, the regular expressions are not very similar and do not suggest clearly the close relationship between the languages.

The last example introduces the notion of equivalence of regular expressions. We say the two regular expressions are equivalent if they denote the same language. One can derive a variety of rules for simplifying regular expressions, but since we have little need for such manipulations we shall not pursue this.

Example 3.6 (continuation)

If we reason slightly differently, we might come up with another answer. If we see L as the repetition of the strings 1 and 01, the shorter expression

$$r = (1 + 01)^*(0 + \lambda)$$

might be reached. Although the two expressions look different, both answers are correct, as they denote the same language. Generally, there are an unlimited number of regular expressions for any given language.

Note that this language is the complement of the language in Example 3.5. However, the regular expressions are not very similar and do not suggest clearly the close relationship between the languages.

The last example introduces the notion of equivalence of regular expressions. We say the two regular expressions are equivalent if they denote the same language. One can derive a variety of rules for simplifying regular expressions, but since we have little need for such manipulations we shall not pursue this.

Example 3.6 (continuation)

If we reason slightly differently, we might come up with another answer. If we see L as the repetition of the strings 1 and 01, the shorter expression

$$r = (1 + 01)^*(0 + \lambda)$$

might be reached. Although the two expressions look different, both answers are correct, as they denote the same language. Generally, there are an unlimited number of regular expressions for any given language.

Note that this language is the complement of the language in Example 3.5. However, the regular expressions are not very similar and do not suggest clearly the close relationship between the languages.

The last example introduces the notion of equivalence of regular expressions. We say the two regular expressions are equivalent if they denote the same language. One can derive a variety of rules for simplifying regular expressions, but since we have little need for such manipulations we shall not pursue this.

Example 3.6 (continuation)

If we reason slightly differently, we might come up with another answer. If we see L as the repetition of the strings 1 and 01, the shorter expression

$$r = (1 + 01)^*(0 + \lambda)$$

might be reached. Although the two expressions look different, both answers are correct, as they denote the same language. Generally, there are an unlimited number of regular expressions for any given language.

Note that this language is the complement of the language in Example 3.5. However, the regular expressions are not very similar and do not suggest clearly the close relationship between the languages.

The last example introduces the notion of equivalence of regular expressions. We say the two regular expressions are equivalent if they denote the same language. One can derive a variety of rules for simplifying regular expressions, but since we have little need for such manipulations we shall not pursue this.

Example 3.6 (continuation)

If we reason slightly differently, we might come up with another answer. If we see L as the repetition of the strings 1 and 01, the shorter expression

$$r = (1 + 01)^*(0 + \lambda)$$

might be reached. Although the two expressions look different, both answers are correct, as they denote the same language. Generally, there are an unlimited number of regular expressions for any given language.

Note that this language is the complement of the language in Example 3.5. However, the regular expressions are not very similar and do not suggest clearly the close relationship between the languages.

The last example introduces the notion of equivalence of regular expressions. We say the two regular expressions are equivalent if they denote the same language. One can derive a variety of rules for simplifying regular expressions, but since we have little need for such manipulations we shall not pursue this.

Example 3.6 (continuation)

If we reason slightly differently, we might come up with another answer. If we see L as the repetition of the strings 1 and 01, the shorter expression

$$r = (1 + 01)^*(0 + \lambda)$$

might be reached. Although the two expressions look different, both answers are correct, as they denote the same language. Generally, there are an unlimited number of regular expressions for any given language.

Note that this language is the complement of the language in Example 3.5. However, the regular expressions are not very similar and do not suggest clearly the close relationship between the languages.

The last example introduces the notion of equivalence of regular expressions. We say the two regular expressions are equivalent if they denote the same language. One can derive a variety of rules for simplifying regular expressions, but since we have little need for such manipulations we shall not pursue this.

Example 3.6 (continuation)

If we reason slightly differently, we might come up with another answer. If we see L as the repetition of the strings 1 and 01, the shorter expression

$$r = (1 + 01)^*(0 + \lambda)$$

might be reached. Although the two expressions look different, both answers are correct, as they denote the same language. Generally, there are an unlimited number of regular expressions for any given language.

Note that this language is the complement of the language in Example 3.5. However, the regular expressions are not very similar and do not suggest clearly the close relationship between the languages.

