Formal Languages, Automata and Codes

Oleg Gutik

Lecture 6

We now come to a fundamental question. In what sense are DFA's and NFA's different? Obviously, there is a difference in their definition, but this does not imply that there is any essential distinction between them. To explore this question, we introduce the concept of equivalence between automata.

Definition 2.7

Two finite accepters, M_1 and M_2, are said to be equivalent if

$$L(M_1) = L(M_2),$$

that is, if they both accept the same language.

As mentioned, there are generally many accepters for a given language, so any DFA or NFA has many equivalent accepters.

We now come to a fundamental question. In what sense are DFA's and NFA's different? Obviously, there is a difference in their definition, but this does not imply that there is any essential distinction between them. To explore this question, we introduce the concept of equivalence between automata.

Definition 2.7

Two finite accepters, M_1 and M_2, are said to be equivalent if

$$L(M_1) = L(M_2),$$

that is, if they both accept the same language.

As mentioned, there are generally many accepters for a given language, so any DFA or NFA has many equivalent accepters.

We now come to a fundamental question. In what sense are DFA's and NFA's different? Obviously, there is a difference in their definition, but this does not imply that there is any essential distinction between them. To explore this question, we introduce the concept of equivalence between automata.

Definition 2.7

Two finite accepters, M_1 and M_2, are said to be equivalent if

$$L(M_1) = L(M_2),$$

that is, if they both accept the same language.

As mentioned, there are generally many accepters for a given language, so any DFA or NFA has many equivalent accepters.

We now come to a fundamental question. In what sense are DFA's and NFA's different? Obviously, there is a difference in their definition, but this does not imply that there is any essential distinction between them. To explore this question, we introduce the concept of equivalence between automata.

Definition 2.7

Two finite accepters, M_1 and M_2, are said to be equivalent if

$$L(M_1) = L(M_2),$$

that is, if they both accept the same language.

As mentioned, there are generally many accepters for a given language, so any DFA or NFA has many equivalent accepters.

We now come to a fundamental question. In what sense are DFA's and NFA's different? Obviously, there is a difference in their definition, but this does not imply that there is any essential distinction between them. To explore this question, we introduce the concept of equivalence between automata.

Definition 2.7

Two finite accepters, M_1 and M_2, are said to be equivalent if

$$L(M_1) = L(M_2),$$

that is, if they both accept the same language.

As mentioned, there are generally many accepters for a given language, so any DFA or NFA has many equivalent accepters.

We now come to a fundamental question. In what sense are DFA's and NFA's different? Obviously, there is a difference in their definition, but this does not imply that there is any essential distinction between them. To explore this question, we introduce the concept of equivalence between automata.

Definition 2.7

Two finite accepters, M_1 and M_2, are said to be equivalent if

$$L(M_1) = L(M_2),$$

that is, if they both accept the same language.

As mentioned, there are generally many accepters for a given language, so any DFA or NFA has many equivalent accepters.

We now come to a fundamental question. In what sense are DFA's and NFA's different? Obviously, there is a difference in their definition, but this does not imply that there is any essential distinction between them. To explore this question, we introduce the concept of equivalence between automata.

Definition 2.7

Two finite accepters, M_1 and M_2, are said to be equivalent if

$$L(M_1) = L(M_2),$$

that is, if they both accept the same language.

As mentioned, there are generally many accepters for a given language, so any DFA or NFA has many equivalent accepters.

We now come to a fundamental question. In what sense are DFA's and NFA's different? Obviously, there is a difference in their definition, but this does not imply that there is any essential distinction between them. To explore this question, we introduce the concept of equivalence between automata.

Definition 2.7

Two finite accepters, M_1 and M_2, are said to be equivalent if

$$L(M_1) = L(M_2),$$

that is, if they both accept the same language.

As mentioned, there are generally many accepters for a given language, so any DFA or NFA has many equivalent accepters.

We now come to a fundamental question. In what sense are DFA's and NFA's different? Obviously, there is a difference in their definition, but this does not imply that there is any essential distinction between them. To explore this question, we introduce the concept of equivalence between automata.

Definition 2.7

Two finite accepters, M_1 and M_2, are said to be equivalent if

$$L(M_1) = L(M_2),$$

that is, if they both accept the same language.

As mentioned, there are generally many accepters for a given language, so any DFA or NFA has many equivalent accepters.

2.3 Equivalence of deterministic and nondeterministic finite accepters

We now come to a fundamental question. In what sense are DFA's and NFA's different? Obviously, there is a difference in their definition, but this does not imply that there is any essential distinction between them. To explore this question, we introduce the concept of equivalence between automata.

Definition 2.7

Two finite accepters, M_1 and M_2, are said to be equivalent if

$$L(M_1) = L(M_2),$$

that is, if they both accept the same language.

As mentioned, there are generally many accepters for a given language, so any DFA or NFA has many equivalent accepters.

We now come to a fundamental question. In what sense are DFA's and NFA's different? Obviously, there is a difference in their definition, but this does not imply that there is any essential distinction between them. To explore this question, we introduce the concept of equivalence between automata.

Definition 2.7

Two finite accepters, M_1 and M_2, are said to be equivalent if

$$L(M_1) = L(M_2),$$

that is, if they both accept the same language.

As mentioned, there are generally many accepters for a given language, so any DFA or NFA has many equivalent accepters.

We now come to a fundamental question. In what sense are DFA's and NFA's different? Obviously, there is a difference in their definition, but this does not imply that there is any essential distinction between them. To explore this question, we introduce the concept of equivalence between automata.

Definition 2.7

Two finite accepters, M_1 and M_2, are said to be equivalent if

$$L(M_1) = L(M_2),$$

that is, if they both accept the same language.

As mentioned, there are generally many accepters for a given language, so any DFA or NFA has many equivalent accepters.

Example 2.11

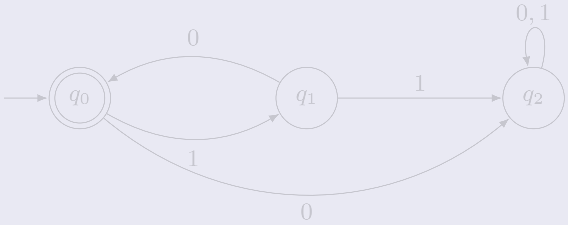
The DFA shown in the following Figure

is equivalent to the NFA in the Figure

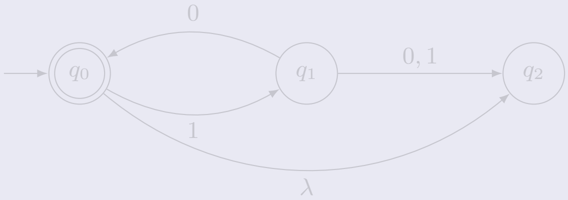
because they both accept the language $\{(10)^n : n \geqslant 0\}$.

Example 2.11

The DFA shown in the following Figure



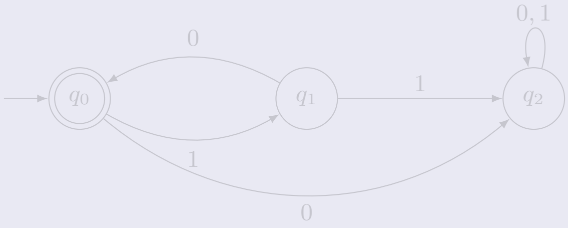
is equivalent to the NFA in the Figure



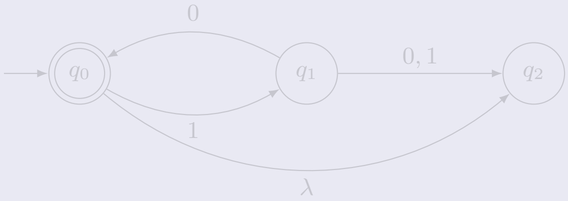
because they both accept the language $\{(10)^n : n \geqslant 0\}$.

Example 2.11

The DFA shown in the following Figure



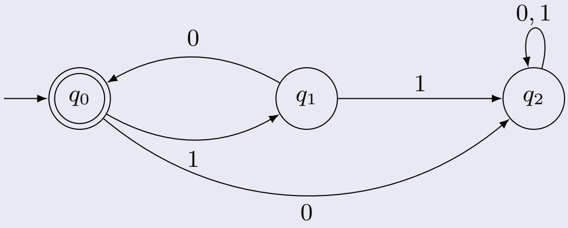
is equivalent to the NFA in the Figure



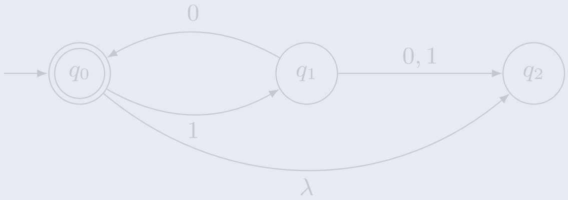
because they both accept the language $\{(10)^n : n \geqslant 0\}$.

Example 2.11

The DFA shown in the following Figure



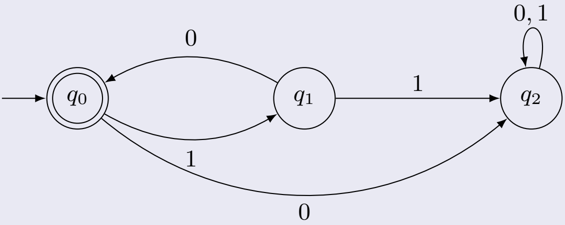
is equivalent to the NFA in the Figure



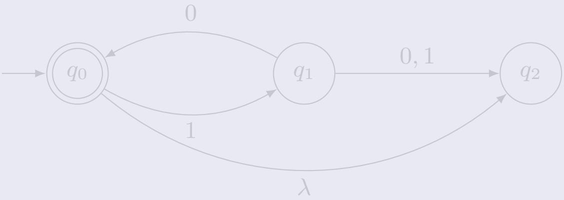
because they both accept the language $\{(10)^n : n \geqslant 0\}$.

Example 2.11

The DFA shown in the following Figure



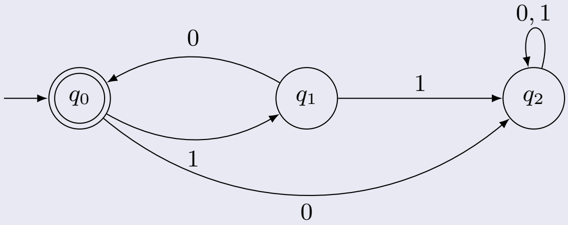
is equivalent to the NFA in the Figure



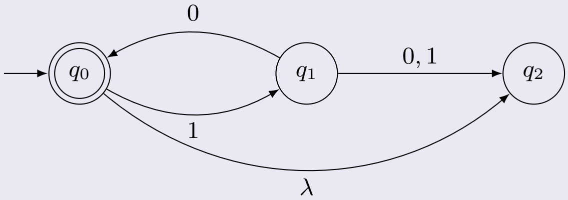
because they both accept the language $\{(10)^n : n \geqslant 0\}$.

Example 2.11

The DFA shown in the following Figure



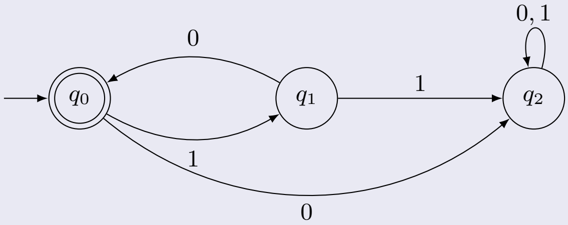
is equivalent to the NFA in the Figure



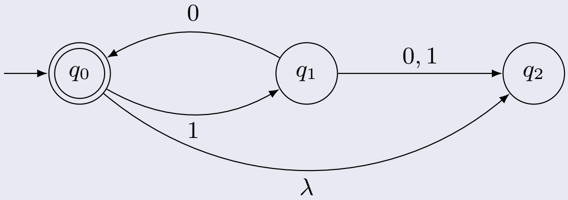
because they both accept the language $\{(10)^n : n \geqslant 0\}$.

Example 2.11

The DFA shown in the following Figure



is equivalent to the NFA in the Figure



because they both accept the language $\{(10)^n : n \geqslant 0\}$.

When we compare different classes of automata, the question invariably arises whether one class is more powerful than the other. By "more powerful" we mean that an automaton of one kind can achieve something that cannot be done by any automaton of the other kind. Let us look at this question for finite accepters. Since a DFA is in essence a restricted kind of NFA, it is clear that any language that is accepted by a DFA is also accepted by some NFA. But the converse is not so obvious. We have added nondeterminism, so it is at least conceivable that there is a language accepted by some NFA for which, in principle, we cannot find a DFA. But it turns out that this is not so. The classes of DFA's and NFA's are equally powerful: For every language accepted by some NFA there is a DFA that accepts the same language.

When we compare different classes of automata, the question invariably arises whether one class is more powerful than the other. By "more powerful" we mean that an automaton of one kind can achieve something that cannot be done by any automaton of the other kind. Let us look at this question for finite accepters. Since a DFA is in essence a restricted kind of NFA, it is clear that any language that is accepted by a DFA is also accepted by some NFA. But the converse is not so obvious. We have added nondeterminism, so it is at least conceivable that there is a language accepted by some NFA for which, in principle, we cannot find a DFA. But it turns out that this is not so. The classes of DFA's and NFA's are equally powerful: For every language accepted by some NFA there is a DFA that accepts the same language.

When we compare different classes of automata, the question invariably arises whether one class is more powerful than the other. By "more powerful" we mean that an automaton of one kind can achieve something that cannot be done by any automaton of the other kind. Let us look at this question for finite accepters. Since a DFA is in essence a restricted kind of NFA, it is clear that any language that is accepted by a DFA is also accepted by some NFA. But the converse is not so obvious. We have added nondeterminism, so it is at least conceivable that there is a language accepted by some NFA for which, in principle, we cannot find a DFA. But it turns out that this is not so. The classes of DFA's and NFA's are equally powerful: For every language accepted by some NFA there is a DFA that accepts the same language.

When we compare different classes of automata, the question invariably arises whether one class is more powerful than the other. By "more powerful" we mean that an automaton of one kind can achieve something that cannot be done by any automaton of the other kind. Let us look at this question for finite accepters. Since a DFA is in essence a restricted kind of NFA, it is clear that any language that is accepted by a DFA is also accepted by some NFA. But the converse is not so obvious. We have added nondeterminism, so it is at least conceivable that there is a language accepted by some NFA for which, in principle, we cannot find a DFA. But it turns out that this is not so. The classes of DFA's and NFA's are equally powerful: For every language accepted by some NFA there is a DFA that accepts the same language.

When we compare different classes of automata, the question invariably arises whether one class is more powerful than the other. By "more powerful" we mean that an automaton of one kind can achieve something that cannot be done by any automaton of the other kind. Let us look at this question for finite accepters. Since a DFA is in essence a restricted kind of NFA, it is clear that any language that is accepted by a DFA is also accepted by some NFA. But the converse is not so obvious. We have added nondeterminism, so it is at least conceivable that there is a language accepted by some NFA for which, in principle, we cannot find a DFA. But it turns out that this is not so. The classes of DFA's and NFA's are equally powerful: For every language accepted by some NFA there is a DFA that accepts the same language.

When we compare different classes of automata, the question invariably arises whether one class is more powerful than the other. By “more powerful” we mean that an automaton of one kind can achieve something that cannot be done by any automaton of the other kind. Let us look at this question for finite accepters. Since a DFA is in essence a restricted kind of NFA, it is clear that any language that is accepted by a DFA is also accepted by some NFA. But the converse is not so obvious. We have added nondeterminism, so it is at least conceivable that there is a language accepted by some NFA for which, in principle, we cannot find a DFA. But it turns out that this is not so. The classes of DFA's and NFA's are equally powerful: For every language accepted by some NFA there is a DFA that accepts the same language.

2.3 Equivalence of deterministic and nondeterministic finite accepters

When we compare different classes of automata, the question invariably arises whether one class is more powerful than the other. By "more powerful" we mean that an automaton of one kind can achieve something that cannot be done by any automaton of the other kind. Let us look at this question for finite accepters. Since a DFA is in essence a restricted kind of NFA, it is clear that any language that is accepted by a DFA is also accepted by some NFA. But the converse is not so obvious. We have added nondeterminism, so it is at least conceivable that there is a language accepted by some NFA for which, in principle, we cannot find a DFA. But it turns out that this is not so. The classes of DFA's and NFA's are equally powerful: For every language accepted by some NFA there is a DFA that accepts the same language.

When we compare different classes of automata, the question invariably arises whether one class is more powerful than the other. By “more powerful” we mean that an automaton of one kind can achieve something that cannot be done by any automaton of the other kind. Let us look at this question for finite accepters. Since a DFA is in essence a restricted kind of NFA, it is clear that any language that is accepted by a DFA is also accepted by some NFA. But the converse is not so obvious. We have added nondeterminism, so it is at least conceivable that there is a language accepted by some NFA for which, in principle, we cannot find a DFA. But it turns out that this is not so. The classes of DFA's and NFA's are equally powerful: For every language accepted by some NFA there is a DFA that accepts the same language.

When we compare different classes of automata, the question invariably arises whether one class is more powerful than the other. By "more powerful" we mean that an automaton of one kind can achieve something that cannot be done by any automaton of the other kind. Let us look at this question for finite accepters. Since a DFA is in essence a restricted kind of NFA, it is clear that any language that is accepted by a DFA is also accepted by some NFA. But the converse is not so obvious. We have added nondeterminism, so it is at least conceivable that there is a language accepted by some NFA for which, in principle, we cannot find a DFA. But it turns out that this is not so. The classes of DFA's and NFA's are equally powerful: For every language accepted by some NFA there is a DFA that accepts the same language.

When we compare different classes of automata, the question invariably arises whether one class is more powerful than the other. By "more powerful" we mean that an automaton of one kind can achieve something that cannot be done by any automaton of the other kind. Let us look at this question for finite accepters. Since a DFA is in essence a restricted kind of NFA, it is clear that any language that is accepted by a DFA is also accepted by some NFA. But the converse is not so obvious. We have added nondeterminism, so it is at least conceivable that there is a language accepted by some NFA for which, in principle, we cannot find a DFA. But it turns out that this is not so. The classes of DFA's and NFA's are equally powerful: For every language accepted by some NFA there is a DFA that accepts the same language.

When we compare different classes of automata, the question invariably arises whether one class is more powerful than the other. By "more powerful" we mean that an automaton of one kind can achieve something that cannot be done by any automaton of the other kind. Let us look at this question for finite accepters. Since a DFA is in essence a restricted kind of NFA, it is clear that any language that is accepted by a DFA is also accepted by some NFA. But the converse is not so obvious. We have added nondeterminism, so it is at least conceivable that there is a language accepted by some NFA for which, in principle, we cannot find a DFA. But it turns out that this is not so. The classes of DFA's and NFA's are equally powerful: For every language accepted by some NFA there is a DFA that accepts the same language.

When we compare different classes of automata, the question invariably arises whether one class is more powerful than the other. By "more powerful" we mean that an automaton of one kind can achieve something that cannot be done by any automaton of the other kind. Let us look at this question for finite accepters. Since a DFA is in essence a restricted kind of NFA, it is clear that any language that is accepted by a DFA is also accepted by some NFA. But the converse is not so obvious. We have added nondeterminism, so it is at least conceivable that there is a language accepted by some NFA for which, in principle, we cannot find a DFA. But it turns out that this is not so. The classes of DFA's and NFA's are equally powerful: For every language accepted by some NFA there is a DFA that accepts the same language.

2.3 Equivalence of deterministic and nondeterministic finite accepters

When we compare different classes of automata, the question invariably arises whether one class is more powerful than the other. By "more powerful" we mean that an automaton of one kind can achieve something that cannot be done by any automaton of the other kind. Let us look at this question for finite accepters. Since a DFA is in essence a restricted kind of NFA, it is clear that any language that is accepted by a DFA is also accepted by some NFA. But the converse is not so obvious. We have added nondeterminism, so it is at least conceivable that there is a language accepted by some NFA for which, in principle, we cannot find a DFA. But it turns out that this is not so. The classes of DFA's and NFA's are equally powerful: For every language accepted by some NFA there is a DFA that accepts the same language.

