Mathematical Concepts (G6012)

Lecture 22

Thomas Nowotny

Chichester I, Room CI-105

Office hours: Tuesday 15:00 - 16:45

T.Nowotny@sussex.ac.uk

REVISIONS

FUNDAMENTALS

Numbers

- There are several **number systems**:

  $\mathbb{N}$ – the **natural** numbers

  $\mathbb{Z}$ – the **integers**

  $\mathbb{Q}$ – the **rational** numbers

  $\mathbb{R}$ – the **real** numbers

  $\mathbb{C}$ – the **complex** numbers

- They contain each other

Read:
“is contained in”
“is subset of”

$$\mathbb{N} \subset \mathbb{Z} \subset \mathbb{Q} \subset \mathbb{R} \subset \mathbb{C}$$

Summation notation ($\sum$ notation)

upper limit

Definition: $$\sum_{j=1}^{3} x_j := x_1 + x_2 + x_3$$

summation index

lower limit

Note: Increment always by 1!

It is like a "for" loop:

```
a= 0;
for ( j=1; j <= 3; j= j + 1) {
    a= a+x_j
}
```

The empty sum is 0

Product notation

Definition:

$$\prod_{j=1}^{5} a_j := a_1 \cdot a_2 \cdot a_3 \cdot a_4 \cdot a_5$$

Think about a for loop, but multiplication inside, instead of summation:

Example:

$$\prod_{j=1}^{5} j = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5$$

```
p= 1;
for ( j=1; j <= 5; j= j + 1) {
    p= p*a_j
}
```

The empty product is 1

Manipulating sums

$$\sum_{i=1}^{n}\sum_{j=1}^{m} x_{ij} = \sum_{i=1}^{n}\left(\sum_{j=1}^{m} x_{ij}\right)$$

(bracketing is implied but doesn’t matter)

$$= \sum_{j=1}^{m}\left(\sum_{i=1}^{n} x_{ij}\right)$$

(sums can be “swapped”)

$$k \cdot \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} k \cdot x_i$$

(common factors can be “pulled out”)

Manipulating sums

- All these rules are the same that you learned in school without the sigma-notation, e.g.

$$k \cdot (x_1 + x_2) = k \cdot x_1 + k \cdot x_2$$

- If in any doubt, use "…" and do what you learned earlier
- This also applies to products

PROOF BY CONTRADICTION

Principle

- We want to demonstrate that statement A is false
- We assume that A is true
- We show that “A true” implies “B true”, where B is known to be false
- This is called a contradiction which can only be resolved if A actually is false
- That completes the proof.

Another example

- Claim: For two positive real numbers a and b,

  $$a + b \geq 2\sqrt{ab}$$

- Proof: Assume $a + b < 2\sqrt{ab}$

$$\Rightarrow \quad (a+b)^2 < 4ab$$

$$\Rightarrow \quad a^2 + b^2 + 2ab < 4ab$$

$$\Rightarrow \quad a^2 + b^2 - 2ab < 0$$

$$\Rightarrow \quad (a-b)^2 < 0$$

Contradiction!

PROOF BY INDUCTION

Principle

- We would like to show a claim P(n) for all natural numbers n.
- If we can show that P(1) is true
- And we can show P(n) implies P(n+1)
- Then P(n) is true for all n.

Recipe: Induction

- Write down the claim you are trying to prove
- Induction start: Show the claim is true for n=0 (or n=1 depending on problem)
- Induction assumption: Assume the claim is true for n. Write it down for n as a reminder.
- Induction step: Show that the claim is true for n+1 using that it is true for n

Another example

- Claim: 6^n-1 is divisible by 5 for all n
- Induction start:
  n = 0: $6^0-1 = 0$ which is divisible by 5.
- Induction Assumption:
  6^n-1 is divisible by 5 for n, i.e. $6^n-1 = 5k$ for some $k \in \mathbb{N}$. Or, equivalently, $6^n = 5k+1$
- Induction Step:Induction step:

$$6^{n+1} - 1 = 6 \cdot 6^n - 1$$

Use assumption here

$$= 6(5k+1) - 1 = 6 \cdot 5k + 6 - 1$$

$$= 6 \cdot 5k + 5 = 5(6k+1)$$

This is divisible by 5!

