CS202 – System Software

Dr. Manish Khare

Lecture 6

Suggested Books/Literatures

- Compilers: Principles, Techniques, and Tools by Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman, 2nd Edition, Pearson.
- Principles of Compiler Design by V. Raghvan, TMH.
- Other Online References

Formal Languages

- A formal language is a set of strings.
- Many infinite language have finite description:
  - Define the language using an automation.
  - Define the language using a grammar.
  - Define the language using a regular expression.

- We can use these compact descriptions of the language to define sets of strings.

Regular Expressions

- **Regular expressions** are a family of descriptions that can be used to capture certain languages (the *regular languages*).

- Often provide a compact and human- readable description of the language.

Atomic Regular Expressions

- The regular expressions we will use in this course begin with two simple building blocks.

- The symbol ε is a regular expression matches the empty string.

- For any symbol **a**, the symbol **a** is a regular expression that just matches **a**.

Compound Regular Expressions

- If R_1 and R_2 are regular expressions, R_1R_2 is a regular expression represents the concatenation of the languages of R_1 and R_2
- If R_1 and R_2 are regular expressions, $R_1|R_2$ is a regular expression representing the union of R_1 and R_2
- If R is a regular expression, R* is a regular expression for the Kleene Closure of R.
- If R is a regular expression, (R) is a regular expression with the same meaning of R.

Operator Precedence

- Regular expression operator precedence is

$$(R)$$

$$R^* \quad R_1R_2$$

$$R_1|R_2$$

So **ab*c|d** is parsed as **((a(b*))c)|d**

- A **Regular Expression** can be recursively defined as follows:
  - ε is a Regular Expression indicates the language containing an empty string. **(L (ε) = {ε})**
  - **φ** is a Regular Expression denoting an empty language. **(L (φ) = { })**
  - **x** is a Regular Expression where **L = {x}**
  - If **X** is a Regular Expression denoting the language **L(X)** and **Y** is a Regular Expression denoting the language **L(Y)**, then
    - **X + Y** is a Regular Expression corresponding to the language **L(X)** ∪ **L(Y)** where **L(X+Y) = L(X)** ∪ **L(Y)**.
    - **X . Y** is a Regular Expression corresponding to the language **L(X) . L(Y)** where **L(X.Y) = L(X) . L(Y)**
    - **R*** is a Regular Expression corresponding to the language **L(R*)**where **L(R*) = (L(R))***
- If we apply any of the rules several times from 1 to 5, they are Regular Expressions.

Regular Expression

- ε is a regular expression, $L(\varepsilon) = \{\varepsilon\}$
- If a is a symbol in ∑then a is a regular expression, $L(a) = \{a\}$
- $(r) \mid (s)$ is a regular expression denoting the language $L(r) \cup L(s)$
- $(r)(s)$ is a regular expression denoting the language $L(r)L(s)$
- $(r)^*$ is a regular expression denoting $(L(r))^*$
- (r) is a regular expression denoting $L(r)$

Regular Expression

Algebraic laws of regular expressions

1) $\alpha|\beta=\beta|\alpha$

2) $\alpha|(\beta|\gamma)=(\alpha|\beta)|\gamma$ $\alpha(\beta\gamma)=(\alpha\beta)\gamma$

3) $\alpha(\beta|\gamma)=\alpha\beta\mid\alpha\gamma$ $(\alpha|\beta)\gamma=\alpha\gamma|\beta\gamma$

4) $\varepsilon\alpha=\alpha\varepsilon=\alpha$

5) $(\alpha^*)^*=\alpha^*$

6) $\alpha^*=\alpha^+|\varepsilon$ $\alpha^+=\alpha\alpha^*=\alpha^*\alpha$

7) $(\alpha|\beta)^*=(\alpha^*|\beta^*)^*=(\alpha^*\beta^*)^*$

Regular Expression

Algebraic laws of regular expressions

8) If $\varepsilon \notin L(\gamma)$, then

$\alpha = \beta | \gamma\alpha$ $\qquad$ $\alpha = \gamma^* \beta$

$\alpha = \beta | \alpha\gamma$ $\qquad$ $\alpha = \beta\gamma^*$

Notes: We assume that the precedence of * is the highest, the precedence of | is the lowest and they are left associative

Regular Expression

- Any set represented by a regular expression is called a *regular*
- *set.*
- If for example, *a, b* ε *L*. then
  - (i) a denotes the set *{a}*,
  - (ii) a + b denotes *{a, b}*,
  - (iii) **ab** denotes *{ab}*,
  - (iv) a* denotes the set *{a, aa. aaa, ... }* and
  - (v) (a + b)* denotes *{a, b}**.
- The set represented by *R* is denoted by L(R)

Operations on RE

- The various operations on languages are:
- Union of two languages L and M is written as
  - L U M = {s | s is in L or s is in M}
- Concatenation of two languages L and M is written as
  - LM = {st | s is in L and t is in M}
- The Kleene Closure of a language L is written as
  - L* = Zero or more occurrence of language L.

- If r and s are regular expressions denoting the languages L(r) and L(s), then
  - **Union** : (r)|(s) is a regular expression denoting L(r) U L(s)
  - **Concatenation** : (r)(s) is a regular expression denoting L(r)L(s)
  - **Kleene closure** : (r)* is a regular expression denoting (L(r))*
  - (r) is a regular expression denoting L(r)

- *, concatenation (.), and | (pipe sign) are left associative
- * has the highest precedence
- Concatenation (.) has the second highest precedence.
- | (pipe sign) has the lowest precedence of all.

Representing valid tokens of a language in RE

- If x is a regular expression, then:
  - x* means zero or more occurrence of x.
    - i.e., it can generate { e, x, xx, xxx, xxxx, … }
  - x? means at most one occurrence of x
    - i.e., it can generate either {x} or {e}.
- [a-z] is all lower-case alphabets of English language.
- [A-Z] is all upper-case alphabets of English language.
- [0-9] is all natural digits used in mathematics.

Representing occurrence of symbols using RE

- letter = [a – z] or [A – Z]

- digit = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 or [0-9]

- sign = [+ | -]

Some RE Examples

Regular Expressions	Regular Set
(0 + 10*)	L = { 0, 1, 10, 100, 1000, 10000, … }
(0*10*)	L = {1, 01, 10, 010, 0010, …}
(0 + ε)(1 + ε)	L = {ε, 0, 1, 01}
(a+b)*	Set of strings of a's and b's of any length including the null string. So L = { ε, a, b, aa , ab , bb , ba, aaa…….}
(a+b)*abb	Set of strings of a's and b's ending with the string abb. So L = {abb, aabb, babb, aaabb, ababb, …………..}
(11)*	Set consisting of even number of 1's including empty string, So L= {ε, 11, 1111, 111111, ……….}
(aa)*(bb)*b	Set of strings consisting of even number of a's followed by odd number of b's , so L = {b, aab, aabbb, aabbbbb, aaaab, aaaabbb, …………..}
(aa + ab + ba + bb)*	String of a's and b's of even length can be obtained by concatenating any combination of the strings aa, ab, ba and bb including null, so L = {aa, ab, ba, bb, aaab, aaba, …………..

Properties of Regular Sets

- **Property 1**. *The union of two regular set is regular*
- *Proof*

Let us take two regular expressions

$RE_1 = a(aa)^*$ and $RE_2 = (aa)^*$

So, $L_1 = \{a, aaa, aaaaa, ...\}$ (Strings of odd length excluding Null)

and $L_2 = \{ \varepsilon, aa, aaaa, aaaaaa, ...\}$ (Strings of even length including Null)

$L_1 \cup L_2 = \{ \varepsilon, a, aa, aaa, aaaa, aaaaa, aaaaaa, ...\}$

(Strings of all possible lengths including Null)

$RE (L_1 \cup L_2) = a^*$ (which is a regular expression itself)

Hence, proved.

- **Property 2.** *The intersection of two regular set is regular.*
- **Proof** –

Let us take two regular expressions

$RE_1 = a(a^*)$ and $RE_2 = (aa)^*$

So, $L_1 = \{ a, aa, aaa, aaaa, \ldots\}$ (Strings of all possible lengths excluding Null)

$L_2 = \{ \varepsilon, aa, aaaa, aaaaaa, \ldots\ldots.\}$ (Strings of even length including Null)

$L_1 \cap L_2 = \{ aa, aaaa, aaaaaa, \ldots\ldots.\}$ (Strings of even length excluding Null)

$RE (L_1 \cap L_2) = aa(aa)^*$ which is a regular expression itself.

Hence, proved.

- **Property 3.** *The complement of a regular set is regular.*
- **Proof** −

Let us take a regular expression −

RE = (aa)*

So, L = {ε, aa, aaaa, aaaaaa, ...} (Strings of even length including Null)

Complement of **L** is all the strings that is not in **L**.

So, L’ = {a, aaa, aaaaa, ...} (Strings of odd length excluding Null)

RE (L’) = a(aa)* which is a regular expression itself.

Hence, proved.

- **Property 4.** *The difference of two regular set is regular.*
- **Proof −**

Let us take two regular expressions −

RE_1 = a (a*) and RE_2 = (aa)*

So, L_1 = {a, aa, aaa, aaaa, ...} (Strings of all possible lengths excluding Null)

L_2 = { ε, aa, aaaa, aaaaaa,.......} (Strings of even length including Null)

$L_1 – L_2$ = {a, aaa, aaaaa, aaaaaaa, ...}

(Strings of all odd lengths excluding Null)

RE ($L_1 – L_2$) = a (aa)* which is a regular expression.

Hence, proved.

- **Property 5.** *The reversal of a regular set is regular.*
- **Proof** –

We have to prove $\mathbf{L}^{\mathbf{R}}$ is also regular if **L** is a regular set.

Let, L = {01, 10, 11, 10}

RE (L) = 01 + 10 + 11 + 10

L^R = {10, 01, 11, 01}

RE (L^R) = 01 + 10 + 11 + 10 which is regular

Hence, proved.

- **Property 6.** *The closure of a regular set is regular.*
- **Proof** –

If L = {a, aaa, aaaaa, ...} (Strings of odd length excluding Null)

i.e., RE (L) = a (aa)*

L* = {a, aa, aaa, aaaa , aaaaa,……………} (Strings of all lengths excluding Null)

RE (L*) = a (a)*

Hence, proved.

- **Property 7.** *The concatenation of two regular sets is regular.*
- **Proof −**

Let RE_1 = (0+1)*0 and RE_2 = 01(0+1)*

Here, L_1 = {0, 00, 10, 000, 010, ...} (Set of strings ending in 0)

and L_2 = {01, 010,011,.....} (Set of strings beginning with 01)

Then, L_1 L_2 =

{001,0010,0011,0001,00010,00011,1001,10010,.............}

Set of strings containing 001 as a substring which can be represented by an RE − (0 + 1)*001(0 + 1)*

Hence, proved.

Simple Regular Expression

- Describe the following sets by regular expressions:

L_1 =the set of all strings of 0's and l's ending in 00.

Any string in L_1 is obtained by concatenating any string over {0,1} and the string 00. {0,1} is represented by 0 + 1. Hence $L\ 1$ is represented by **(0 + 1)* 00**.

- Describe the following sets by regular expressions:

L_2 = the set of all strings of 0's and l's beginning with 0 and ending with 1.

As any element of L_2 is obtained by concatenating 0, any string over {0,1} and 1, L_2 can be represented by **0(0 + 1)* 1**.

- Suppose the only characters are `0` and `1`.
- Here is a regular expression for strings containing `00` as a substring:

(0 | 1)*00(0 | 1)*

11011100101
0000
11111011110011111

Suppose the only characters are `0` and `1`.

Here is a regular expression for strings of length exactly four:

(0|1)(0|1)(0|1)(0|1)

0000
1010
1111
1000

Suppose the only characters are `0` and `1`.

Here is a regular expression for strings that contain at most one zero:

1*(0 | ε)1*

11110111
111111
0111
0

Implementing Regular Expression

- Regular expression can be implemented using Finite Automata.
- There are two main kinds of finite automata.
  - DFAs (Deterministic Finite Automata)
  - NFAs (Non Deterministic Finite Automata)

DFAs (Deterministic Finite Automata)

- In DFA, for each input symbol, one can determine the state to which the machine will move. Hence, it is called **Deterministic Automaton**.

- As it has a finite number of states, the machine is called **Deterministic Finite Automaton.**

DFAs (Deterministic Finite Automata)

- Analytically. a finite automaton can be represented by a 5-tuple *(*Q, ∑, δ, q_0, F*)*. where
  - (i) Q is a finite nonempty set of states.
  - (ii) ∑ is a finite nonempty set of inputs called the *input alphabet.*
  - (iii) δ is a function which maps Q x ∑ into Q and is usually called the *direct transition function.* This is the function which describes the change of states during the transition. This mapping is usually represented by a transition table or a transition diagram.

    $$\delta: Q \times \sum \rightarrow Q$$
  - (iv) $\mathbf{q_0}$ is the initial state from where any input is processed ($q_0 \in Q$).
  - (v) **F** is a set of final state/states of Q ($F \subseteq Q$).

DFAs (Deterministic Finite Automata)

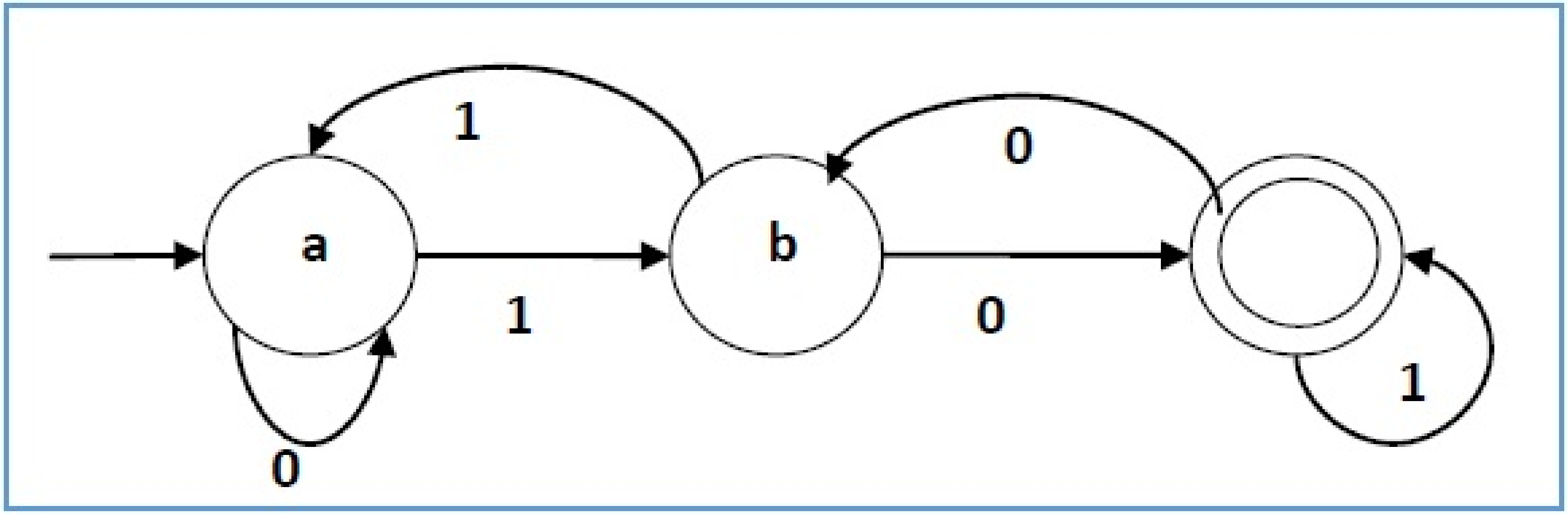
- DFA is represented by digraphs called **state diagram**.
  - The vertices represent the states.
  - The arcs labeled with an input alphabet show the transitions.
  - The initial state is denoted by an empty single incoming arc.
  - The final state is indicated by double circles.

- Example: Let a deterministic finite automaton be →
  - $Q = \{a, b, c\}$,
  - $\sum = \{0, 1\}$,
  - $q_0 = \{a\}$,
  - $F = \{c\}$, and

Transition function δ as shown by the following table −

Present State	Next State for Input 0	Next State for Input 1
a	a	b
b	c	a
c	b	c

- Its graphical representation would be as follows



NFAs (Non Deterministic Finite Automata)

- In NDFA, for a particular input symbol, the machine can move to any combination of the states in the machine. In other words, the exact state to which the machine moves cannot be determined. Hence, it is called **Non-deterministic Automaton**.

- As it has finite number of states, the machine is called **Non-deterministic Finite Automaton**.

- A nondeterministic finite automaton (NDFA) is a 5-tuple *(*Q, ∑, δ, q_0, F), where
  - (i) Q is a finite nonempty set of states;
  - (ii) ∑ is a finite set of symbols called the alphabets;
  - (iii) **δ** is the transition function where δ: Q × ∑ → 2^Q (Here the power set of Q (2^Q) has been taken because in case of NDFA, from a state, transition can occur to any combination of Q states)*;*
  - (iv) $\mathbf{q_0}$ is the initial state from where any input is processed ($q_0 \in Q$);
  - (v) **F** is a set of final state/states of Q ($F \subseteq Q$);

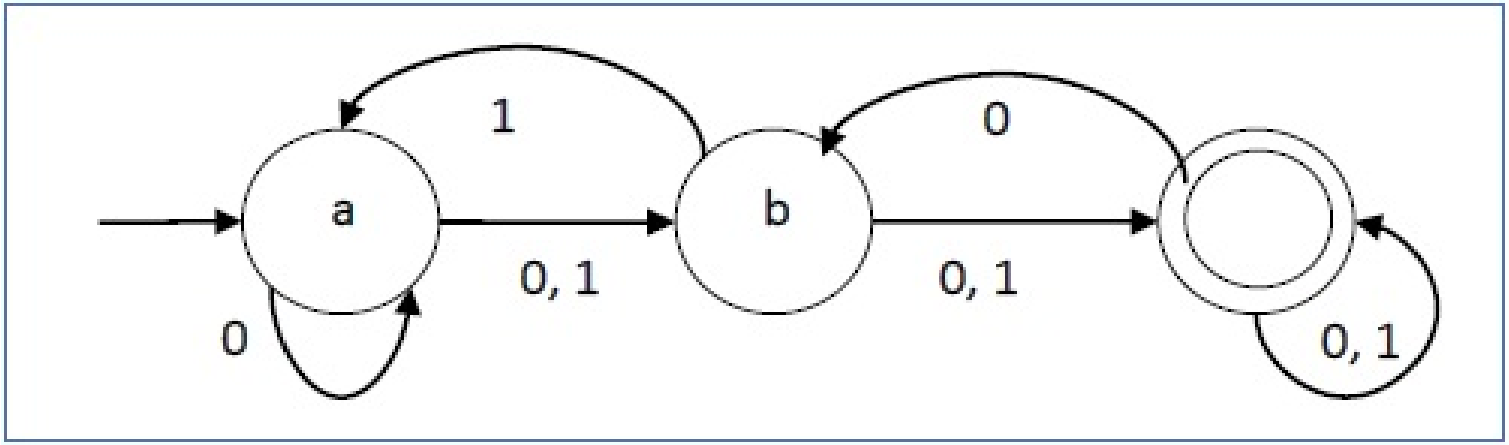
- An NDFA is represented by digraphs called state diagram.
  - The vertices represent the states.
  - The arcs labeled with an input alphabet show the transitions.
  - The initial state is denoted by an empty single incoming arc.
  - The final state is indicated by double circles.

- **Example:** Let a non-deterministic finite automaton be →
  - Q = {a, b, c}
  - ∑ = {0, 1}
  - q_0 = {a}
  - F = {c}

The transition function δ as shown below –

Present State	Next State for Input 0	Next State for Input 1
a	a, b	b
b	c	a, c
c	b, c	c

- Its graphical representation would be as follows −



DFA vs NDFA

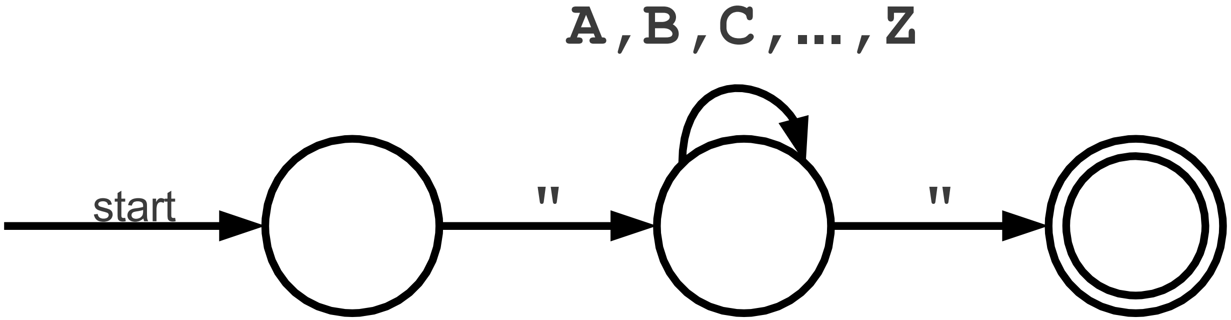
- The following table lists the differences between DFA and NDFA

DFA	NDFA
The transition from a state is to a single particular next state for each input symbol. Hence it is called *deterministic*.	The transition from a state can be to multiple next states for each input symbol. Hence it is called *non-deterministic*.
Empty string transitions are not seen in DFA.	NDFA permits empty string transitions.
Backtracking is allowed in DFA	In NDFA, backtracking is not always possible.
Requires more space.	Requires less space.
A string is accepted by a DFA, if it transits to a final state.	A string is accepted by a NDFA, if at least one of all possible transitions ends in a final state.

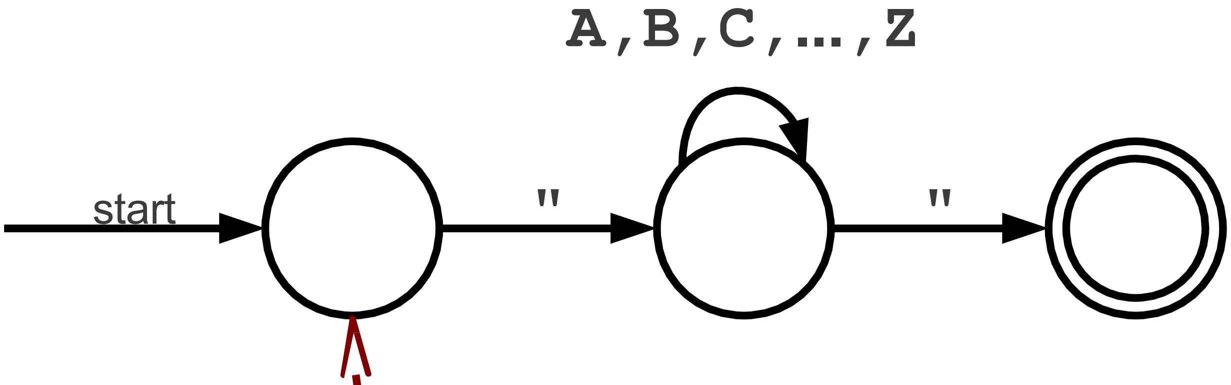
Acceptability by DFA and NDFA

- A string is accepted by a DFA/NDFA iff the DFA/NDFA starting at the initial state ends in an accepting state (any of the final states) after reading the string wholly.
- A string S is accepted by a DFA/NDFA (Q, ∑, δ, q_0, F), iff
  - **$\delta^*(q_0, S) \in F$**
- The language **L** accepted by DFA/NDFA is
  - **$\{S \mid S \in \sum^* \text{ and } \delta^*(q_0, S) \in F\}$**
- A string S′ is not accepted by a DFA/NDFA (Q, ∑, δ, q_0, F), iff
  - **$\delta^*(q_0, S') \notin F$**
- The language L′ not accepted by DFA/NDFA (Complement of accepted language L) is
  - **$\{S \mid S \in \sum^* \text{ and } \delta^*(q_0, S) \notin F\}$**

A Simple Automaton

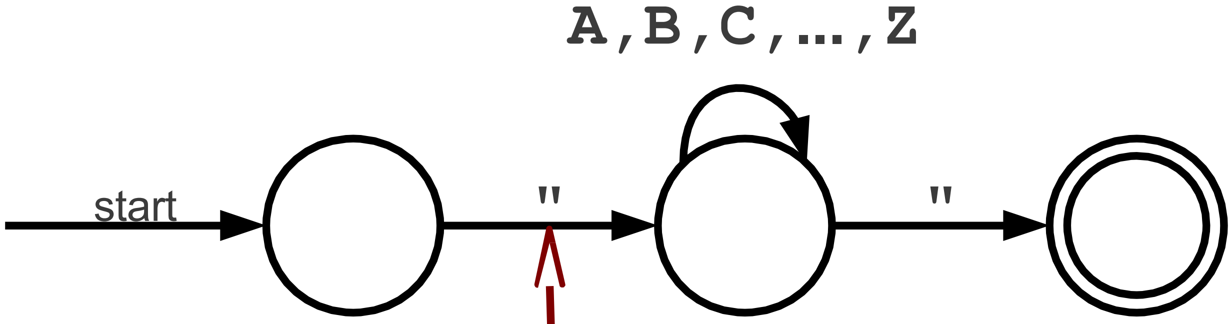


A Simple Automaton



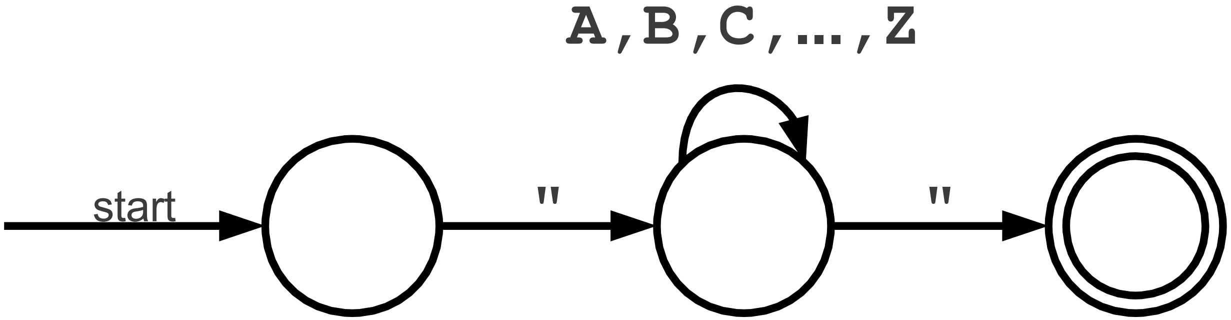
Each circle is a **state** of the automaton. The automaton's configuration is determined by what state(s) it is in.