This result is not obvious and certainly has to be demonstrated. The argument, like most arguments in this book, will be constructive. This means that we can actually give a way of converting any NFA into an equivalent DFA. The construction is not hard to understand; once the idea is clear it becomes the starting point for a rigorous argument. The rationale for the construction is the following. After an NFA has read a string w, we may not know exactly what state it will be in, but we can say that it must be in one state of a set of possible states, say $\{q_i, q_j, \ldots, q_k\}$. An equivalent DFA after reading the same string must be in some definite state. How can we make these two situations correspond? The answer is a nice trick: Label the states of the DFA with a set of states in such a way that, after reading w, the equivalent DFA will be in a single state labeled $\{q_i, q_j, \ldots, q_k\}$. Since for a set of $|Q|$ states there are exactly $2^{|Q|}$ subsets, the corresponding DFA will have a finite number of states.

Most of the work in this suggested construction lies in the analysis of the NFA to get the correspondence between possible states and inputs. Before getting to the formal description of this, let us illustrate it with a simple example.

This result is not obvious and certainly has to be demonstrated. The argument, like most arguments in this book, will be constructive. This means that we can actually give a way of converting any NFA into an equivalent DFA. The construction is not hard to understand; once the idea is clear it becomes the starting point for a rigorous argument. The rationale for the construction is the following. After an NFA has read a string w, we may not know exactly what state it will be in, but we can say that it must be in one state of a set of possible states, say $\{q_i, q_j, \ldots, q_k\}$. An equivalent DFA after reading the same string must be in some definite state. How can we make these two situations correspond? The answer is a nice trick: Label the states of the DFA with a set of states in such a way that, after reading w, the equivalent DFA will be in a single state labeled $\{q_i, q_j, \ldots, q_k\}$. Since for a set of $|Q|$ states there are exactly $2^{|Q|}$ subsets, the corresponding DFA will have a finite number of states.

Most of the work in this suggested construction lies in the analysis of the NFA to get the correspondence between possible states and inputs. Before getting to the formal description of this, let us illustrate it with a simple example.

This result is not obvious and certainly has to be demonstrated. The argument, like most arguments in this book, will be constructive. This means that we can actually give a way of converting any NFA into an equivalent DFA. The construction is not hard to understand; once the idea is clear it becomes the starting point for a rigorous argument. The rationale for the construction is the following. After an NFA has read a string w, we may not know exactly what state it will be in, but we can say that it must be in one state of a set of possible states, say $\{q_i, q_j, \ldots, q_k\}$. An equivalent DFA after reading the same string must be in some definite state. How can we make these two situations correspond? The answer is a nice trick: Label the states of the DFA with a set of states in such a way that, after reading w, the equivalent DFA will be in a single state labeled $\{q_i, q_j, \ldots, q_k\}$. Since for a set of $|Q|$ states there are exactly $2^{|Q|}$ subsets, the corresponding DFA will have a finite number of states.

Most of the work in this suggested construction lies in the analysis of the NFA to get the correspondence between possible states and inputs. Before getting to the formal description of this, let us illustrate it with a simple example.

This result is not obvious and certainly has to be demonstrated. The argument, like most arguments in this book, will be constructive. This means that we can actually give a way of converting any NFA into an equivalent DFA. The construction is not hard to understand; once the idea is clear it becomes the starting point for a rigorous argument. The rationale for the construction is the following. After an NFA has read a string w, we may not know exactly what state it will be in, but we can say that it must be in one state of a set of possible states, say $\{q_i, q_j, \ldots, q_k\}$. An equivalent DFA after reading the same string must be in some definite state. How can we make these two situations correspond? The answer is a nice trick: Label the states of the DFA with a set of states in such a way that, after reading w, the equivalent DFA will be in a single state labeled $\{q_i, q_j, \ldots, q_k\}$. Since for a set of $|Q|$ states there are exactly $2^{|Q|}$ subsets, the corresponding DFA will have a finite number of states.

Most of the work in this suggested construction lies in the analysis of the NFA to get the correspondence between possible states and inputs. Before getting to the formal description of this, let us illustrate it with a simple example.

This result is not obvious and certainly has to be demonstrated. The argument, like most arguments in this book, will be constructive. This means that we can actually give a way of converting any NFA into an equivalent DFA. The construction is not hard to understand; once the idea is clear it becomes the starting point for a rigorous argument. The rationale for the construction is the following. After an NFA has read a string w, we may not know exactly what state it will be in, but we can say that it must be in one state of a set of possible states, say $\{q_i, q_j, \ldots, q_k\}$. An equivalent DFA after reading the same string must be in some definite state. How can we make these two situations correspond? The answer is a nice trick: Label the states of the DFA with a set of states in such a way that, after reading w, the equivalent DFA will be in a single state labeled $\{q_i, q_j, \ldots, q_k\}$. Since for a set of $|Q|$ states there are exactly $2^{|Q|}$ subsets, the corresponding DFA will have a finite number of states.

Most of the work in this suggested construction lies in the analysis of the NFA to get the correspondence between possible states and inputs. Before getting to the formal description of this, let us illustrate it with a simple example.

This result is not obvious and certainly has to be demonstrated. The argument, like most arguments in this book, will be constructive. This means that we can actually give a way of converting any NFA into an equivalent DFA. The construction is not hard to understand; once the idea is clear it becomes the starting point for a rigorous argument. The rationale for the construction is the following. After an NFA has read a string w, we may not know exactly what state it will be in, but we can say that it must be in one state of a set of possible states, say $\{q_i, q_j, \ldots, q_k\}$. An equivalent DFA after reading the same string must be in some definite state. How can we make these two situations correspond? The answer is a nice trick: Label the states of the DFA with a set of states in such a way that, after reading w, the equivalent DFA will be in a single state labeled $\{q_i, q_j, \ldots, q_k\}$. Since for a set of $|Q|$ states there are exactly $2^{|Q|}$ subsets, the corresponding DFA will have a finite number of states.

Most of the work in this suggested construction lies in the analysis of the NFA to get the correspondence between possible states and inputs. Before getting to the formal description of this, let us illustrate it with a simple example.

This result is not obvious and certainly has to be demonstrated. The argument, like most arguments in this book, will be constructive. This means that we can actually give a way of converting any NFA into an equivalent DFA. The construction is not hard to understand; once the idea is clear it becomes the starting point for a rigorous argument. The rationale for the construction is the following. After an NFA has read a string w, we may not know exactly what state it will be in, but we can say that it must be in one state of a set of possible states, say $\{q_i, q_j, \ldots, q_k\}$. An equivalent DFA after reading the same string must be in some definite state. How can we make these two situations correspond? The answer is a nice trick: Label the states of the DFA with a set of states in such a way that, after reading w, the equivalent DFA will be in a single state labeled $\{q_i, q_j, \ldots, q_k\}$. Since for a set of $|Q|$ states there are exactly $2^{|Q|}$ subsets, the corresponding DFA will have a finite number of states.

Most of the work in this suggested construction lies in the analysis of the NFA to get the correspondence between possible states and inputs. Before getting to the formal description of this, let us illustrate it with a simple example.

This result is not obvious and certainly has to be demonstrated. The argument, like most arguments in this book, will be constructive. This means that we can actually give a way of converting any NFA into an equivalent DFA. The construction is not hard to understand; once the idea is clear it becomes the starting point for a rigorous argument. The rationale for the construction is the following. After an NFA has read a string w, we may not know exactly what state it will be in, but we can say that it must be in one state of a set of possible states, say $\{q_i, q_j, \ldots, q_k\}$. An equivalent DFA after reading the same string must be in some definite state. How can we make these two situations correspond? The answer is a nice trick: Label the states of the DFA with a set of states in such a way that, after reading w, the equivalent DFA will be in a single state labeled $\{q_i, q_j, \ldots, q_k\}$. Since for a set of $|Q|$ states there are exactly $2^{|Q|}$ subsets, the corresponding DFA will have a finite number of states.

Most of the work in this suggested construction lies in the analysis of the NFA to get the correspondence between possible states and inputs. Before getting to the formal description of this, let us illustrate it with a simple example.

This result is not obvious and certainly has to be demonstrated. The argument, like most arguments in this book, will be constructive. This means that we can actually give a way of converting any NFA into an equivalent DFA. The construction is not hard to understand; once the idea is clear it becomes the starting point for a rigorous argument. The rationale for the construction is the following. After an NFA has read a string w, we may not know exactly what state it will be in, but we can say that it must be in one state of a set of possible states, say $\{q_i, q_j, \dots, q_k\}$. An equivalent DFA after reading the same string must be in some definite state. How can we make these two situations correspond? The answer is a nice trick: Label the states of the DFA with a set of states in such a way that, after reading w, the equivalent DFA will be in a single state labeled $\{q_i, q_j, \dots, q_k\}$. Since for a set of $|Q|$ states there are exactly $2^{|Q|}$ subsets, the corresponding DFA will have a finite number of states.

Most of the work in this suggested construction lies in the analysis of the NFA to get the correspondence between possible states and inputs. Before getting to the formal description of this, let us illustrate it with a simple example.

This result is not obvious and certainly has to be demonstrated. The argument, like most arguments in this book, will be constructive. This means that we can actually give a way of converting any NFA into an equivalent DFA. The construction is not hard to understand; once the idea is clear it becomes the starting point for a rigorous argument. The rationale for the construction is the following. After an NFA has read a string w, we may not know exactly what state it will be in, but we can say that it must be in one state of a set of possible states, say $\{q_i, q_j, \ldots, q_k\}$. An equivalent DFA after reading the same string must be in some definite state. How can we make these two situations correspond? The answer is a nice trick: Label the states of the DFA with a set of states in such a way that, after reading w, the equivalent DFA will be in a single state labeled $\{q_i, q_j, \ldots, q_k\}$. Since for a set of $|Q|$ states there are exactly $2^{|Q|}$ subsets, the corresponding DFA will have a finite number of states.

Most of the work in this suggested construction lies in the analysis of the NFA to get the correspondence between possible states and inputs. Before getting to the formal description of this, let us illustrate it with a simple example.

This result is not obvious and certainly has to be demonstrated. The argument, like most arguments in this book, will be constructive. This means that we can actually give a way of converting any NFA into an equivalent DFA. The construction is not hard to understand; once the idea is clear it becomes the starting point for a rigorous argument. The rationale for the construction is the following. After an NFA has read a string w, we may not know exactly what state it will be in, but we can say that it must be in one state of a set of possible states, say $\{q_i, q_j, \ldots, q_k\}$. An equivalent DFA after reading the same string must be in some definite state. How can we make these two situations correspond? The answer is a nice trick: Label the states of the DFA with a set of states in such a way that, after reading w, the equivalent DFA will be in a single state labeled $\{q_i, q_j, \ldots, q_k\}$. Since for a set of $|Q|$ states there are exactly $2^{|Q|}$ subsets, the corresponding DFA will have a finite number of states.

Most of the work in this suggested construction lies in the analysis of the NFA to get the correspondence between possible states and inputs. Before getting to the formal description of this, let us illustrate it with a simple example.

This result is not obvious and certainly has to be demonstrated. The argument, like most arguments in this book, will be constructive. This means that we can actually give a way of converting any NFA into an equivalent DFA. The construction is not hard to understand; once the idea is clear it becomes the starting point for a rigorous argument. The rationale for the construction is the following. After an NFA has read a string w, we may not know exactly what state it will be in, but we can say that it must be in one state of a set of possible states, say $\{q_i, q_j, \dots, q_k\}$. An equivalent DFA after reading the same string must be in some definite state. How can we make these two situations correspond? The answer is a nice trick: Label the states of the DFA with a set of states in such a way that, after reading w, the equivalent DFA will be in a single state labeled $\{q_i, q_j, \dots, q_k\}$. Since for a set of $|Q|$ states there are exactly $2^{|Q|}$ subsets, the corresponding DFA will have a finite number of states.

Most of the work in this suggested construction lies in the analysis of the NFA to get the correspondence between possible states and inputs. Before getting to the formal description of this, let us illustrate it with a simple example.

This result is not obvious and certainly has to be demonstrated. The argument, like most arguments in this book, will be constructive. This means that we can actually give a way of converting any NFA into an equivalent DFA. The construction is not hard to understand; once the idea is clear it becomes the starting point for a rigorous argument. The rationale for the construction is the following. After an NFA has read a string w, we may not know exactly what state it will be in, but we can say that it must be in one state of a set of possible states, say $\{q_i, q_j, \ldots, q_k\}$. An equivalent DFA after reading the same string must be in some definite state. How can we make these two situations correspond? The answer is a nice trick: Label the states of the DFA with a set of states in such a way that, after reading w, the equivalent DFA will be in a single state labeled $\{q_i, q_j, \ldots, q_k\}$. Since for a set of $|Q|$ states there are exactly $2^{|Q|}$ subsets, the corresponding DFA will have a finite number of states.

Most of the work in this suggested construction lies in the analysis of the NFA to get the correspondence between possible states and inputs. Before getting to the formal description of this, let us illustrate it with a simple example.

This result is not obvious and certainly has to be demonstrated. The argument, like most arguments in this book, will be constructive. This means that we can actually give a way of converting any NFA into an equivalent DFA. The construction is not hard to understand; once the idea is clear it becomes the starting point for a rigorous argument. The rationale for the construction is the following. After an NFA has read a string w, we may not know exactly what state it will be in, but we can say that it must be in one state of a set of possible states, say $\{q_i, q_j, \ldots, q_k\}$. An equivalent DFA after reading the same string must be in some definite state. How can we make these two situations correspond? The answer is a nice trick: Label the states of the DFA with a set of states in such a way that, after reading w, the equivalent DFA will be in a single state labeled $\{q_i, q_j, \ldots, q_k\}$. Since for a set of $|Q|$ states there are exactly $2^{|Q|}$ subsets, the corresponding DFA will have a finite number of states.

Most of the work in this suggested construction lies in the analysis of the NFA to get the correspondence between possible states and inputs. Before getting to the formal description of this, let us illustrate it with a simple example.

This result is not obvious and certainly has to be demonstrated. The argument, like most arguments in this book, will be constructive. This means that we can actually give a way of converting any NFA into an equivalent DFA. The construction is not hard to understand; once the idea is clear it becomes the starting point for a rigorous argument. The rationale for the construction is the following. After an NFA has read a string w, we may not know exactly what state it will be in, but we can say that it must be in one state of a set of possible states, say $\{q_i, q_j, \ldots, q_k\}$. An equivalent DFA after reading the same string must be in some definite state. How can we make these two situations correspond? The answer is a nice trick: Label the states of the DFA with a set of states in such a way that, after reading w, the equivalent DFA will be in a single state labeled $\{q_i, q_j, \ldots, q_k\}$. Since for a set of $|Q|$ states there are exactly $2^{|Q|}$ subsets, the corresponding DFA will have a finite number of states.

Most of the work in this suggested construction lies in the analysis of the NFA to get the correspondence between possible states and inputs. Before getting to the formal description of this, let us illustrate it with a simple example.

This result is not obvious and certainly has to be demonstrated. The argument, like most arguments in this book, will be constructive. This means that we can actually give a way of converting any NFA into an equivalent DFA. The construction is not hard to understand; once the idea is clear it becomes the starting point for a rigorous argument. The rationale for the construction is the following. After an NFA has read a string w, we may not know exactly what state it will be in, but we can say that it must be in one state of a set of possible states, say $\{q_i, q_j, \ldots, q_k\}$. An equivalent DFA after reading the same string must be in some definite state. How can we make these two situations correspond? The answer is a nice trick: Label the states of the DFA with a set of states in such a way that, after reading w, the equivalent DFA will be in a single state labeled $\{q_i, q_j, \ldots, q_k\}$. Since for a set of $|Q|$ states there are exactly $2^{|Q|}$ subsets, the corresponding DFA will have a finite number of states.

Most of the work in this suggested construction lies in the analysis of the NFA to get the correspondence between possible states and inputs. Before getting to the formal description of this, let us illustrate it with a simple example.

This result is not obvious and certainly has to be demonstrated. The argument, like most arguments in this book, will be constructive. This means that we can actually give a way of converting any NFA into an equivalent DFA. The construction is not hard to understand; once the idea is clear it becomes the starting point for a rigorous argument. The rationale for the construction is the following. After an NFA has read a string w, we may not know exactly what state it will be in, but we can say that it must be in one state of a set of possible states, say $\{q_i, q_j, \ldots, q_k\}$. An equivalent DFA after reading the same string must be in some definite state. How can we make these two situations correspond? The answer is a nice trick: Label the states of the DFA with a set of states in such a way that, after reading w, the equivalent DFA will be in a single state labeled $\{q_i, q_j, \ldots, q_k\}$. Since for a set of $|Q|$ states there are exactly $2^{|Q|}$ subsets, the corresponding DFA will have a finite number of states.

Most of the work in this suggested construction lies in the analysis of the NFA to get the correspondence between possible states and inputs. Before getting to the formal description of this, let us illustrate it with a simple example.

This result is not obvious and certainly has to be demonstrated. The argument, like most arguments in this book, will be constructive. This means that we can actually give a way of converting any NFA into an equivalent DFA. The construction is not hard to understand; once the idea is clear it becomes the starting point for a rigorous argument. The rationale for the construction is the following. After an NFA has read a string w, we may not know exactly what state it will be in, but we can say that it must be in one state of a set of possible states, say $\{q_i, q_j, \ldots, q_k\}$. An equivalent DFA after reading the same string must be in some definite state. How can we make these two situations correspond? The answer is a nice trick: Label the states of the DFA with a set of states in such a way that, after reading w, the equivalent DFA will be in a single state labeled $\{q_i, q_j, \ldots, q_k\}$. Since for a set of $|Q|$ states there are exactly $2^{|Q|}$ subsets, the corresponding DFA will have a finite number of states.

Most of the work in this suggested construction lies in the analysis of the NFA to get the correspondence between possible states and inputs. Before getting to the formal description of this, let us illustrate it with a simple example.

This result is not obvious and certainly has to be demonstrated. The argument, like most arguments in this book, will be constructive. This means that we can actually give a way of converting any NFA into an equivalent DFA. The construction is not hard to understand; once the idea is clear it becomes the starting point for a rigorous argument. The rationale for the construction is the following. After an NFA has read a string w, we may not know exactly what state it will be in, but we can say that it must be in one state of a set of possible states, say $\{q_i, q_j, \ldots, q_k\}$. An equivalent DFA after reading the same string must be in some definite state. How can we make these two situations correspond? The answer is a nice trick: Label the states of the DFA with a set of states in such a way that, after reading w, the equivalent DFA will be in a single state labeled $\{q_i, q_j, \ldots, q_k\}$. Since for a set of $|Q|$ states there are exactly $2^{|Q|}$ subsets, the corresponding DFA will have a finite number of states.

Most of the work in this suggested construction lies in the analysis of the NFA to get the correspondence between possible states and inputs. Before getting to the formal description of this, let us illustrate it with a simple example.

This result is not obvious and certainly has to be demonstrated. The argument, like most arguments in this book, will be constructive. This means that we can actually give a way of converting any NFA into an equivalent DFA. The construction is not hard to understand; once the idea is clear it becomes the starting point for a rigorous argument. The rationale for the construction is the following. After an NFA has read a string w, we may not know exactly what state it will be in, but we can say that it must be in one state of a set of possible states, say $\{q_i, q_j, \ldots, q_k\}$. An equivalent DFA after reading the same string must be in some definite state. How can we make these two situations correspond? The answer is a nice trick: Label the states of the DFA with a set of states in such a way that, after reading w, the equivalent DFA will be in a single state labeled $\{q_i, q_j, \ldots, q_k\}$. Since for a set of $|Q|$ states there are exactly $2^{|Q|}$ subsets, the corresponding DFA will have a finite number of states.

Most of the work in this suggested construction lies in the analysis of the NFA to get the correspondence between possible states and inputs. Before getting to the formal description of this, let us illustrate it with a simple example.

This result is not obvious and certainly has to be demonstrated. The argument, like most arguments in this book, will be constructive. This means that we can actually give a way of converting any NFA into an equivalent DFA. The construction is not hard to understand; once the idea is clear it becomes the starting point for a rigorous argument. The rationale for the construction is the following. After an NFA has read a string w, we may not know exactly what state it will be in, but we can say that it must be in one state of a set of possible states, say $\{q_i, q_j, \ldots, q_k\}$. An equivalent DFA after reading the same string must be in some definite state. How can we make these two situations correspond? The answer is a nice trick: Label the states of the DFA with a set of states in such a way that, after reading w, the equivalent DFA will be in a single state labeled $\{q_i, q_j, \ldots, q_k\}$. Since for a set of $|Q|$ states there are exactly $2^{|Q|}$ subsets, the corresponding DFA will have a finite number of states.

