# Mathematics for Computing Science

Dictionary order: Strings are ordered ‘alphabetically’

Lexical order: Strings ordered by their length, shortest first. Lambda is always the first.

Finite State Automation- abstract model of a simple machine. (FSM)

A Finite State Automaton (FSA) is a 5-tuple (Q, I , F ,T, E) where:

Q = States - a finite set

I = Initial states - a non empty set of Q

F = Final states - a subset of Q

T = An alphabet

E = Edges - a subset of Q x (T + lambda) x Q

E.g.

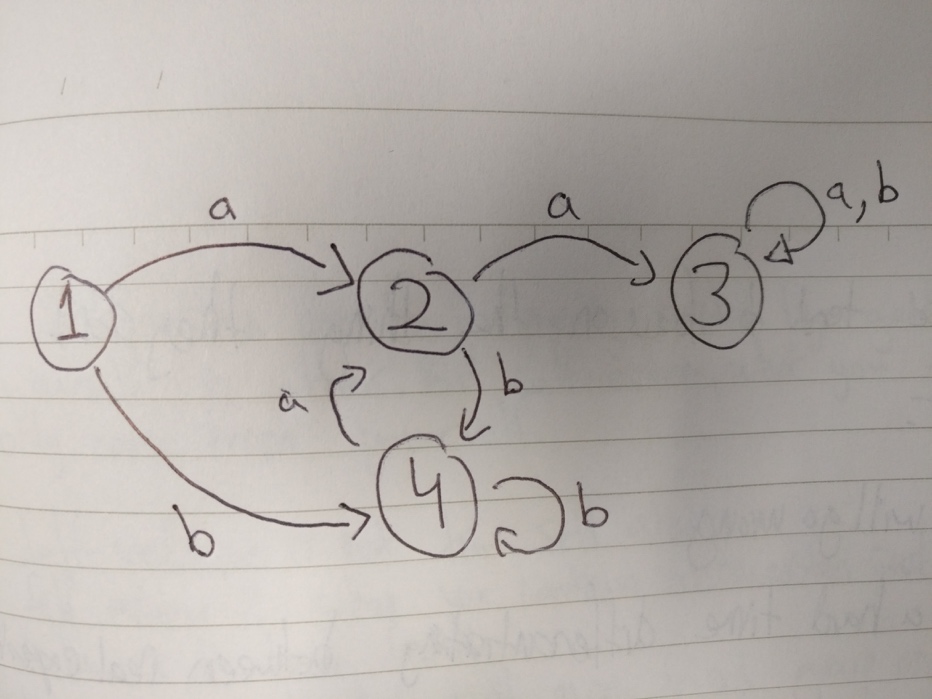
Q = {1,2,3,4}

I = {1}

F = {4}

T = {a, b}

E = {(1, a, 2), (1, b, 4), (2, a, 3), (2, b, 4), (3, a, 3), (3, b, 3), (4, a, 2), (4, b, 4)}



# Module 4 – Functions

Cartesian product is a mathematical operation which returns a set from multiple sets. For sets A and B, the Cartesian product A \* B is the set of ordered pairs.

* The Cartesian product of n sets

A1 \* A2 \* …. \* An

A binary relation R from A to B is a subset of A x B.

Can be used to model real-life facts.

E.g. Scored = {(x<Student, y<CAS): x scored y in last years CS2013 exam}

Inverse relations

Any binary relation R: A x B has an inverse relation R-1: B x A

Relations can have lots of properties.

Binary relations can be symmetric or asymmetric.

# Back to FSH’s

Non determinism

1. From one state there could be a number of edges with the same label
2. Some of the edges could be labelled with lambda, the empty string
3. May be more than one initial state

Our aim is to specify languages for use in the computer

Sketch of an FSA is easy for us to understand.

# Propositional Logic

Formal language for computers

All propositional letters are statements, then we can use these statemets to make shit

# REGEX

The actually useful stuff.

Let T be an alphabet. A regular expression over T defines a language over T as follows:

1. Lambda λ {λ}, ∅ denotes {}, t denotes {t} for t < T:
2. If r and s are regular expressions denoting languages R and S then

(r + s) denoting R + S

(rs) denoting RS, and

(r\*) denoting R\* are regular expressions;

1. Nothing else is a regular expression over T.

# Lecture X

# Kleene’s Theorem

Now we have defined regular languages.

We have also defined language which are accepted by FSAs

**Theorem: A language L is accepted by FSA iff L is regular**

Recall regular expressions form last lecture.

You can join languages/FSAs together.

## Revision

Alphabets

If T is an alphabet, then T\* is the set of all strings over T

T = {a, b}, T\* = {lambda, a, b, aa, ab, bb, ba, aaa, … }

A language over an alphabet T is a set of strings over T. Also called a T – language – all the possible solutions from an alphabet.

A + B = Union of a and B

A (u upside down) B = intersection of A and B

## Finite State Automaton

Can be in a finite number of states. Receives symbols as input, and particular inputs move the machine to particular states.

Deterministic FSA – well behaved FSA

FSA is a 5-tuple Q, I, F, T, E

Q = States , finite set

I = Initial states, nonempty subset of Q

F = Final states = subset of Q

T = an alphabet

E = edges = subset of Q \* (T + lambda) \* Q

If (x, a, y) is an edge, x is start and y is its end state.

A path is a sequence of edges such that the end state of one is the start state of the next.

A path is successful if the start state of the first edge is an initial state, and the final is the final state.

The label of a path is the sequence of edge labels.

A string is accepted by a FSA if it is the label of a successful path.

Let A be a FSA. The language accepted by A is the set of strings accepted by A, denoted by L(A)

## Nondeterministic FSA

1. From one state there could be a number of edges with the same label
2. Some edges can be labelled with lambda, the empty string
3. May be more than one initial state

FSA is deterministic if:

1. There are no empty labelled edges
2. For any pair of state and symbol (q, t) there is at most one edge (q, t, p)
3. Only one initial state.

## Nondeterministic transition functions

* We don’t do this

Minimum size of FSA’s

Lecture III FOPL

## Finite order predicate logic

More practise with FOPL semantics: handling strange formulas

Some FOPL equivalence laws

Deduction

Quantifier equivalence laws:

If u.d. = a, b, c, d …

More equivalence laws (Check slides)