A Simple Automaton



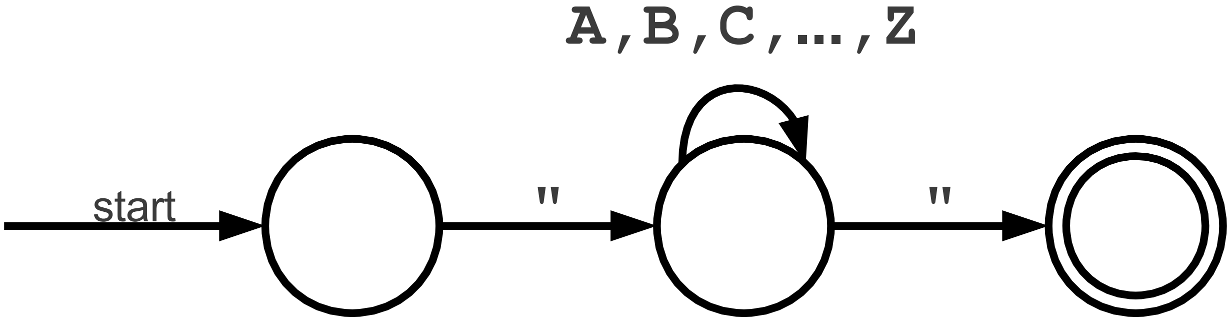
These arrows are called **transitions**. The automaton changes which state(s) it is in by following transitions.

A Simple Automaton



"HEYA"

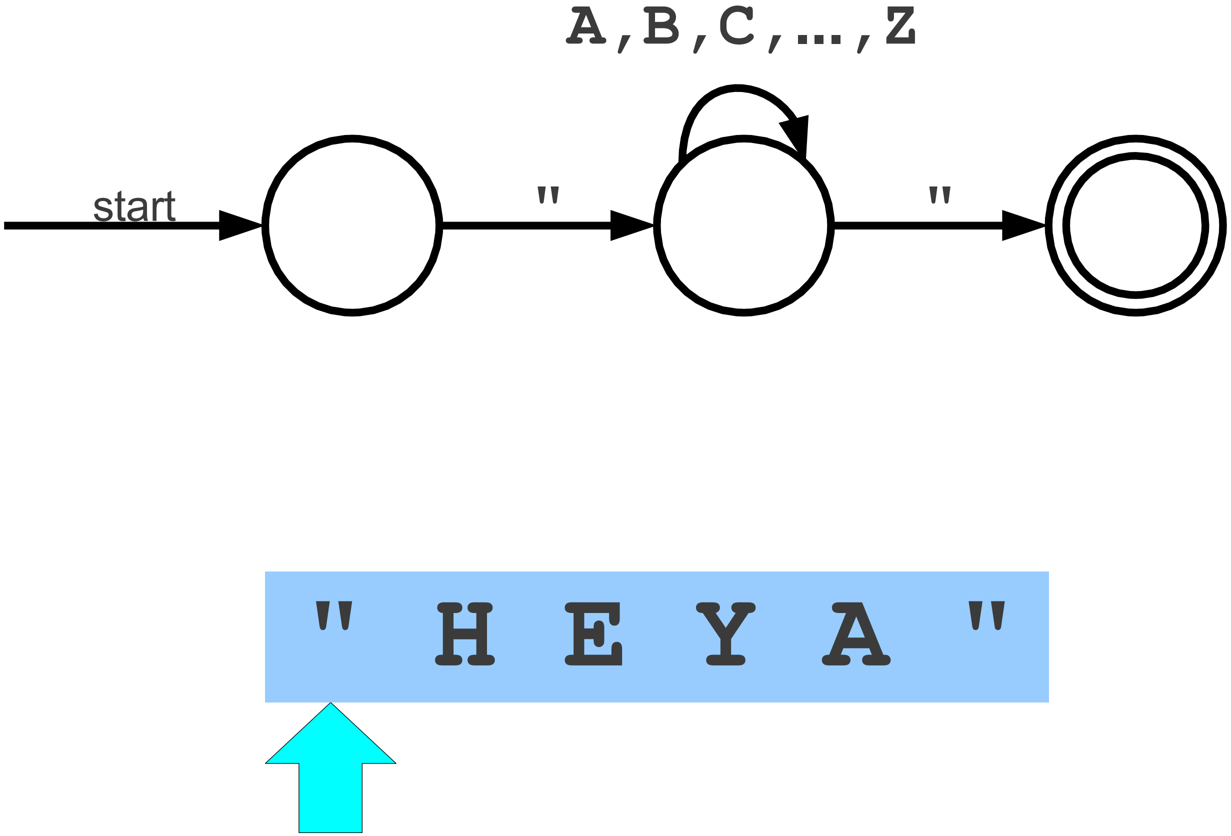
A Simple Automaton



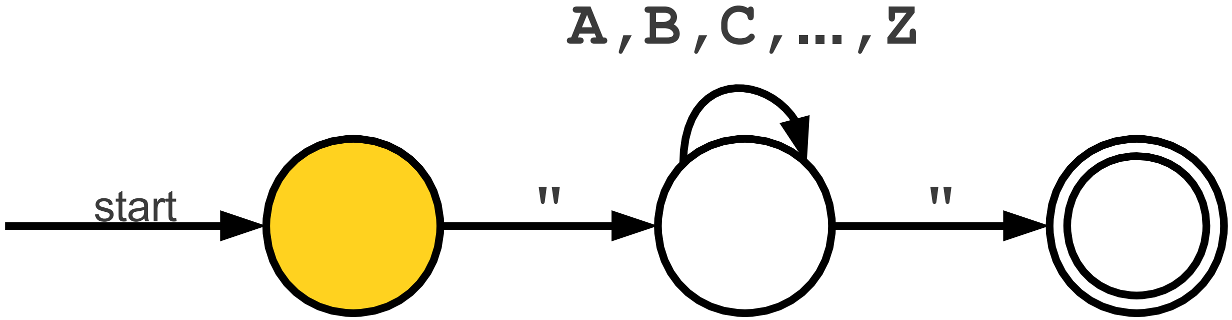
" H E Y A "

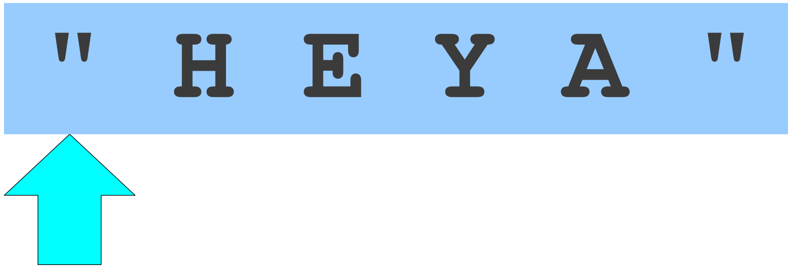
The automaton takes a string as input and decides whether to accept or reject the string.

A Simple Automaton

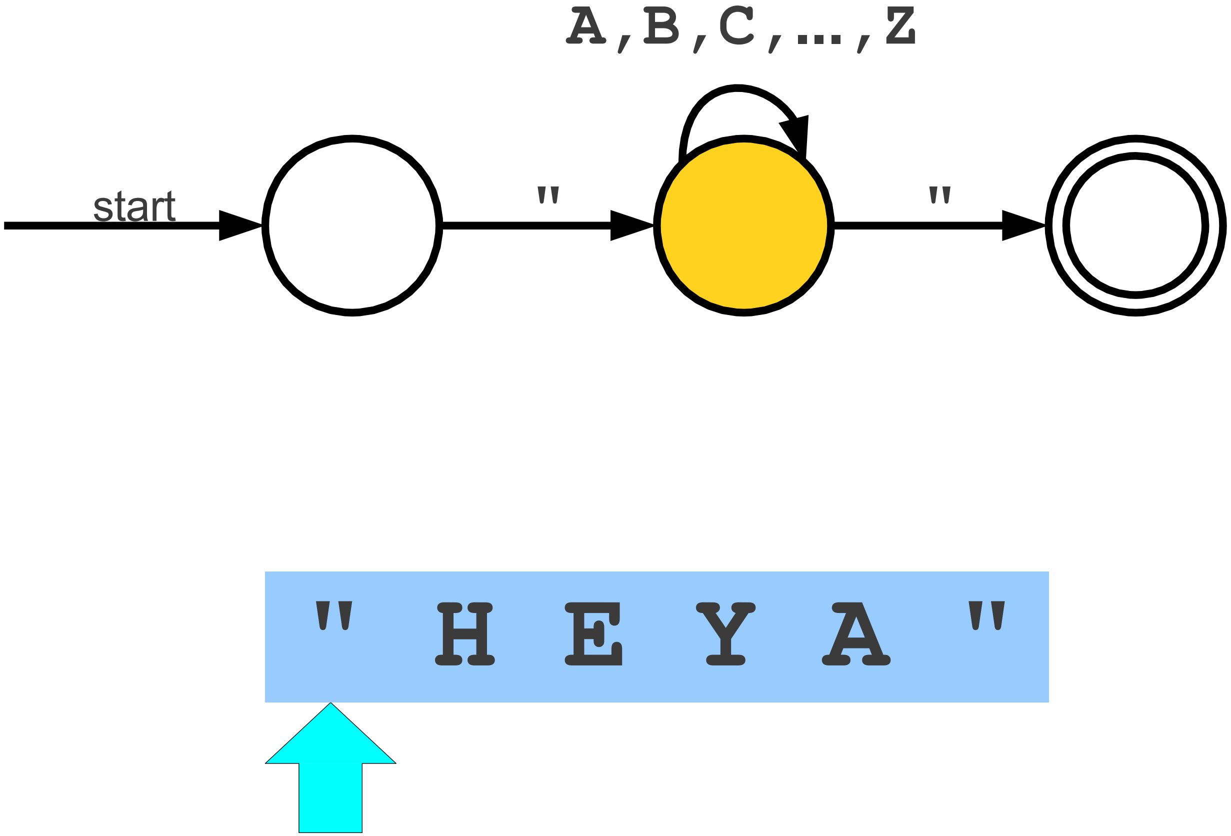


A Simple Automaton

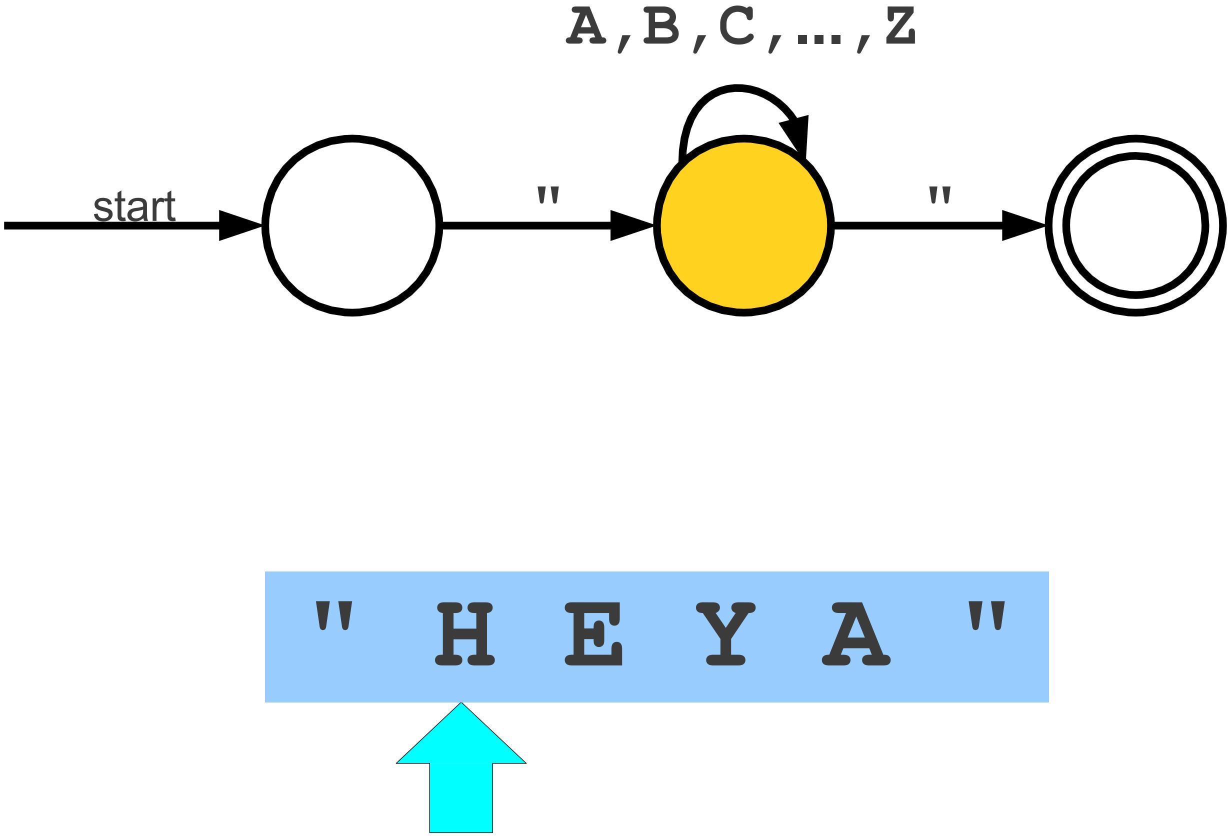




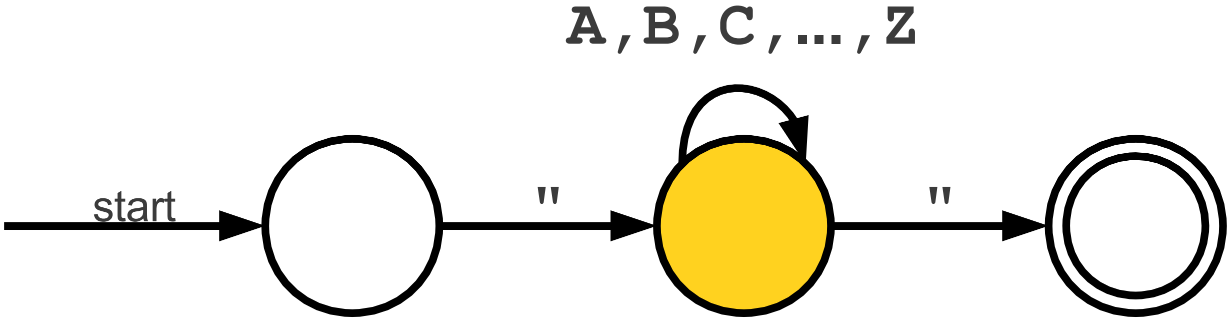
A Simple Automaton



A Simple Automaton

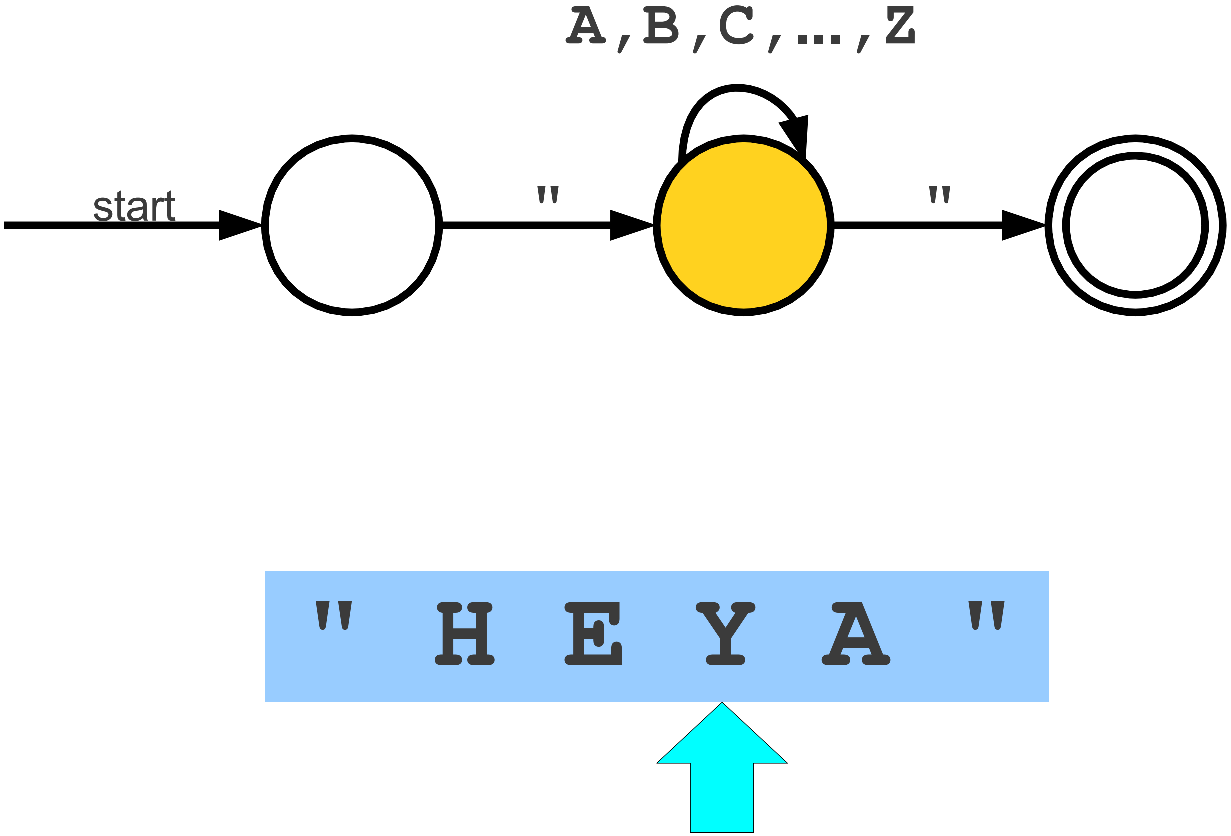


A Simple Automaton

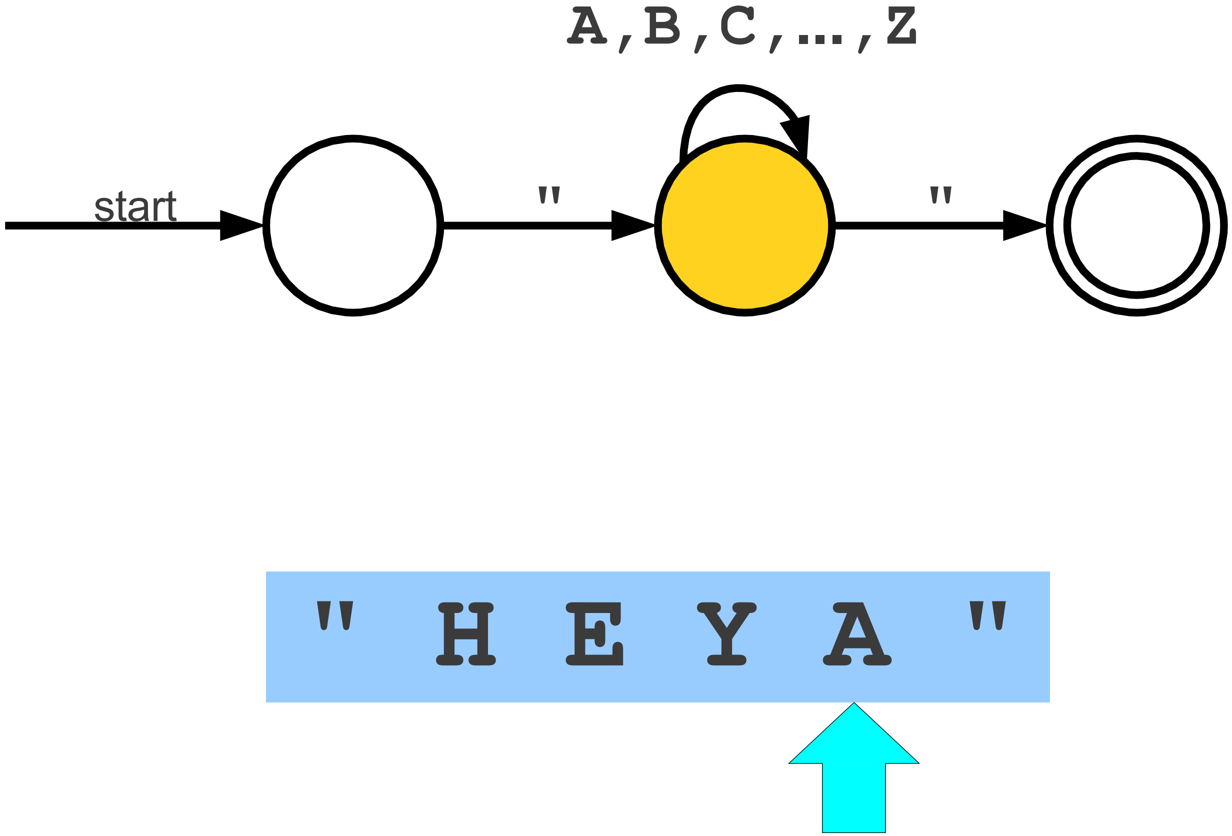


" H E Y A "