Tips to remember

- Sometimes it is easiest to "work from both sides" to complete the Induction Step
- You must use the assumption, otherwise it's not a proof by induction (and likely will not work).
- Therefore, when doing the Induction Step, look for an opportunity to use the assumption
- All three parts must be there: Start, Assumption & Step – otherwise it is meaningless

SETS

Summary

- Notation to define a set
- Cardinality
- Relationships between sets: Subset, subsetEqual, superset, supersetEqual
- Set operations: Union, intersection, subtraction, complement
- Intervals

Dictionary of set theory

Symbol	**Meaning**	**Symbol**	**Meaning**
$\in$	Element of	$\{\ldots\}$	Set of elements
$\subset, \subseteq$	Subset	$\setminus$	Subtract, “without”
$\supset, \supseteq$	Superset	c	Complement
$\cap$	Intersection	card	Cardinality
$\cup$	Union	$\emptyset$	Empty set
[a,b]	Interval (a,b included)	]a,b[	Interval (a,b excluded)

Brackets do matter

- Different brackets mean completely different things!

$$[x, y]$$

All numbers of $\mathbb{R}$ between x and y, including x and y. Infinitely many numbers!

$$\{x, y\}$$

Literally, just the numbers x and y. 2 numbers

REGULAR EXPRESSIONS

Fundamentals on Languages

- Alphabet= set of symbols
- Word= a sequence of symbols
- Singleton word= word with one symbol
- Language= set of words
- Regular language= language assembled from singleton words using
  - Union
  - Concatenation
  - Kleene *

Some examples for the operations

Alphabet $\mathcal{S} = \{\mathrm{a}, \mathrm{b}\}$

Language $\mathcal{A} = \{\mathrm{a}, \mathrm{aa}\}$

Language $\mathcal{B} = \{\mathrm{b}, \mathrm{bb}\}$

- Union:

  $\mathcal{A} \cup \mathcal{B} = \{\mathrm{a}, \mathrm{aa}, \mathrm{b}, \mathrm{bb}\}$
- Concatenation:

  $\mathcal{A} \circ \mathcal{B} = \{\mathrm{ab}, \mathrm{abb}, \mathrm{aab}, \mathrm{aabb}\}$
- Kleene star:

  $\mathcal{A}* = \{\mathrm{a}, \mathrm{aa}, \mathrm{aaa}, \mathrm{aaaa}, \ldots\}$

Regular expressions

- A regular expression describes the legal words in a language by a matching operation:
  - ‘*a*’ matches the symbol ‘a’ in the alphabet
  - The ‘|’ denotes alternatives (Boolean (x)or)
  - Brackets ‘*(*‘ and ‘*)*’ are used for grouping
  - ‘$*$’ matches zero or more of the preceding symbol
  - ‘+’ matches one or more of the preceding symbol
  - ‘?’ matches 0 or 1 of the preceding symbol

Precedence of operators

Precedence	Operator
Highest	( )
Middle	? * +
Low	concatenation
Lowest	|

Quick test (common mistakes)

- ab+ = ???
  1. (ab)+
  2. a(b+)