Most of the work in this suggested construction lies in the analysis of the NFA to get the correspondence between possible states and inputs. Before getting to the formal description of this, let us illustrate it with a simple example.

Example 2.12

Convert the NFA in the following Figure to an equivalent DFA.

The NFA starts in state q_0, so the initial state of the DFA will be labeled $\{q_0\}$. After reading a symbol a, the NFA can be in state q_1 or, by making a λ-transition, in state q_2. Therefore, the corresponding DFA must have a state labeled $\{q_1, q_2\}$ and a transition

$$\delta(\{q_0\}, a) = \{q_1, q_2\}.$$

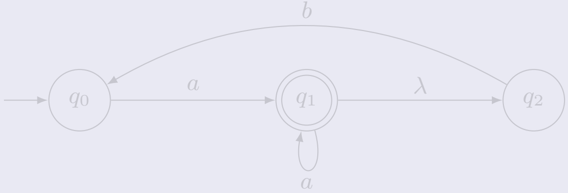
In state q_0, the NFA has no specified transition when the input is b; therefore,

$$\delta(\{q_0\}, b) = \varnothing.$$

A state labeled $\varnothing$ represents an impossible move for the NFA and, therefore, means nonacceptance of the string. Consequently, this state in the DFA must be a nonfinal trap state.

Example 2.12

Convert the NFA in the following Figure to an equivalent DFA.



The NFA starts in state q_0, so the initial state of the DFA will be labeled $\{q_0\}$. After reading a symbol a, the NFA can be in state q_1 or, by making a λ-transition, in state q_2. Therefore, the corresponding DFA must have a state labeled $\{q_1, q_2\}$ and a transition

$$\delta(\{q_0\}, a) = \{q_1, q_2\}.$$

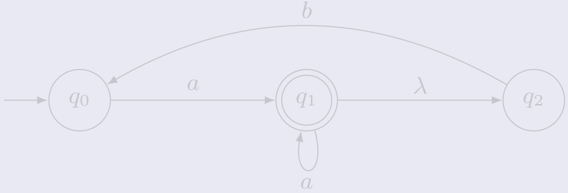
In state q_0, the NFA has no specified transition when the input is b; therefore,

$$\delta(\{q_0\}, b) = \varnothing.$$

A state labeled $\varnothing$ represents an impossible move for the NFA and, therefore, means nonacceptance of the string. Consequently, this state in the DFA must be a nonfinal trap state.

Example 2.12

Convert the NFA in the following Figure to an equivalent DFA.



The NFA starts in state q_0, so the initial state of the DFA will be labeled $\{q_0\}$. After reading a symbol a, the NFA can be in state q_1 or, by making a λ-transition, in state q_2. Therefore, the corresponding DFA must have a state labeled $\{q_1, q_2\}$ and a transition

$$\delta(\{q_0\}, a) = \{q_1, q_2\}.$$

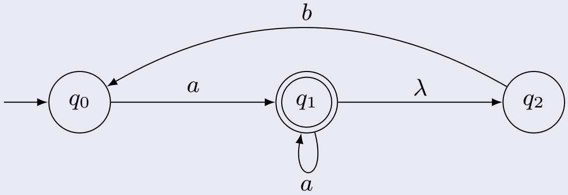
In state q_0, the NFA has no specified transition when the input is b; therefore,

$$\delta(\{q_0\}, b) = \varnothing.$$

A state labeled $\varnothing$ represents an impossible move for the NFA and, therefore, means nonacceptance of the string. Consequently, this state in the DFA must be a nonfinal trap state.

Example 2.12

Convert the NFA in the following Figure to an equivalent DFA.



The NFA starts in state q_0, so the initial state of the DFA will be labeled $\{q_0\}$. After reading a symbol a, the NFA can be in state q_1 or, by making a λ-transition, in state q_2. Therefore, the corresponding DFA must have a state labeled $\{q_1, q_2\}$ and a transition

$$\delta(\{q_0\}, a) = \{q_1, q_2\}.$$

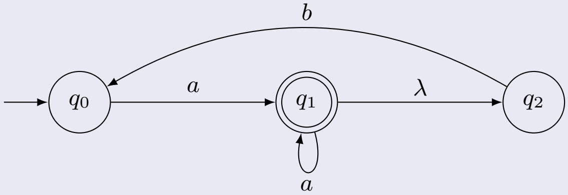
In state q_0, the NFA has no specified transition when the input is b; therefore,

$$\delta(\{q_0\}, b) = \varnothing.$$

A state labeled $\varnothing$ represents an impossible move for the NFA and, therefore, means nonacceptance of the string. Consequently, this state in the DFA must be a nonfinal trap state.

Example 2.12

Convert the NFA in the following Figure to an equivalent DFA.



The NFA starts in state q_0, so the initial state of the DFA will be labeled $\{q_0\}$. After reading a symbol a, the NFA can be in state q_1 or, by making a λ-transition, in state q_2. Therefore, the corresponding DFA must have a state labeled $\{q_1, q_2\}$ and a transition

$$\delta(\{q_0\}, a) = \{q_1, q_2\}.$$

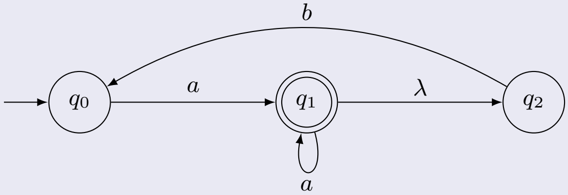
In state q_0, the NFA has no specified transition when the input is b; therefore,

$$\delta(\{q_0\}, b) = \varnothing.$$

A state labeled $\varnothing$ represents an impossible move for the NFA and, therefore, means nonacceptance of the string. Consequently, this state in the DFA must be a nonfinal trap state.

Example 2.12

Convert the NFA in the following Figure to an equivalent DFA.



The NFA starts in state q_0, so the initial state of the DFA will be labeled $\{q_0\}$. After reading a symbol a, the NFA can be in state q_1 or, by making a λ-transition, in state q_2. Therefore, the corresponding DFA must have a state labeled $\{q_1, q_2\}$ and a transition

$$\delta(\{q_0\}, a) = \{q_1, q_2\}.$$

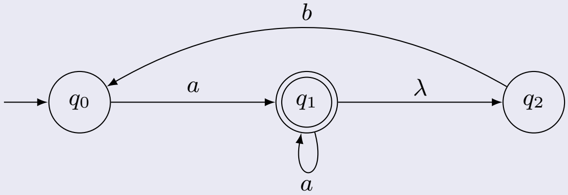
In state q_0, the NFA has no specified transition when the input is b; therefore,

$$\delta(\{q_0\}, b) = \varnothing.$$

A state labeled $\varnothing$ represents an impossible move for the NFA and, therefore, means nonacceptance of the string. Consequently, this state in the DFA must be a nonfinal trap state.

Example 2.12

Convert the NFA in the following Figure to an equivalent DFA.



The NFA starts in state q_0, so the initial state of the DFA will be labeled $\{q_0\}$. After reading a symbol a, the NFA can be in state q_1 or, by making a λ-transition, in state q_2. Therefore, the corresponding DFA must have a state labeled $\{q_1, q_2\}$ and a transition

$$\delta(\{q_0\}, a) = \{q_1, q_2\}.$$

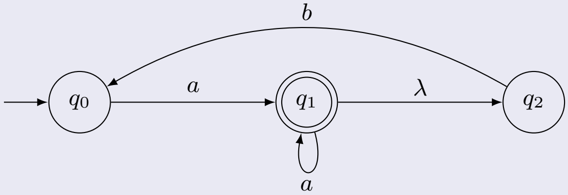
In state q_0, the NFA has no specified transition when the input is b; therefore,

$$\delta(\{q_0\}, b) = \varnothing.$$

A state labeled $\varnothing$ represents an impossible move for the NFA and, therefore, means nonacceptance of the string. Consequently, this state in the DFA must be a nonfinal trap state.

Example 2.12

Convert the NFA in the following Figure to an equivalent DFA.



The NFA starts in state q_0, so the initial state of the DFA will be labeled $\{q_0\}$. After reading a symbol a, the NFA can be in state q_1 or, by making a λ-transition, in state q_2. Therefore, the corresponding DFA must have a state labeled $\{q_1, q_2\}$ and a transition

$$\delta(\{q_0\}, a) = \{q_1, q_2\}.$$

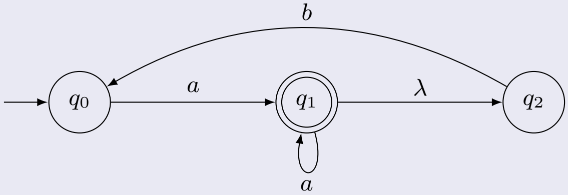
In state q_0, the NFA has no specified transition when the input is b; therefore,

$$\delta(\{q_0\}, b) = \varnothing.$$

A state labeled $\varnothing$ represents an impossible move for the NFA and, therefore, means nonacceptance of the string. Consequently, this state in the DFA must be a nonfinal trap state.

Example 2.12

Convert the NFA in the following Figure to an equivalent DFA.



The NFA starts in state q_0, so the initial state of the DFA will be labeled $\{q_0\}$. After reading a symbol a, the NFA can be in state q_1 or, by making a λ-transition, in state q_2. Therefore, the corresponding DFA must have a state labeled $\{q_1, q_2\}$ and a transition

$$\delta(\{q_0\}, a) = \{q_1, q_2\}.$$

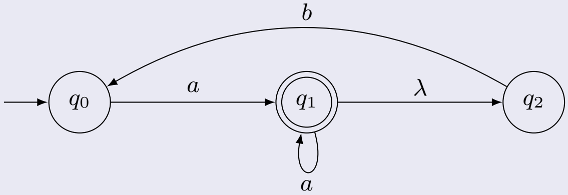
In state q_0, the NFA has no specified transition when the input is b; therefore,

$$\delta(\{q_0\}, b) = \varnothing.$$

A state labeled $\varnothing$ represents an impossible move for the NFA and, therefore, means nonacceptance of the string. Consequently, this state in the DFA must be a nonfinal trap state.

Example 2.12

Convert the NFA in the following Figure to an equivalent DFA.



The NFA starts in state q_0, so the initial state of the DFA will be labeled $\{q_0\}$. After reading a symbol a, the NFA can be in state q_1 or, by making a λ-transition, in state q_2. Therefore, the corresponding DFA must have a state labeled $\{q_1, q_2\}$ and a transition

$$\delta(\{q_0\}, a) = \{q_1, q_2\}.$$

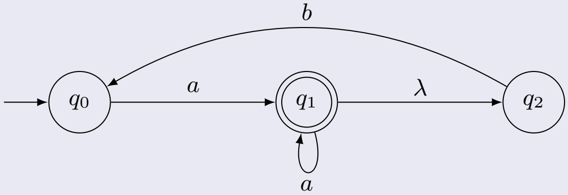
In state q_0, the NFA has no specified transition when the input is b; therefore,

$$\delta(\{q_0\}, b) = \varnothing.$$

A state labeled $\varnothing$ represents an impossible move for the NFA and, therefore, means nonacceptance of the string. Consequently, this state in the DFA must be a nonfinal trap state.

Example 2.12

Convert the NFA in the following Figure to an equivalent DFA.



The NFA starts in state q_0, so the initial state of the DFA will be labeled $\{q_0\}$. After reading a symbol a, the NFA can be in state q_1 or, by making a λ-transition, in state q_2. Therefore, the corresponding DFA must have a state labeled $\{q_1, q_2\}$ and a transition

$$\delta(\{q_0\}, a) = \{q_1, q_2\}.$$

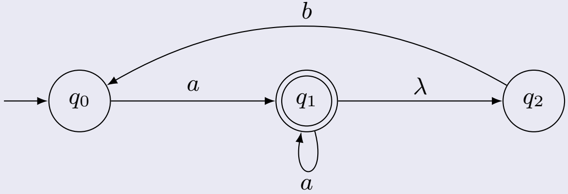
In state q_0, the NFA has no specified transition when the input is b; therefore,

$$\delta(\{q_0\}, b) = \varnothing.$$

A state labeled $\varnothing$ represents an impossible move for the NFA and, therefore, means nonacceptance of the string. Consequently, this state in the DFA must be a nonfinal trap state.

Example 2.12

Convert the NFA in the following Figure to an equivalent DFA.



The NFA starts in state q_0, so the initial state of the DFA will be labeled $\{q_0\}$. After reading a symbol a, the NFA can be in state q_1 or, by making a λ-transition, in state q_2. Therefore, the corresponding DFA must have a state labeled $\{q_1, q_2\}$ and a transition

$$\delta(\{q_0\}, a) = \{q_1, q_2\}.$$

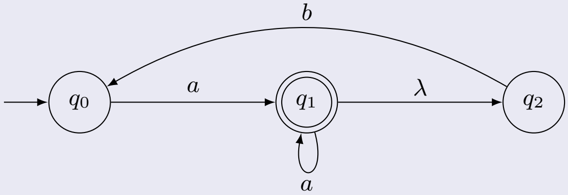
In state q_0, the NFA has no specified transition when the input is b; therefore,

$$\delta(\{q_0\}, b) = \varnothing.$$

A state labeled $\varnothing$ represents an impossible move for the NFA and, therefore, means nonacceptance of the string. Consequently, this state in the DFA must be a nonfinal trap state.

Example 2.12

Convert the NFA in the following Figure to an equivalent DFA.



The NFA starts in state q_0, so the initial state of the DFA will be labeled $\{q_0\}$. After reading a symbol a, the NFA can be in state q_1 or, by making a λ-transition, in state q_2. Therefore, the corresponding DFA must have a state labeled $\{q_1, q_2\}$ and a transition

$$\delta(\{q_0\}, a) = \{q_1, q_2\}.$$

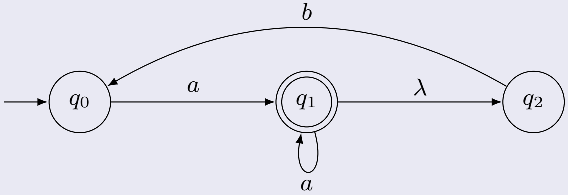
In state q_0, the NFA has no specified transition when the input is b; therefore,

$$\delta(\{q_0\}, b) = \varnothing.$$

A state labeled $\varnothing$ represents an impossible move for the NFA and, therefore, means nonacceptance of the string. Consequently, this state in the DFA must be a nonfinal trap state.

Example 2.12

Convert the NFA in the following Figure to an equivalent DFA.



The NFA starts in state q_0, so the initial state of the DFA will be labeled $\{q_0\}$. After reading a symbol a, the NFA can be in state q_1 or, by making a λ-transition, in state q_2. Therefore, the corresponding DFA must have a state labeled $\{q_1, q_2\}$ and a transition

$$\delta(\{q_0\}, a) = \{q_1, q_2\}.$$

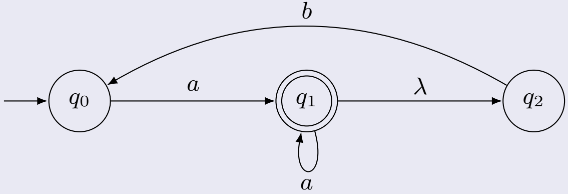
In state q_0, the NFA has no specified transition when the input is b; therefore,

$$\delta(\{q_0\}, b) = \varnothing.$$

A state labeled $\varnothing$ represents an impossible move for the NFA and, therefore, means nonacceptance of the string. Consequently, this state in the DFA must be a nonfinal trap state.

Example 2.12

Convert the NFA in the following Figure to an equivalent DFA.



The NFA starts in state q_0, so the initial state of the DFA will be labeled $\{q_0\}$. After reading a symbol a, the NFA can be in state q_1 or, by making a λ-transition, in state q_2. Therefore, the corresponding DFA must have a state labeled $\{q_1, q_2\}$ and a transition

$$\delta(\{q_0\}, a) = \{q_1, q_2\}.$$

In state q_0, the NFA has no specified transition when the input is b; therefore,

$$\delta(\{q_0\}, b) = \varnothing.$$

A state labeled $\varnothing$ represents an impossible move for the NFA and, therefore, means nonacceptance of the string. Consequently, this state in the DFA must be a nonfinal trap state.

Example 2.12 (continuation)

We have now introduced into the DFA the state $\{q_1, q_2\}$, so we need to find the transitions out of this state. Remember that this state of the DFA corresponds to two possible states of the NFA, so we must refer back to the NFA. If the NFA is in state q_1 and reads a symbol a, it can go to q_1. Furthermore, from q_1 the NFA can make a λ-transition to q_2. If, for the same input, the NFA is in state q_2, then there is no specified transition. Therefore,

$$\delta(\{q_1, q_2\}, a) = \{q_1, q_2\}.$$

Similarly,

$$\delta(\{q_1, q_2\}, b) = \{q_0\}.$$

Example 2.12 (continuation)

We have now introduced into the DFA the state $\{q_1, q_2\}$, so we need to find the transitions out of this state. Remember that this state of the DFA corresponds to two possible states of the NFA, so we must refer back to the NFA. If the NFA is in state q_1 and reads a symbol a, it can go to q_1. Furthermore, from q_1 the NFA can make a λ-transition to q_2. If, for the same input, the NFA is in state q_2, then there is no specified transition. Therefore,

$$\delta(\{q_1, q_2\}, a) = \{q_1, q_2\}.$$

Similarly,

$$\delta(\{q_1, q_2\}, b) = \{q_0\}.$$

Example 2.12 (continuation)

We have now introduced into the DFA the state $\{q_1, q_2\}$, so we need to find the transitions out of this state. Remember that this state of the DFA corresponds to two possible states of the NFA, so we must refer back to the NFA. If the NFA is in state q_1 and reads a symbol a, it can go to q_1. Furthermore, from q_1 the NFA can make a λ-transition to q_2. If, for the same input, the NFA is in state q_2, then there is no specified transition. Therefore,

$$\delta(\{q_1, q_2\}, a) = \{q_1, q_2\}.$$

Similarly,

$$\delta(\{q_1, q_2\}, b) = \{q_0\}.$$

Example 2.12 (continuation)

We have now introduced into the DFA the state $\{q_1, q_2\}$, so we need to find the transitions out of this state. Remember that this state of the DFA corresponds to two possible states of the NFA, so we must refer back to the NFA. If the NFA is in state q_1 and reads a symbol a, it can go to q_1. Furthermore, from q_1 the NFA can make a λ-transition to q_2. If, for the same input, the NFA is in state q_2, then there is no specified transition. Therefore,

$$\delta(\{q_1, q_2\}, a) = \{q_1, q_2\}.$$

Similarly,

$$\delta(\{q_1, q_2\}, b) = \{q_0\}.$$

Example 2.12 (continuation)

We have now introduced into the DFA the state $\{q_1, q_2\}$, so we need to find the transitions out of this state. Remember that this state of the DFA corresponds to two possible states of the NFA, so we must refer back to the NFA. If the NFA is in state q_1 and reads a symbol a, it can go to q_1. Furthermore, from q_1 the NFA can make a λ-transition to q_2. If, for the same input, the NFA is in state q_2, then there is no specified transition. Therefore,

$$\delta(\{q_1, q_2\}, a) = \{q_1, q_2\}.$$

Similarly,

$$\delta(\{q_1, q_2\}, b) = \{q_0\}.$$

Example 2.12 (continuation)

We have now introduced into the DFA the state $\{q_1, q_2\}$, so we need to find the transitions out of this state. Remember that this state of the DFA corresponds to two possible states of the NFA, so we must refer back to the NFA. If the NFA is in state q_1 and reads a symbol a, it can go to q_1. Furthermore, from q_1 the NFA can make a λ-transition to q_2. If, for the same input, the NFA is in state q_2, then there is no specified transition. Therefore,

$$\delta(\{q_1, q_2\}, a) = \{q_1, q_2\}.$$

Similarly,

$$\delta(\{q_1, q_2\}, b) = \{q_0\}.$$

Example 2.12 (continuation)