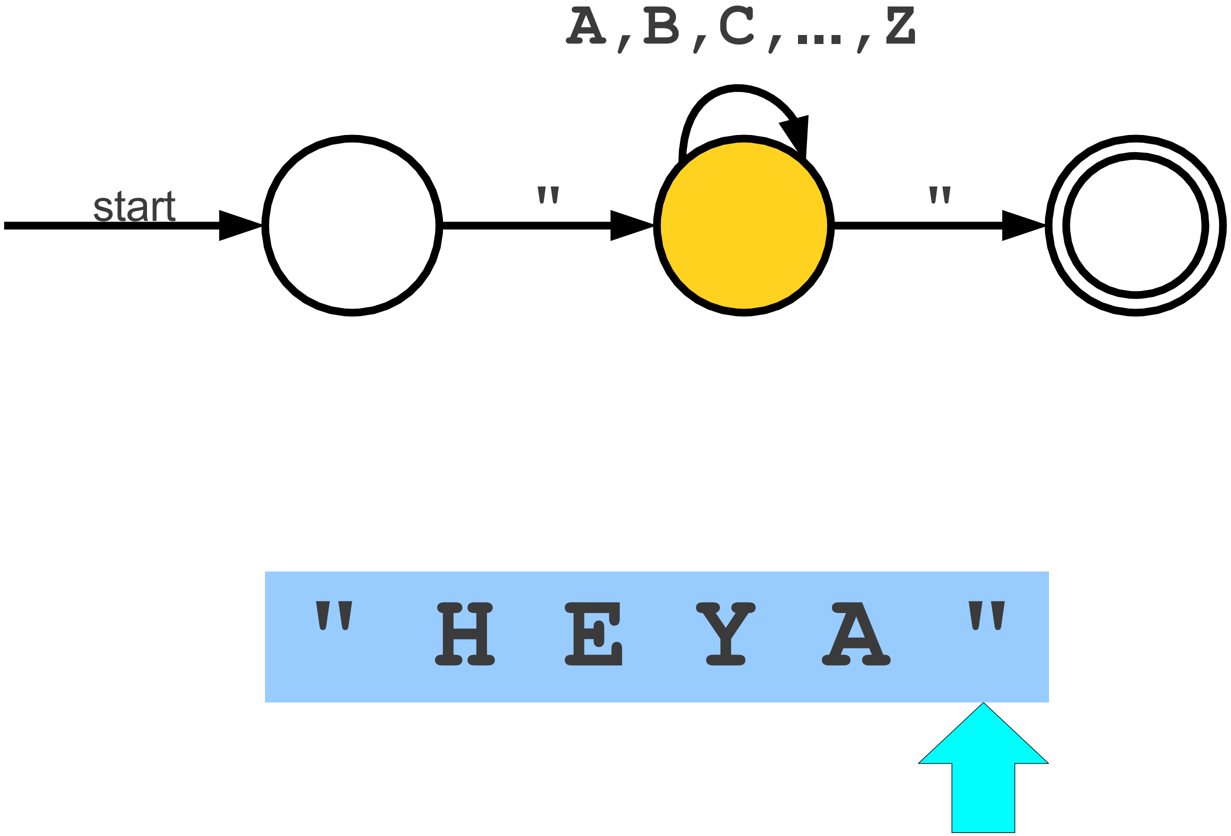
A Simple Automaton



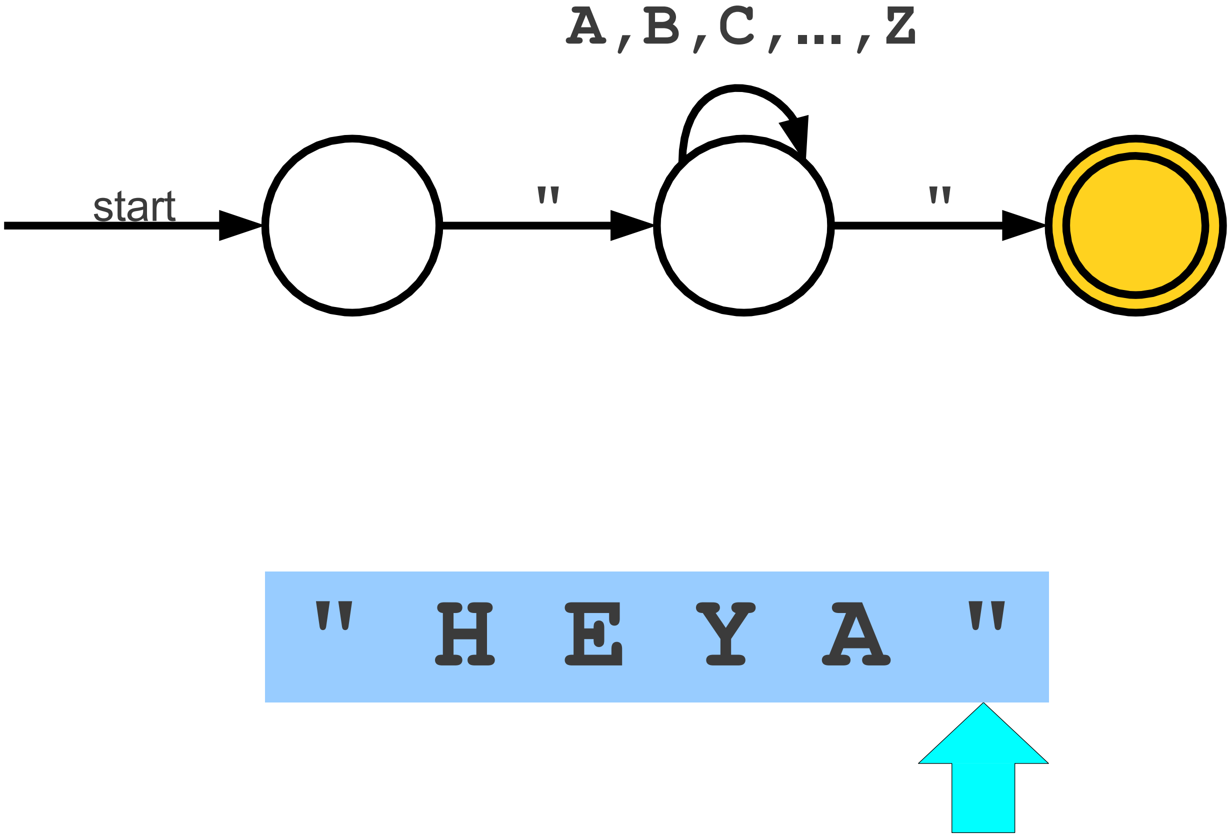
A Simple Automaton



A Simple Automaton

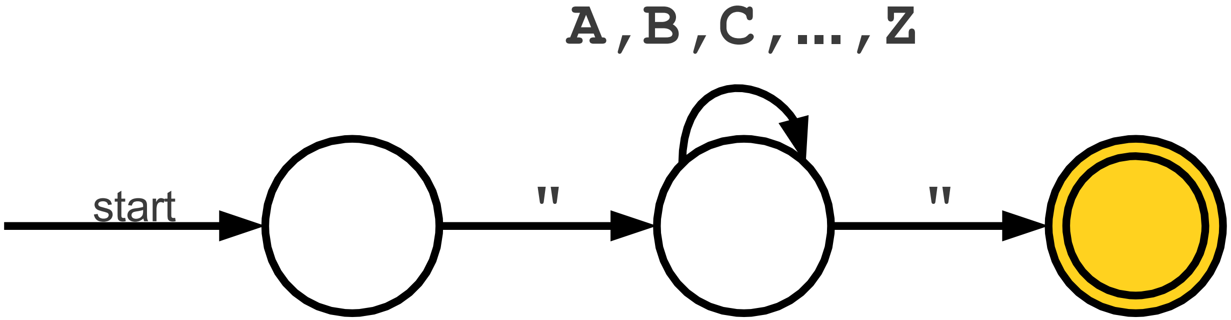


A Simple Automaton



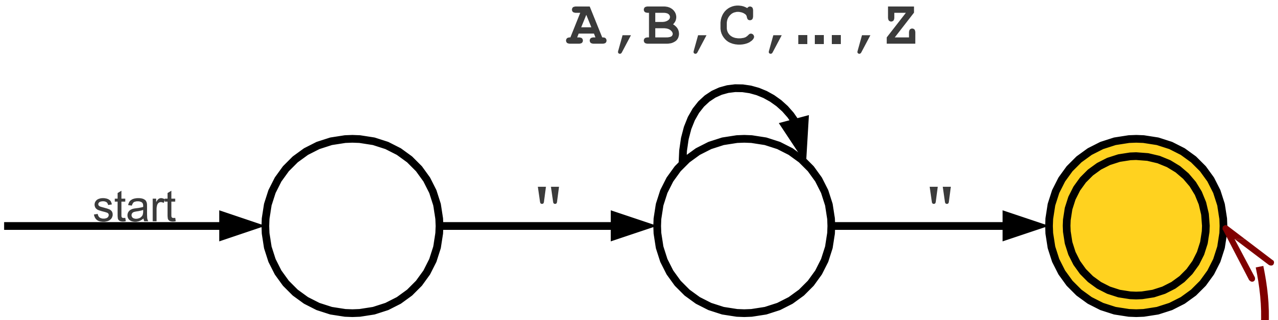
" H E Y A "

A Simple Automaton



" H E Y A "

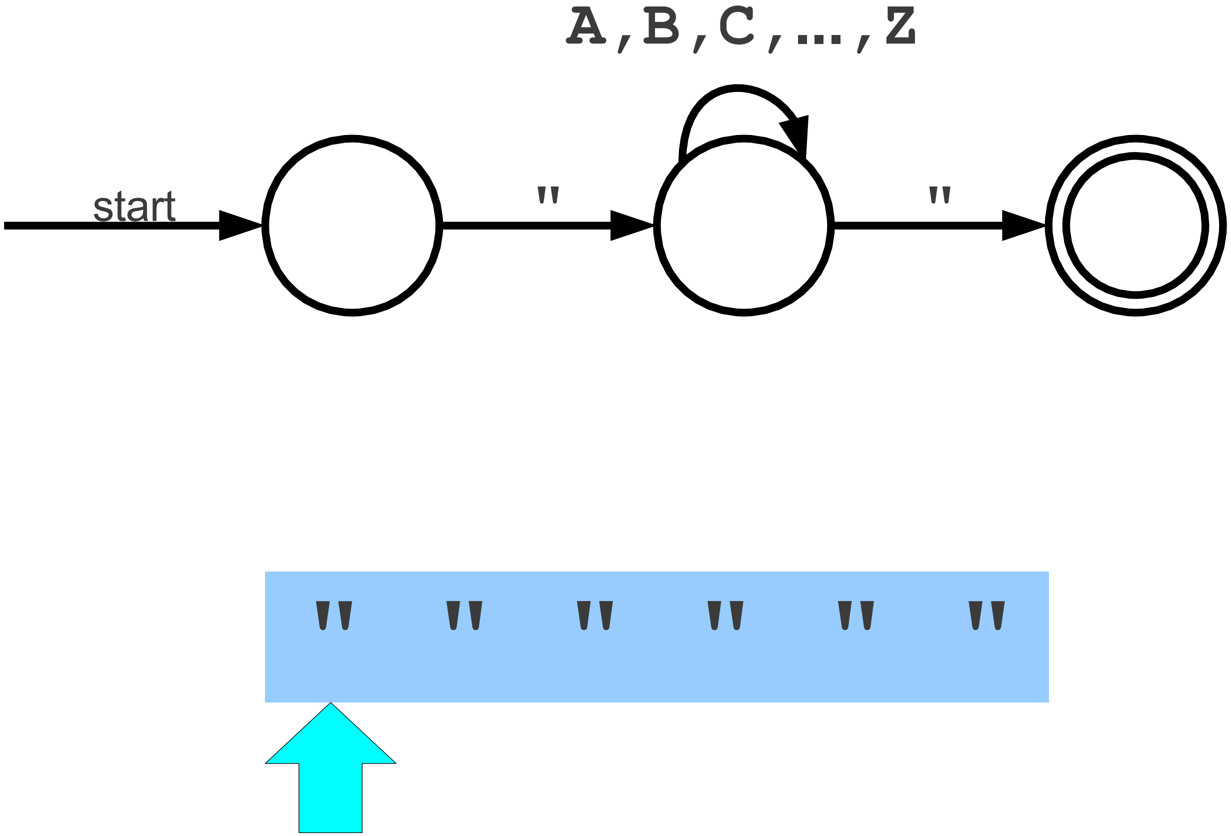
A Simple Automaton



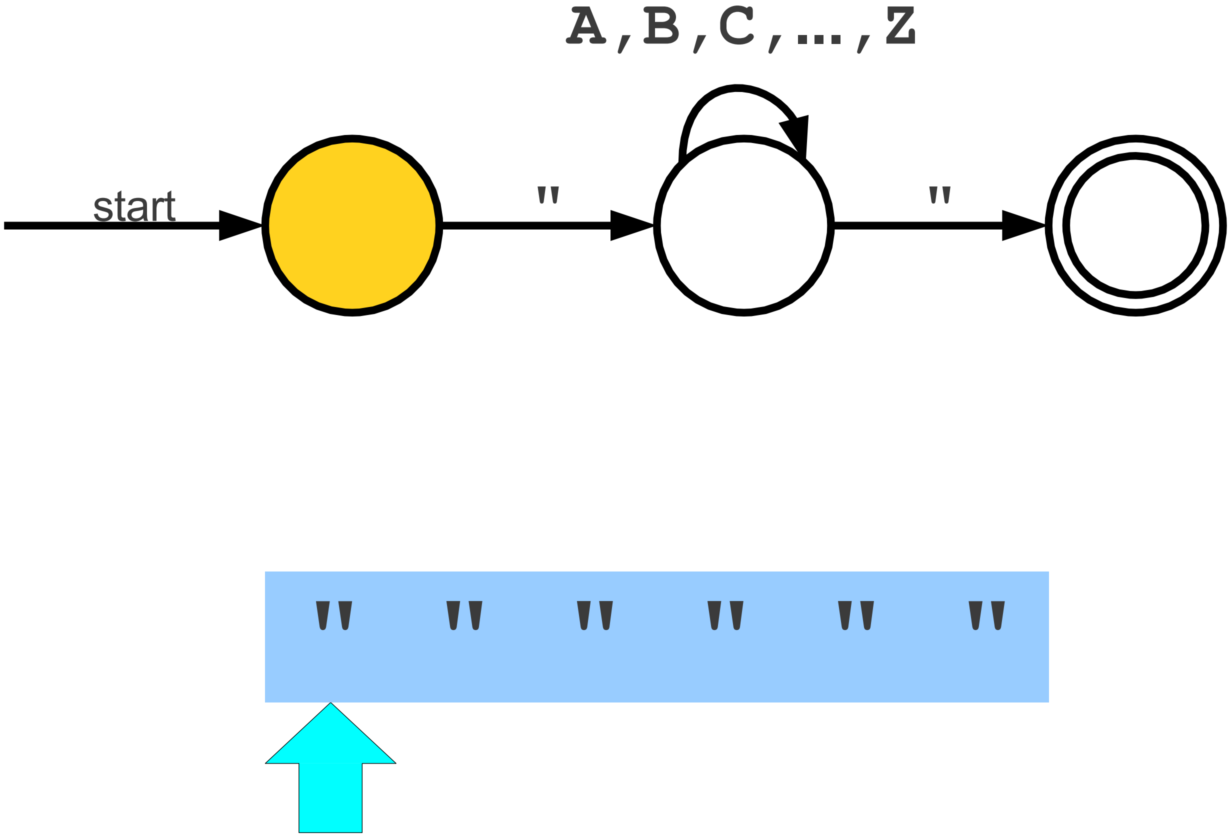
"HEYA"

The double circle indicates that this state is an **accepting state**. The automaton accepts the string if it ends in an accepting state.