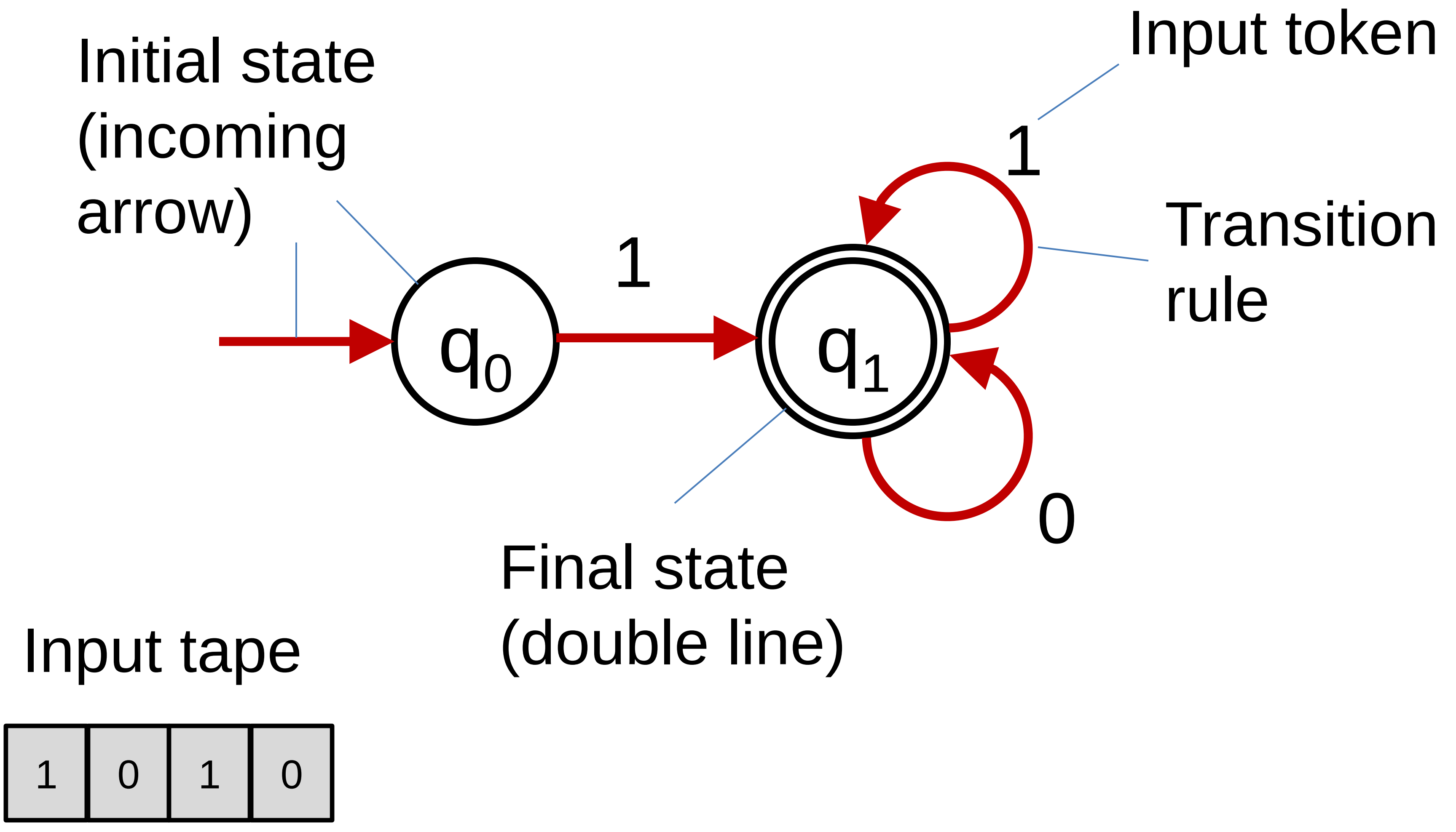
- ab*|ba* = ???
  1. ((ab)*)|((ba)*)
  2. (a(b*))|(b(a*))
  3. a(b*|b)a*
  4. (((ab)*)|b)a)*

FINITE STATE AUTOMATA

Finite State Automata (FSA)

- Finite number of states
- One Initial state
- One ore more final states
- Input tape
- Transitions defined by an input symbol that is “consumed”