We have now introduced into the DFA the state $\{q_1, q_2\}$, so we need to find the transitions out of this state. Remember that this state of the DFA corresponds to two possible states of the NFA, so we must refer back to the NFA. If the NFA is in state q_1 and reads a symbol a, it can go to q_1. Furthermore, from q_1 the NFA can make a λ-transition to q_2. If, for the same input, the NFA is in state q_2, then there is no specified transition. Therefore,

$$\delta(\{q_1, q_2\}, a) = \{q_1, q_2\}.$$

Similarly,

$$\delta(\{q_1, q_2\}, b) = \{q_0\}.$$

Example 2.12 (continuation)

We have now introduced into the DFA the state $\{q_1, q_2\}$, so we need to find the transitions out of this state. Remember that this state of the DFA corresponds to two possible states of the NFA, so we must refer back to the NFA. If the NFA is in state q_1 and reads a symbol a, it can go to q_1. Furthermore, from q_1 the NFA can make a λ-transition to q_2. If, for the same input, the NFA is in state q_2, then there is no specified transition. Therefore,

$$\delta(\{q_1, q_2\}, a) = \{q_1, q_2\}.$$

Similarly,

$$\delta(\{q_1, q_2\}, b) = \{q_0\}.$$

Example 2.12 (continuation)

We have now introduced into the DFA the state $\{q_1, q_2\}$, so we need to find the transitions out of this state. Remember that this state of the DFA corresponds to two possible states of the NFA, so we must refer back to the NFA. If the NFA is in state q_1 and reads a symbol a, it can go to q_1. Furthermore, from q_1 the NFA can make a λ-transition to q_2. If, for the same input, the NFA is in state q_2, then there is no specified transition. Therefore,

$$\delta(\{q_1, q_2\}, a) = \{q_1, q_2\}.$$

Similarly,

$$\delta(\{q_1, q_2\}, b) = \{q_0\}.$$

Example 2.12 (continuation)

We have now introduced into the DFA the state $\{q_1, q_2\}$, so we need to find the transitions out of this state. Remember that this state of the DFA corresponds to two possible states of the NFA, so we must refer back to the NFA. If the NFA is in state q_1 and reads a symbol a, it can go to q_1. Furthermore, from q_1 the NFA can make a λ-transition to q_2. If, for the same input, the NFA is in state q_2, then there is no specified transition. Therefore,

$$\delta(\{q_1, q_2\}, a) = \{q_1, q_2\}.$$

Similarly,

$$\delta(\{q_1, q_2\}, b) = \{q_0\}.$$

Example 2.12 (continuation)

We have now introduced into the DFA the state $\{q_1, q_2\}$, so we need to find the transitions out of this state. Remember that this state of the DFA corresponds to two possible states of the NFA, so we must refer back to the NFA. If the NFA is in state q_1 and reads a symbol a, it can go to q_1. Furthermore, from q_1 the NFA can make a λ-transition to q_2. If, for the same input, the NFA is in state q_2, then there is no specified transition. Therefore,

$$\delta(\{q_1, q_2\}, a) = \{q_1, q_2\}.$$

Similarly,

$$\delta(\{q_1, q_2\}, b) = \{q_0\}.$$

Example 2.12 (continuation)

We have now introduced into the DFA the state $\{q_1, q_2\}$, so we need to find the transitions out of this state. Remember that this state of the DFA corresponds to two possible states of the NFA, so we must refer back to the NFA. If the NFA is in state q_1 and reads a symbol a, it can go to q_1. Furthermore, from q_1 the NFA can make a λ-transition to q_2. If, for the same input, the NFA is in state q_2, then there is no specified transition. Therefore,

$$\delta(\{q_1, q_2\}, a) = \{q_1, q_2\}.$$

Similarly,

$$\delta(\{q_1, q_2\}, b) = \{q_0\}.$$

Example 2.12 (continuation)

We have now introduced into the DFA the state $\{q_1, q_2\}$, so we need to find the transitions out of this state. Remember that this state of the DFA corresponds to two possible states of the NFA, so we must refer back to the NFA. If the NFA is in state q_1 and reads a symbol a, it can go to q_1. Furthermore, from q_1 the NFA can make a λ-transition to q_2. If, for the same input, the NFA is in state q_2, then there is no specified transition. Therefore,

$$\delta(\{q_1, q_2\}, a) = \{q_1, q_2\}.$$

Similarly,

$$\delta(\{q_1, q_2\}, b) = \{q_0\}.$$

Example 2.12 (continuation)

We have now introduced into the DFA the state $\{q_1, q_2\}$, so we need to find the transitions out of this state. Remember that this state of the DFA corresponds to two possible states of the NFA, so we must refer back to the NFA. If the NFA is in state q_1 and reads a symbol a, it can go to q_1. Furthermore, from q_1 the NFA can make a λ-transition to q_2. If, for the same input, the NFA is in state q_2, then there is no specified transition. Therefore,

$$\delta(\{q_1, q_2\}, a) = \{q_1, q_2\}.$$

Similarly,

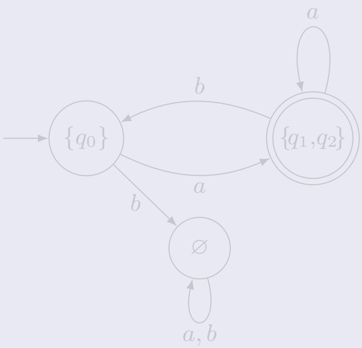
$$\delta(\{q_1, q_2\}, b) = \{q_0\}.$$

Example 2.12 (continuation)

At this point, every state has all transitions defined. The result, shown in the following Figure, is a DFA, equivalent to the NFA with which we started. The NFA in the Figure accepts any string for which $\delta^*(q_0, w)$ contains q_1. For the corresponding DFA to accept every such w, any state whose label includes q_1 must be made a final state.

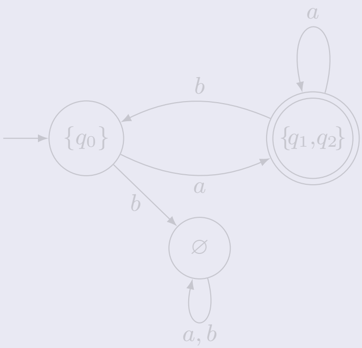
Example 2.12 (continuation)

At this point, every state has all transitions defined. The result, shown in the following Figure, is a DFA, equivalent to the NFA with which we started. The NFA in the Figure accepts any string for which $\delta^*(q_0, w)$ contains q_1. For the corresponding DFA to accept every such w, any state whose label includes q_1 must be made a final state.



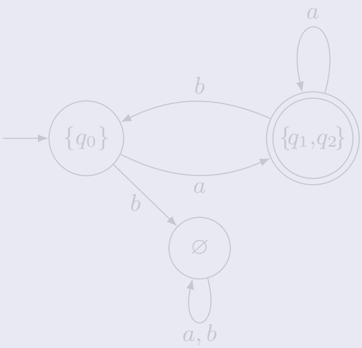
Example 2.12 (continuation)

At this point, every state has all transitions defined. The result, shown in the following Figure, is a DFA, equivalent to the NFA with which we started. The NFA in the Figure accepts any string for which $\delta^*(q_0, w)$ contains q_1. For the corresponding DFA to accept every such w, any state whose label includes q_1 must be made a final state.



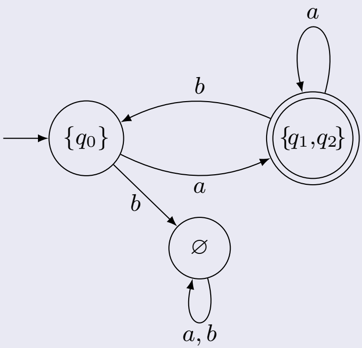
Example 2.12 (continuation)

At this point, every state has all transitions defined. The result, shown in the following Figure, is a DFA, equivalent to the NFA with which we started. The NFA in the Figure accepts any string for which $\delta^*(q_0, w)$ contains q_1. For the corresponding DFA to accept every such w, any state whose label includes q_1 must be made a final state.



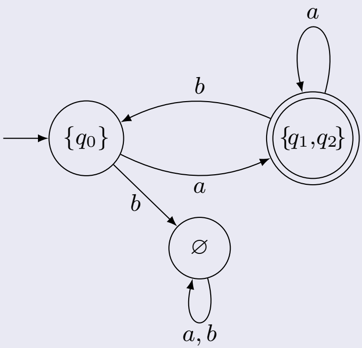
Example 2.12 (continuation)

At this point, every state has all transitions defined. The result, shown in the following Figure, is a DFA, equivalent to the NFA with which we started. The NFA in the Figure accepts any string for which $\delta^*(q_0, w)$ contains q_1. For the corresponding DFA to accept every such w, any state whose label includes q_1 must be made a final state.



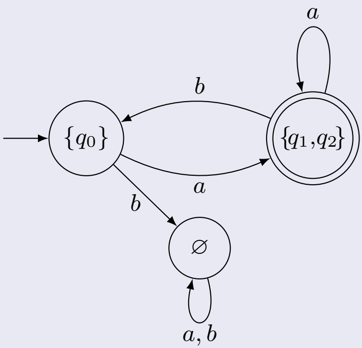
Example 2.12 (continuation)

At this point, every state has all transitions defined. The result, shown in the following Figure, is a DFA, equivalent to the NFA with which we started. The NFA in the Figure accepts any string for which $\delta^*(q_0, w)$ contains q_1. For the corresponding DFA to accept every such w, any state whose label includes q_1 must be made a final state.



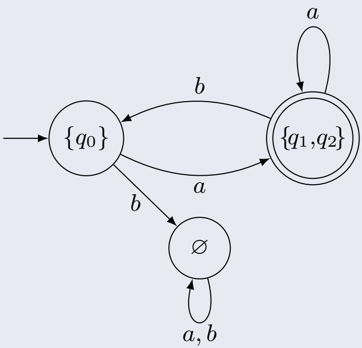
Example 2.12 (continuation)

At this point, every state has all transitions defined. The result, shown in the following Figure, is a DFA, equivalent to the NFA with which we started. The NFA in the Figure accepts any string for which $\delta^*(q_0, w)$ contains q_1. For the corresponding DFA to accept every such w, any state whose label includes q_1 must be made a final state.



Example 2.12 (continuation)

At this point, every state has all transitions defined. The result, shown in the following Figure, is a DFA, equivalent to the NFA with which we started. The NFA in the Figure accepts any string for which $\delta^*(q_0, w)$ contains q_1. For the corresponding DFA to accept every such w, any state whose label includes q_1 must be made a final state.



Theorem 2.2

Let L be the language accepted by a nondeterministic finite accepter $M_N = (Q_N, \Sigma, \delta_N, q_0, F_N)$. Then there exists a deterministic finite accepter $M_D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$ such that

$$L(M_N) = L(M_D).$$

Proof. Given M_N, we use the procedure *NFA-to-DFA* below to construct the transition graph G_D for M_D. To understand the construction, remember that G_D has to have certain properties. Every vertex must have exactly $|\Sigma|$ outgoing edges, each labeled with a different element of Σ. During the construction, some of the edges may be missing, but the procedure continues until they are all there.

Theorem 2.2

Let L be the language accepted by a nondeterministic finite accepter $M_N = (Q_N, \Sigma, \delta_N, q_0, F_N)$. Then there exists a deterministic finite accepter $M_D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$ such that

$$L(M_N) = L(M_D).$$

Proof. Given M_N, we use the procedure *NFA-to-DFA* below to construct the transition graph G_D for M_D. To understand the construction, remember that G_D has to have certain properties. Every vertex must have exactly $|\Sigma|$ outgoing edges, each labeled with a different element of Σ. During the construction, some of the edges may be missing, but the procedure continues until they are all there.

Theorem 2.2

Let L be the language accepted by a nondeterministic finite accepter $M_N = (Q_N, \Sigma, \delta_N, q_0, F_N)$. Then there exists a deterministic finite accepter $M_D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$ such that

$$L(M_N) = L(M_D).$$

Proof. Given M_N, we use the procedure *NFA-to-DFA* below to construct the transition graph G_D for M_D. To understand the construction, remember that G_D has to have certain properties. Every vertex must have exactly $|\Sigma|$ outgoing edges, each labeled with a different element of Σ. During the construction, some of the edges may be missing, but the procedure continues until they are all there.

Theorem 2.2

Let L be the language accepted by a nondeterministic finite accepter $M_N = (Q_N, \Sigma, \delta_N, q_0, F_N)$. Then there exists a deterministic finite accepter $M_D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$ such that

$$L(M_N) = L(M_D).$$

Proof. Given M_N, we use the procedure *NFA-to-DFA* below to construct the transition graph G_D for M_D. To understand the construction, remember that G_D has to have certain properties. Every vertex must have exactly $|\Sigma|$ outgoing edges, each labeled with a different element of Σ. During the construction, some of the edges may be missing, but the procedure continues until they are all there.

Theorem 2.2

Let L be the language accepted by a nondeterministic finite accepter $M_N = (Q_N, \Sigma, \delta_N, q_0, F_N)$. Then there exists a deterministic finite accepter $M_D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$ such that

$$L(M_N) = L(M_D).$$

Proof. Given M_N, we use the procedure *NFA-to-DFA* below to construct the transition graph G_D for M_D. To understand the construction, remember that G_D has to have certain properties. Every vertex must have exactly $|\Sigma|$ outgoing edges, each labeled with a different element of Σ. During the construction, some of the edges may be missing, but the procedure continues until they are all there.

Theorem 2.2

Let L be the language accepted by a nondeterministic finite accepter $M_N = (Q_N, \Sigma, \delta_N, q_0, F_N)$. Then there exists a deterministic finite accepter $M_D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$ such that

$$L(M_N) = L(M_D).$$

Proof. Given M_N, we use the procedure *NFA-to-DFA* below to construct the transition graph G_D for M_D. To understand the construction, remember that G_D has to have certain properties. Every vertex must have exactly $|\Sigma|$ outgoing edges, each labeled with a different element of Σ. During the construction, some of the edges may be missing, but the procedure continues until they are all there.

Theorem 2.2

Let L be the language accepted by a nondeterministic finite accepter $M_N = (Q_N, \Sigma, \delta_N, q_0, F_N)$. Then there exists a deterministic finite accepter $M_D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$ such that

$$L(M_N) = L(M_D).$$

Proof. Given M_N, we use the procedure *NFA-to-DFA* below to construct the transition graph G_D for M_D. To understand the construction, remember that G_D has to have certain properties. Every vertex must have exactly $|\Sigma|$ outgoing edges, each labeled with a different element of Σ. During the construction, some of the edges may be missing, but the procedure continues until they are all there.

Theorem 2.2

Let L be the language accepted by a nondeterministic finite accepter $M_N = (Q_N, \Sigma, \delta_N, q_0, F_N)$. Then there exists a deterministic finite accepter $M_D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$ such that

$$L(M_N) = L(M_D).$$

Proof. Given M_N, we use the procedure *NFA-to-DFA* below to construct the transition graph G_D for M_D. To understand the construction, remember that G_D has to have certain properties. Every vertex must have exactly $|\Sigma|$ outgoing edges, each labeled with a different element of Σ. During the construction, some of the edges may be missing, but the procedure continues until they are all there.

Theorem 2.2

Let L be the language accepted by a nondeterministic finite accepter $M_N = (Q_N, \Sigma, \delta_N, q_0, F_N)$. Then there exists a deterministic finite accepter $M_D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$ such that

$$L(M_N) = L(M_D).$$

Proof. Given M_N, we use the procedure *NFA-to-DFA* below to construct the transition graph G_D for M_D. To understand the construction, remember that G_D has to have certain properties. Every vertex must have exactly $|\Sigma|$ outgoing edges, each labeled with a different element of Σ. During the construction, some of the edges may be missing, but the procedure continues until they are all there.

Theorem 2.2

Let L be the language accepted by a nondeterministic finite accepter $M_N = (Q_N, \Sigma, \delta_N, q_0, F_N)$. Then there exists a deterministic finite accepter $M_D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$ such that

$$L(M_N) = L(M_D).$$

Proof. Given M_N, we use the procedure *NFA-to-DFA* below to construct the transition graph G_D for M_D. To understand the construction, remember that G_D has to have certain properties. Every vertex must have exactly $|\Sigma|$ outgoing edges, each labeled with a different element of Σ. During the construction, some of the edges may be missing, but the procedure continues until they are all there.

Theorem 2.2

Let L be the language accepted by a nondeterministic finite accepter $M_N = (Q_N, \Sigma, \delta_N, q_0, F_N)$. Then there exists a deterministic finite accepter $M_D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$ such that

$$L(M_N) = L(M_D).$$

Proof. Given M_N, we use the procedure *NFA-to-DFA* below to construct the transition graph G_D for M_D. To understand the construction, remember that G_D has to have certain properties. Every vertex must have exactly $|\Sigma|$ outgoing edges, each labeled with a different element of Σ. During the construction, some of the edges may be missing, but the procedure continues until they are all there.

Theorem 2.2

Let L be the language accepted by a nondeterministic finite accepter $M_N = (Q_N, \Sigma, \delta_N, q_0, F_N)$. Then there exists a deterministic finite accepter $M_D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$ such that

$$L(M_N) = L(M_D).$$

Proof. Given M_N, we use the procedure *NFA-to-DFA* below to construct the transition graph G_D for M_D. To understand the construction, remember that G_D has to have certain properties. Every vertex must have exactly $|\Sigma|$ outgoing edges, each labeled with a different element of Σ. During the construction, some of the edges may be missing, but the procedure continues until they are all there.

Procedure ***NFA-to-DFA***

1. Create a graph G_D with vertex $\{q_0\}$.

2. Repeat the following steps until no more edges are missing.

   Take any vertex $\{q_i, q_j, \ldots, q_k\}$ of G_D that has no outgoing edge for some $a \in \Sigma$.

   Add to G_D an edge from $\{q_i, q_j, \ldots, q_k\}$ to $\{q_l, q_m, \ldots, q_n\}$ and label it with a.

3. Every state of G_D whose label contains any $q_f \in F_N$ is identified as a final vertex.

4. If M_N accepts λ, the vertex $\{q_0\}$ in G_D is also made a final vertex.

Procedure *NFA-to-DFA*

1. Create a graph G_D with vertex $\{q_0\}$.
2. Repeat the following steps until no more edges are missing.

   Take any vertex $\{q_i, q_j, \ldots, q_k\}$ of G_D that has no outgoing edge for some $a \in \Sigma$.

   Add to G_D an edge from $\{q_i, q_j, \ldots, q_k\}$ to $\{q_l, q_m, \ldots, q_n\}$ and label it with a.
3. Every state of G_D whose label contains any $q_f \in F_N$ is identified as a final vertex.
4. If M_N accepts λ, the vertex q_0 in G_D is also made a final vertex.

Procedure *NFA-to-DFA*

1. Create a graph G_D with vertex $\{q_0\}$. Identify this vertex as the initial vertex.
2. Repeat the following steps until no more edges are missing.

   Take any vertex $\{q_i, q_j, \ldots, q_k\}$ of G_D that has no outgoing edge for some $a \in \Sigma$. Compute $\delta_N^*(q_i, a)$, $\delta_N^*(q_j, a)$, ..., $\delta_N^*(q_k, a)$. If

   $$\delta_N^*(q_i, a) \cup \delta_N^*(q_j, a) \cup \cdots \cup \delta_N^*(q_k, a) = \{q_l, q_m, \ldots, q_n\},$$

   create a vertex for G_D labeled $\{q_l, q_m, \ldots, q_n\}$ if it does not already exist.

   Add to G_D an edge from $\{q_i, q_j, \ldots, q_k\}$ to $\{q_l, q_m, \ldots, q_n\}$ and label it with a.
3. Every state of G_D whose label contains any $q_f \in F_N$ is identified as a final vertex.
4. If M_N accepts λ, the vertex q_0 in G_D is also made a final vertex.

Procedure *NFA-to-DFA*

1. Create a graph G_D with vertex $\{q_0\}$. Identify this vertex as the initial vertex.
2. Repeat the following steps until no more edges are missing.

   Take any vertex $\{q_i, q_j, \ldots, q_k\}$ of G_D that has no outgoing edge for some $a \in \Sigma$. Compute $\delta_N^*(q_i, a)$, $\delta_N^*(q_j, a)$, ..., $\delta_N^*(q_k, a)$. If

   $$\delta_N^*(q_i, a) \cup \delta_N^*(q_j, a) \cup \cdots \cup \delta_N^*(q_k, a) = \{q_l, q_m, \ldots, q_n\},$$

   create a vertex for G_D labeled $\{q_l, q_m, \ldots, q_n\}$ if it does not already exist.