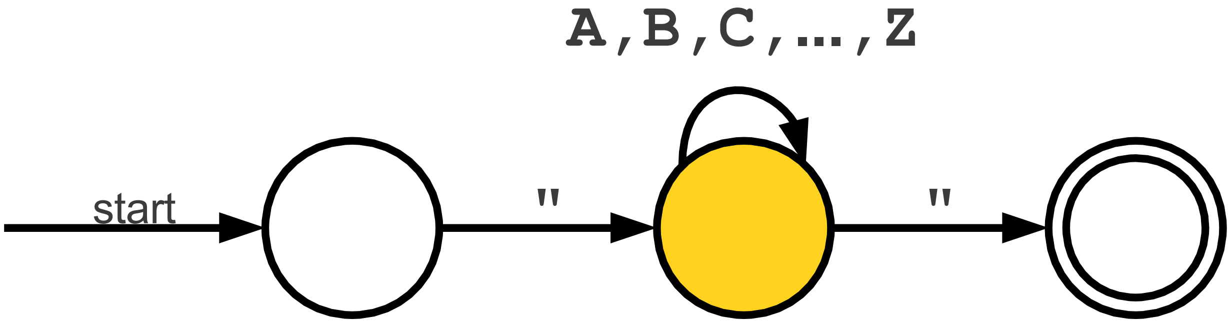
A Simple Automaton

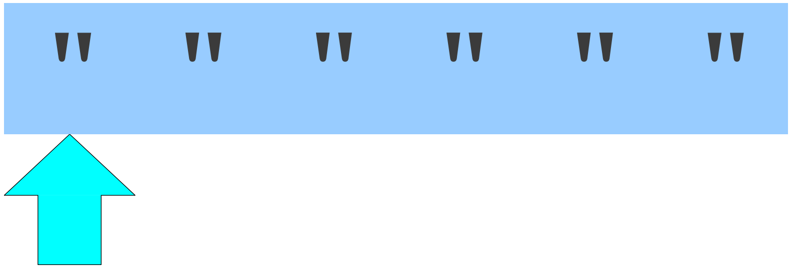


A Simple Automaton

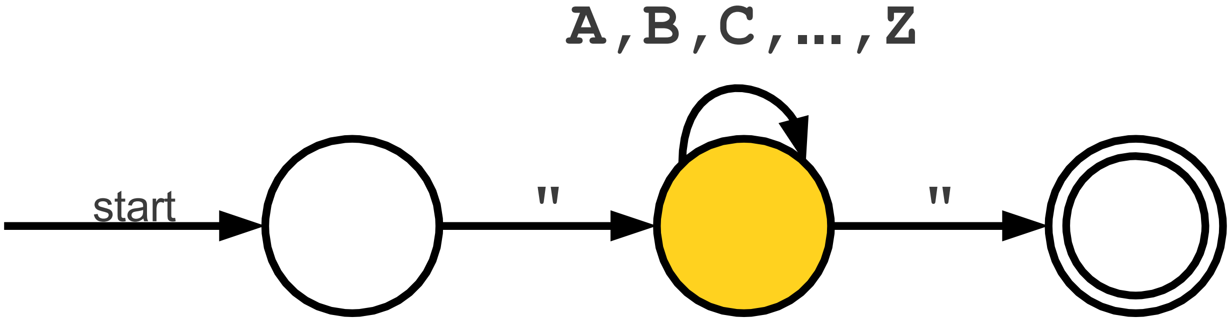


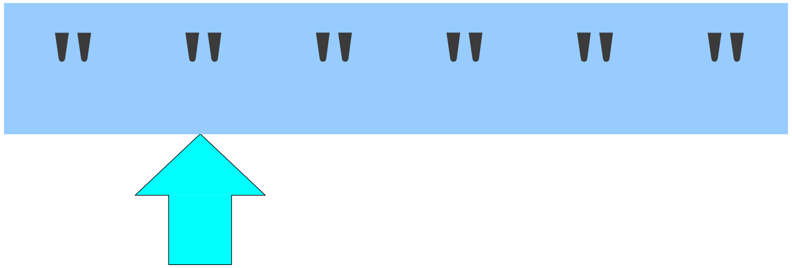
A Simple Automaton



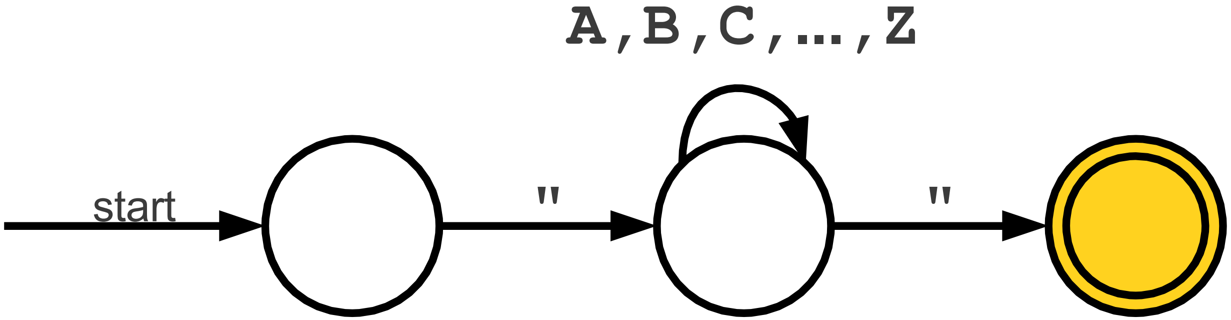


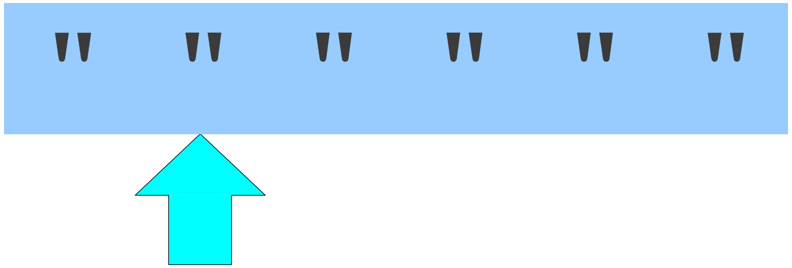
A Simple Automaton



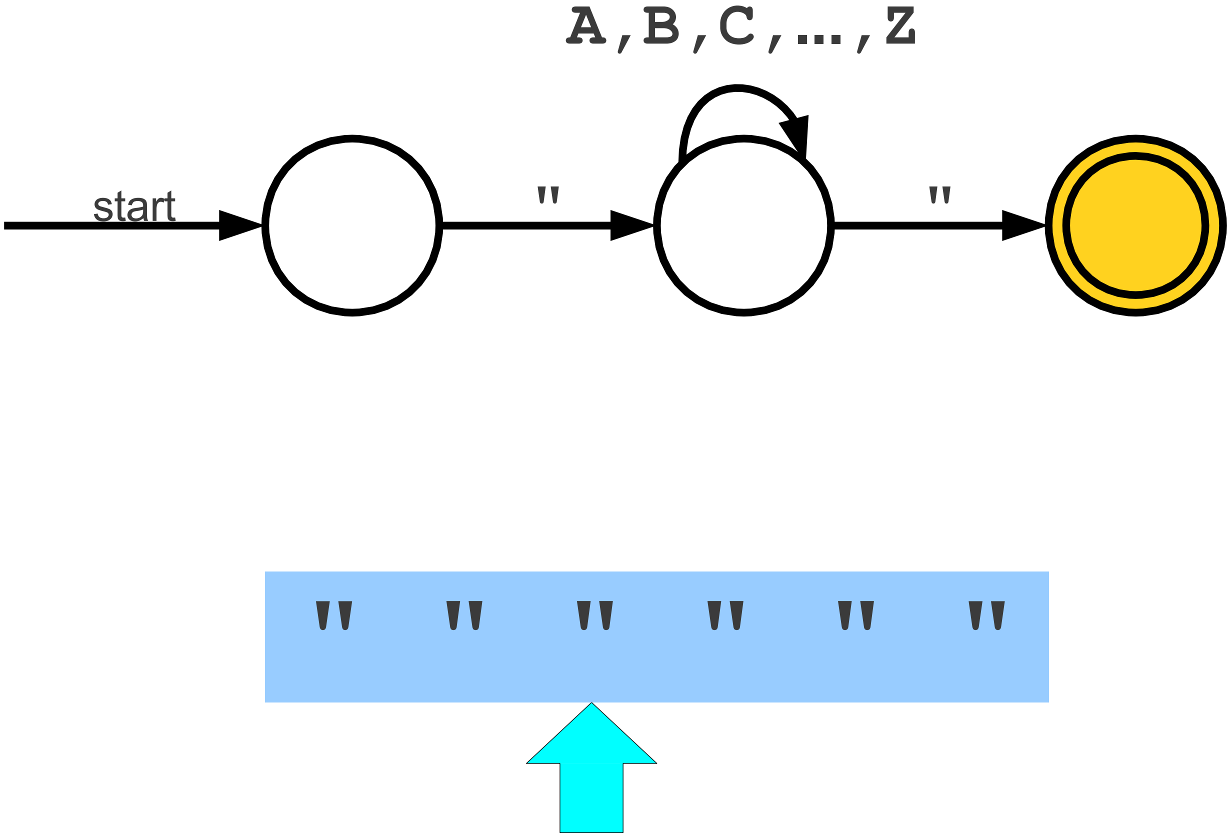


A Simple Automaton

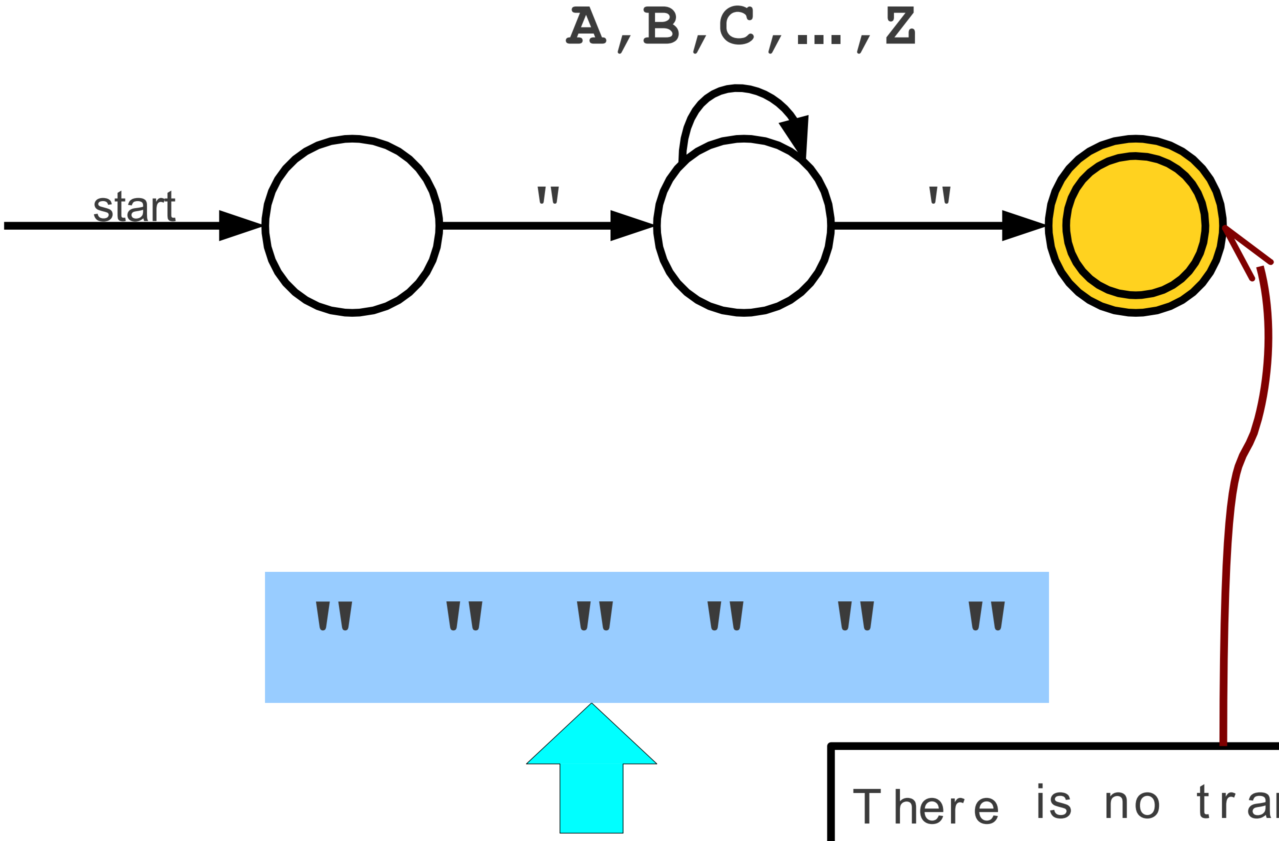




A Simple Automaton

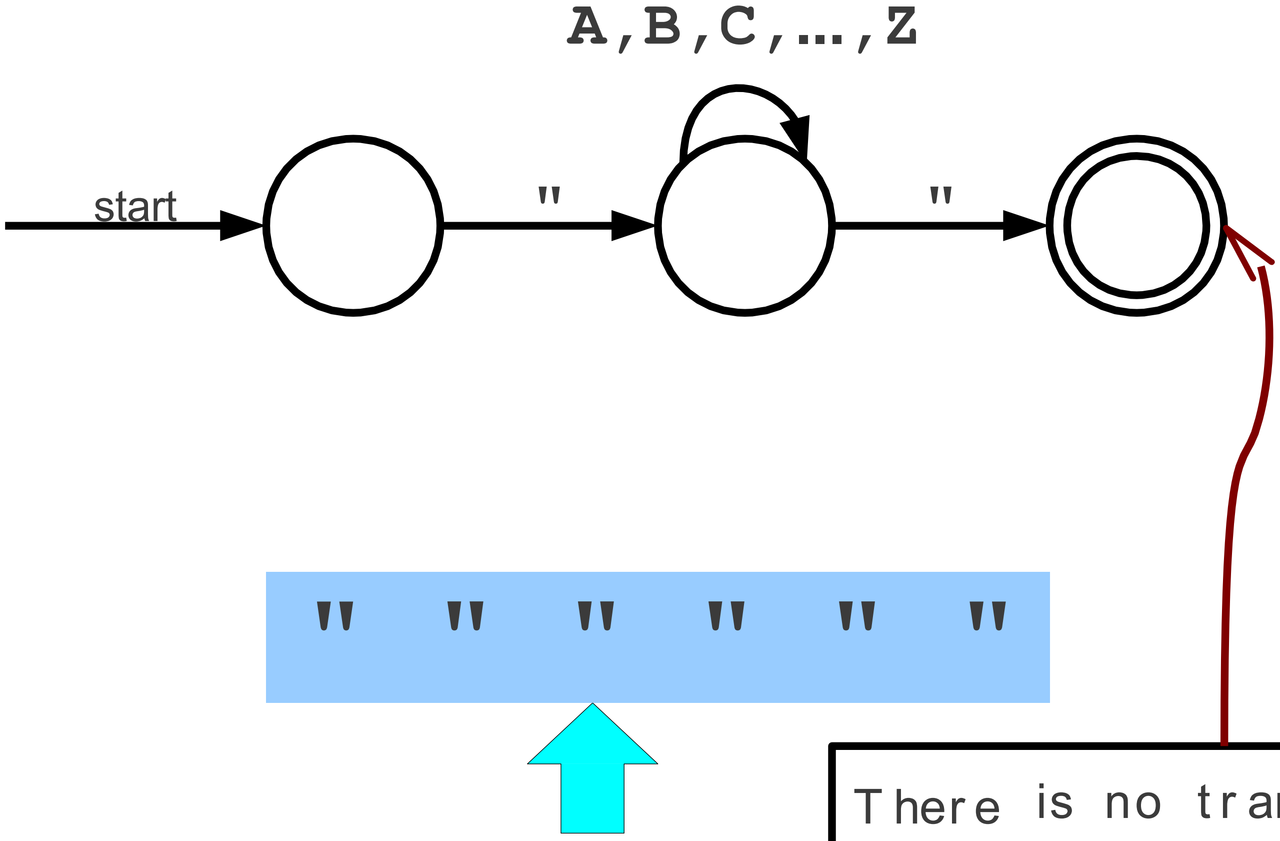


A Simple Automaton



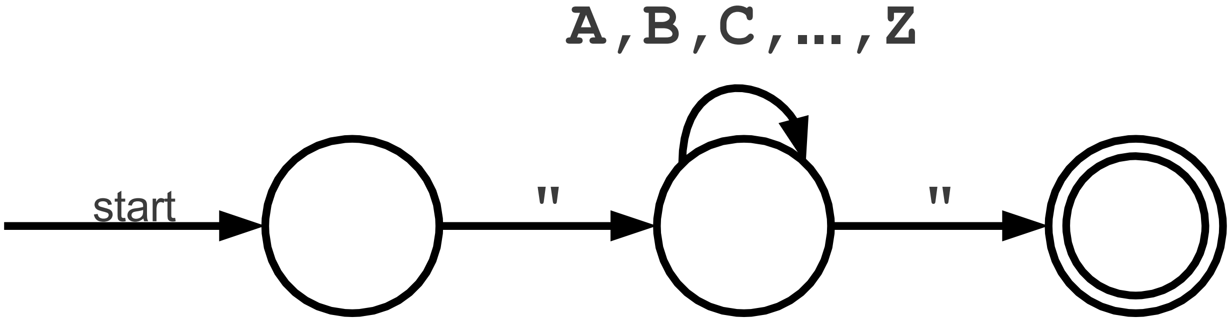
There is no transition on " here, sothe automaton **dies** and rejects.

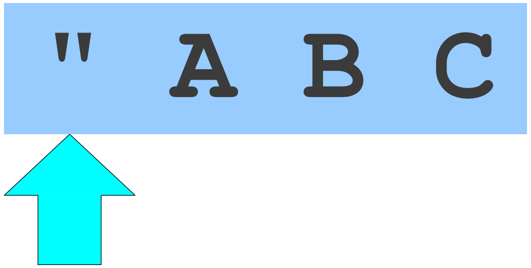
A Simple Automaton



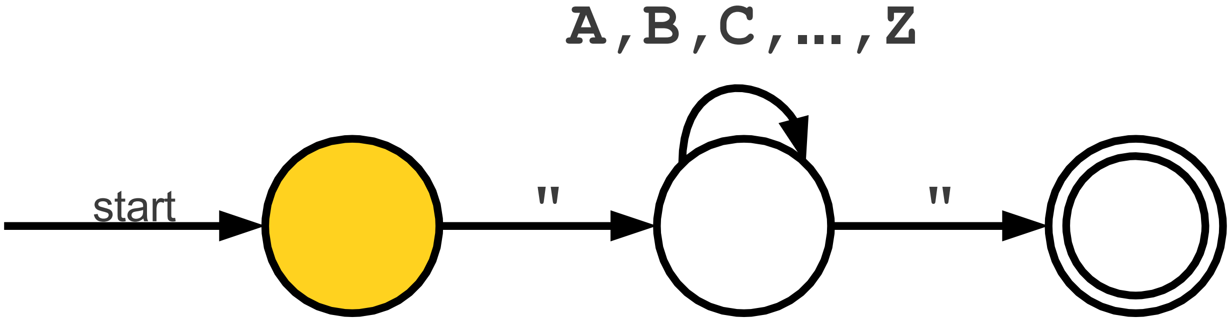
There is no transition on " here, sothe automaton **dies** and rejects.

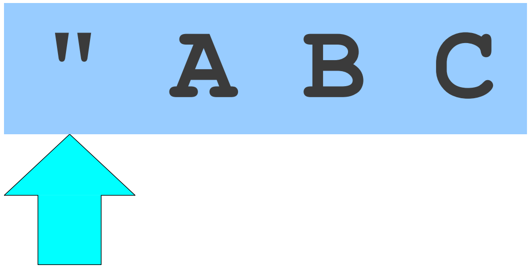
A Simple Automaton



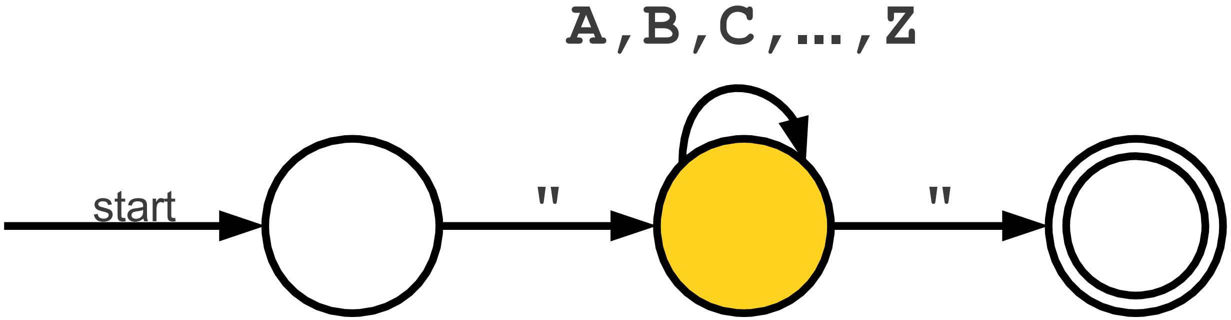

" A B C

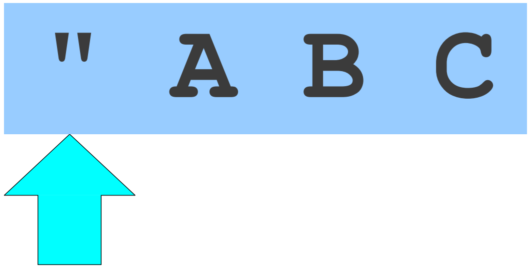
A Simple Automaton



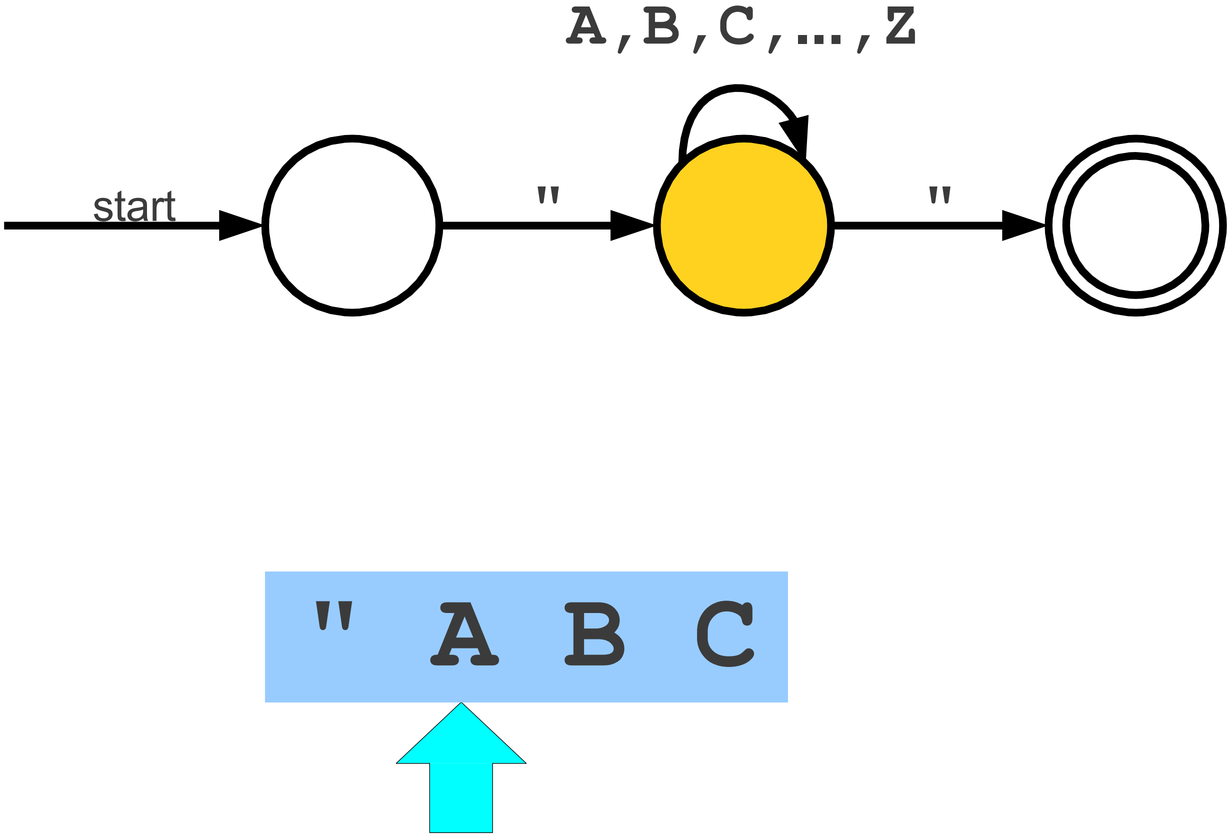


A Simple Automaton

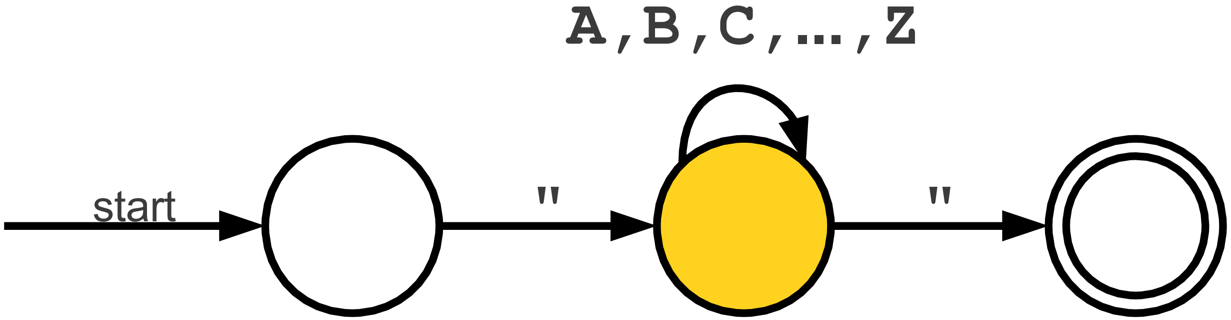


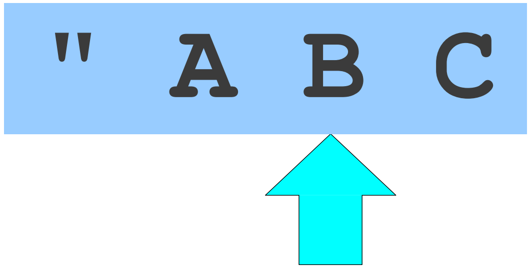


A Simple Automaton

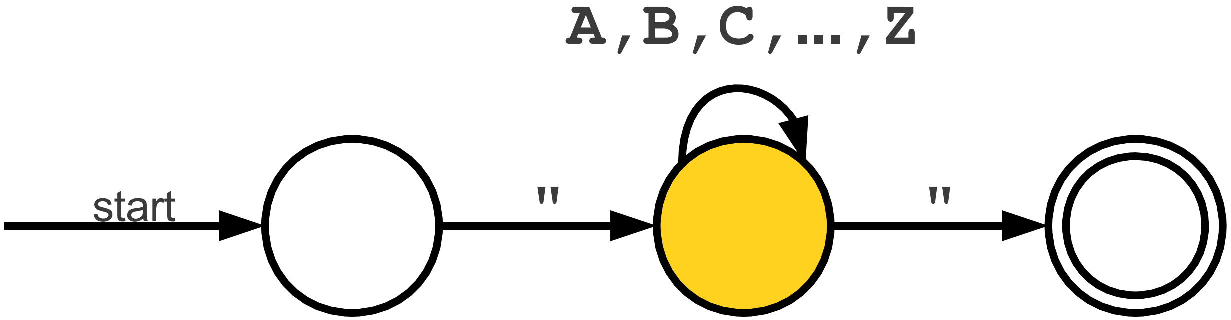


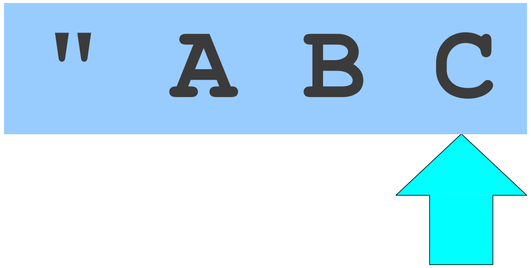
A Simple Automaton



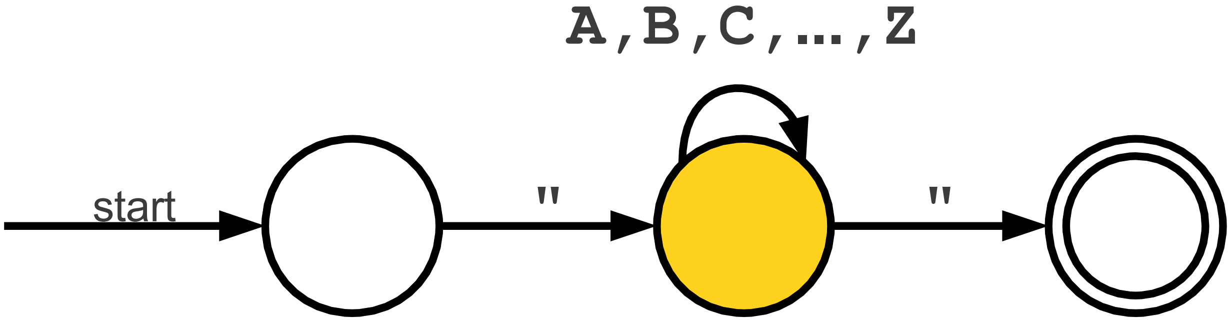


A Simple Automaton



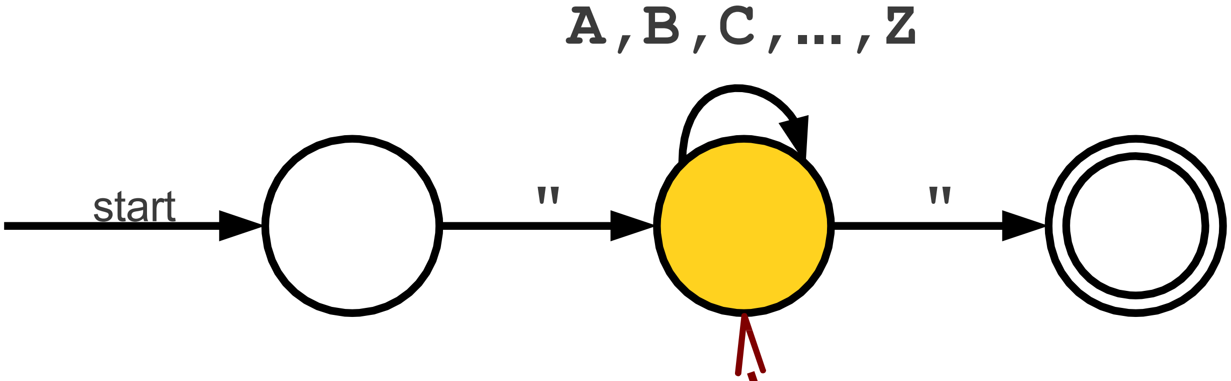


A Simple Automaton



" A B C

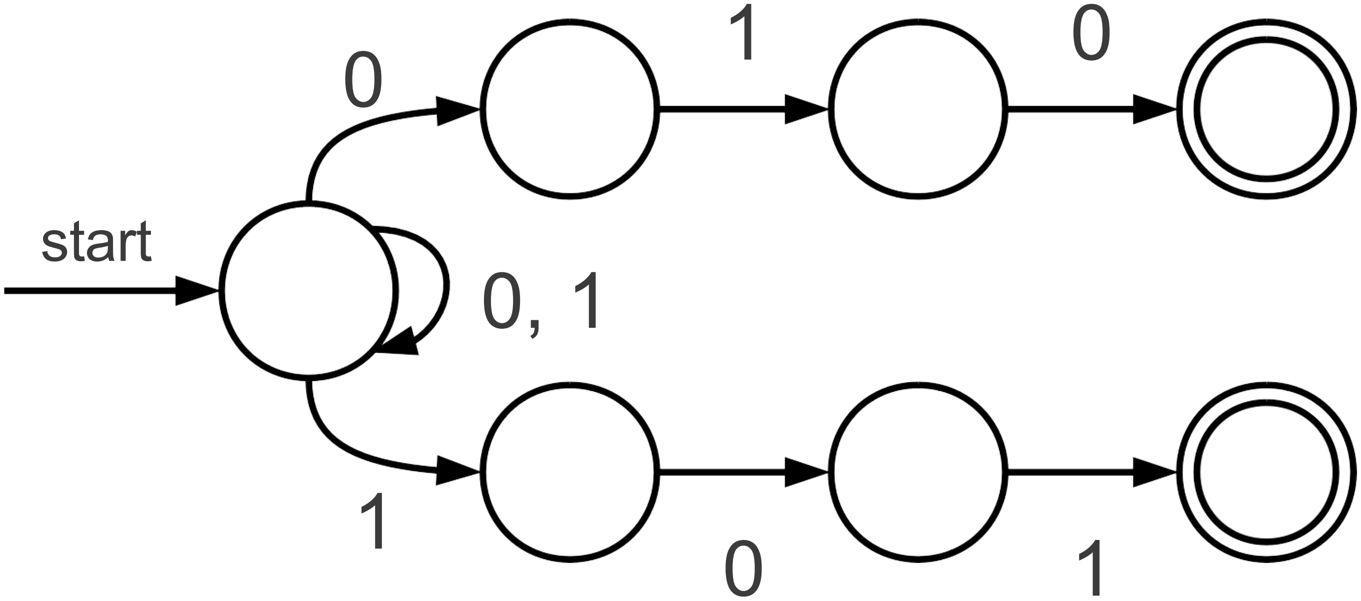
A Simple Automaton



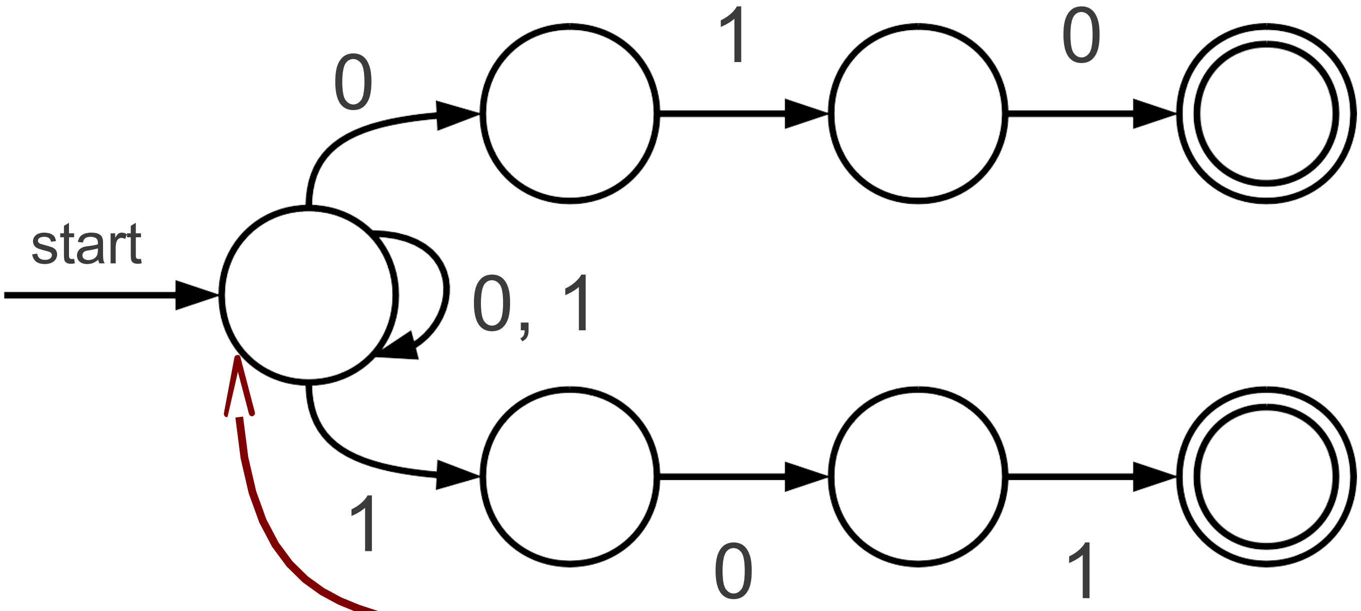
" A B C

This is not an accepting state, so the automation reject.