DSA: Graphic Representation



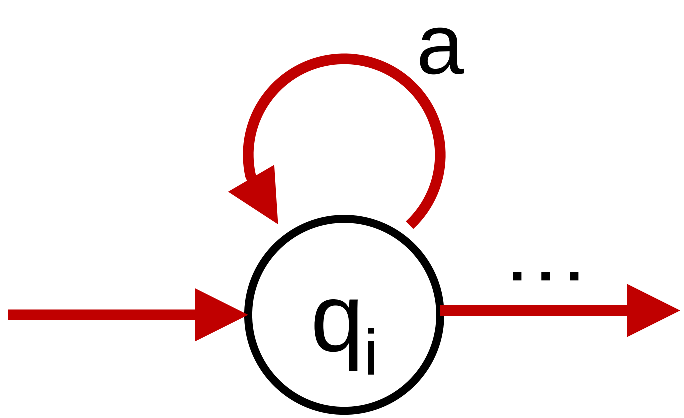
FSA and regular expressions

FSA accept regular languages
Regular expressions define regular languages

➡ For any regular expression we can find a FSA that accepts the corresponding language

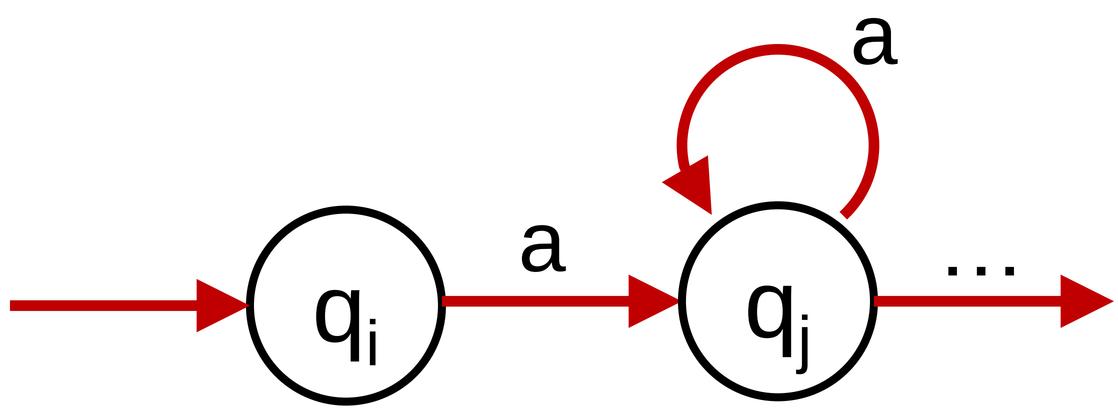
Some pointers:

a*…

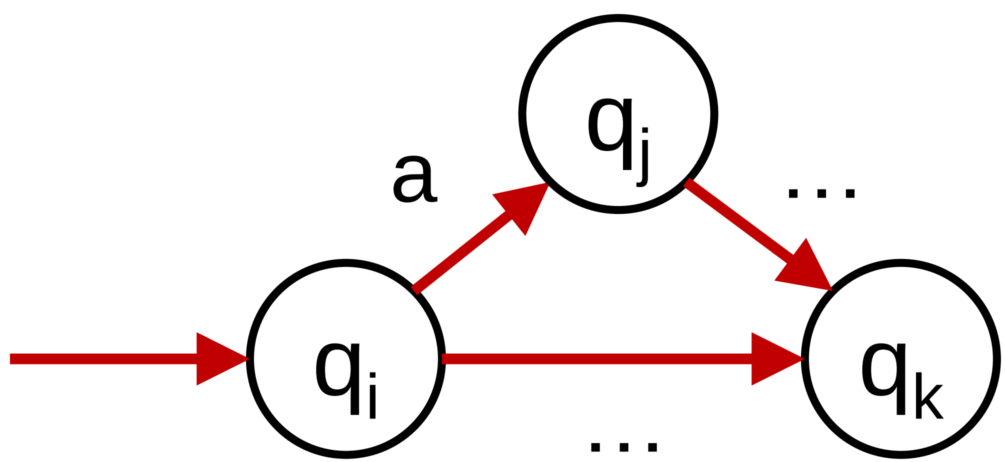


More hints

a+…

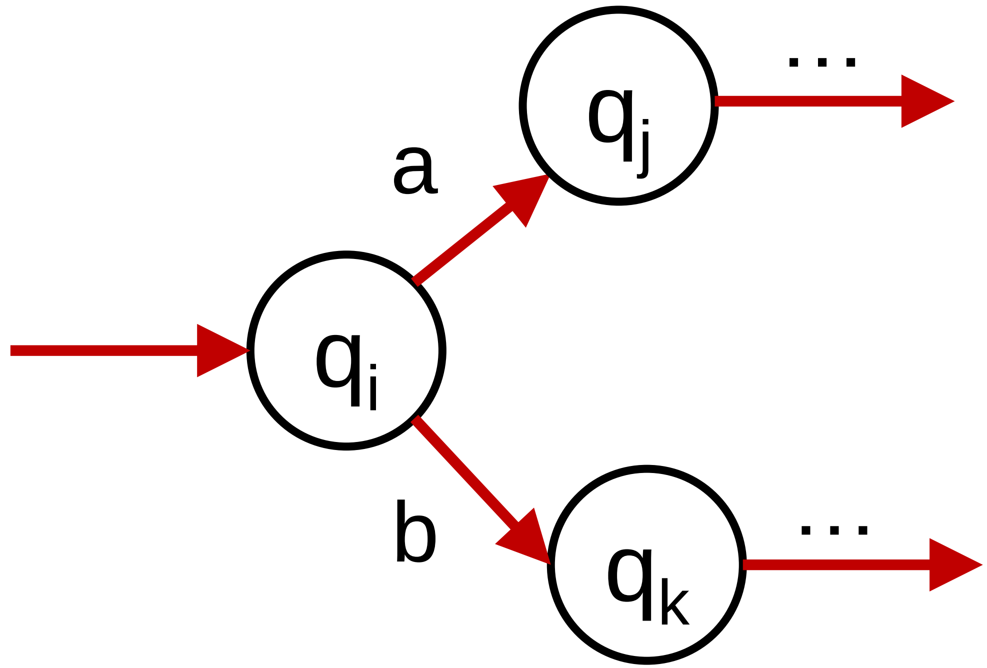


a?…



One more …

(a|b)…



Accepting computation

- FSA must be in a final state
- The input must have been consumed

Error states

- No rule applies for input symbol (stuck)
- Tape is empty but not in a final state

Other things about FSA

- Deterministic/ indeterministic
- Input symbols are consumed/disappear
- Can use empty string for indeterministic FSA

Non-determinism

- Occurs whenever the same “situation” has several possible transitions
- For FSA: A state has two rules with the same input symbol or a rule with the empty string which is not the only rule of the state.
- Non-deterministic automata examine all possible computations to find a successful one
- For FSA we showed that non-deterministic FSA are not more powerful than deterministic ones