   Add to G_D an edge from $\{q_i, q_j, \ldots, q_k\}$ to $\{q_l, q_m, \ldots, q_n\}$ and label it with a.
3. Every state of G_D whose label contains any $q_f \in F_N$ is identified as a final vertex.
4. If M_N accepts λ, the vertex q_0 in G_D is also made a final vertex.

Procedure *NFA-to-DFA*

1. Create a graph G_D with vertex $\{q_0\}$. Identify this vertex as the initial vertex.
2. Repeat the following steps until no more edges are missing.

   Take any vertex $\{q_i, q_j, \ldots, q_k\}$ of G_D that has no outgoing edge for some $a \in \Sigma$. Compute $\delta_N^*(q_i, a)$, $\delta_N^*(q_j, a)$, ..., $\delta_N^*(q_k, a)$. If

   $$\delta_N^*(q_i, a) \cup \delta_N^*(q_j, a) \cup \cdots \cup \delta_N^*(q_k, a) = \{q_l, q_m, \ldots, q_n\},$$

   create a vertex for G_D labeled $\{q_l, q_m, \ldots, q_n\}$ if it does not already exist.

   Add to G_D an edge from $\{q_i, q_j, \ldots, q_k\}$ to $\{q_l, q_m, \ldots, q_n\}$ and label it with a.
3. Every state of G_D whose label contains any $q_f \in F_N$ is identified as a final vertex.
4. If M_N accepts λ, the vertex q_0 in G_D is also made a final vertex.

Procedure *NFA-to-DFA*

1. Create a graph G_D with vertex $\{q_0\}$. Identify this vertex as the initial vertex.
2. Repeat the following steps until no more edges are missing.

   Take any vertex $\{q_i, q_j, \ldots, q_k\}$ of G_D that has no outgoing edge for some $a \in \Sigma$. Compute $\delta_N^*(q_i, a)$, $\delta_N^*(q_j, a)$, $\ldots$, $\delta_N^*(q_k, a)$. If

   $$\delta_N^*(q_i, a) \cup \delta_N^*(q_j, a) \cup \cdots \cup \delta_N^*(q_k, a) = \{q_l, q_m, \ldots, q_n\},$$

   create a vertex for G_D labeled $\{q_l, q_m, \ldots, q_n\}$ if it does not already exist.

   Add to G_D an edge from $\{q_i, q_j, \ldots, q_k\}$ to $\{q_l, q_m, \ldots, q_n\}$ and label it with a.
3. Every state of G_D whose label contains any $q_f \in F_N$ is identified as a final vertex.
4. If M_N accepts λ, the vertex q_0 in G_D is also made a final vertex.

Procedure *NFA-to-DFA*

1. Create a graph G_D with vertex $\{q_0\}$. Identify this vertex as the initial vertex.
2. Repeat the following steps until no more edges are missing.

   Take any vertex $\{q_i, q_j, \ldots, q_k\}$ of G_D that has no outgoing edge for some $a \in \Sigma$. Compute $\delta_N^*(q_i, a)$, $\delta_N^*(q_j, a)$, …, $\delta_N^*(q_k, a)$. If

   $$\delta_N^*(q_i, a) \cup \delta_N^*(q_j, a) \cup \cdots \cup \delta_N^*(q_k, a) = \{q_l, q_m, \ldots, q_n\},$$

   create a vertex for G_D labeled $\{q_l, q_m, \ldots, q_n\}$ if it does not already exist.

   Add to G_D an edge from $\{q_i, q_j, \ldots, q_k\}$ to $\{q_l, q_m, \ldots, q_n\}$ and label it with a.
3. Every state of G_D whose label contains any $q_f \in F_N$ is identified as a final vertex.
4. If M_N accepts λ, the vertex q_0 in G_D is also made a final vertex.

Procedure *NFA-to-DFA*

1. Create a graph G_D with vertex $\{q_0\}$. Identify this vertex as the initial vertex.
2. Repeat the following steps until no more edges are missing.

   Take any vertex $\{q_i, q_j, \ldots, q_k\}$ of G_D that has no outgoing edge for some $a \in \Sigma$. Compute $\delta_N^*(q_i, a)$, $\delta_N^*(q_j, a)$, …, $\delta_N^*(q_k, a)$. If

   $$\delta_N^*(q_i, a) \cup \delta_N^*(q_j, a) \cup \cdots \cup \delta_N^*(q_k, a) = \{q_l, q_m, \ldots, q_n\},$$

   create a vertex for G_D labeled $\{q_l, q_m, \ldots, q_n\}$ if it does not already exist.

   Add to G_D an edge from $\{q_i, q_j, \ldots, q_k\}$ to $\{q_l, q_m, \ldots, q_n\}$ and label it with a.
3. Every state of G_D whose label contains any $q_f \in F_N$ is identified as a final vertex.
4. If M_N accepts λ, the vertex q_0 in G_D is also made a final vertex.

Procedure *NFA-to-DFA*

1. Create a graph G_D with vertex $\{q_0\}$. Identify this vertex as the initial vertex.
2. Repeat the following steps until no more edges are missing.

   Take any vertex $\{q_i, q_j, \ldots, q_k\}$ of G_D that has no outgoing edge for some $a \in \Sigma$. Compute $\delta_N^*(q_i, a)$, $\delta_N^*(q_j, a)$, …, $\delta_N^*(q_k, a)$. If

   $$\delta_N^*(q_i, a) \cup \delta_N^*(q_j, a) \cup \cdots \cup \delta_N^*(q_k, a) = \{q_l, q_m, \ldots, q_n\},$$

   create a vertex for G_D labeled $\{q_l, q_m, \ldots, q_n\}$ if it does not already exist.

   Add to G_D an edge from $\{q_i, q_j, \ldots, q_k\}$ to $\{q_l, q_m, \ldots, q_n\}$ and label it with a.
3. Every state of G_D whose label contains any $q_f \in F_N$ is identified as a final vertex.
4. If M_N accepts λ, the vertex q_0 in G_D is also made a final vertex.

Procedure *NFA-to-DFA*

1. Create a graph G_D with vertex $\{q_0\}$. Identify this vertex as the initial vertex.
2. Repeat the following steps until no more edges are missing.

   Take any vertex $\{q_i, q_j, \ldots, q_k\}$ of G_D that has no outgoing edge for some $a \in \Sigma$. Compute $\delta_N^*(q_i, a)$, $\delta_N^*(q_j, a)$, …, $\delta_N^*(q_k, a)$. If

   $$\delta_N^*(q_i, a) \cup \delta_N^*(q_j, a) \cup \cdots \cup \delta_N^*(q_k, a) = \{q_l, q_m, \ldots, q_n\},$$

   create a vertex for G_D labeled $\{q_l, q_m, \ldots, q_n\}$ if it does not already exist.

   Add to G_D an edge from $\{q_i, q_j, \ldots, q_k\}$ to $\{q_l, q_m, \ldots, q_n\}$ and label it with a.
3. Every state of G_D whose label contains any $q_f \in F_N$ is identified as a final vertex.
4. If M_N accepts λ, the vertex q_0 in G_D is also made a final vertex.

Procedure *NFA-to-DFA*

1. Create a graph G_D with vertex $\{q_0\}$. Identify this vertex as the initial vertex.
2. Repeat the following steps until no more edges are missing.

   Take any vertex $\{q_i, q_j, \dots, q_k\}$ of G_D that has no outgoing edge for some $a \in \Sigma$. Compute $\delta_N^*(q_i, a)$, $\delta_N^*(q_j, a)$, ..., $\delta_N^*(q_k, a)$. If

   $$\delta_N^*(q_i, a) \cup \delta_N^*(q_j, a) \cup \dots \cup \delta_N^*(q_k, a) = \{q_l, q_m, \dots, q_n\},$$

   create a vertex for G_D labeled $\{q_l, q_m, \dots, q_n\}$ if it does not already exist.

   Add to G_D an edge from $\{q_i, q_j, \dots, q_k\}$ to $\{q_l, q_m, \dots, q_n\}$ and label it with a.
3. Every state of G_D whose label contains any $q_f \in F_N$ is identified as a final vertex.
4. If M_N accepts λ, the vertex q_0 in G_D is also made a final vertex.

Procedure *NFA-to-DFA*

1. Create a graph G_D with vertex $\{q_0\}$. Identify this vertex as the initial vertex.
2. Repeat the following steps until no more edges are missing.

   Take any vertex $\{q_i, q_j, \ldots, q_k\}$ of G_D that has no outgoing edge for some $a \in \Sigma$. Compute $\delta_N^*(q_i, a)$, $\delta_N^*(q_j, a)$, ..., $\delta_N^*(q_k, a)$. If

   $$\delta_N^*(q_i, a) \cup \delta_N^*(q_j, a) \cup \cdots \cup \delta_N^*(q_k, a) = \{q_l, q_m, \ldots, q_n\},$$

   create a vertex for G_D labeled $\{q_l, q_m, \ldots, q_n\}$ if it does not already exist.

   Add to G_D an edge from $\{q_i, q_j, \ldots, q_k\}$ to $\{q_l, q_m, \ldots, q_n\}$ and label it with a.
3. Every state of G_D whose label contains any $q_f \in F_N$ is identified as a final vertex.
4. If M_N accepts λ, the vertex q_0 in G_D is also made a final vertex.

Procedure *NFA-to-DFA*

1. Create a graph G_D with vertex $\{q_0\}$. Identify this vertex as the initial vertex.
2. Repeat the following steps until no more edges are missing.

   Take any vertex $\{q_i, q_j, \ldots, q_k\}$ of G_D that has no outgoing edge for some $a \in \Sigma$. Compute $\delta_N^*(q_i, a)$, $\delta_N^*(q_j, a)$, …, $\delta_N^*(q_k, a)$. If

   $$\delta_N^*(q_i, a) \cup \delta_N^*(q_j, a) \cup \cdots \cup \delta_N^*(q_k, a) = \{q_l, q_m, \ldots, q_n\},$$

   create a vertex for G_D labeled $\{q_l, q_m, \ldots, q_n\}$ if it does not already exist.

   Add to G_D an edge from $\{q_i, q_j, \ldots, q_k\}$ to $\{q_l, q_m, \ldots, q_n\}$ and label it with a.
3. Every state of G_D whose label contains any $q_f \in F_N$ is identified as a final vertex.
4. If M_N accepts λ, the vertex q_0 in G_D is also made a final vertex.

Procedure *NFA-to-DFA*

1. Create a graph G_D with vertex $\{q_0\}$. Identify this vertex as the initial vertex.
2. Repeat the following steps until no more edges are missing.

   Take any vertex $\{q_i, q_j, \ldots, q_k\}$ of G_D that has no outgoing edge for some $a \in \Sigma$. Compute $\delta_N^*(q_i, a)$, $\delta_N^*(q_j, a)$, ..., $\delta_N^*(q_k, a)$. If

   $$\delta_N^*(q_i, a) \cup \delta_N^*(q_j, a) \cup \cdots \cup \delta_N^*(q_k, a) = \{q_l, q_m, \ldots, q_n\},$$

   create a vertex for G_D labeled $\{q_l, q_m, \ldots, q_n\}$ if it does not already exist.

   Add to G_D an edge from $\{q_i, q_j, \ldots, q_k\}$ to $\{q_l, q_m, \ldots, q_n\}$ and label it with a.
3. Every state of G_D whose label contains any $q_f \in F_N$ is identified as a final vertex.
4. If M_N accepts λ, the vertex q_0 in G_D is also made a final vertex.

It is clear that this procedure always terminates. Each pass through the loop in Step 2 adds an edge to G_D. But G_D has at most $2^{|Q_N|}|\Sigma|$ edges, so that the loop eventually stops. To show that the construction also gives the correct answer, we argue by induction on the length of the input string.

Assume that for every v of length less than or equal to n, the presence in G_N of a walk labeled by v from q_0 to q_i implies that in G_D there is a walk labeled by v from $\{q_0\}$ to a state $Q_i = \{\ldots, q_i, \ldots\}$. Consider now any $w = va$ and look at a walk in G_N labeled by w from q_0 to q_l. There must then be a walk labeled by v from q_0 to q_i and an edge (or a sequence of edges) labeled by a from q_i to q_l. By the inductive assumption, in G_D there will be a walk labeled v from $\{q_0\}$ to Q_i. But by construction, there will be an edge from Q_i to some state whose label contains q_l. Thus, the inductive assumption holds for all strings of length $n+1$. As it is obviously true for $n = 1$, it is true for all n. The result then is that whenever $\delta_N^*(q_0, w)$ contains a final state q_f, so does the label of $\delta_D^*(q_0, w)$. To complete the proof, we reverse the argument to show that if the label of $\delta_D^*(q_0, w)$ contains q_f, so must $\delta_N^*(q_0, w)$. ■

The arguments in this proof, although correct, are admittedly somewhat terse, showing only the major steps. We shall follow this practice in the rest of the course of lectures, emphasizing the basic ideas in a proof and omitting minor details, which you may want to fill in yourself.

It is clear that this procedure always terminates. Each pass through the loop in Step 2 adds an edge to G_D. But G_D has at most $2^{|Q_N|}|\Sigma|$ edges, so that the loop eventually stops. To show that the construction also gives the correct answer, we argue by induction on the length of the input string.

Assume that for every v of length less than or equal to n, the presence in G_N of a walk labeled by v from q_0 to q_i implies that in G_D there is a walk labeled by v from $\{q_0\}$ to a state $Q_i = \{\ldots, q_i, \ldots\}$. Consider now any $w = va$ and look at a walk in G_N labeled by w from q_0 to q_l. There must then be a walk labeled by v from q_0 to q_i and an edge (or a sequence of edges) labeled by a from q_i to q_l. By the inductive assumption, in G_D there will be a walk labeled v from $\{q_0\}$ to Q_i. But by construction, there will be an edge from Q_i to some state whose label contains q_l. Thus, the inductive assumption holds for all strings of length $n+1$. As it is obviously true for $n = 1$, it is true for all n. The result then is that whenever $\delta_N^*(q_0, w)$ contains a final state q_f, so does the label of $\delta_D^*(q_0, w)$. To complete the proof, we reverse the argument to show that if the label of $\delta_D^*(q_0, w)$ contains q_f, so must $\delta_N^*(q_0, w)$. ■

The arguments in this proof, although correct, are admittedly somewhat terse, showing only the major steps. We shall follow this practice in the rest of the course of lectures, emphasizing the basic ideas in a proof and omitting minor details, which you may want to fill in yourself.

It is clear that this procedure always terminates. Each pass through the loop in Step 2 adds an edge to G_D. But G_D has at most $2^{|Q_N|}|\Sigma|$ edges, so that the loop eventually stops. To show that the construction also gives the correct answer, we argue by induction on the length of the input string.

Assume that for every v of length less than or equal to n, the presence in G_N of a walk labeled by v from q_0 to q_i implies that in G_D there is a walk labeled by v from $\{q_0\}$ to a state $Q_i = \{\ldots, q_i, \ldots\}$. Consider now any $w = va$ and look at a walk in G_N labeled by w from q_0 to q_l. There must then be a walk labeled by v from q_0 to q_i and an edge (or a sequence of edges) labeled by a from q_i to q_l. By the inductive assumption, in G_D there will be a walk labeled v from $\{q_0\}$ to Q_i. But by construction, there will be an edge from Q_i to some state whose label contains q_l. Thus, the inductive assumption holds for all strings of length $n + 1$. As it is obviously true for $n = 1$, it is true for all n. The result then is that whenever $\delta_N^*(q_0, w)$ contains a final state q_f, so does the label of $\delta_D^*(q_0, w)$. To complete the proof, we reverse the argument to show that if the label of $\delta_D^*(q_0, w)$ contains q_f, so must $\delta_N^*(q_0, w)$. ■

The arguments in this proof, although correct, are admittedly somewhat terse, showing only the major steps. We shall follow this practice in the rest of the course of lectures, emphasizing the basic ideas in a proof and omitting minor details, which you may want to fill in yourself.

It is clear that this procedure always terminates. Each pass through the loop in Step 2 adds an edge to G_D. But G_D has at most $2^{|Q_N|}|\Sigma|$ edges, so that the loop eventually stops. To show that the construction also gives the correct answer, we argue by induction on the length of the input string.

Assume that for every v of length less than or equal to n, the presence in G_N of a walk labeled by v from q_0 to q_i implies that in G_D there is a walk labeled by v from $\{q_0\}$ to a state $Q_i = \{\ldots, q_i, \ldots\}$. Consider now any $w = va$ and look at a walk in G_N labeled by w from q_0 to q_l. There must then be a walk labeled by v from q_0 to q_i and an edge (or a sequence of edges) labeled by a from q_i to q_l. By the inductive assumption, in G_D there will be a walk labeled v from $\{q_0\}$ to Q_i. But by construction, there will be an edge from Q_i to some state whose label contains q_l. Thus, the inductive assumption holds for all strings of length $n+1$. As it is obviously true for $n = 1$, it is true for all n. The result then is that whenever $\delta_N^*(q_0, w)$ contains a final state q_f, so does the label of $\delta_D^*(q_0, w)$. To complete the proof, we reverse the argument to show that if the label of $\delta_D^*(q_0, w)$ contains q_f, so must $\delta_N^*(q_0, w)$. ■

The arguments in this proof, although correct, are admittedly somewhat terse, showing only the major steps. We shall follow this practice in the rest of the course of lectures, emphasizing the basic ideas in a proof and omitting minor details, which you may want to fill in yourself.

It is clear that this procedure always terminates. Each pass through the loop in Step 2 adds an edge to G_D. But G_D has at most $2^{|Q_N|}|\Sigma|$ edges, so that the loop eventually stops. To show that the construction also gives the correct answer, we argue by induction on the length of the input string.

Assume that for every v of length less than or equal to n, the presence in G_N of a walk labeled by v from q_0 to q_i implies that in G_D there is a walk labeled by v from $\{q_0\}$ to a state $Q_i = \{\ldots, q_i, \ldots\}$. Consider now any $w = va$ and look at a walk in G_N labeled by w from q_0 to q_l. There must then be a walk labeled by v from q_0 to q_i and an edge (or a sequence of edges) labeled by a from q_i to q_l. By the inductive assumption, in G_D there will be a walk labeled v from $\{q_0\}$ to Q_i. But by construction, there will be an edge from Q_i to some state whose label contains q_l. Thus, the inductive assumption holds for all strings of length $n+1$. As it is obviously true for $n = 1$, it is true for all n. The result then is that whenever $\delta_N^*(q_0, w)$ contains a final state q_f, so does the label of $\delta_D^*(q_0, w)$. To complete the proof, we reverse the argument to show that if the label of $\delta_D^*(q_0, w)$ contains q_f, so must $\delta_N^*(q_0, w)$. ■

The arguments in this proof, although correct, are admittedly somewhat terse, showing only the major steps. We shall follow this practice in the rest of the course of lectures, emphasizing the basic ideas in a proof and omitting minor details, which you may want to fill in yourself.

It is clear that this procedure always terminates. Each pass through the loop in Step 2 adds an edge to G_D. But G_D has at most $2^{|Q_N|}|\Sigma|$ edges, so that the loop eventually stops. To show that the construction also gives the correct answer, we argue by induction on the length of the input string.

Assume that for every v of length less than or equal to n, the presence in G_N of a walk labeled by v from q_0 to q_i implies that in G_D there is a walk labeled by v from $\{q_0\}$ to a state $Q_i = \{\ldots, q_i, \ldots\}$. Consider now any $w = va$ and look at a walk in G_N labeled by w from q_0 to q_l. There must then be a walk labeled by v from q_0 to q_i and an edge (or a sequence of edges) labeled by a from q_i to q_l. By the inductive assumption, in G_D there will be a walk labeled v from $\{q_0\}$ to Q_i. But by construction, there will be an edge from Q_i to some state whose label contains q_l. Thus, the inductive assumption holds for all strings of length $n+1$. As it is obviously true for $n = 1$, it is true for all n. The result then is that whenever $\delta_N^*(q_0, w)$ contains a final state q_f, so does the label of $\delta_D^*(q_0, w)$. To complete the proof, we reverse the argument to show that if the label of $\delta_D^*(q_0, w)$ contains q_f, so must $\delta_N^*(q_0, w)$. ∎

The arguments in this proof, although correct, are admittedly somewhat terse, showing only the major steps. We shall follow this practice in the rest of the course of lectures, emphasizing the basic ideas in a proof and omitting minor details, which you may want to fill in yourself.