A More Complex Automaton

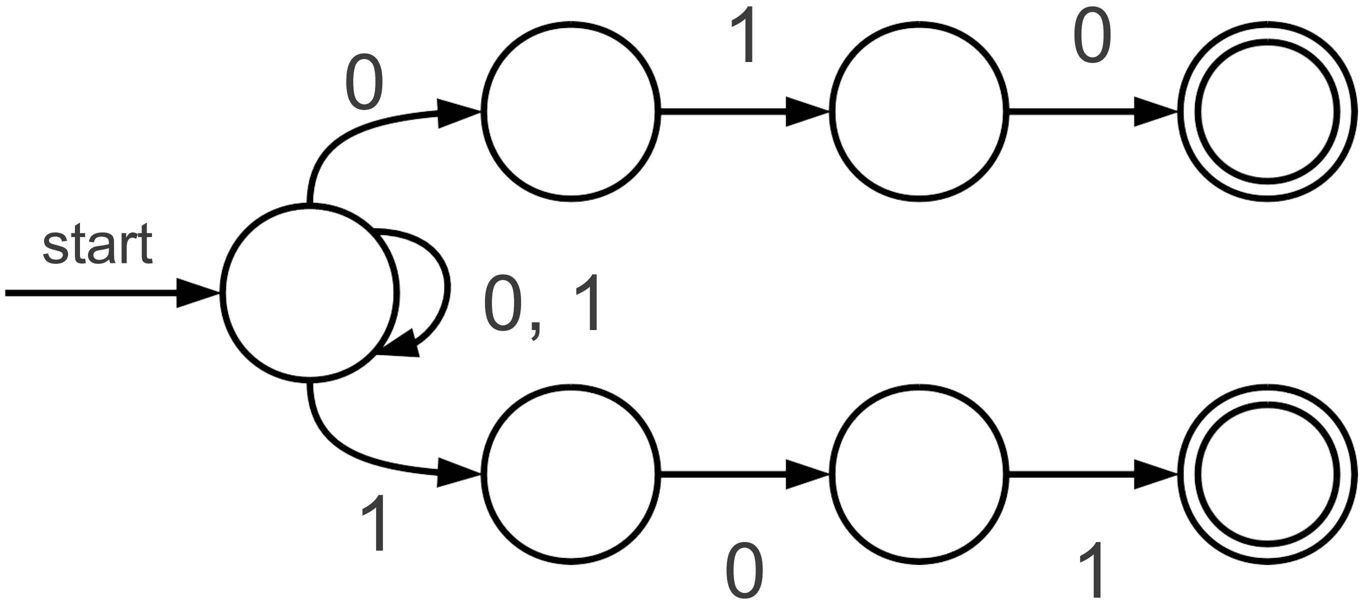


A More Complex Automaton

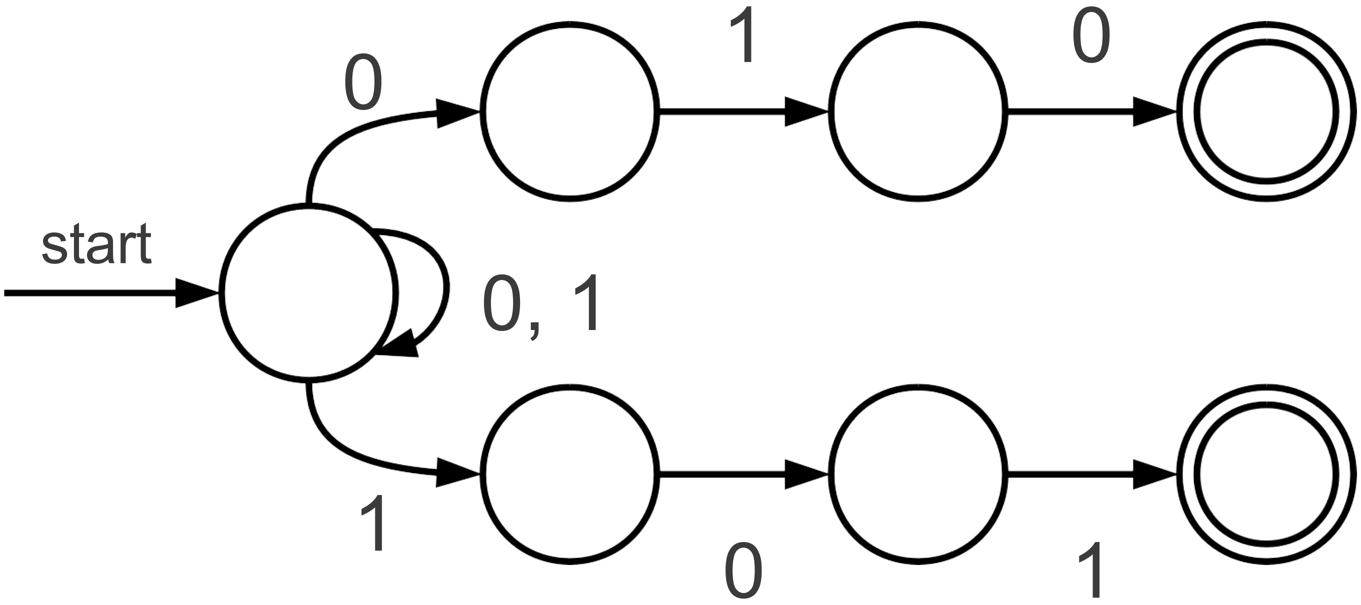


Notice that there are multiple transactions defined here on 0 and 1. If we read a 0 or 1 here, we follow both transition and enter multiple paths.