The last example introduces the notion of equivalence of regular expressions. We say the two regular expressions are equivalent if they denote the same language. One can derive a variety of rules for simplifying regular expressions, but since we have little need for such manipulations we shall not pursue this.

Example 3.6 (continuation)

If we reason slightly differently, we might come up with another answer. If we see L as the repetition of the strings 1 and 01, the shorter expression

$$r = (1 + 01)^*(0 + \lambda)$$

might be reached. Although the two expressions look different, both answers are correct, as they denote the same language. Generally, there are an unlimited number of regular expressions for any given language.

Note that this language is the complement of the language in Example 3.5. However, the regular expressions are not very similar and do not suggest clearly the close relationship between the languages.

The last example introduces the notion of equivalence of regular expressions. We say the two regular expressions are equivalent if they denote the same language. One can derive a variety of rules for simplifying regular expressions, but since we have little need for such manipulations we shall not pursue this.

Example 3.6 (continuation)

If we reason slightly differently, we might come up with another answer. If we see L as the repetition of the strings 1 and 01, the shorter expression

$$r = (1 + 01)^*(0 + \lambda)$$

might be reached. Although the two expressions look different, both answers are correct, as they denote the same language. Generally, there are an unlimited number of regular expressions for any given language.

Note that this language is the complement of the language in Example 3.5. However, the regular expressions are not very similar and do not suggest clearly the close relationship between the languages.

The last example introduces the notion of equivalence of regular expressions. We say the two regular expressions are equivalent if they denote the same language. One can derive a variety of rules for simplifying regular expressions, but since we have little need for such manipulations we shall not pursue this.

Example 3.6 (continuation)

If we reason slightly differently, we might come up with another answer. If we see L as the repetition of the strings 1 and 01, the shorter expression

$$r = (1 + 01)^*(0 + \lambda)$$

might be reached. Although the two expressions look different, both answers are correct, as they denote the same language. Generally, there are an unlimited number of regular expressions for any given language.

Note that this language is the complement of the language in Example 3.5. However, the regular expressions are not very similar and do not suggest clearly the close relationship between the languages.

The last example introduces the notion of equivalence of regular expressions. We say the two regular expressions are equivalent if they denote the same language. One can derive a variety of rules for simplifying regular expressions, but since we have little need for such manipulations we shall not pursue this.

Example 3.6 (continuation)

If we reason slightly differently, we might come up with another answer. If we see L as the repetition of the strings 1 and 01, the shorter expression

$$r = (1 + 01)^*(0 + \lambda)$$

might be reached. Although the two expressions look different, both answers are correct, as they denote the same language. Generally, there are an unlimited number of regular expressions for any given language.

Note that this language is the complement of the language in Example 3.5. However, the regular expressions are not very similar and do not suggest clearly the close relationship between the languages.

The last example introduces the notion of equivalence of regular expressions. We say the two regular expressions are equivalent if they denote the same language. One can derive a variety of rules for simplifying regular expressions, but since we have little need for such manipulations we shall not pursue this.

Example 3.6 (continuation)

If we reason slightly differently, we might come up with another answer. If we see L as the repetition of the strings 1 and 01, the shorter expression

$$r = (1 + 01)^*(0 + \lambda)$$

might be reached. Although the two expressions look different, both answers are correct, as they denote the same language. Generally, there are an unlimited number of regular expressions for any given language.

Note that this language is the complement of the language in Example 3.5. However, the regular expressions are not very similar and do not suggest clearly the close relationship between the languages.

The last example introduces the notion of equivalence of regular expressions. We say the two regular expressions are equivalent if they denote the same language. One can derive a variety of rules for simplifying regular expressions, but since we have little need for such manipulations we shall not pursue this.

Example 3.6 (continuation)

If we reason slightly differently, we might come up with another answer. If we see L as the repetition of the strings 1 and 01, the shorter expression

$$r = (1 + 01)^*(0 + \lambda)$$

might be reached. Although the two expressions look different, both answers are correct, as they denote the same language. Generally, there are an unlimited number of regular expressions for any given language.

Note that this language is the complement of the language in Example 3.5. However, the regular expressions are not very similar and do not suggest clearly the close relationship between the languages.

The last example introduces the notion of equivalence of regular expressions. We say the two regular expressions are equivalent if they denote the same language. One can derive a variety of rules for simplifying regular expressions, but since we have little need for such manipulations we shall not pursue this.

Thank You for attention!