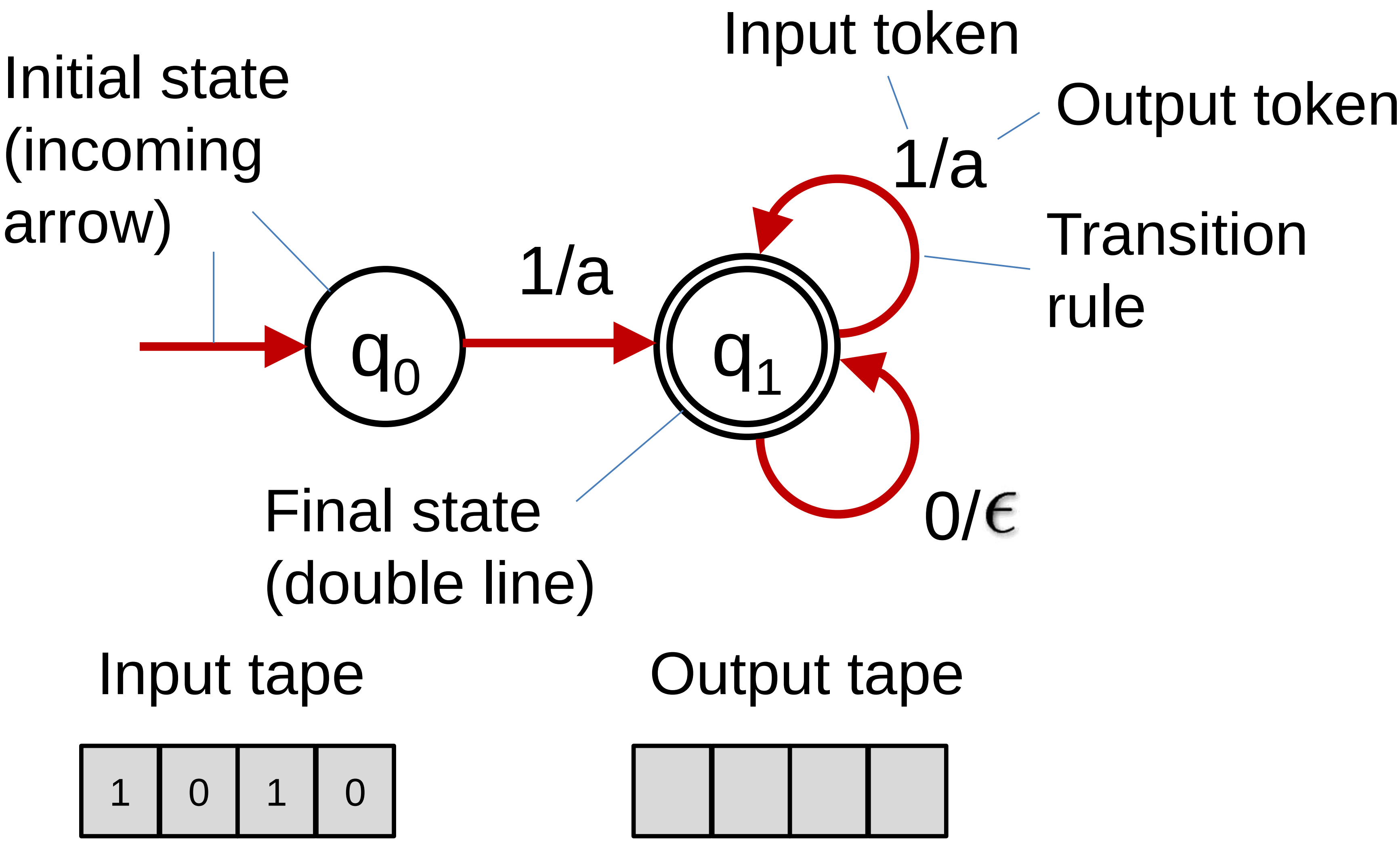
FINITE STATE TRANSDUCERS

FST

- FST are like FSA with one addition:
  Each input symbol is mapped to an output symbol, i.e. we have two tapes: Input tape and output tape
- The output tape is filled left-to-right
- Output tape has unlimited length
- We allow the empty string as output, i.e. rules like a/

- Everything else as FSA, e.g. determinism, non-determinism etc.

FST: Graphical Representation



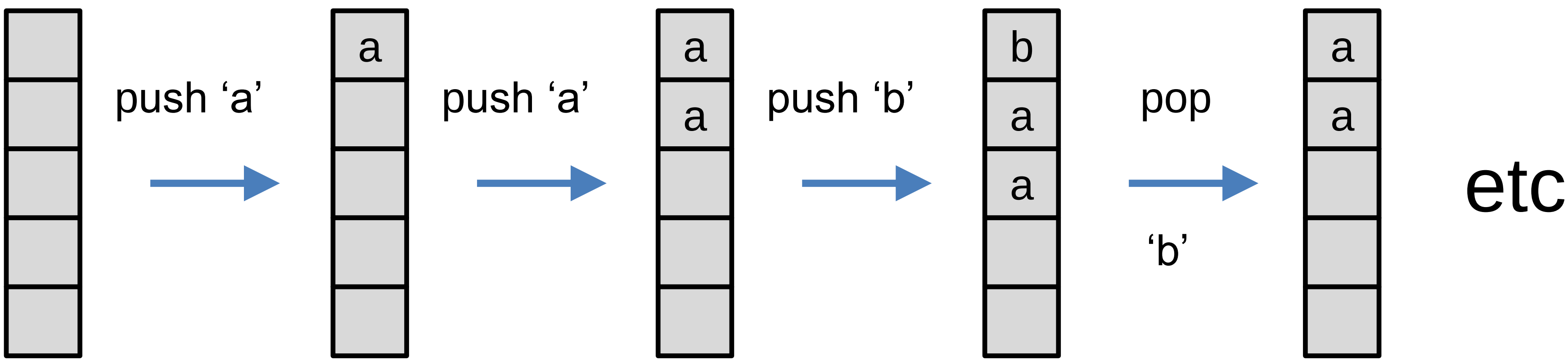
PUSHDOWN AUTOMATA

Pushdown (storage)