A More Complex Automaton

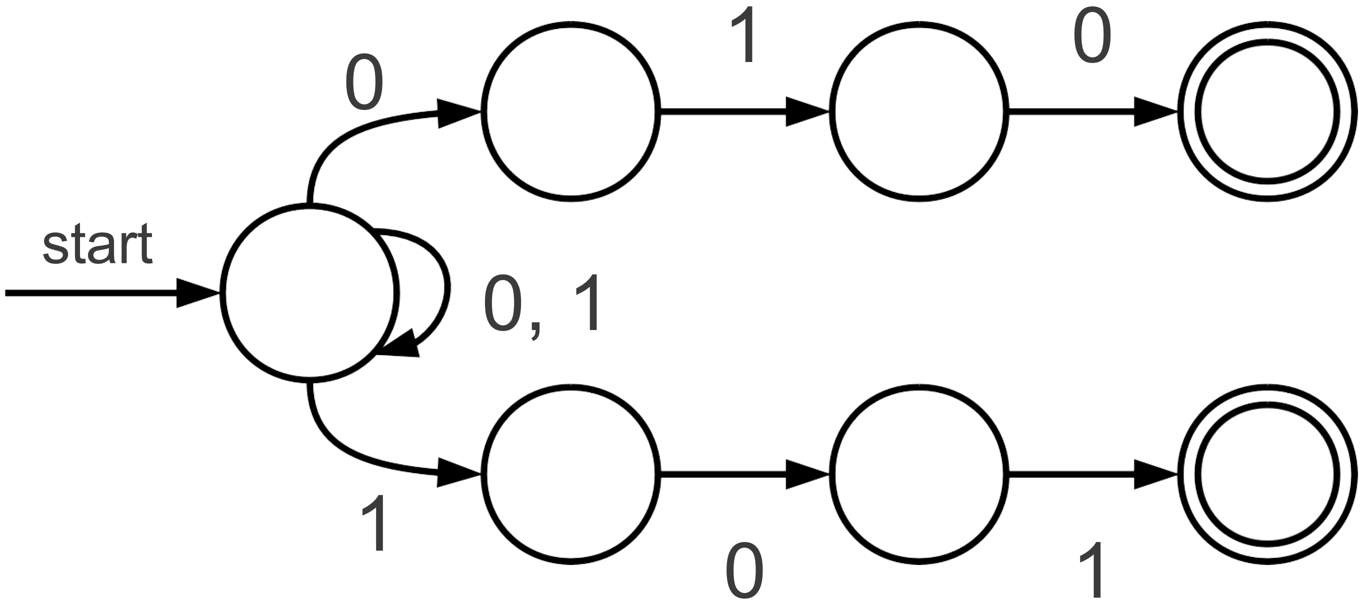


A More Complex Automaton



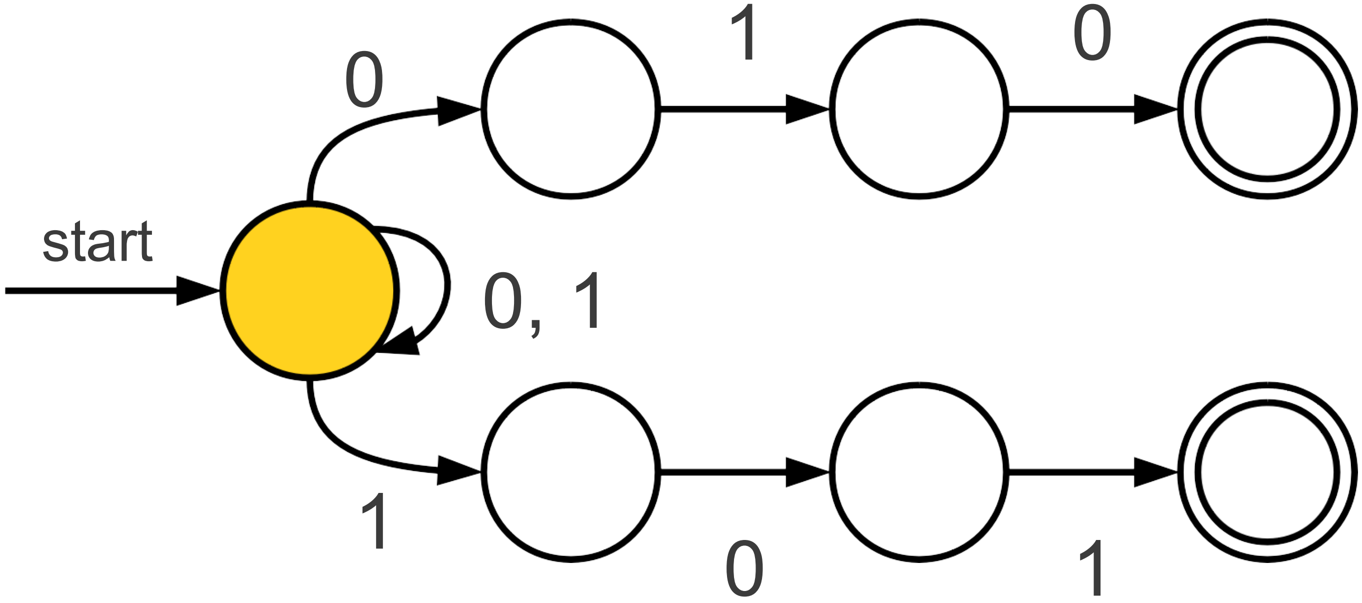
0 1 1 1 0 1

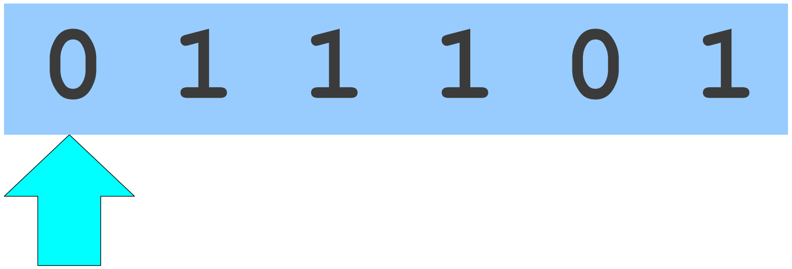
A More Complex Automaton



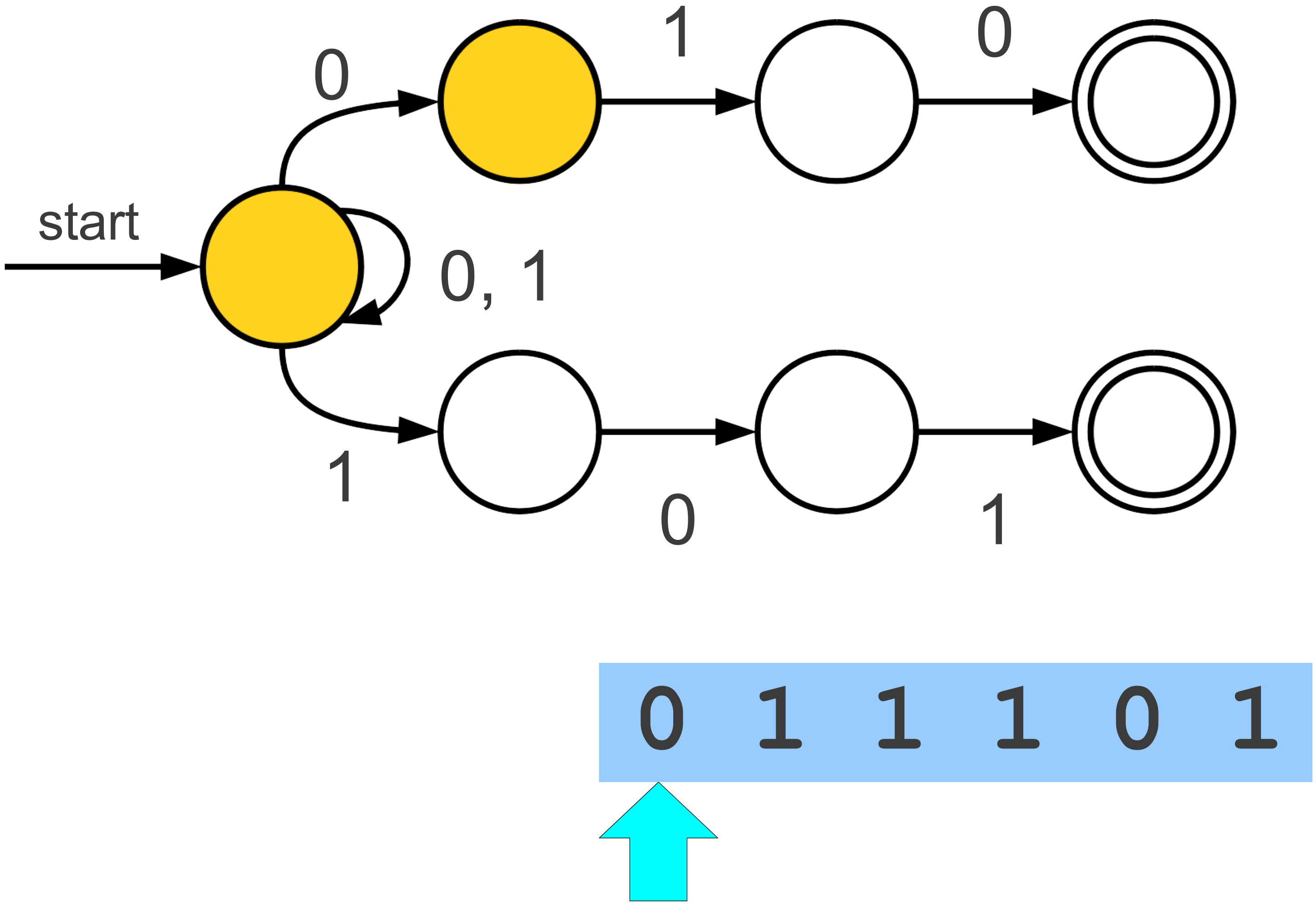
0 1 1 1 0 1

A More Complex Automaton



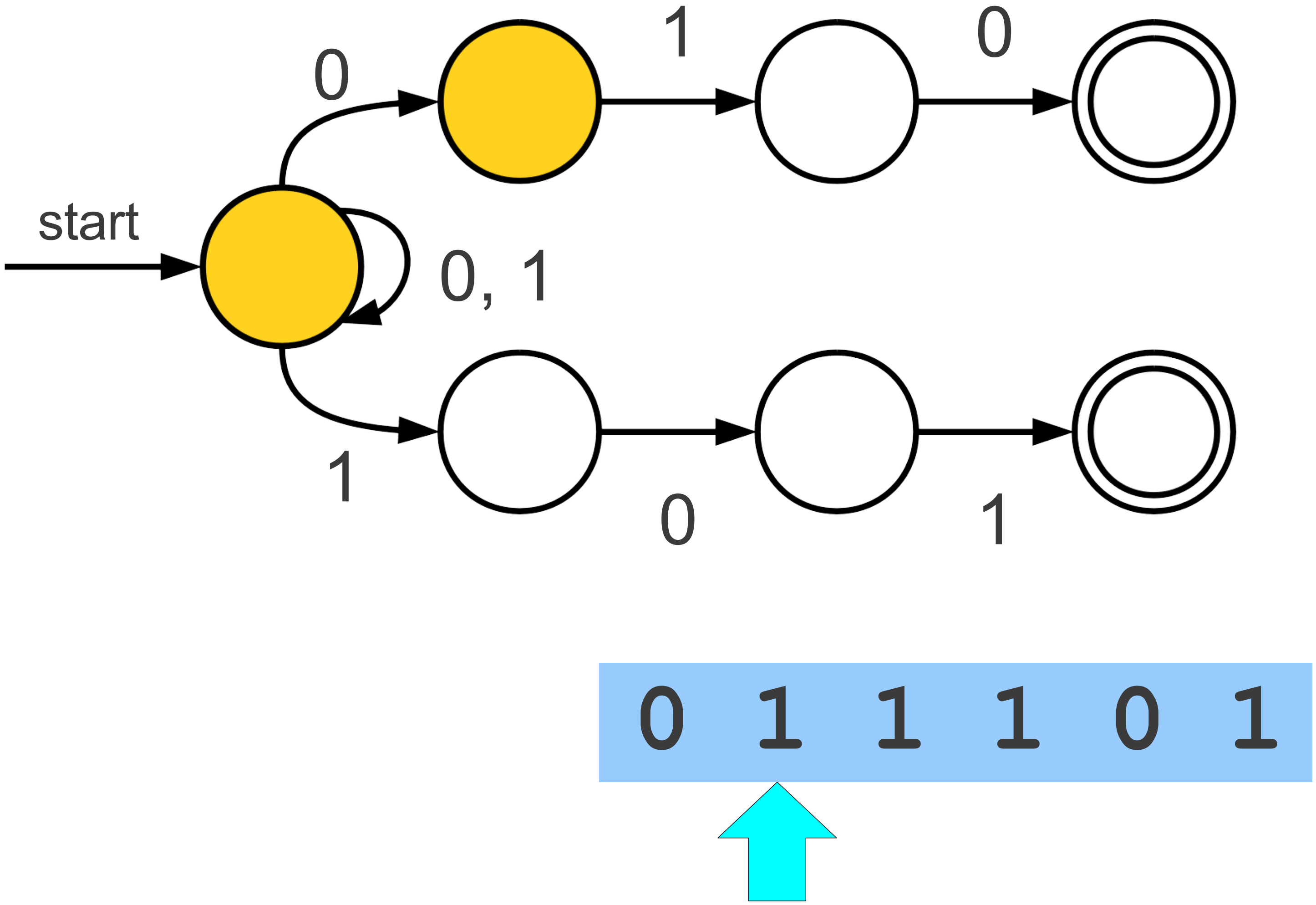


A More Complex Automaton

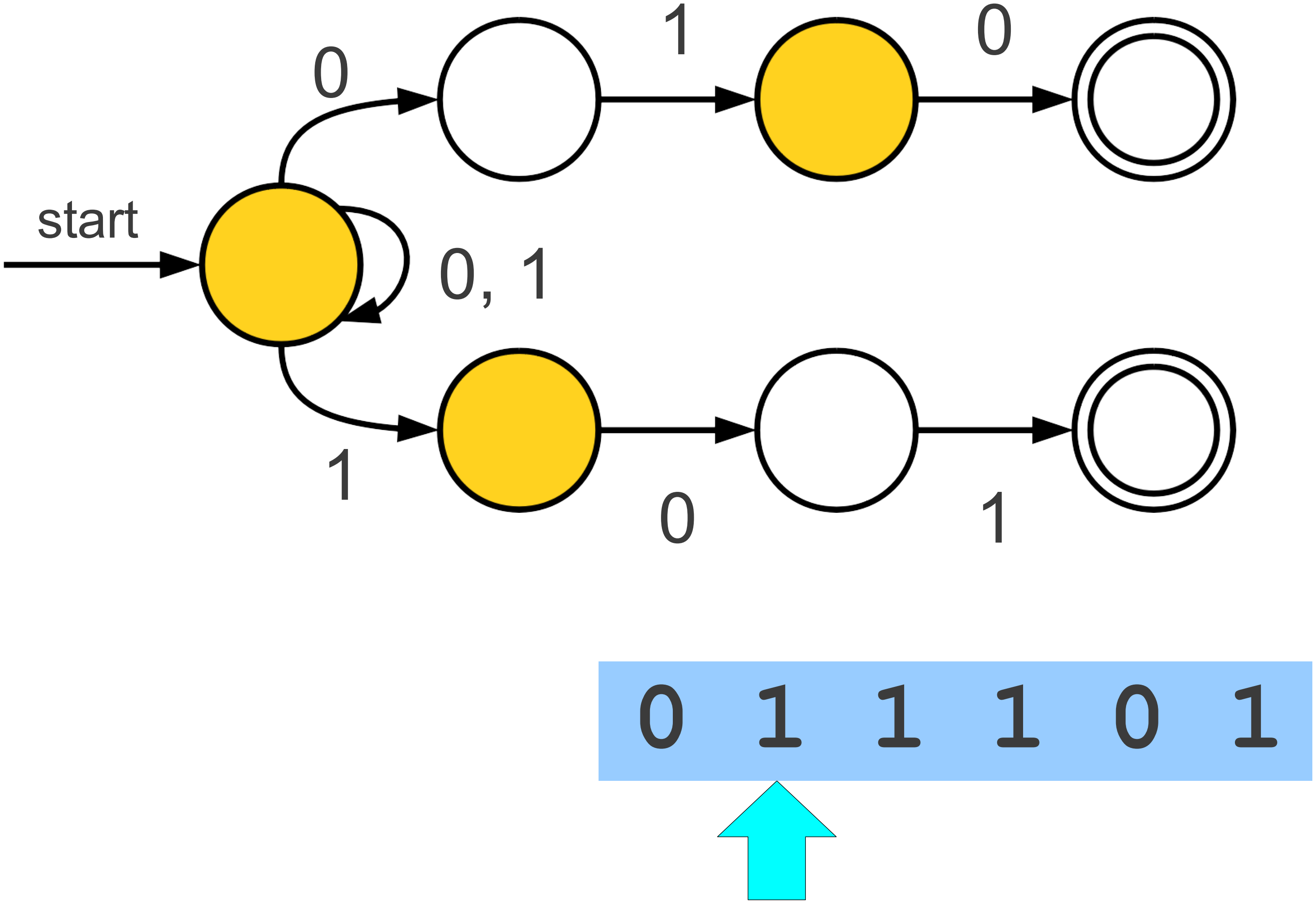


0 1 1 1 0 1

A More Complex Automaton

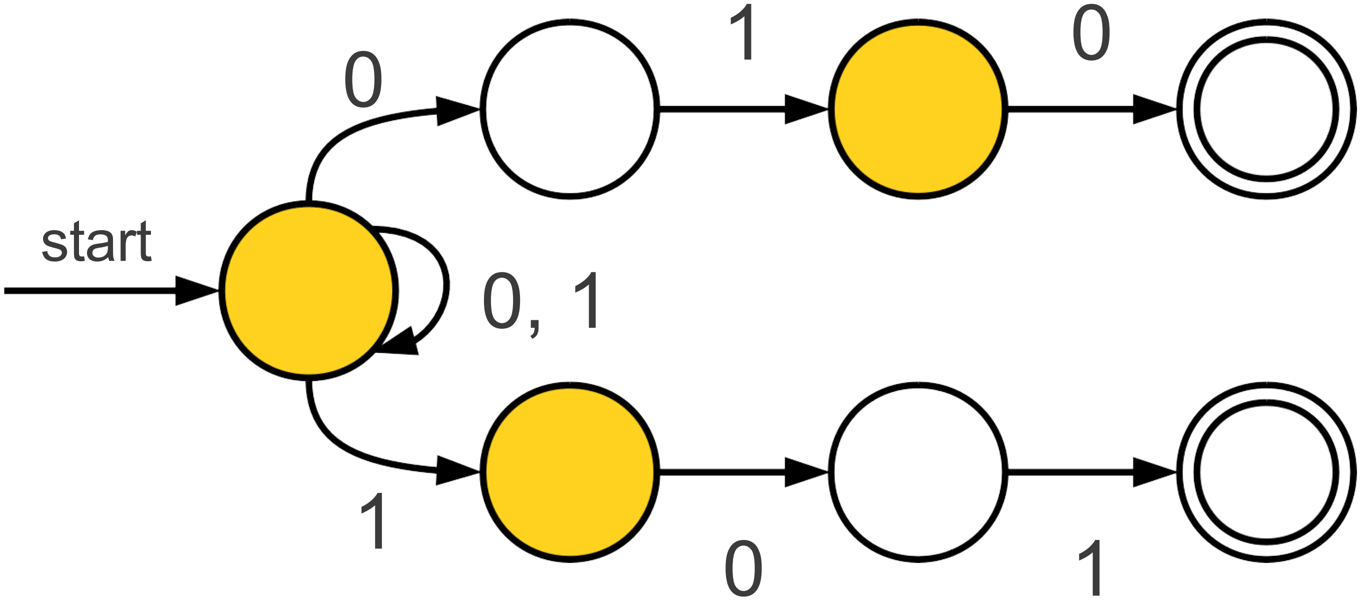


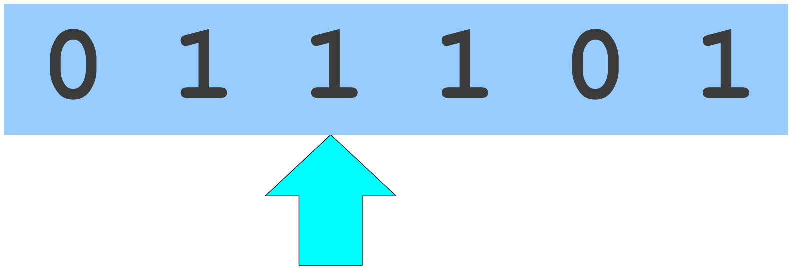
A More Complex Automaton



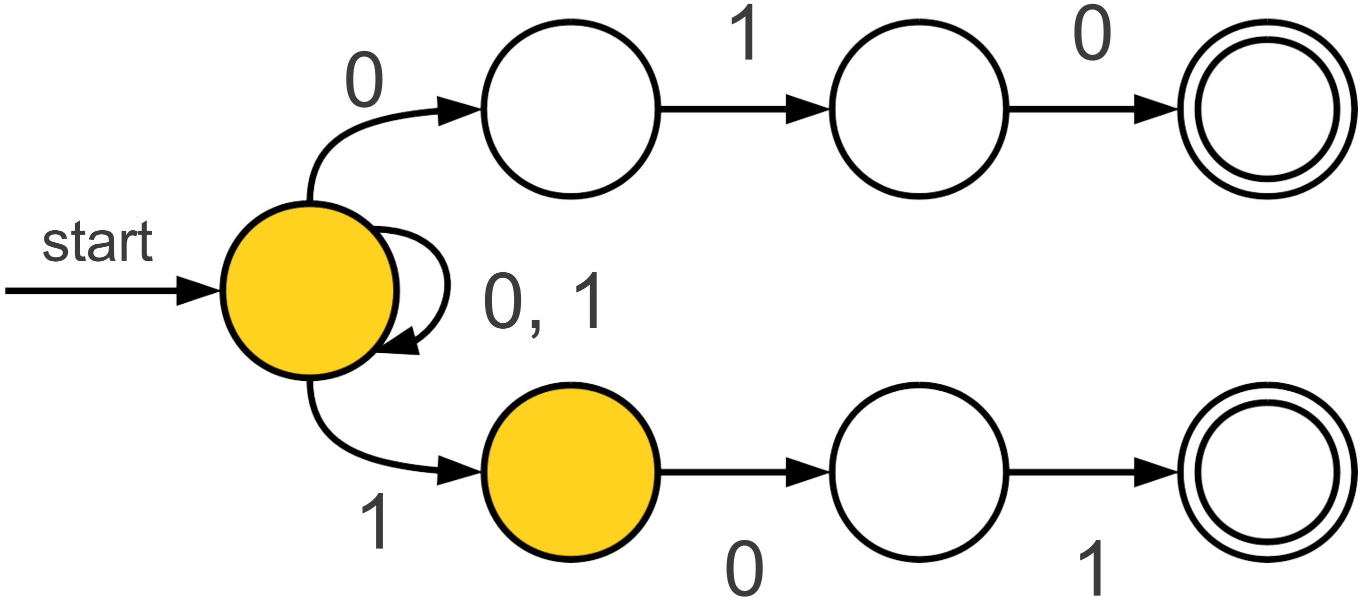
0 1 1 1 0 1

A More Complex Automaton



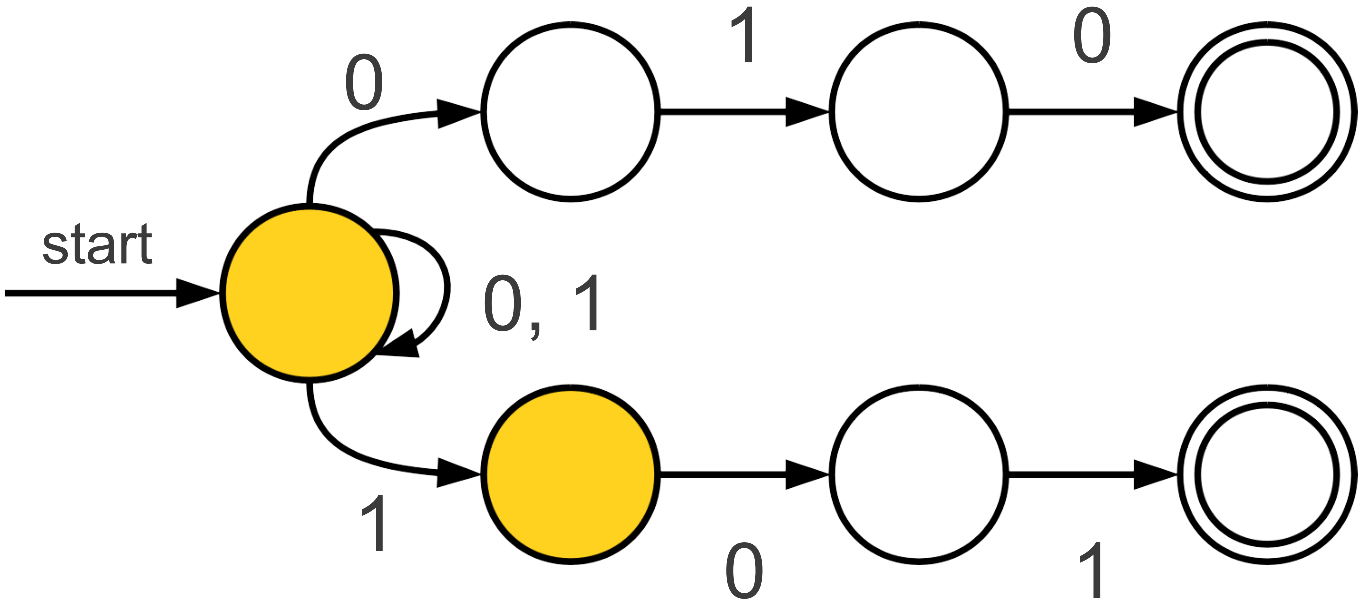


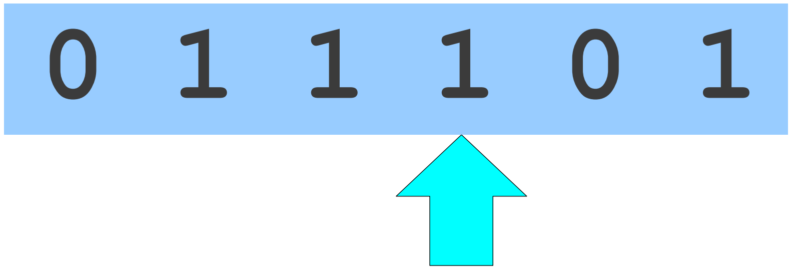
A More Complex Automaton



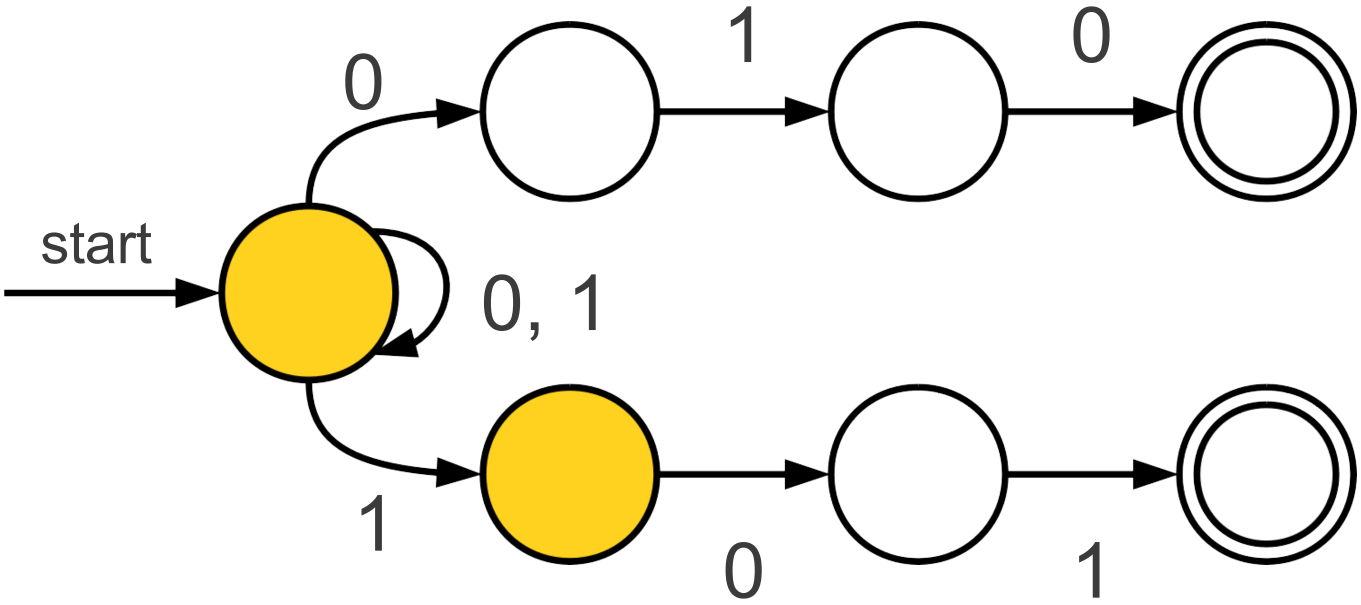
0 1 1 1 0 1

A More Complex Automaton



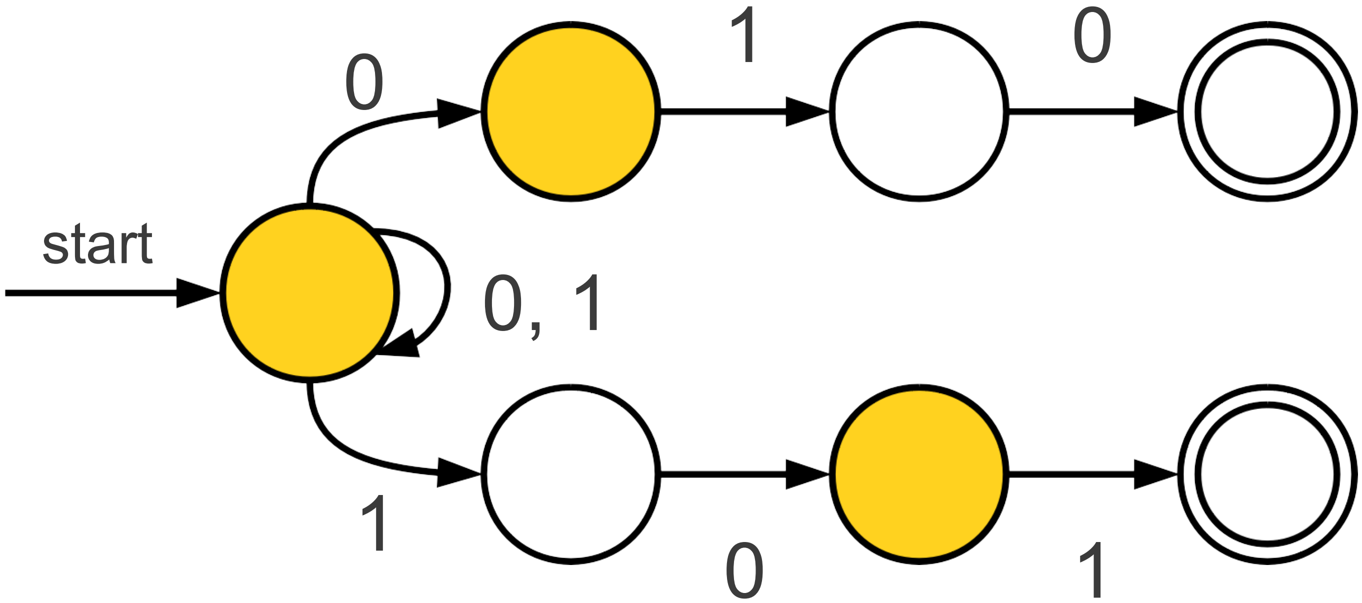


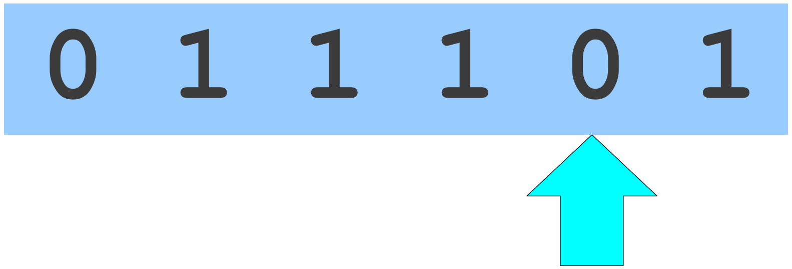