It is clear that this procedure always terminates. Each pass through the loop in Step 2 adds an edge to G_D. But G_D has at most $2^{|Q_N|}|\Sigma|$ edges, so that the loop eventually stops. To show that the construction also gives the correct answer, we argue by induction on the length of the input string.

Assume that for every v of length less than or equal to n, the presence in G_N of a walk labeled by v from q_0 to q_i implies that in G_D there is a walk labeled by v from $\{q_0\}$ to a state $Q_i = \{\ldots, q_i, \ldots\}$. Consider now any $w = va$ and look at a walk in G_N labeled by w from q_0 to q_l. There must then be a walk labeled by v from q_0 to q_i and an edge (or a sequence of edges) labeled by a from q_i to q_l. By the inductive assumption, in G_D there will be a walk labeled v from $\{q_0\}$ to Q_i. But by construction, there will be an edge from Q_i to some state whose label contains q_l. Thus, the inductive assumption holds for all strings of length $n+1$. As it is obviously true for $n = 1$, it is true for all n. The result then is that whenever $\delta_N^*(q_0, w)$ contains a final state q_f, so does the label of $\delta_D^*(q_0, w)$. To complete the proof, we reverse the argument to show that if the label of $\delta_D^*(q_0, w)$ contains q_f, so must $\delta_N^*(q_0, w)$. ■

The arguments in this proof, although correct, are admittedly somewhat terse, showing only the major steps. We shall follow this practice in the rest of the course of lectures, emphasizing the basic ideas in a proof and omitting minor details, which you may want to fill in yourself.

It is clear that this procedure always terminates. Each pass through the loop in Step 2 adds an edge to G_D. But G_D has at most $2^{|Q_N|}|\Sigma|$ edges, so that the loop eventually stops. To show that the construction also gives the correct answer, we argue by induction on the length of the input string.

Assume that for every v of length less than or equal to n, the presence in G_N of a walk labeled by v from q_0 to q_i implies that in G_D there is a walk labeled by v from $\{q_0\}$ to a state $Q_i = \{\ldots, q_i, \ldots\}$. Consider now any $w = va$ and look at a walk in G_N labeled by w from q_0 to q_l. There must then be a walk labeled by v from q_0 to q_i and an edge (or a sequence of edges) labeled by a from q_i to q_l. By the inductive assumption, in G_D there will be a walk labeled v from $\{q_0\}$ to Q_i. But by construction, there will be an edge from Q_i to some state whose label contains q_l. Thus, the inductive assumption holds for all strings of length $n+1$. As it is obviously true for $n = 1$, it is true for all n. The result then is that whenever $\delta_N^*(q_0, w)$ contains a final state q_f, so does the label of $\delta_D^*(q_0, w)$. To complete the proof, we reverse the argument to show that if the label of $\delta_D^*(q_0, w)$ contains q_f, so must $\delta_N^*(q_0, w)$. ■

The arguments in this proof, although correct, are admittedly somewhat terse, showing only the major steps. We shall follow this practice in the rest of the course of lectures, emphasizing the basic ideas in a proof and omitting minor details, which you may want to fill in yourself.

It is clear that this procedure always terminates. Each pass through the loop in Step 2 adds an edge to G_D. But G_D has at most $2^{|Q_N|}|\Sigma|$ edges, so that the loop eventually stops. To show that the construction also gives the correct answer, we argue by induction on the length of the input string.

Assume that for every v of length less than or equal to n, the presence in G_N of a walk labeled by v from q_0 to q_i implies that in G_D there is a walk labeled by v from $\{q_0\}$ to a state $Q_i = \{\ldots, q_i, \ldots\}$. Consider now any $w = va$ and look at a walk in G_N labeled by w from q_0 to q_l. There must then be a walk labeled by v from q_0 to q_i and an edge (or a sequence of edges) labeled by a from q_i to q_l. By the inductive assumption, in G_D there will be a walk labeled v from $\{q_0\}$ to Q_i. But by construction, there will be an edge from Q_i to some state whose label contains q_l. Thus, the inductive assumption holds for all strings of length $n+1$. As it is obviously true for $n = 1$, it is true for all n. The result then is that whenever $\delta_N^*(q_0, w)$ contains a final state q_f, so does the label of $\delta_D^*(q_0, w)$. To complete the proof, we reverse the argument to show that if the label of $\delta_D^*(q_0, w)$ contains q_f, so must $\delta_N^*(q_0, w)$. ■

The arguments in this proof, although correct, are admittedly somewhat terse, showing only the major steps. We shall follow this practice in the rest of the course of lectures, emphasizing the basic ideas in a proof and omitting minor details, which you may want to fill in yourself.

It is clear that this procedure always terminates. Each pass through the loop in Step 2 adds an edge to G_D. But G_D has at most $2^{|Q_N|}|\Sigma|$ edges, so that the loop eventually stops. To show that the construction also gives the correct answer, we argue by induction on the length of the input string.

Assume that for every v of length less than or equal to n, the presence in G_N of a walk labeled by v from q_0 to q_i implies that in G_D there is a walk labeled by v from $\{q_0\}$ to a state $Q_i = \{\ldots, q_i, \ldots\}$. Consider now any $w = va$ and look at a walk in G_N labeled by w from q_0 to q_l. There must then be a walk labeled by v from q_0 to q_i and an edge (or a sequence of edges) labeled by a from q_i to q_l. By the inductive assumption, in G_D there will be a walk labeled v from $\{q_0\}$ to Q_i. But by construction, there will be an edge from Q_i to some state whose label contains q_l. Thus, the inductive assumption holds for all strings of length $n+1$. As it is obviously true for $n = 1$, it is true for all n. The result then is that whenever $\delta_N^*(q_0, w)$ contains a final state q_f, so does the label of $\delta_D^*(q_0, w)$. To complete the proof, we reverse the argument to show that if the label of $\delta_D^*(q_0, w)$ contains q_f, so must $\delta_N^*(q_0, w)$. ■

The arguments in this proof, although correct, are admittedly somewhat terse, showing only the major steps. We shall follow this practice in the rest of the course of lectures, emphasizing the basic ideas in a proof and omitting minor details, which you may want to fill in yourself.

It is clear that this procedure always terminates. Each pass through the loop in Step 2 adds an edge to G_D. But G_D has at most $2^{|Q_N|}|\Sigma|$ edges, so that the loop eventually stops. To show that the construction also gives the correct answer, we argue by induction on the length of the input string.

Assume that for every v of length less than or equal to n, the presence in G_N of a walk labeled by v from q_0 to q_i implies that in G_D there is a walk labeled by v from $\{q_0\}$ to a state $Q_i = \{\ldots, q_i, \ldots\}$. Consider now any $w = va$ and look at a walk in G_N labeled by w from q_0 to q_l. There must then be a walk labeled by v from q_0 to q_i and an edge (or a sequence of edges) labeled by a from q_i to q_l. By the inductive assumption, in G_D there will be a walk labeled v from $\{q_0\}$ to Q_i. But by construction, there will be an edge from Q_i to some state whose label contains q_l. Thus, the inductive assumption holds for all strings of length $n+1$. As it is obviously true for $n = 1$, it is true for all n. The result then is that whenever $\delta_N^*(q_0, w)$ contains a final state q_f, so does the label of $\delta_D^*(q_0, w)$. To complete the proof, we reverse the argument to show that if the label of $\delta_D^*(q_0, w)$ contains q_f, so must $\delta_N^*(q_0, w)$. ∎

The arguments in this proof, although correct, are admittedly somewhat terse, showing only the major steps. We shall follow this practice in the rest of the course of lectures, emphasizing the basic ideas in a proof and omitting minor details, which you may want to fill in yourself.

It is clear that this procedure always terminates. Each pass through the loop in Step 2 adds an edge to G_D. But G_D has at most $2^{|Q_N|}|\Sigma|$ edges, so that the loop eventually stops. To show that the construction also gives the correct answer, we argue by induction on the length of the input string.

Assume that for every v of length less than or equal to n, the presence in G_N of a walk labeled by v from q_0 to q_i implies that in G_D there is a walk labeled by v from $\{q_0\}$ to a state $Q_i = \{\ldots, q_i, \ldots\}$. Consider now any $w = va$ and look at a walk in G_N labeled by w from q_0 to q_l. There must then be a walk labeled by v from q_0 to q_i and an edge (or a sequence of edges) labeled by a from q_i to q_l. By the inductive assumption, in G_D there will be a walk labeled v from $\{q_0\}$ to Q_i. But by construction, there will be an edge from Q_i to some state whose label contains q_l. Thus, the inductive assumption holds for all strings of length $n+1$. As it is obviously true for $n = 1$, it is true for all n. The result then is that whenever $\delta_N^*(q_0, w)$ contains a final state q_f, so does the label of $\delta_D^*(q_0, w)$. To complete the proof, we reverse the argument to show that if the label of $\delta_D^*(q_0, w)$ contains q_f, so must $\delta_N^*(q_0, w)$. ■

The arguments in this proof, although correct, are admittedly somewhat terse, showing only the major steps. We shall follow this practice in the rest of the course of lectures, emphasizing the basic ideas in a proof and omitting minor details, which you may want to fill in yourself.

It is clear that this procedure always terminates. Each pass through the loop in Step 2 adds an edge to G_D. But G_D has at most $2^{|Q_N|}|\Sigma|$ edges, so that the loop eventually stops. To show that the construction also gives the correct answer, we argue by induction on the length of the input string.

Assume that for every v of length less than or equal to n, the presence in G_N of a walk labeled by v from q_0 to q_i implies that in G_D there is a walk labeled by v from $\{q_0\}$ to a state $Q_i = \{\ldots, q_i, \ldots\}$. Consider now any $w = va$ and look at a walk in G_N labeled by w from q_0 to q_l. There must then be a walk labeled by v from q_0 to q_i and an edge (or a sequence of edges) labeled by a from q_i to q_l. By the inductive assumption, in G_D there will be a walk labeled v from $\{q_0\}$ to Q_i. But by construction, there will be an edge from Q_i to some state whose label contains q_l. Thus, the inductive assumption holds for all strings of length $n+1$. As it is obviously true for $n = 1$, it is true for all n. The result then is that whenever $\delta_N^*(q_0, w)$ contains a final state q_f, so does the label of $\delta_D^*(q_0, w)$. To complete the proof, we reverse the argument to show that if the label of $\delta_D^*(q_0, w)$ contains q_f, so must $\delta_N^*(q_0, w)$. ■

The arguments in this proof, although correct, are admittedly somewhat terse, showing only the major steps. We shall follow this practice in the rest of the course of lectures, emphasizing the basic ideas in a proof and omitting minor details, which you may want to fill in yourself.

It is clear that this procedure always terminates. Each pass through the loop in Step 2 adds an edge to G_D. But G_D has at most $2^{|Q_N|}|\Sigma|$ edges, so that the loop eventually stops. To show that the construction also gives the correct answer, we argue by induction on the length of the input string.

Assume that for every v of length less than or equal to n, the presence in G_N of a walk labeled by v from q_0 to q_i implies that in G_D there is a walk labeled by v from $\{q_0\}$ to a state $Q_i = \{\ldots, q_i, \ldots\}$. Consider now any $w = va$ and look at a walk in G_N labeled by w from q_0 to q_l. There must then be a walk labeled by v from q_0 to q_i and an edge (or a sequence of edges) labeled by a from q_i to q_l. By the inductive assumption, in G_D there will be a walk labeled v from $\{q_0\}$ to Q_i. But by construction, there will be an edge from Q_i to some state whose label contains q_l. Thus, the inductive assumption holds for all strings of length $n+1$. As it is obviously true for $n = 1$, it is true for all n. The result then is that whenever $\delta_N^*(q_0, w)$ contains a final state q_f, so does the label of $\delta_D^*(q_0, w)$. To complete the proof, we reverse the argument to show that if the label of $\delta_D^*(q_0, w)$ contains q_f, so must $\delta_N^*(q_0, w)$. ■

The arguments in this proof, although correct, are admittedly somewhat terse, showing only the major steps. We shall follow this practice in the rest of the course of lectures, emphasizing the basic ideas in a proof and omitting minor details, which you may want to fill in yourself.

It is clear that this procedure always terminates. Each pass through the loop in Step 2 adds an edge to G_D. But G_D has at most $2^{|Q_N|}|\Sigma|$ edges, so that the loop eventually stops. To show that the construction also gives the correct answer, we argue by induction on the length of the input string.

Assume that for every v of length less than or equal to n, the presence in G_N of a walk labeled by v from q_0 to q_i implies that in G_D there is a walk labeled by v from $\{q_0\}$ to a state $Q_i = \{\ldots, q_i, \ldots\}$. Consider now any $w = va$ and look at a walk in G_N labeled by w from q_0 to q_l. There must then be a walk labeled by v from q_0 to q_i and an edge (or a sequence of edges) labeled by a from q_i to q_l. By the inductive assumption, in G_D there will be a walk labeled v from $\{q_0\}$ to Q_i. But by construction, there will be an edge from Q_i to some state whose label contains q_l. Thus, the inductive assumption holds for all strings of length $n+1$. As it is obviously true for $n = 1$, it is true for all n. The result then is that whenever $\delta_N^*(q_0, w)$ contains a final state q_f, so does the label of $\delta_D^*(q_0, w)$. To complete the proof, we reverse the argument to show that if the label of $\delta_D^*(q_0, w)$ contains q_f, so must $\delta_N^*(q_0, w)$. ■

The arguments in this proof, although correct, are admittedly somewhat terse, showing only the major steps. We shall follow this practice in the rest of the course of lectures, emphasizing the basic ideas in a proof and omitting minor details, which you may want to fill in yourself.

It is clear that this procedure always terminates. Each pass through the loop in Step 2 adds an edge to G_D. But G_D has at most $2^{|Q_N|}|\Sigma|$ edges, so that the loop eventually stops. To show that the construction also gives the correct answer, we argue by induction on the length of the input string.

Assume that for every v of length less than or equal to n, the presence in G_N of a walk labeled by v from q_0 to q_i implies that in G_D there is a walk labeled by v from $\{q_0\}$ to a state $Q_i = \{\ldots, q_i, \ldots\}$. Consider now any $w = va$ and look at a walk in G_N labeled by w from q_0 to q_l. There must then be a walk labeled by v from q_0 to q_i and an edge (or a sequence of edges) labeled by a from q_i to q_l. By the inductive assumption, in G_D there will be a walk labeled v from $\{q_0\}$ to Q_i. But by construction, there will be an edge from Q_i to some state whose label contains q_l. Thus, the inductive assumption holds for all strings of length $n+1$. As it is obviously true for $n = 1$, it is true for all n. The result then is that whenever $\delta_N^*(q_0, w)$ contains a final state q_f, so does the label of $\delta_D^*(q_0, w)$. To complete the proof, we reverse the argument to show that if the label of $\delta_D^*(q_0, w)$ contains q_f, so must $\delta_N^*(q_0, w)$. ∎

The arguments in this proof, although correct, are admittedly somewhat terse, showing only the major steps. We shall follow this practice in the rest of the course of lectures, emphasizing the basic ideas in a proof and omitting minor details, which you may want to fill in yourself.

It is clear that this procedure always terminates. Each pass through the loop in Step 2 adds an edge to G_D. But G_D has at most $2^{|Q_N|}|\Sigma|$ edges, so that the loop eventually stops. To show that the construction also gives the correct answer, we argue by induction on the length of the input string.

Assume that for every v of length less than or equal to n, the presence in G_N of a walk labeled by v from q_0 to q_i implies that in G_D there is a walk labeled by v from $\{q_0\}$ to a state $Q_i = \{\ldots, q_i, \ldots\}$. Consider now any $w = va$ and look at a walk in G_N labeled by w from q_0 to q_l. There must then be a walk labeled by v from q_0 to q_i and an edge (or a sequence of edges) labeled by a from q_i to q_l. By the inductive assumption, in G_D there will be a walk labeled v from $\{q_0\}$ to Q_i. But by construction, there will be an edge from Q_i to some state whose label contains q_l. Thus, the inductive assumption holds for all strings of length $n+1$. As it is obviously true for $n = 1$, it is true for all n. The result then is that whenever $\delta_N^*(q_0, w)$ contains a final state q_f, so does the label of $\delta_D^*(q_0, w)$. To complete the proof, we reverse the argument to show that if the label of $\delta_D^*(q_0, w)$ contains q_f, so must $\delta_N^*(q_0, w)$. ■

The arguments in this proof, although correct, are admittedly somewhat terse, showing only the major steps. We shall follow this practice in the rest of the course of lectures, emphasizing the basic ideas in a proof and omitting minor details, which you may want to fill in yourself.

It is clear that this procedure always terminates. Each pass through the loop in Step 2 adds an edge to G_D. But G_D has at most $2^{|Q_N|}|\Sigma|$ edges, so that the loop eventually stops. To show that the construction also gives the correct answer, we argue by induction on the length of the input string.

Assume that for every v of length less than or equal to n, the presence in G_N of a walk labeled by v from q_0 to q_i implies that in G_D there is a walk labeled by v from $\{q_0\}$ to a state $Q_i = \{\ldots, q_i, \ldots\}$. Consider now any $w = va$ and look at a walk in G_N labeled by w from q_0 to q_l. There must then be a walk labeled by v from q_0 to q_i and an edge (or a sequence of edges) labeled by a from q_i to q_l. By the inductive assumption, in G_D there will be a walk labeled v from $\{q_0\}$ to Q_i. But by construction, there will be an edge from Q_i to some state whose label contains q_l. Thus, the inductive assumption holds for all strings of length $n+1$. As it is obviously true for $n = 1$, it is true for all n. The result then is that whenever $\delta_N^*(q_0, w)$ contains a final state q_f, so does the label of $\delta_D^*(q_0, w)$. To complete the proof, we reverse the argument to show that if the label of $\delta_D^*(q_0, w)$ contains q_f, so must $\delta_N^*(q_0, w)$. ■

The arguments in this proof, although correct, are admittedly somewhat terse, showing only the major steps. We shall follow this practice in the rest of the course of lectures, emphasizing the basic ideas in a proof and omitting minor details, which you may want to fill in yourself.

It is clear that this procedure always terminates. Each pass through the loop in Step 2 adds an edge to G_D. But G_D has at most $2^{|Q_N|}|\Sigma|$ edges, so that the loop eventually stops. To show that the construction also gives the correct answer, we argue by induction on the length of the input string.

Assume that for every v of length less than or equal to n, the presence in G_N of a walk labeled by v from q_0 to q_i implies that in G_D there is a walk labeled by v from $\{q_0\}$ to a state $Q_i = \{\ldots, q_i, \ldots\}$. Consider now any $w = va$ and look at a walk in G_N labeled by w from q_0 to q_l. There must then be a walk labeled by v from q_0 to q_i and an edge (or a sequence of edges) labeled by a from q_i to q_l. By the inductive assumption, in G_D there will be a walk labeled v from $\{q_0\}$ to Q_i. But by construction, there will be an edge from Q_i to some state whose label contains q_l. Thus, the inductive assumption holds for all strings of length $n+1$. As it is obviously true for $n = 1$, it is true for all n. The result then is that whenever $\delta_N^*(q_0, w)$ contains a final state q_f, so does the label of $\delta_D^*(q_0, w)$. To complete the proof, we reverse the argument to show that if the label of $\delta_D^*(q_0, w)$ contains q_f, so must $\delta_N^*(q_0, w)$. ■

The arguments in this proof, although correct, are admittedly somewhat terse, showing only the major steps. We shall follow this practice in the rest of the course of lectures, emphasizing the basic ideas in a proof and omitting minor details, which you may want to fill in yourself.

It is clear that this procedure always terminates. Each pass through the loop in Step 2 adds an edge to G_D. But G_D has at most $2^{|Q_N|}|\Sigma|$ edges, so that the loop eventually stops. To show that the construction also gives the correct answer, we argue by induction on the length of the input string.