- A special kind of list
- Provides (in principle) unbounded storage
- Last in, first out (LIFO)
- Add/remove items only from one end (“top”)
- Push – add an element to the top of PD
- Pop – remove and element from the top of PD
- No other editing or browsing allowed

Example

- It's like a stack of paper where you stack stuff on the top and take it away from the top:



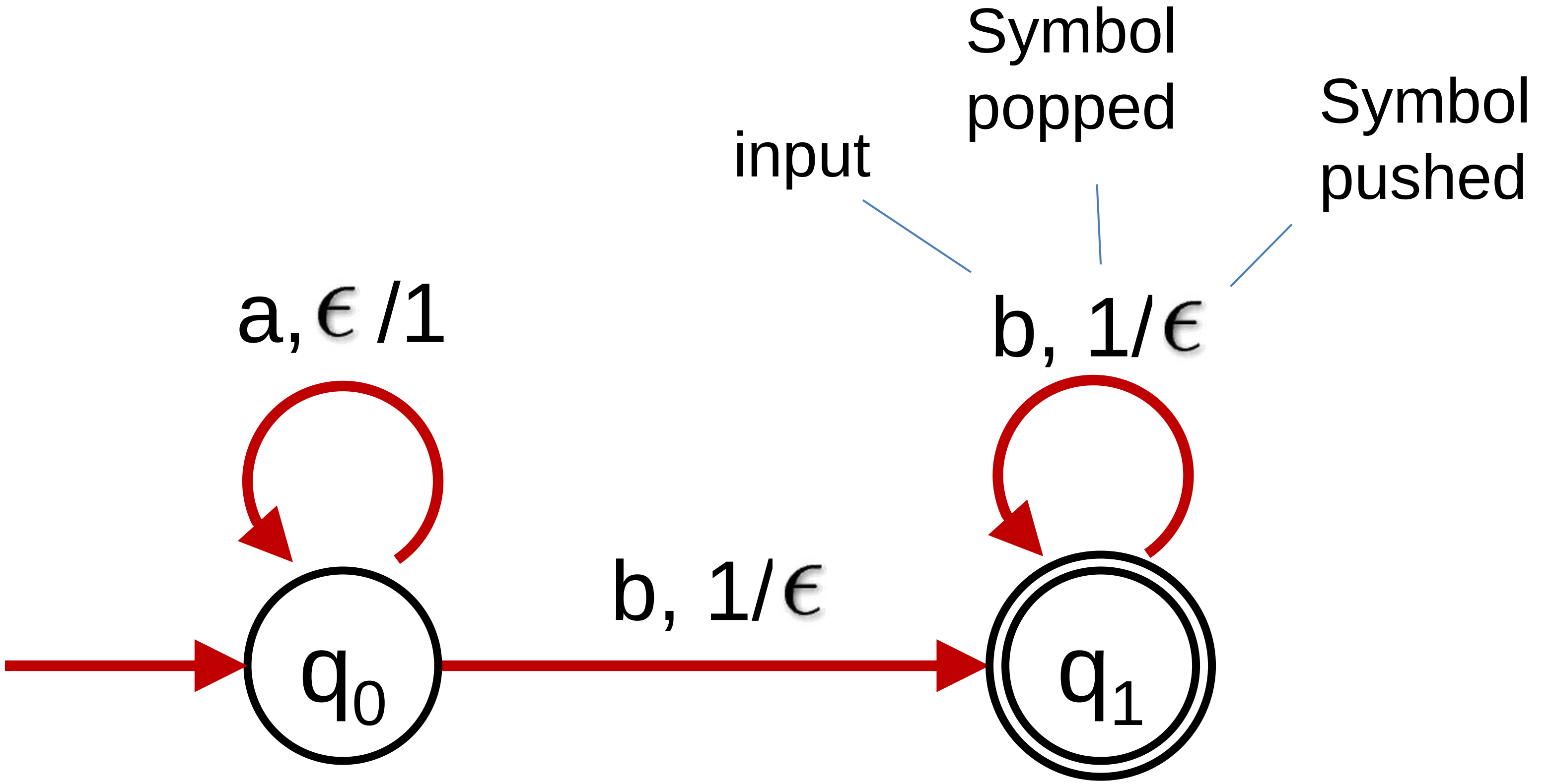
- Alternative notation:

$$( ) \xrightarrow{\text{push 'a'}} (a) \xrightarrow{\text{push 'a'}} (aa) \xrightarrow{\text{push 'b'}} (baa) \xrightarrow[\text{'b'}]{\text{pop}} (aa)$$

Pushdown Automata (PDA)

- FSA + pushdown storage (unlimited size)
- Much more powerful than FSA
- Can build a PDA to accept any context free language (language defined by a context free grammar see below for summary)
- The empty string can be used both for input and for pushdown
- There is no output
- Non-determinism adds power here

Graphical representation



Accepting computation

- Must be in a final state
- The input must have been consumed
- The pushdown must be empty

Error states

- No rule applies for input symbol (stuck)
- Cannot pop correct symbol (error)
(includes trying to pop a symbol other than ϵ from empty pushdown)

Context-free grammar

- Is defined through
  - one or more no-final symbols (one of them initial symbol), we always used ‘S’
  - Productions (replacement rules)
- A context free grammar defines a context-free language: All strings that can be produced by it
- When asked to define a context-free grammar for a particular language, it must produce all words of the language and nothing else.

Example

- Grammar given by S with productions
  1. $S \mapsto aSb$
  2. $S \mapsto ab$
- Generates the language $\{a^n b^n \ : \ n > 0\}$
- While
  1. $S \mapsto aSb$
  2. $S \mapsto \epsilon$

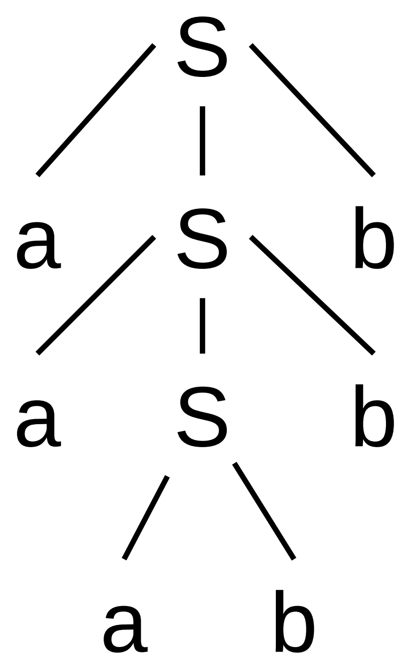
  Generates the language $\{a^n b^n \ : \ n \geq 0\}$
- (one contains the empty string, the other not)

Example mistake

- Define a context-free grammar for $\{a^n b^{2n} \ : \ n \geq 0\}$
- Wrong solution:
  1. S $\mapsto$ aS
  2. S $\mapsto$ bS
  3. S $\mapsto$ ϵ
- These rules generate all needed words, but also a lot of illegal ones

Derivation tree

- Generating a word can be visualised as a tree:



Back to PDA: Protocol of Computation

- Can give “protocol of computation” by making a list of
  - States
  - Input tape content
  - Pushdown content

Example protocol of a computation

Input tape

State

Content of PD

$$(q_0, aabb, \epsilon) \mapsto (q_0, abb, 1)$$
$$\mapsto (q_0, bb, 11)$$
$$\mapsto (q_1, b, 1)$$
$$\mapsto (q_1, \epsilon, \epsilon)$$

- Pushdown here represented as a string of pushdown symbols