A More Complex Automaton



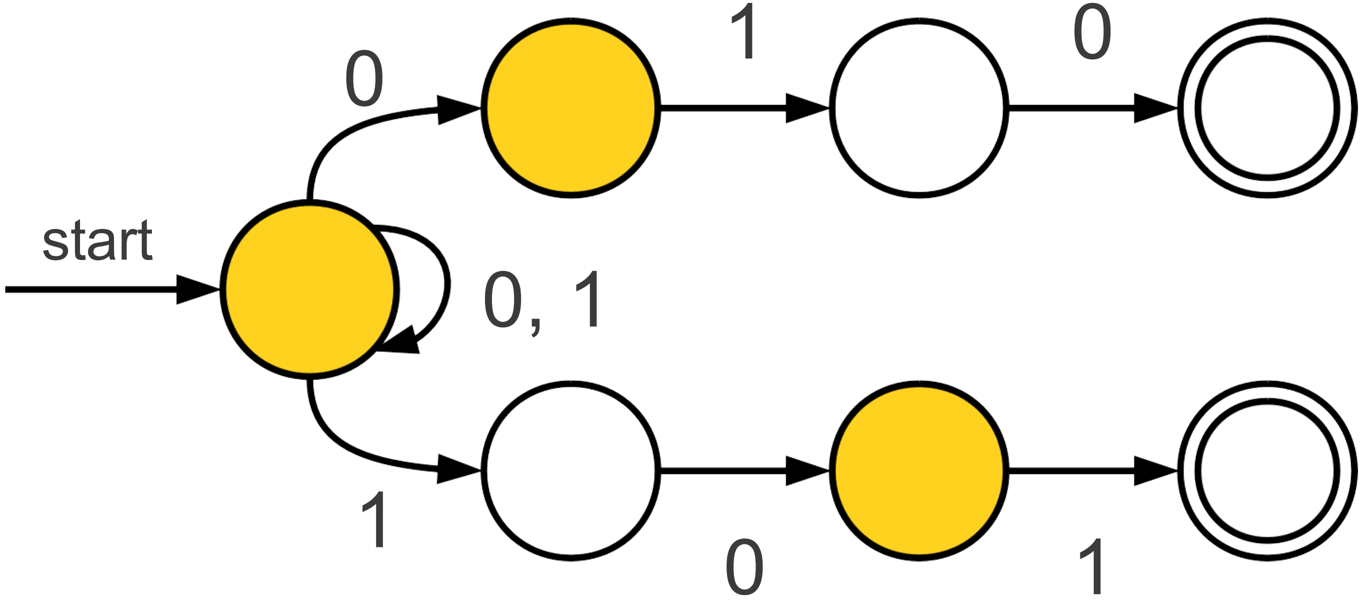
0 1 1 1 0 1

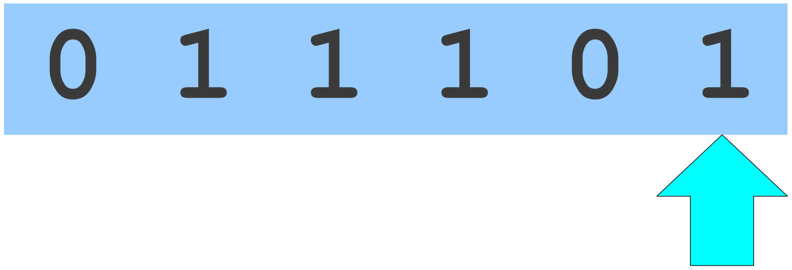
A More Complex Automaton



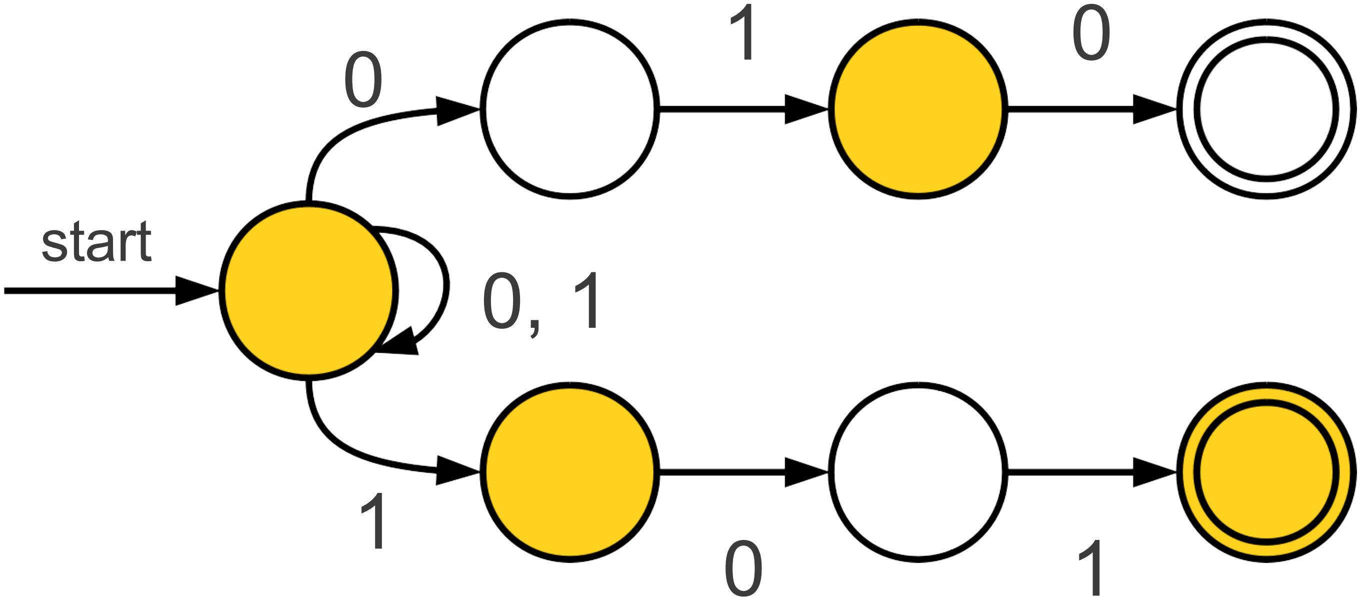


A More Complex Automaton



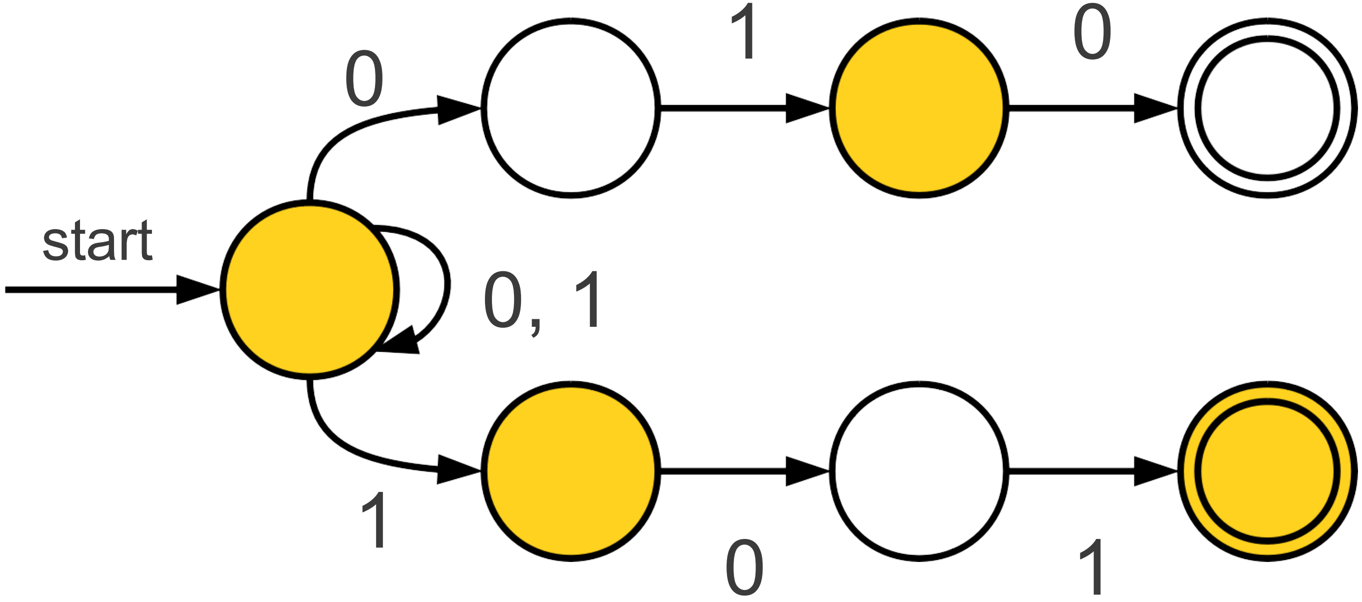


A More Complex Automaton



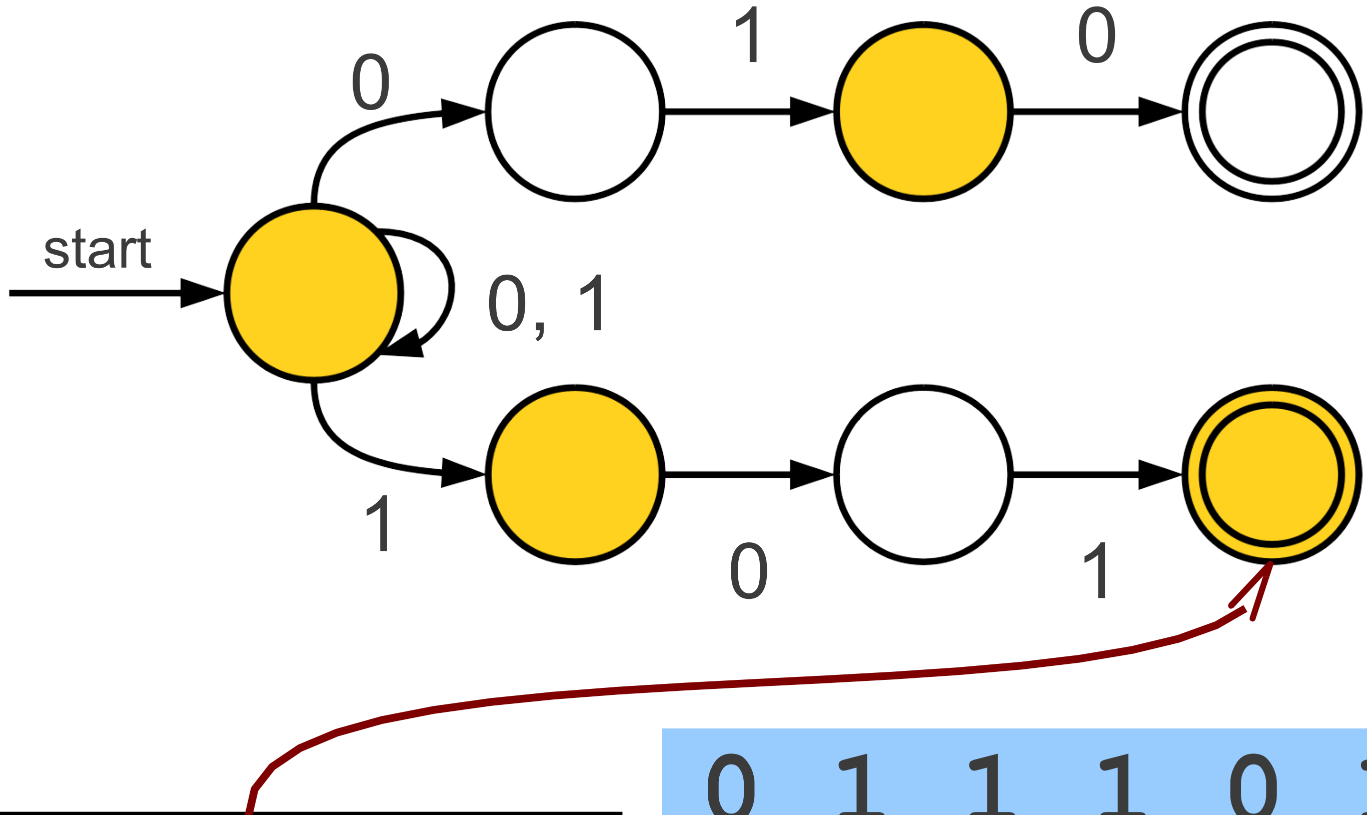
0 1 1 1 0 1

A More Complex Automaton



0 1 1 1 0 1

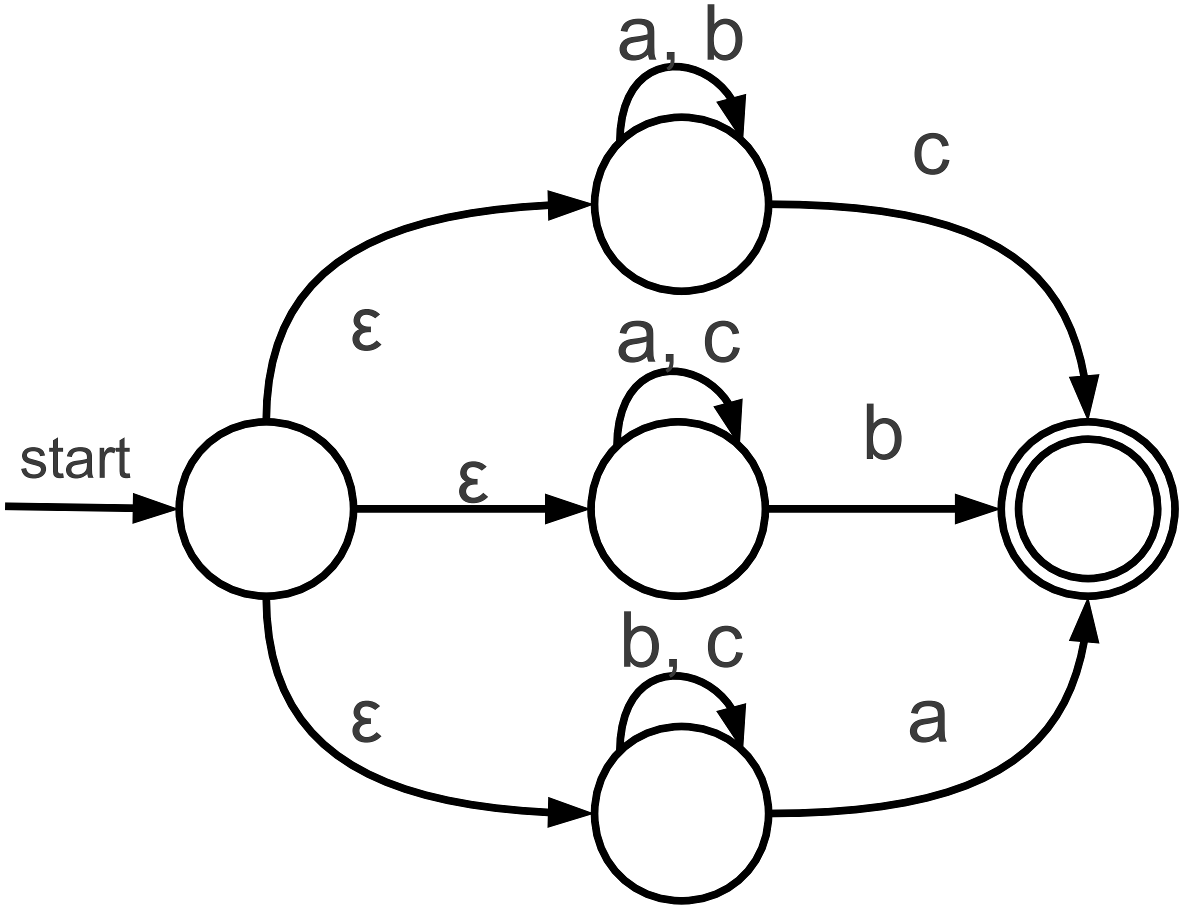
A More Complex Automaton



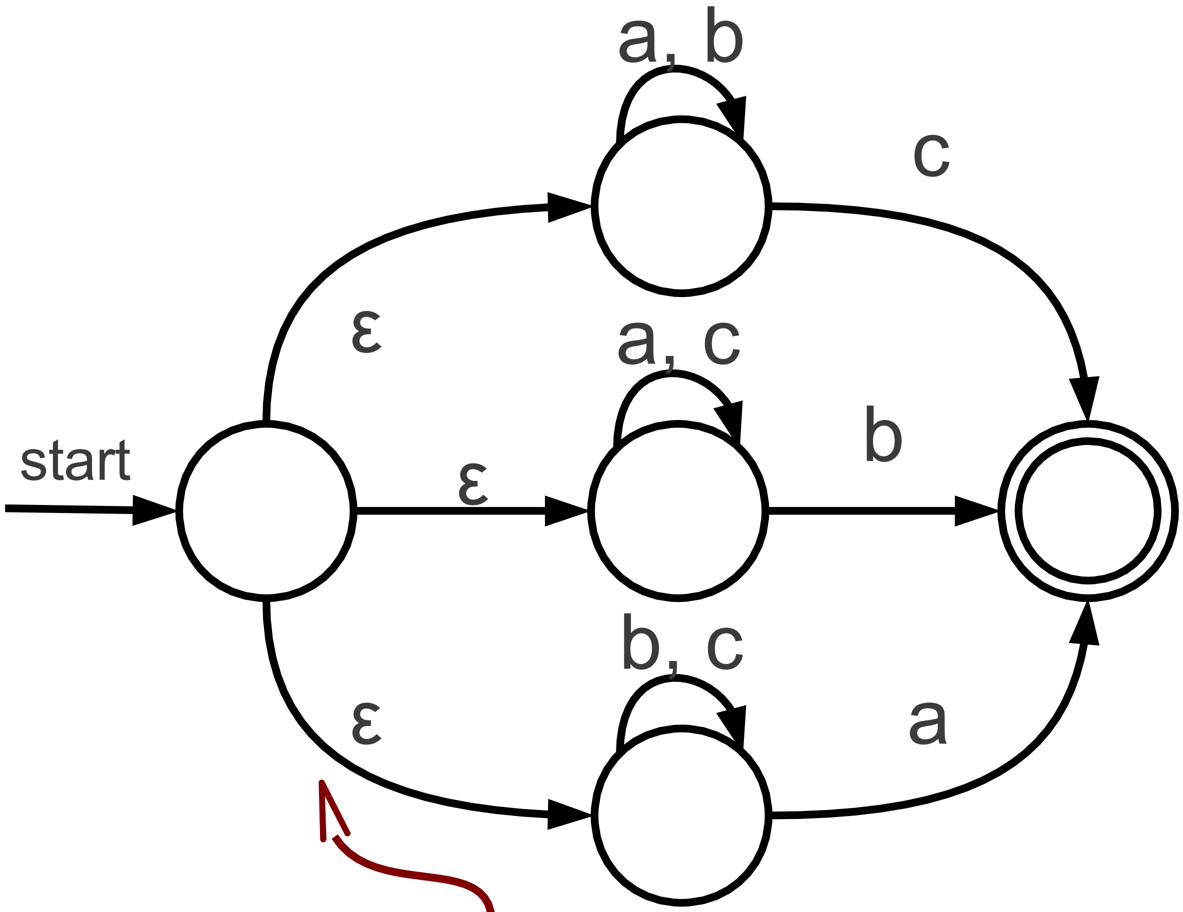
0 1 1 1 0 1

Sincewe are in at least one accepting state, the automaton accepts.

An Even More Complex Automaton

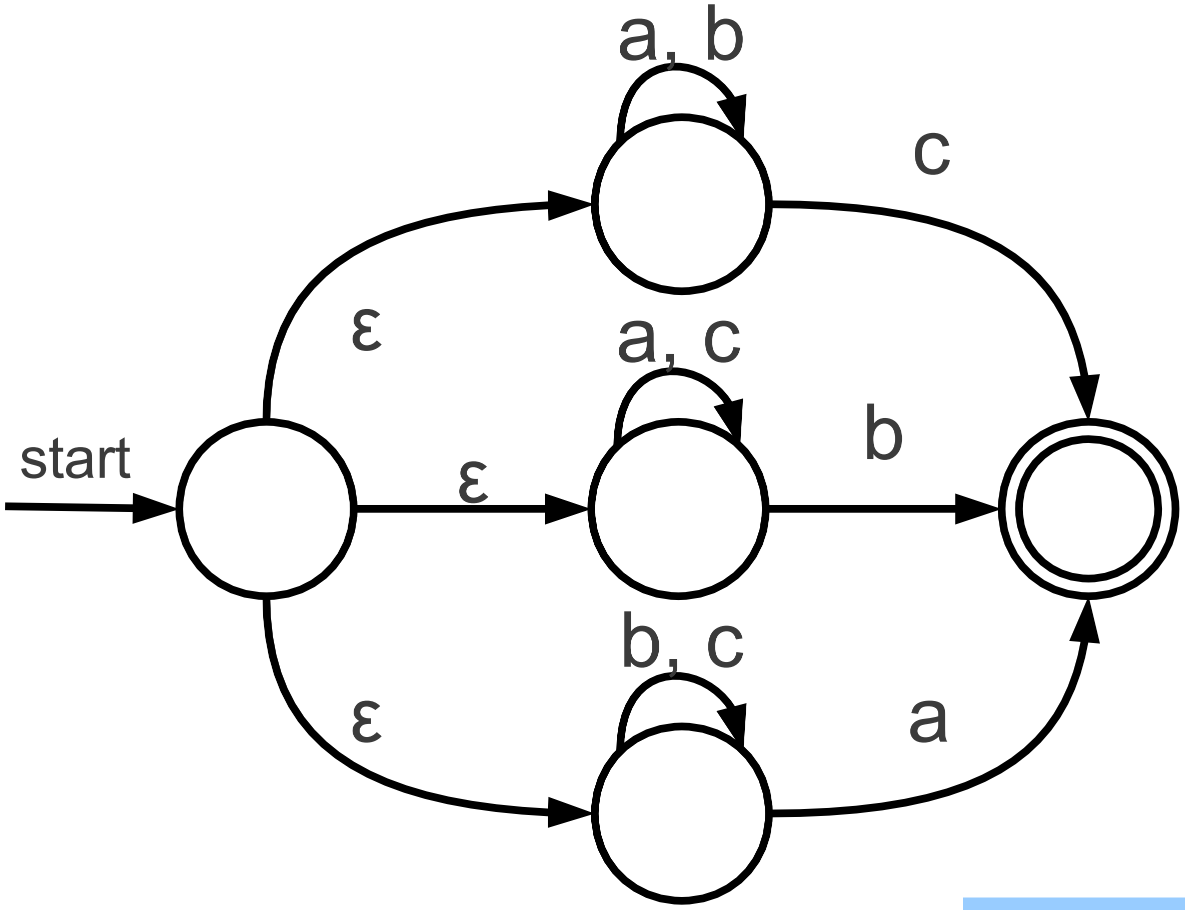


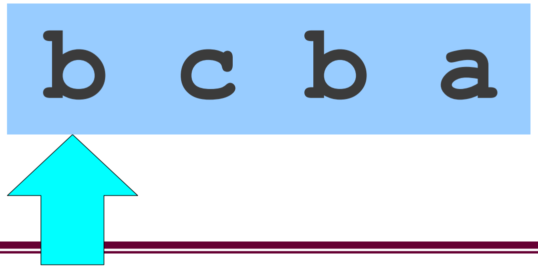
An Even More Complex Automaton



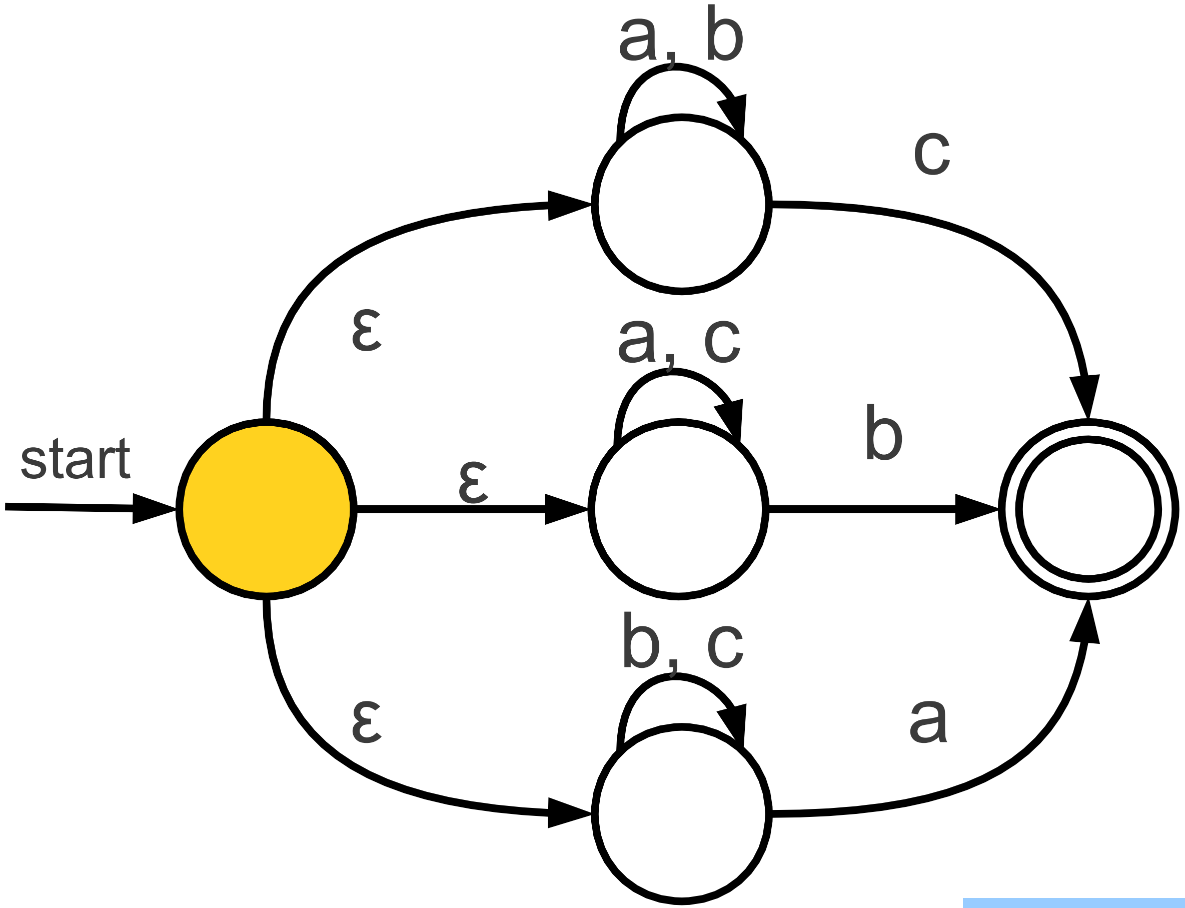
These are called **ε-transitions**. These transitions are followed automatically and without consuming any input.