Assume that for every v of length less than or equal to n, the presence in G_N of a walk labeled by v from q_0 to q_i implies that in G_D there is a walk labeled by v from $\{q_0\}$ to a state $Q_i = \{\ldots, q_i, \ldots\}$. Consider now any $w = va$ and look at a walk in G_N labeled by w from q_0 to q_l. There must then be a walk labeled by v from q_0 to q_i and an edge (or a sequence of edges) labeled by a from q_i to q_l. By the inductive assumption, in G_D there will be a walk labeled v from $\{q_0\}$ to Q_i. But by construction, there will be an edge from Q_i to some state whose label contains q_l. Thus, the inductive assumption holds for all strings of length $n+1$. As it is obviously true for $n = 1$, it is true for all n. The result then is that whenever $\delta_N^*(q_0, w)$ contains a final state q_f, so does the label of $\delta_D^*(q_0, w)$. To complete the proof, we reverse the argument to show that if the label of $\delta_D^*(q_0, w)$ contains q_f, so must $\delta_N^*(q_0, w)$. ■

The arguments in this proof, although correct, are admittedly somewhat terse, showing only the major steps. We shall follow this practice in the rest of the course of lectures, emphasizing the basic ideas in a proof and omitting minor details, which you may want to fill in yourself.

It is clear that this procedure always terminates. Each pass through the loop in Step 2 adds an edge to G_D. But G_D has at most $2^{|Q_N|}|\Sigma|$ edges, so that the loop eventually stops. To show that the construction also gives the correct answer, we argue by induction on the length of the input string.

Assume that for every v of length less than or equal to n, the presence in G_N of a walk labeled by v from q_0 to q_i implies that in G_D there is a walk labeled by v from $\{q_0\}$ to a state $Q_i = \{\ldots, q_i, \ldots\}$. Consider now any $w = va$ and look at a walk in G_N labeled by w from q_0 to q_l. There must then be a walk labeled by v from q_0 to q_i and an edge (or a sequence of edges) labeled by a from q_i to q_l. By the inductive assumption, in G_D there will be a walk labeled v from $\{q_0\}$ to Q_i. But by construction, there will be an edge from Q_i to some state whose label contains q_l. Thus, the inductive assumption holds for all strings of length $n+1$. As it is obviously true for $n = 1$, it is true for all n. The result then is that whenever $\delta_N^*(q_0, w)$ contains a final state q_f, so does the label of $\delta_D^*(q_0, w)$. To complete the proof, we reverse the argument to show that if the label of $\delta_D^*(q_0, w)$ contains q_f, so must $\delta_N^*(q_0, w)$. ■

The arguments in this proof, although correct, are admittedly somewhat terse, showing only the major steps. We shall follow this practice in the rest of the course of lectures, emphasizing the basic ideas in a proof and omitting minor details, which you may want to fill in yourself.

It is clear that this procedure always terminates. Each pass through the loop in Step 2 adds an edge to G_D. But G_D has at most $2^{|Q_N|}|\Sigma|$ edges, so that the loop eventually stops. To show that the construction also gives the correct answer, we argue by induction on the length of the input string.

Assume that for every v of length less than or equal to n, the presence in G_N of a walk labeled by v from q_0 to q_i implies that in G_D there is a walk labeled by v from $\{q_0\}$ to a state $Q_i = \{\ldots, q_i, \ldots\}$. Consider now any $w = va$ and look at a walk in G_N labeled by w from q_0 to q_l. There must then be a walk labeled by v from q_0 to q_i and an edge (or a sequence of edges) labeled by a from q_i to q_l. By the inductive assumption, in G_D there will be a walk labeled v from $\{q_0\}$ to Q_i. But by construction, there will be an edge from Q_i to some state whose label contains q_l. Thus, the inductive assumption holds for all strings of length $n+1$. As it is obviously true for $n=1$, it is true for all n. The result then is that whenever $\delta_N^*(q_0, w)$ contains a final state q_f, so does the label of $\delta_D^*(q_0, w)$. To complete the proof, we reverse the argument to show that if the label of $\delta_D^*(q_0, w)$ contains q_f, so must $\delta_N^*(q_0, w)$. ■

The arguments in this proof, although correct, are admittedly somewhat terse, showing only the major steps. We shall follow this practice in the rest of the course of lectures, emphasizing the basic ideas in a proof and omitting minor details, which you may want to fill in yourself.

It is clear that this procedure always terminates. Each pass through the loop in Step 2 adds an edge to G_D. But G_D has at most $2^{|Q_N|}|\Sigma|$ edges, so that the loop eventually stops. To show that the construction also gives the correct answer, we argue by induction on the length of the input string.

Assume that for every v of length less than or equal to n, the presence in G_N of a walk labeled by v from q_0 to q_i implies that in G_D there is a walk labeled by v from $\{q_0\}$ to a state $Q_i = \{\ldots, q_i, \ldots\}$. Consider now any $w = va$ and look at a walk in G_N labeled by w from q_0 to q_l. There must then be a walk labeled by v from q_0 to q_i and an edge (or a sequence of edges) labeled by a from q_i to q_l. By the inductive assumption, in G_D there will be a walk labeled v from $\{q_0\}$ to Q_i. But by construction, there will be an edge from Q_i to some state whose label contains q_l. Thus, the inductive assumption holds for all strings of length $n+1$. As it is obviously true for $n = 1$, it is true for all n. The result then is that whenever $\delta_N^*(q_0, w)$ contains a final state q_f, so does the label of $\delta_D^*(q_0, w)$. To complete the proof, we reverse the argument to show that if the label of $\delta_D^*(q_0, w)$ contains q_f, so must $\delta_N^*(q_0, w)$. ■

The arguments in this proof, although correct, are admittedly somewhat terse, showing only the major steps. We shall follow this practice in the rest of the course of lectures, emphasizing the basic ideas in a proof and omitting minor details, which you may want to fill in yourself.

It is clear that this procedure always terminates. Each pass through the loop in Step 2 adds an edge to G_D. But G_D has at most $2^{|Q_N|}|\Sigma|$ edges, so that the loop eventually stops. To show that the construction also gives the correct answer, we argue by induction on the length of the input string.

Assume that for every v of length less than or equal to n, the presence in G_N of a walk labeled by v from q_0 to q_i implies that in G_D there is a walk labeled by v from $\{q_0\}$ to a state $Q_i = \{\ldots, q_i, \ldots\}$. Consider now any $w = va$ and look at a walk in G_N labeled by w from q_0 to q_l. There must then be a walk labeled by v from q_0 to q_i and an edge (or a sequence of edges) labeled by a from q_i to q_l. By the inductive assumption, in G_D there will be a walk labeled v from $\{q_0\}$ to Q_i. But by construction, there will be an edge from Q_i to some state whose label contains q_l. Thus, the inductive assumption holds for all strings of length $n+1$. As it is obviously true for $n=1$, it is true for all n. The result then is that whenever $\delta_N^*(q_0, w)$ contains a final state q_f, so does the label of $\delta_D^*(q_0, w)$. To complete the proof, we reverse the argument to show that if the label of $\delta_D^*(q_0, w)$ contains q_f, so must $\delta_N^*(q_0, w)$. ■

The arguments in this proof, although correct, are admittedly somewhat terse, showing only the major steps. We shall follow this practice in the rest of the course of lectures, emphasizing the basic ideas in a proof and omitting minor details, which you may want to fill in yourself.

It is clear that this procedure always terminates. Each pass through the loop in Step 2 adds an edge to G_D. But G_D has at most $2^{|Q_N|}|\Sigma|$ edges, so that the loop eventually stops. To show that the construction also gives the correct answer, we argue by induction on the length of the input string.

Assume that for every v of length less than or equal to n, the presence in G_N of a walk labeled by v from q_0 to q_i implies that in G_D there is a walk labeled by v from $\{q_0\}$ to a state $Q_i = \{\ldots, q_i, \ldots\}$. Consider now any $w = va$ and look at a walk in G_N labeled by w from q_0 to q_l. There must then be a walk labeled by v from q_0 to q_i and an edge (or a sequence of edges) labeled by a from q_i to q_l. By the inductive assumption, in G_D there will be a walk labeled v from $\{q_0\}$ to Q_i. But by construction, there will be an edge from Q_i to some state whose label contains q_l. Thus, the inductive assumption holds for all strings of length $n+1$. As it is obviously true for $n = 1$, it is true for all n. The result then is that whenever $\delta_N^*(q_0, w)$ contains a final state q_f, so does the label of $\delta_D^*(q_0, w)$. To complete the proof, we reverse the argument to show that if the label of $\delta_D^*(q_0, w)$ contains q_f, so must $\delta_N^*(q_0, w)$. ■

The arguments in this proof, although correct, are admittedly somewhat terse, showing only the major steps. We shall follow this practice in the rest of the course of lectures, emphasizing the basic ideas in a proof and omitting minor details, which you may want to fill in yourself.

The construction in the previous proof is tedious but important. Let us do another example to make sure we understand all the steps.

Example 2.13

Convert the NFA in the following Figure to an equivalent DFA.

Since $\delta_N(q_0, 0) = \{q_0, q_1\}$, we introduce the state $\{q_0, q_1\}$ in G_D and add an edge labeled by 0 between $\{q_0\}$ and $\{q_0, q_1\}$. In the same way, considering $\delta_N(q_0, 1) = \{q_1\}$ gives us the new state $\{q_1\}$ and an edge labeled by 1 between it and $\{q_0\}$.

There are now a number of missing edges, so we continue, using the construction of Theorem 2.2. Looking at the state $\{q_0, q_1\}$, we see that there is no outgoing edge labeled by 0, so we compute

$$\delta_N^*(q_0, 0) \cup \delta_N^*(q_1, 0) = \{q_0, q_1, q_2\}.$$

This gives us the new state $\{q_0, q_1, q_2\}$ and the transition

$$\delta_N^*(\{q_0, q_1\}, 0) = \{q_0, q_1, q_2\}.$$

The construction in the previous proof is tedious but important. Let us do another example to make sure we understand all the steps.

Example 2.13

Convert the NFA in the following Figure to an equivalent DFA.

Since $\delta_N(q_0, 0) = \{q_0, q_1\}$, we introduce the state $\{q_0, q_1\}$ in G_D and add an edge labeled by 0 between $\{q_0\}$ and $\{q_0, q_1\}$. In the same way, considering $\delta_N(q_0, 1) = \{q_1\}$ gives us the new state $\{q_1\}$ and an edge labeled by 1 between it and $\{q_0\}$.

There are now a number of missing edges, so we continue, using the construction of Theorem 2.2. Looking at the state $\{q_0, q_1\}$, we see that there is no outgoing edge labeled by 0, so we compute

$$\delta_N^*(q_0, 0) \cup \delta_N^*(q_1, 0) = \{q_0, q_1, q_2\}.$$

This gives us the new state $\{q_0, q_1, q_2\}$ and the transition

$$\delta_N^*(\{q_0, q_1\}, 0) = \{q_0, q_1, q_2\}.$$

The construction in the previous proof is tedious but important. Let us do another example to make sure we understand all the steps.

Example 2.13

Convert the NFA in the following Figure to an equivalent DFA.

Since $\delta_N(q_0, 0) = \{q_0, q_1\}$, we introduce the state $\{q_0, q_1\}$ in G_D and add an edge labeled by 0 between $\{q_0\}$ and $\{q_0, q_1\}$. In the same way, considering $\delta_N(q_0, 1) = \{q_1\}$ gives us the new state $\{q_1\}$ and an edge labeled by 1 between it and $\{q_0\}$.

There are now a number of missing edges, so we continue, using the construction of Theorem 2.2. Looking at the state $\{q_0, q_1\}$, we see that there is no outgoing edge labeled by 0, so we compute

$$\delta_N^*(q_0, 0) \cup \delta_N^*(q_1, 0) = \{q_0, q_1, q_2\}.$$

This gives us the new state $\{q_0, q_1, q_2\}$ and the transition

$$\delta_N^*(\{q_0, q_1\}, 0) = \{q_0, q_1, q_2\}.$$

The construction in the previous proof is tedious but important. Let us do another example to make sure we understand all the steps.

Example 2.13

Convert the NFA in the following Figure to an equivalent DFA.

Since $\delta_N(q_0, 0) = \{q_0, q_1\}$, we introduce the state $\{q_0, q_1\}$ in G_D and add an edge labeled by 0 between $\{q_0\}$ and $\{q_0, q_1\}$. In the same way, considering $\delta_N(q_0, 1) = \{q_1\}$ gives us the new state $\{q_1\}$ and an edge labeled by 1 between it and $\{q_0\}$.

There are now a number of missing edges, so we continue, using the construction of Theorem 2.2. Looking at the state $\{q_0, q_1\}$, we see that there is no outgoing edge labeled by 0, so we compute

$$\delta_N^*(q_0, 0) \cup \delta_N^*(q_1, 0) = \{q_0, q_1, q_2\}.$$

This gives us the new state $\{q_0, q_1, q_2\}$ and the transition

$$\delta_N^*(\{q_0, q_1\}, 0) = \{q_0, q_1, q_2\}.$$

The construction in the previous proof is tedious but important. Let us do another example to make sure we understand all the steps.

Example 2.13

Convert the NFA in the following Figure to an equivalent DFA.

Since $\delta_N(q_0, 0) = \{q_0, q_1\}$, we introduce the state $\{q_0, q_1\}$ in G_D and add an edge labeled by 0 between $\{q_0\}$ and $\{q_0, q_1\}$. In the same way, considering $\delta_N(q_0, 1) = \{q_1\}$ gives us the new state $\{q_1\}$ and an edge labeled by 1 between it and $\{q_0\}$.

There are now a number of missing edges, so we continue, using the construction of Theorem 2.2. Looking at the state $\{q_0, q_1\}$, we see that there is no outgoing edge labeled by 0, so we compute

$$\delta_N^*(q_0, 0) \cup \delta_N^*(q_1, 0) = \{q_0, q_1, q_2\}.$$

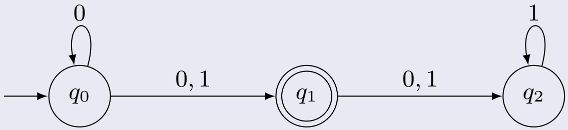
This gives us the new state $\{q_0, q_1, q_2\}$ and the transition

$$\delta_N^*(\{q_0, q_1\}, 0) = \{q_0, q_1, q_2\}.$$

The construction in the previous proof is tedious but important. Let us do another example to make sure we understand all the steps.

Example 2.13

Convert the NFA in the following Figure to an equivalent DFA.



Since $\delta_N(q_0, 0) = \{q_0, q_1\}$, we introduce the state $\{q_0, q_1\}$ in G_D and add an edge labeled by 0 between $\{q_0\}$ and $\{q_0, q_1\}$. In the same way, considering $\delta_N(q_0, 1) = \{q_1\}$ gives us the new state $\{q_1\}$ and an edge labeled by 1 between it and $\{q_0\}$.

There are now a number of missing edges, so we continue, using the construction of Theorem 2.2. Looking at the state $\{q_0, q_1\}$, we see that there is no outgoing edge labeled by 0, so we compute

$$\delta_N^*(q_0, 0) \cup \delta_N^*(q_1, 0) = \{q_0, q_1, q_2\}.$$

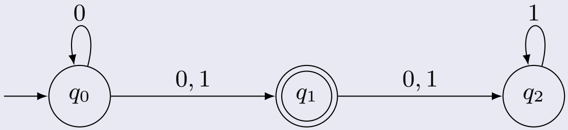
This gives us the new state $\{q_0, q_1, q_2\}$ and the transition

$$\delta_N^*(\{q_0, q_1\}, 0) = \{q_0, q_1, q_2\}.$$

The construction in the previous proof is tedious but important. Let us do another example to make sure we understand all the steps.

Example 2.13

Convert the NFA in the following Figure to an equivalent DFA.



Since $\delta_N(q_0, 0) = \{q_0, q_1\}$, we introduce the state $\{q_0, q_1\}$ in G_D and add an edge labeled by 0 between $\{q_0\}$ and $\{q_0, q_1\}$. In the same way, considering $\delta_N(q_0, 1) = \{q_1\}$ gives us the new state $\{q_1\}$ and an edge labeled by 1 between it and $\{q_0\}$.

There are now a number of missing edges, so we continue, using the construction of Theorem 2.2. Looking at the state $\{q_0, q_1\}$, we see that there is no outgoing edge labeled by 0, so we compute

$$\delta_N^*(q_0, 0) \cup \delta_N^*(q_1, 0) = \{q_0, q_1, q_2\}.$$

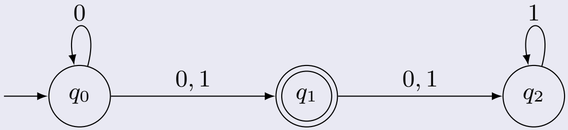
This gives us the new state $\{q_0, q_1, q_2\}$ and the transition

$$\delta_N^*(\{q_0, q_1\}, 0) = \{q_0, q_1, q_2\}.$$

The construction in the previous proof is tedious but important. Let us do another example to make sure we understand all the steps.

Example 2.13

Convert the NFA in the following Figure to an equivalent DFA.



Since $\delta_N(q_0, 0) = \{q_0, q_1\}$, we introduce the state $\{q_0, q_1\}$ in G_D and add an edge labeled by 0 between $\{q_0\}$ and $\{q_0, q_1\}$. In the same way, considering $\delta_N(q_0, 1) = \{q_1\}$ gives us the new state $\{q_1\}$ and an edge labeled by 1 between it and $\{q_0\}$.

There are now a number of missing edges, so we continue, using the construction of Theorem 2.2. Looking at the state $\{q_0, q_1\}$, we see that there is no outgoing edge labeled by 0, so we compute

$$\delta_N^*(q_0, 0) \cup \delta_N^*(q_1, 0) = \{q_0, q_1, q_2\}.$$

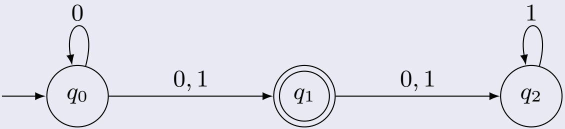
This gives us the new state $\{q_0, q_1, q_2\}$ and the transition

$$\delta_N^*(\{q_0, q_1\}, 0) = \{q_0, q_1, q_2\}.$$

The construction in the previous proof is tedious but important. Let us do another example to make sure we understand all the steps.

Example 2.13

Convert the NFA in the following Figure to an equivalent DFA.



Since $\delta_N(q_0, 0) = \{q_0, q_1\}$, we introduce the state $\{q_0, q_1\}$ in G_D and add an edge labeled by 0 between $\{q_0\}$ and $\{q_0, q_1\}$. In the same way, considering $\delta_N(q_0, 1) = \{q_1\}$ gives us the new state $\{q_1\}$ and an edge labeled by 1 between it and $\{q_0\}$.

There are now a number of missing edges, so we continue, using the construction of Theorem 2.2. Looking at the state $\{q_0, q_1\}$, we see that there is no outgoing edge labeled by 0, so we compute

$$\delta_N^*(q_0, 0) \cup \delta_N^*(q_1, 0) = \{q_0, q_1, q_2\}.$$

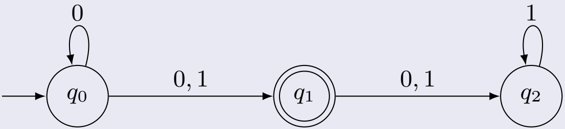
This gives us the new state $\{q_0, q_1, q_2\}$ and the transition

$$\delta_N^*(\{q_0, q_1\}, 0) = \{q_0, q_1, q_2\}.$$

The construction in the previous proof is tedious but important. Let us do another example to make sure we understand all the steps.

Example 2.13

Convert the NFA in the following Figure to an equivalent DFA.



Since $\delta_N(q_0, 0) = \{q_0, q_1\}$, we introduce the state $\{q_0, q_1\}$ in G_D and add an edge labeled by 0 between $\{q_0\}$ and $\{q_0, q_1\}$. In the same way, considering $\delta_N(q_0, 1) = \{q_1\}$ gives us the new state $\{q_1\}$ and an edge labeled by 1 between it and $\{q_0\}$.

There are now a number of missing edges, so we continue, using the construction of Theorem 2.2. Looking at the state $\{q_0, q_1\}$, we see that there is no outgoing edge labeled by 0, so we compute

$$\delta_N^*(q_0, 0) \cup \delta_N^*(q_1, 0) = \{q_0, q_1, q_2\}.$$

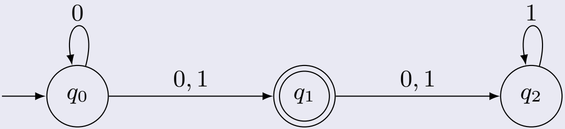
This gives us the new state $\{q_0, q_1, q_2\}$ and the transition

$$\delta_N^*(\{q_0, q_1\}, 0) = \{q_0, q_1, q_2\}.$$

The construction in the previous proof is tedious but important. Let us do another example to make sure we understand all the steps.

Example 2.13

Convert the NFA in the following Figure to an equivalent DFA.



Since $\delta_N(q_0, 0) = \{q_0, q_1\}$, we introduce the state $\{q_0, q_1\}$ in G_D and add an edge labeled by 0 between $\{q_0\}$ and $\{q_0, q_1\}$. In the same way, considering $\delta_N(q_0, 1) = \{q_1\}$ gives us the new state $\{q_1\}$ and an edge labeled by 1 between it and $\{q_0\}$.

There are now a number of missing edges, so we continue, using the construction of Theorem 2.2. Looking at the state $\{q_0, q_1\}$, we see that there is no outgoing edge labeled by 0, so we compute

$$\delta_N^*(q_0, 0) \cup \delta_N^*(q_1, 0) = \{q_0, q_1, q_2\}.$$

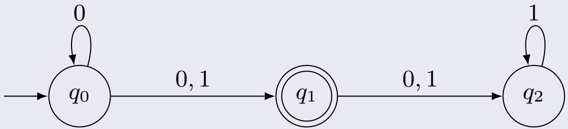
This gives us the new state $\{q_0, q_1, q_2\}$ and the transition

$$\delta_N^*(\{q_0, q_1\}, 0) = \{q_0, q_1, q_2\}.$$

The construction in the previous proof is tedious but important. Let us do another example to make sure we understand all the steps.

Example 2.13

Convert the NFA in the following Figure to an equivalent DFA.



Since $\delta_N(q_0, 0) = \{q_0, q_1\}$, we introduce the state $\{q_0, q_1\}$ in G_D and add an edge labeled by 0 between $\{q_0\}$ and $\{q_0, q_1\}$. In the same way, considering $\delta_N(q_0, 1) = \{q_1\}$ gives us the new state $\{q_1\}$ and an edge labeled by 1 between it and $\{q_0\}$.

There are now a number of missing edges, so we continue, using the construction of Theorem 2.2. Looking at the state $\{q_0, q_1\}$, we see that there is no outgoing edge labeled by 0, so we compute

$$\delta_N^*(q_0, 0) \cup \delta_N^*(q_1, 0) = \{q_0, q_1, q_2\}.$$

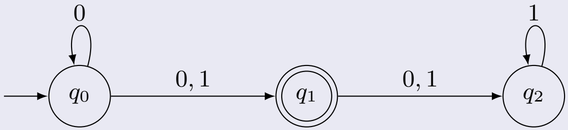
This gives us the new state $\{q_0, q_1, q_2\}$ and the transition

$$\delta_N^*(\{q_0, q_1\}, 0) = \{q_0, q_1, q_2\}.$$

The construction in the previous proof is tedious but important. Let us do another example to make sure we understand all the steps.

Example 2.13

Convert the NFA in the following Figure to an equivalent DFA.



Since $\delta_N(q_0, 0) = \{q_0, q_1\}$, we introduce the state $\{q_0, q_1\}$ in G_D and add an edge labeled by 0 between $\{q_0\}$ and $\{q_0, q_1\}$. In the same way, considering $\delta_N(q_0, 1) = \{q_1\}$ gives us the new state $\{q_1\}$ and an edge labeled by 1 between it and $\{q_0\}$.

There are now a number of missing edges, so we continue, using the construction of Theorem 2.2. Looking at the state $\{q_0, q_1\}$, we see that there is no outgoing edge labeled by 0, so we compute

$$\delta_N^*(q_0, 0) \cup \delta_N^*(q_1, 0) = \{q_0, q_1, q_2\}.$$

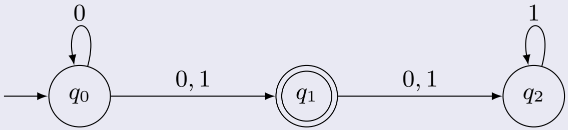
This gives us the new state $\{q_0, q_1, q_2\}$ and the transition

$$\delta_N^*(\{q_0, q_1\}, 0) = \{q_0, q_1, q_2\}.$$

The construction in the previous proof is tedious but important. Let us do another example to make sure we understand all the steps.

Example 2.13

Convert the NFA in the following Figure to an equivalent DFA.



Since $\delta_N(q_0, 0) = \{q_0, q_1\}$, we introduce the state $\{q_0, q_1\}$ in G_D and add an edge labeled by 0 between $\{q_0\}$ and $\{q_0, q_1\}$. In the same way, considering $\delta_N(q_0, 1) = \{q_1\}$ gives us the new state $\{q_1\}$ and an edge labeled by 1 between it and $\{q_0\}$.

There are now a number of missing edges, so we continue, using the construction of Theorem 2.2. Looking at the state $\{q_0, q_1\}$, we see that there is no outgoing edge labeled by 0, so we compute

$$\delta_N^*(q_0, 0) \cup \delta_N^*(q_1, 0) = \{q_0, q_1, q_2\}.$$

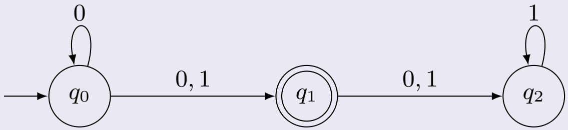
This gives us the new state $\{q_0, q_1, q_2\}$ and the transition

$$\delta_N^*(\{q_0, q_1\}, 0) = \{q_0, q_1, q_2\}.$$

The construction in the previous proof is tedious but important. Let us do another example to make sure we understand all the steps.

Example 2.13

Convert the NFA in the following Figure to an equivalent DFA.



Since $\delta_N(q_0, 0) = \{q_0, q_1\}$, we introduce the state $\{q_0, q_1\}$ in G_D and add an edge labeled by 0 between $\{q_0\}$ and $\{q_0, q_1\}$. In the same way, considering $\delta_N(q_0, 1) = \{q_1\}$ gives us the new state $\{q_1\}$ and an edge labeled by 1 between it and $\{q_0\}$.

There are now a number of missing edges, so we continue, using the construction of Theorem 2.2. Looking at the state $\{q_0, q_1\}$, we see that there is no outgoing edge labeled by 0, so we compute

$$\delta_N^*(q_0, 0) \cup \delta_N^*(q_1, 0) = \{q_0, q_1, q_2\}.$$

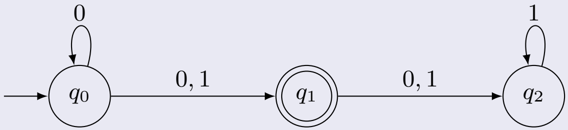
This gives us the new state $\{q_0, q_1, q_2\}$ and the transition

$$\delta_N^*(\{q_0, q_1\}, 0) = \{q_0, q_1, q_2\}.$$

The construction in the previous proof is tedious but important. Let us do another example to make sure we understand all the steps.

Example 2.13

Convert the NFA in the following Figure to an equivalent DFA.



Since $\delta_N(q_0, 0) = \{q_0, q_1\}$, we introduce the state $\{q_0, q_1\}$ in G_D and add an edge labeled by 0 between $\{q_0\}$ and $\{q_0, q_1\}$. In the same way, considering $\delta_N(q_0, 1) = \{q_1\}$ gives us the new state $\{q_1\}$ and an edge labeled by 1 between it and $\{q_0\}$.

There are now a number of missing edges, so we continue, using the construction of Theorem 2.2. Looking at the state $\{q_0, q_1\}$, we see that there is no outgoing edge labeled by 0, so we compute

$$\delta_N^*(q_0, 0) \cup \delta_N^*(q_1, 0) = \{q_0, q_1, q_2\}.$$

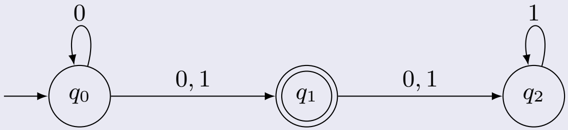
This gives us the new state $\{q_0, q_1, q_2\}$ and the transition

$$\delta_N^*(\{q_0, q_1\}, 0) = \{q_0, q_1, q_2\}.$$

The construction in the previous proof is tedious but important. Let us do another example to make sure we understand all the steps.

Example 2.13

Convert the NFA in the following Figure to an equivalent DFA.



Since $\delta_N(q_0, 0) = \{q_0, q_1\}$, we introduce the state $\{q_0, q_1\}$ in G_D and add an edge labeled by 0 between $\{q_0\}$ and $\{q_0, q_1\}$. In the same way, considering $\delta_N(q_0, 1) = \{q_1\}$ gives us the new state $\{q_1\}$ and an edge labeled by 1 between it and $\{q_0\}$.

There are now a number of missing edges, so we continue, using the construction of Theorem 2.2. Looking at the state $\{q_0, q_1\}$, we see that there is no outgoing edge labeled by 0, so we compute

$$\delta_N^*(q_0, 0) \cup \delta_N^*(q_1, 0) = \{q_0, q_1, q_2\}.$$

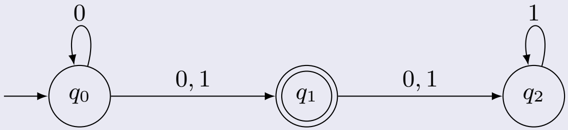
This gives us the new state $\{q_0, q_1, q_2\}$ and the transition

$$\delta_N^*(\{q_0, q_1\}, 0) = \{q_0, q_1, q_2\}.$$

The construction in the previous proof is tedious but important. Let us do another example to make sure we understand all the steps.

Example 2.13

Convert the NFA in the following Figure to an equivalent DFA.



Since $\delta_N(q_0, 0) = \{q_0, q_1\}$, we introduce the state $\{q_0, q_1\}$ in G_D and add an edge labeled by 0 between $\{q_0\}$ and $\{q_0, q_1\}$. In the same way, considering $\delta_N(q_0, 1) = \{q_1\}$ gives us the new state $\{q_1\}$ and an edge labeled by 1 between it and $\{q_0\}$.

There are now a number of missing edges, so we continue, using the construction of Theorem 2.2. Looking at the state $\{q_0, q_1\}$, we see that there is no outgoing edge labeled by 0, so we compute

$$\delta_N^*(q_0, 0) \cup \delta_N^*(q_1, 0) = \{q_0, q_1, q_2\}.$$

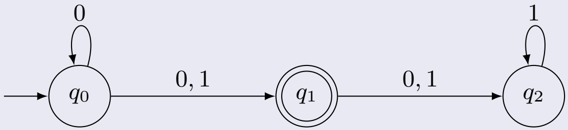
This gives us the new state $\{q_0, q_1, q_2\}$ and the transition

$$\delta_N^*(\{q_0, q_1\}, 0) = \{q_0, q_1, q_2\}.$$

The construction in the previous proof is tedious but important. Let us do another example to make sure we understand all the steps.

Example 2.13

Convert the NFA in the following Figure to an equivalent DFA.



Since $\delta_N(q_0, 0) = \{q_0, q_1\}$, we introduce the state $\{q_0, q_1\}$ in G_D and add an edge labeled by 0 between $\{q_0\}$ and $\{q_0, q_1\}$. In the same way, considering $\delta_N(q_0, 1) = \{q_1\}$ gives us the new state $\{q_1\}$ and an edge labeled by 1 between it and $\{q_0\}$.

There are now a number of missing edges, so we continue, using the construction of Theorem 2.2. Looking at the state $\{q_0, q_1\}$, we see that there is no outgoing edge labeled by 0, so we compute

$$\delta_N^*(q_0, 0) \cup \delta_N^*(q_1, 0) = \{q_0, q_1, q_2\}.$$

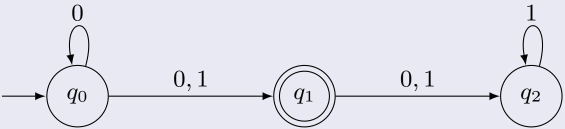
This gives us the new state $\{q_0, q_1, q_2\}$ and the transition

$$\delta_N^*(\{q_0, q_1\}, 0) = \{q_0, q_1, q_2\}.$$

The construction in the previous proof is tedious but important. Let us do another example to make sure we understand all the steps.

Example 2.13

Convert the NFA in the following Figure to an equivalent DFA.



Since $\delta_N(q_0, 0) = \{q_0, q_1\}$, we introduce the state $\{q_0, q_1\}$ in G_D and add an edge labeled by 0 between $\{q_0\}$ and $\{q_0, q_1\}$. In the same way, considering $\delta_N(q_0, 1) = \{q_1\}$ gives us the new state $\{q_1\}$ and an edge labeled by 1 between it and $\{q_0\}$.

There are now a number of missing edges, so we continue, using the construction of Theorem 2.2. Looking at the state $\{q_0, q_1\}$, we see that there is no outgoing edge labeled by 0, so we compute

$$\delta_N^*(q_0, 0) \cup \delta_N^*(q_1, 0) = \{q_0, q_1, q_2\}.$$

This gives us the new state $\{q_0, q_1, q_2\}$ and the transition

$$\delta_N^*(\{q_0, q_1\}, 0) = \{q_0, q_1, q_2\}.$$

Example 2.13 (continuation)

Then, using $a = 1$, $i = 0$, $j = 1$, $k = 2$,

$$\delta_N^*(q_0, 1) \cup \delta_N^*(q_1, 1) \cup \delta_N^*(q_2, 1) = \{q_1, q_2\}$$

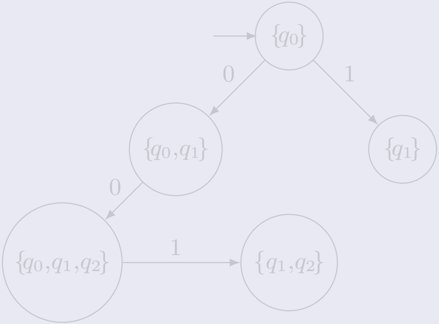
makes it necessary to introduce yet another state $\{q_1, q_2\}$. At this point, we have the partially constructed automaton shown in the following Figure.

Example 2.13 (continuation)

Then, using $a = 1$, $i = 0$, $j = 1$, $k = 2$,

$$\delta_N^*(q_0, 1) \cup \delta_N^*(q_1, 1) \cup \delta_N^*(q_2, 1) = \{q_1, q_2\}$$

makes it necessary to introduce yet another state $\{q_1, q_2\}$. At this point, we have the partially constructed automaton shown in the following Figure.

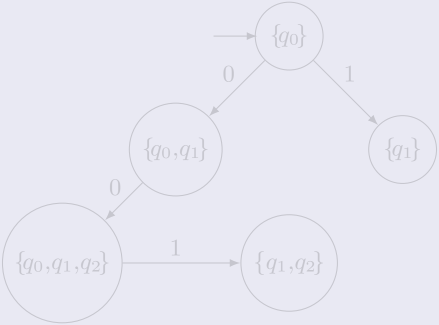


Example 2.13 (continuation)

Then, using $a = 1$, $i = 0$, $j = 1$, $k = 2$,

$$\delta_N^*(q_0, 1) \cup \delta_N^*(q_1, 1) \cup \delta_N^*(q_2, 1) = \{q_1, q_2\}$$

makes it necessary to introduce yet another state $\{q_1, q_2\}$. At this point, we have the partially constructed automaton shown in the following Figure.

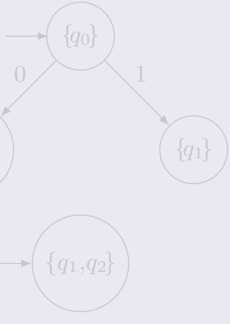


Example 2.13 (continuation)

Then, using $a = 1$, $i = 0$, $j = 1$, $k = 2$,

$$\delta_N^*(q_0, 1) \cup \delta_N^*(q_1, 1) \cup \delta_N^*(q_2, 1) = \{q_1, q_2\}$$

makes it necessary to introduce yet another state $\{q_1, q_2\}$. At this point, we have the partially constructed automaton shown in the following Figure.

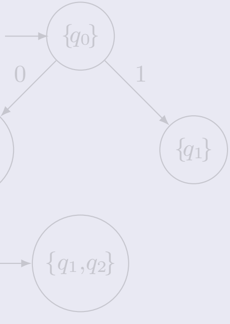


Example 2.13 (continuation)

Then, using $a = 1$, $i = 0$, $j = 1$, $k = 2$,

$$\delta_N^*(q_0, 1) \cup \delta_N^*(q_1, 1) \cup \delta_N^*(q_2, 1) = \{q_1, q_2\}$$

makes it necessary to introduce yet another state $\{q_1, q_2\}$. At this point, we have the partially constructed automaton shown in the following Figure.

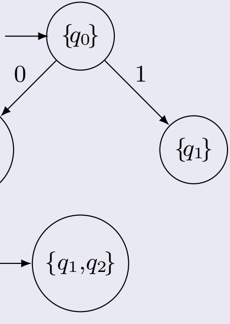


Example 2.13 (continuation)

Then, using $a = 1$, $i = 0$, $j = 1$, $k = 2$,

$$\delta_N^*(q_0, 1) \cup \delta_N^*(q_1, 1) \cup \delta_N^*(q_2, 1) = \{q_1, q_2\}$$

makes it necessary to introduce yet another state $\{q_1, q_2\}$. At this point, we have the partially constructed automaton shown in the following Figure.

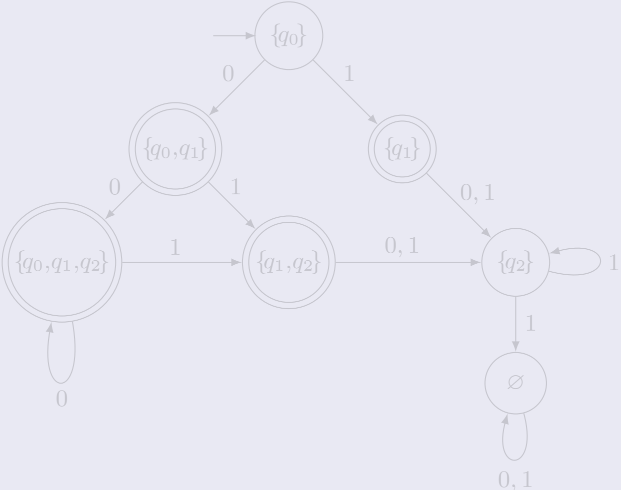


Example 2.13 (continuation)

Since there are still some missing edges, we continue until we obtain the complete solution in the following Figure.

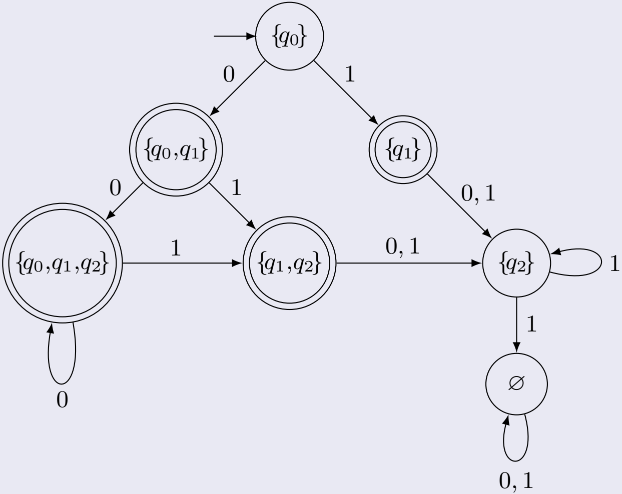
Example 2.13 (continuation)

Since there are still some missing edges, we continue until we obtain the complete solution in the following Figure.



Example 2.13 (continuation)

Since there are still some missing edges, we continue until we obtain the complete solution in the following Figure.



One important conclusion we can draw from Theorem 2.2 is that every language accepted by an NFA is regular.

One important conclusion we can draw from Theorem 2.2 is that every language accepted by an NFA is regular.

Thank You for attention!