An Even More Complex Automaton



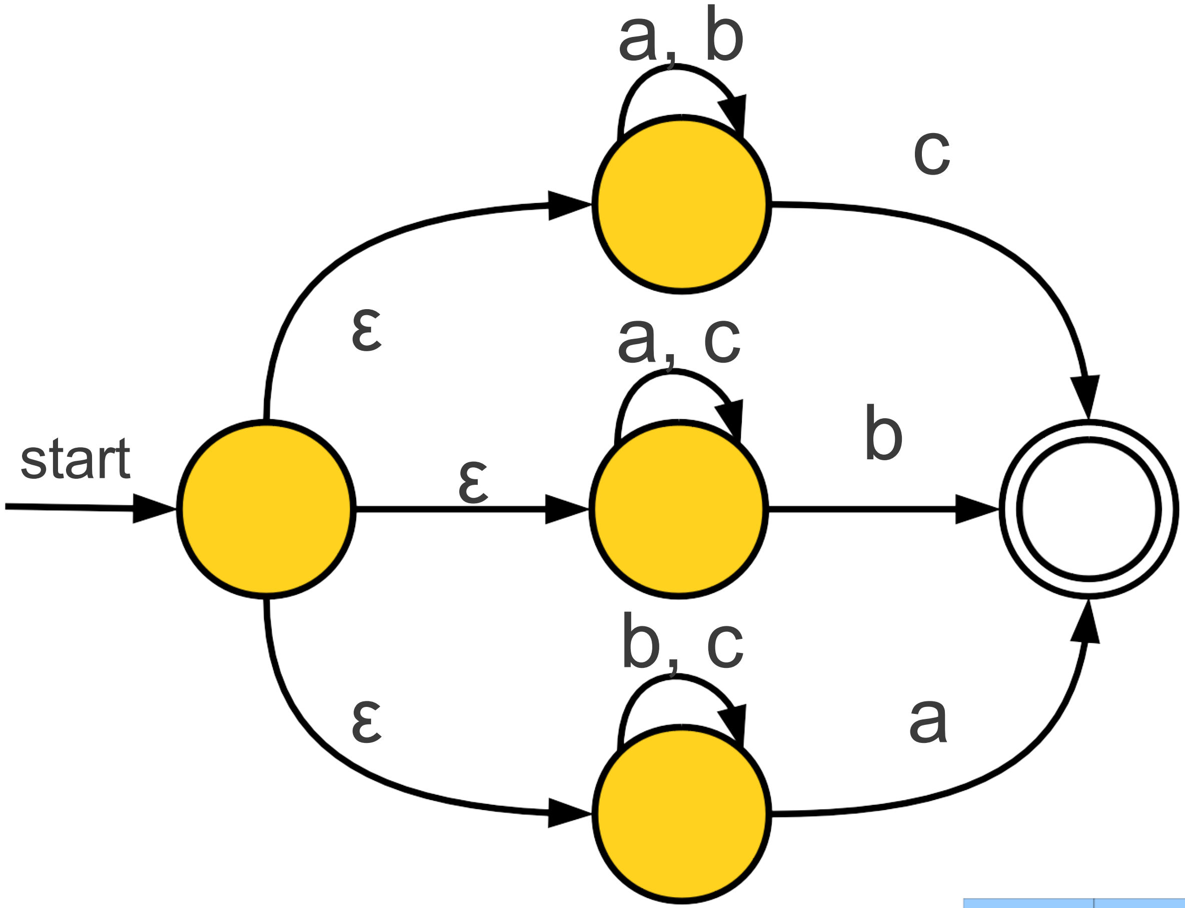


An Even More Complex Automaton



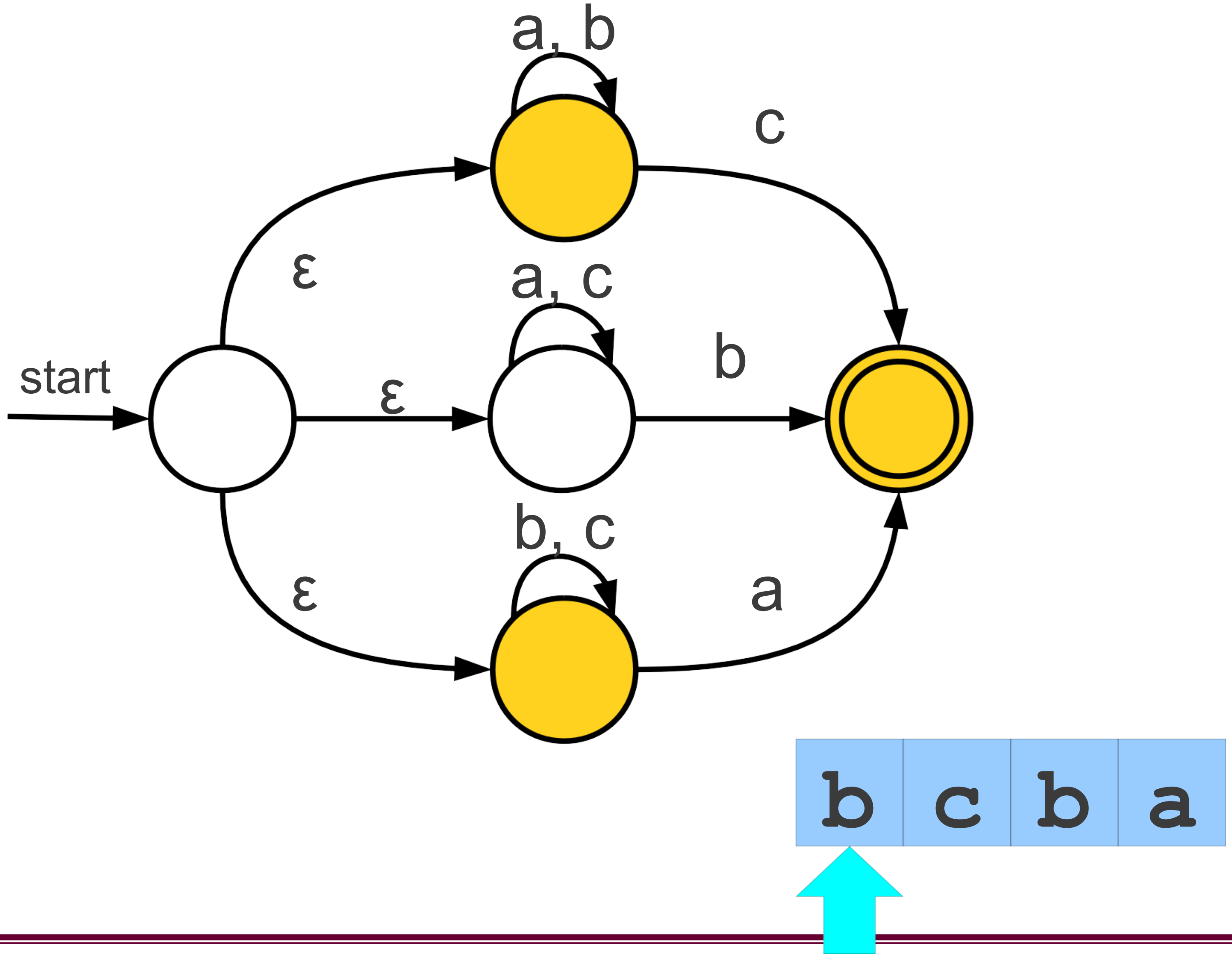
b c b a

An Even More Complex Automaton

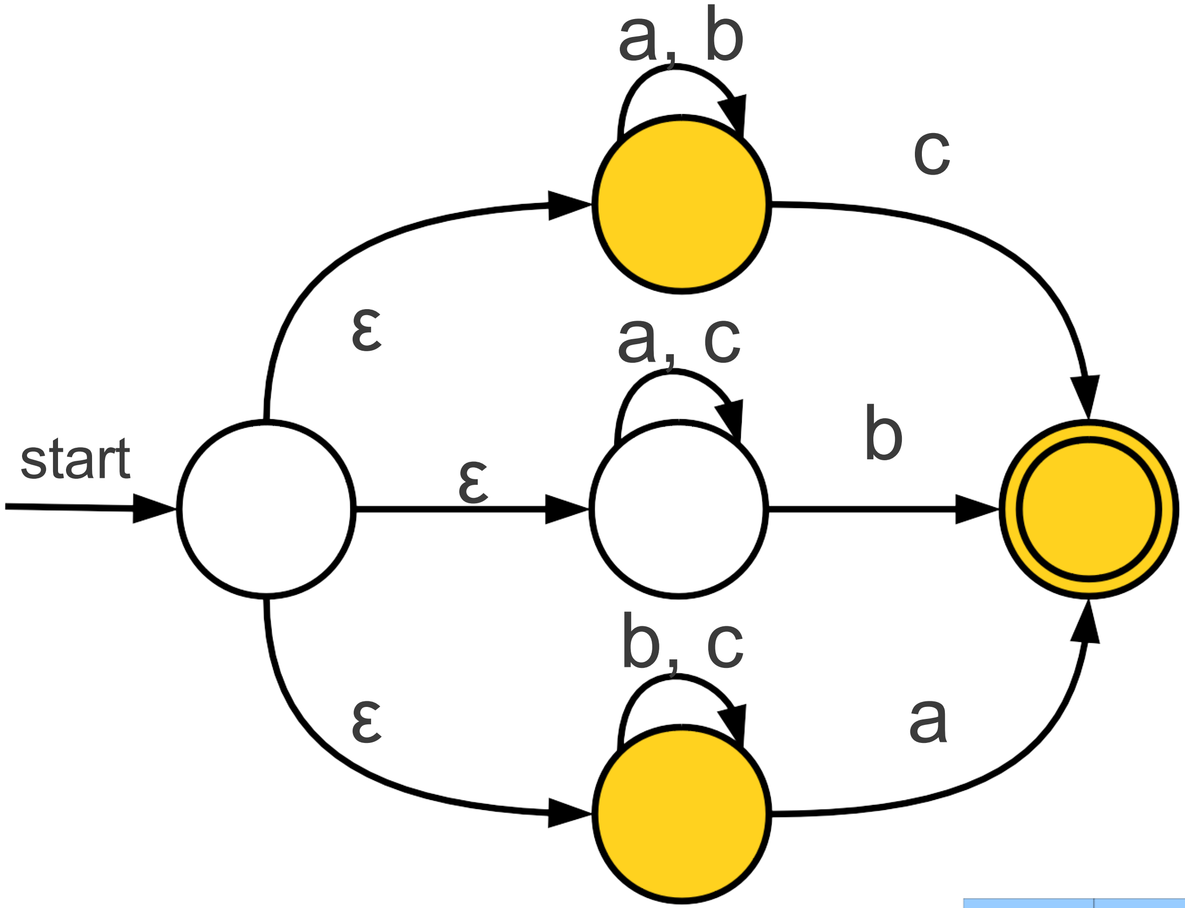


b	c	b	a

An Even More Complex Automaton

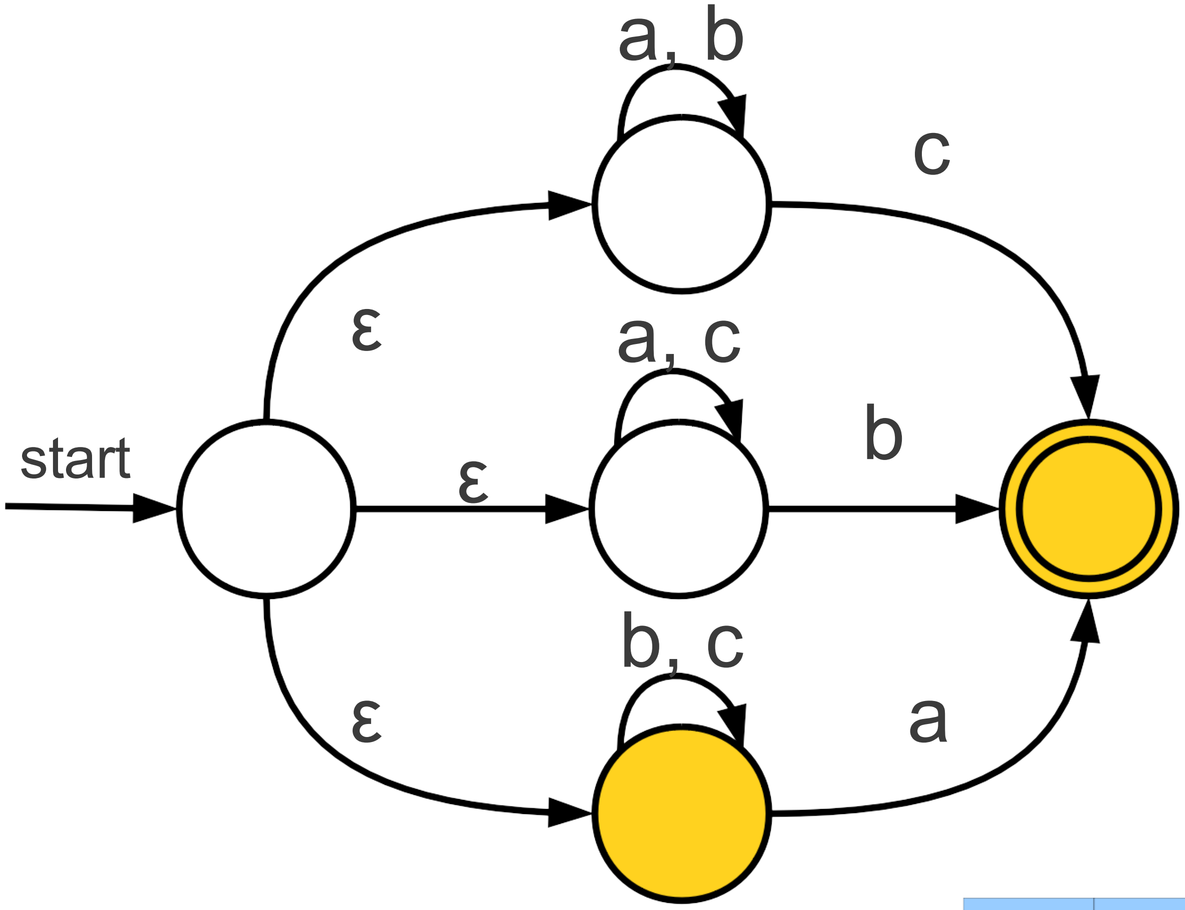


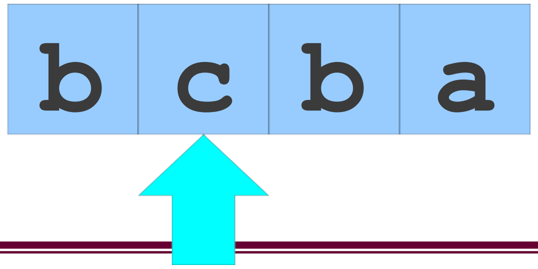
An Even More Complex Automaton



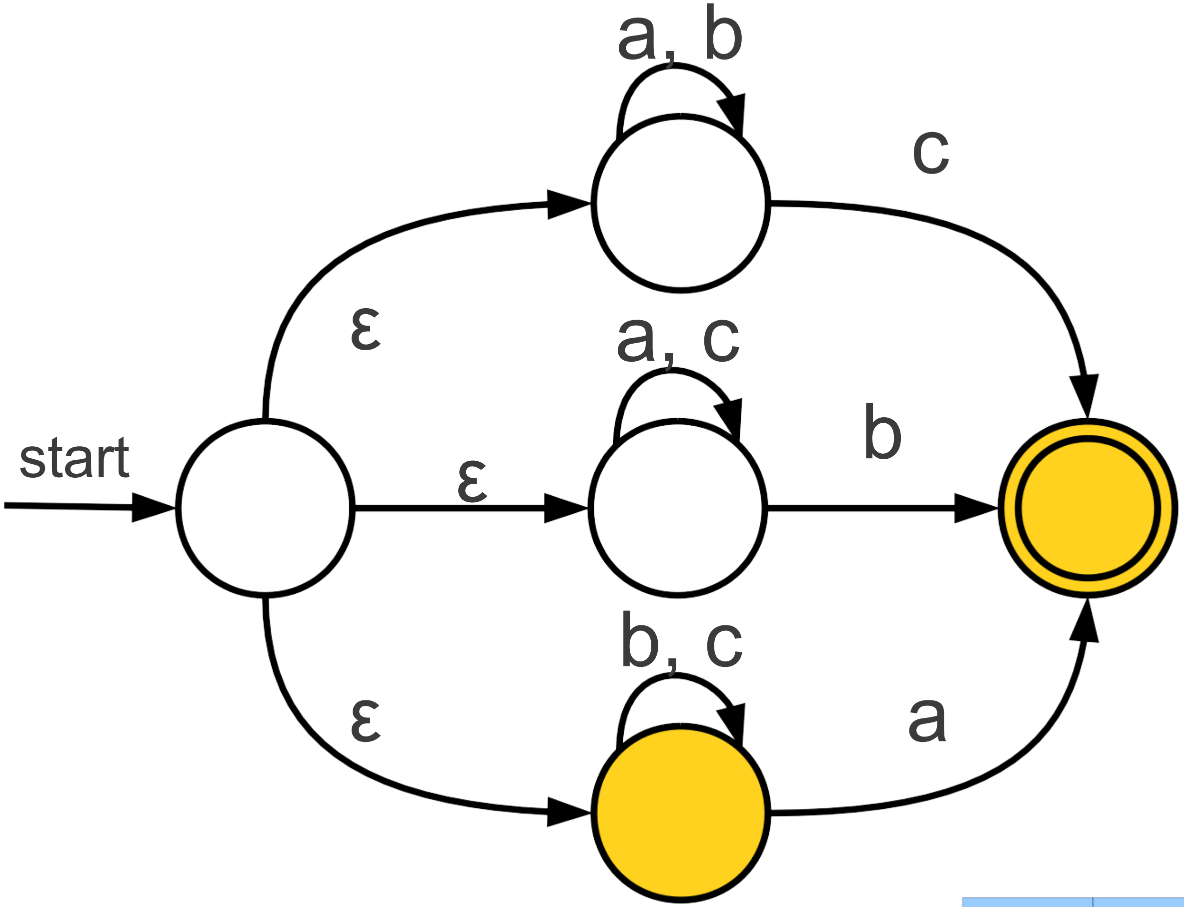
b	c	b	a

An Even More Complex Automaton



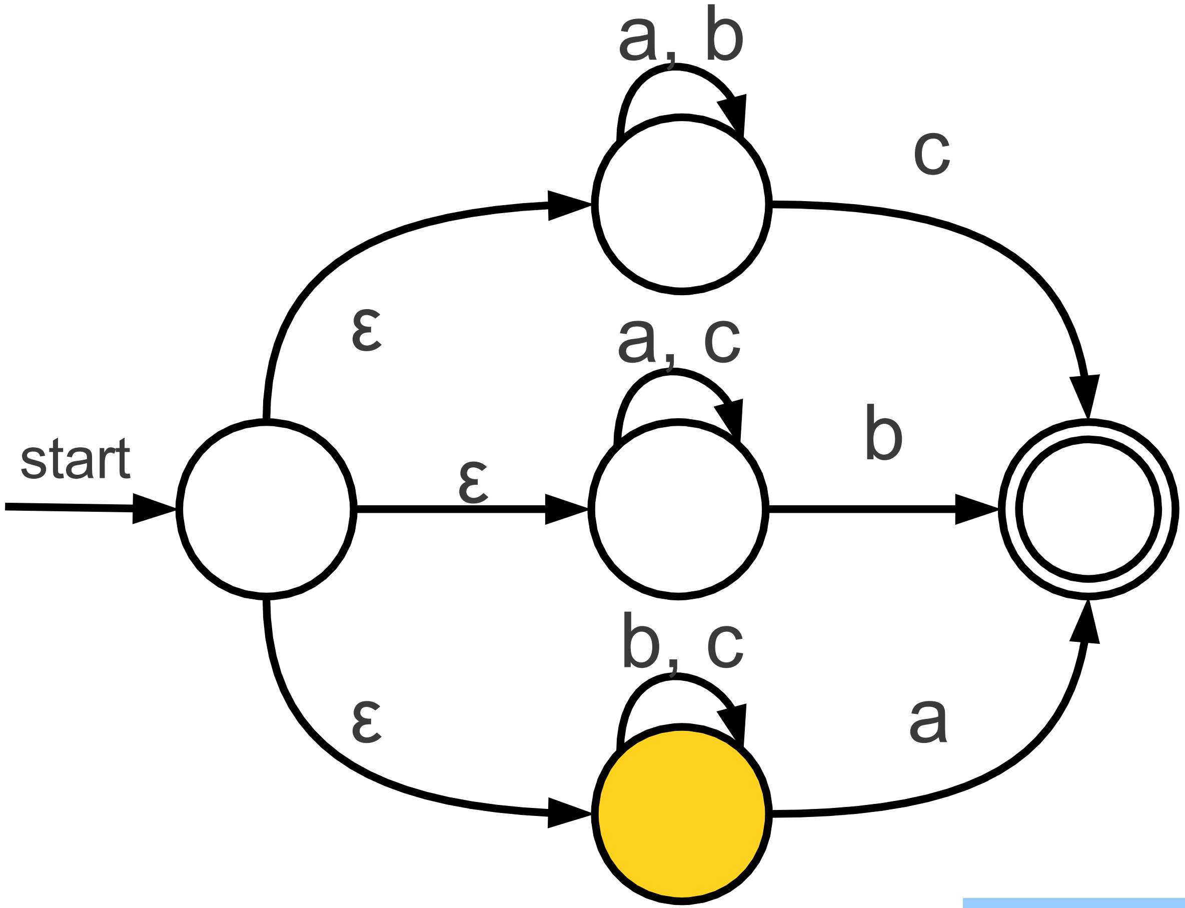


An Even More Complex Automaton



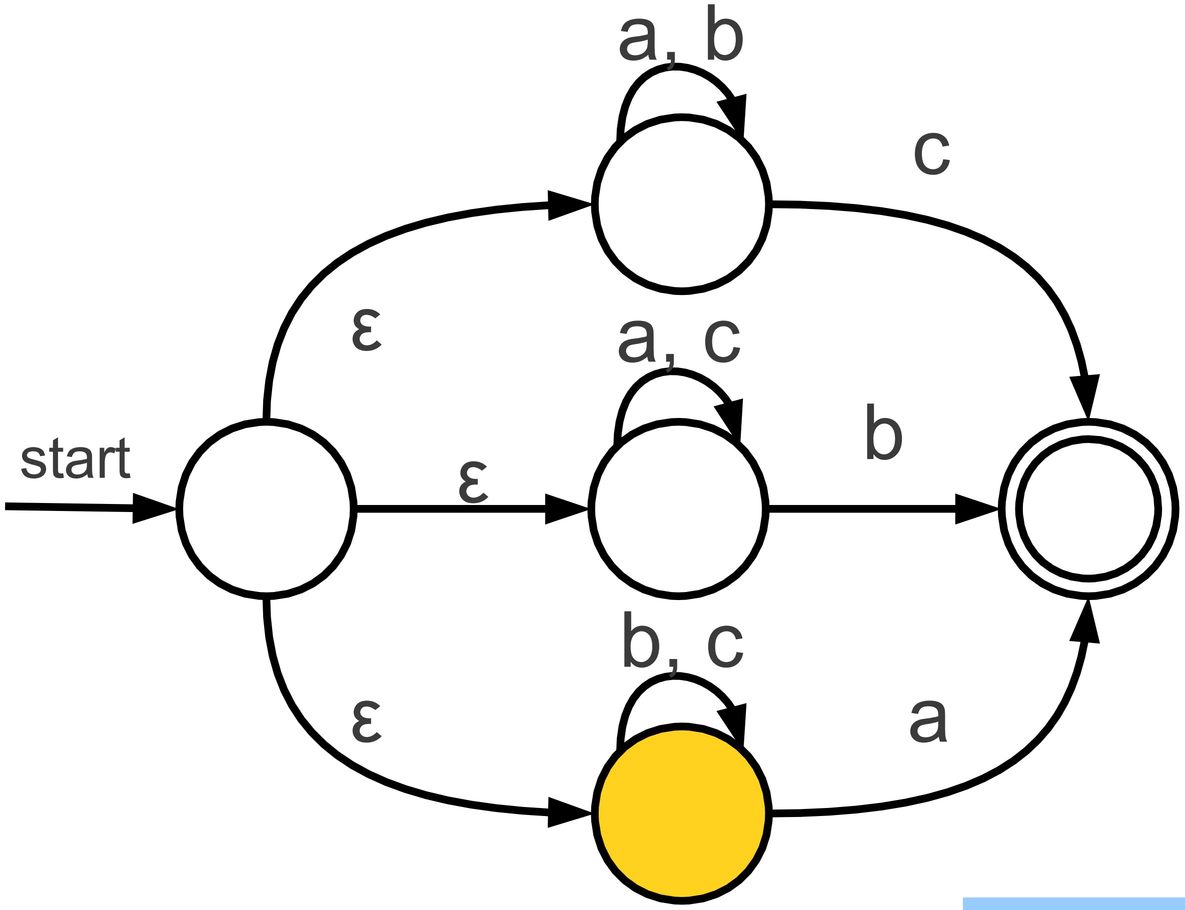
b	c	b	a

An Even More Complex Automaton



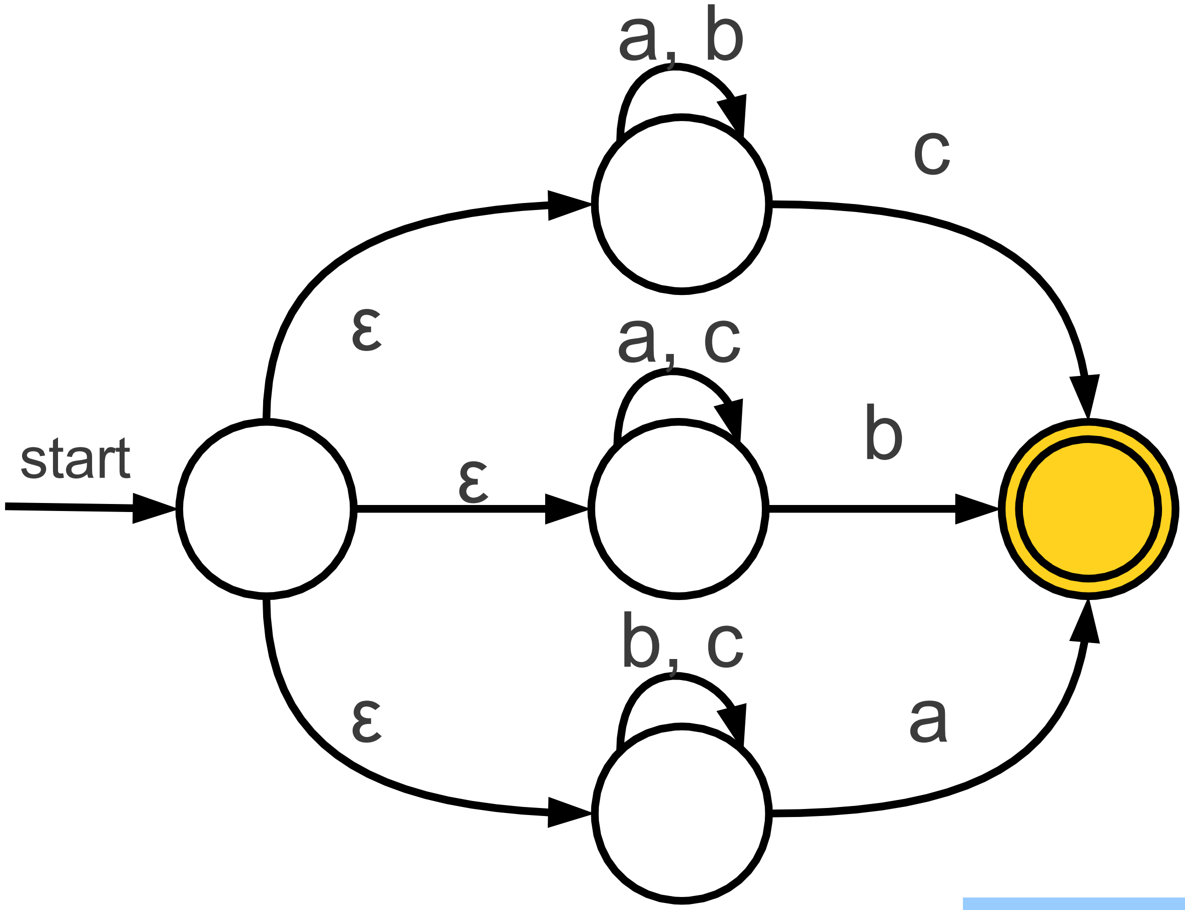
b c b a

An Even More Complex Automaton



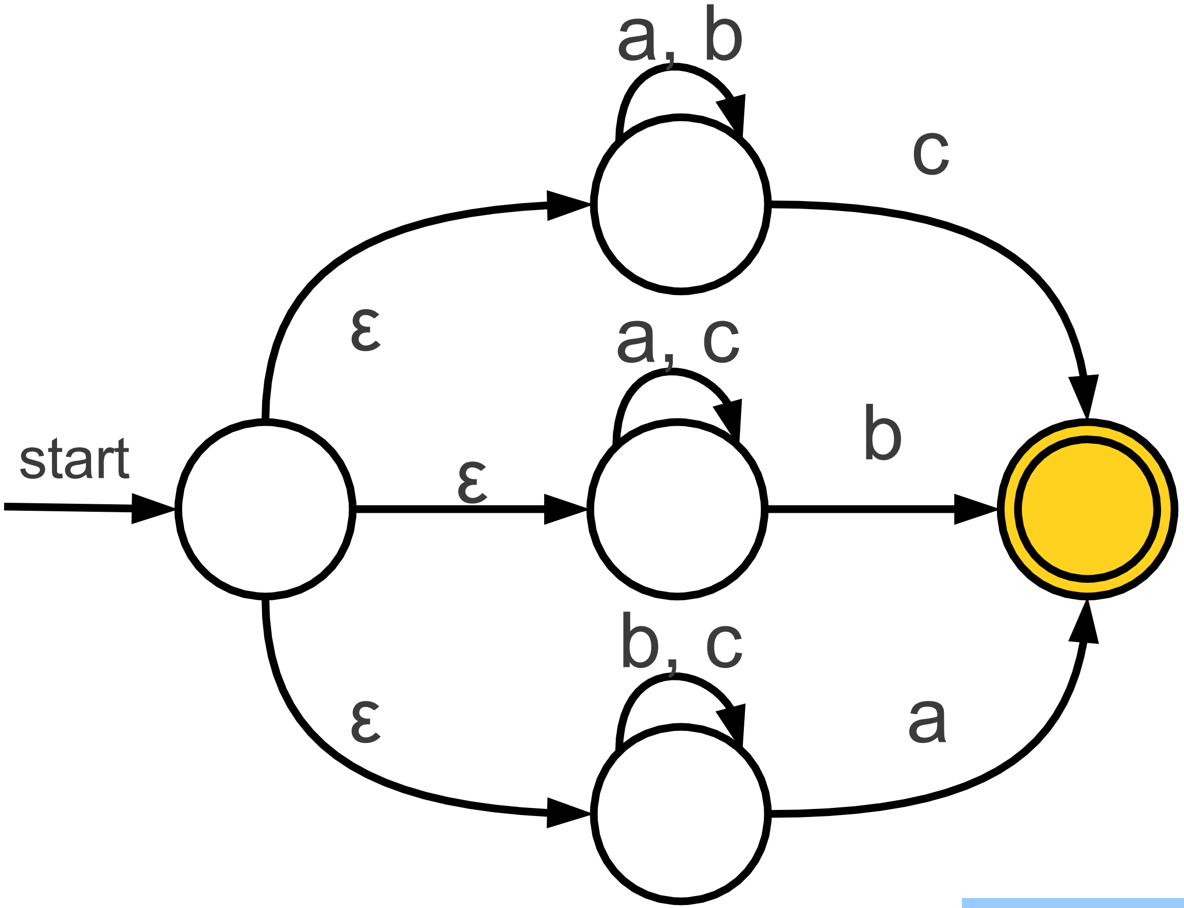
b c b a

An Even More Complex Automaton



`b c b a`

An Even More Complex Automaton



b c b a