Comp 598 Notes

Cole Killian

Summer 2020

This is McGill's undergraduate Math 598, "Topics in Science 1" instructed by Prakash Panangaden. This semester it is focusing on Automata and Computability. You can find this and other course notes here: https://colekillian.com/course-notes

Contents

§1 05-04

§1.1 Logisitics

1. Potentially an oral final exam
2. Too many assignments to grade at the fast pace. Students will self grade their homeworks, and a few will be chosen at random to see if people are being honest in their self evaluation
3. All assignments are to be completed in Latex
4. Fact means he'll say it and not prove it.

§1.2 Schedule

Lecture Schedule for COMP 598 Summer 2020

Week 1	Lecture 1	4 May	Induction, strings, automata, monoids
	Lecture 2	5 May	NFA, Kleene theorem, regexp, Kleene algebra
	Lecture 3	6 May	Myhill-Nerode, minimization, duality
	Lecture 4	7 May	Pumping lemma
Week 2	Lecture 1	11 May	Temporal logic
	Lecture 2	12 May	Bisimulation
	Lecture 3	13 May	Learning automata
	Lecture 4	14 May	Weighted automata
Week 3	Lecture 1	18 May	Context-free languages, grammars and parsing
	Lecture 2	19 May	Pumping lemma for CFLs, DCFLs
	Lecture 3	20 May	Models of computation
	Lecture 4	21 May	λ-calculus
Week 4	Lecture 1	25 May	Computability, reductions
	Lecture 2	26 May	Logic and unsolvability, arithmetic hierarchy
	Lecture 3	27 May	Fixed-point theory
	Lecture 4	28 May	Godel's and Tarski's theorems

Definition 1.1 (Equivalence Relation)**.** An equivalence relation R on a set S is a set of pairs $R \subseteq S \times S$ such that (write xRy for $(x, y) \in R$)

1. xRx
2. $xRy \Rightarrow yRx$
3. $xRy \ \wedge \ yRz \Rightarrow xRz$

Definition 1.2 (Equivalence Relation)**.** $[x] \coloneqq \{y | xRy\}$ is the equivalence class of x
The set of equivalence classes forms a partition of S. Prove this as an exercise.

Definition 1.3 (Partial Order)**.** Abstraction of comparison. Not all elements can be compared. A binary relation on a set S. Big difference is no symmetry.

1. $x \leq x$
2. $x \leq y \wedge y \leq x \Rightarrow x = y$

3. $x \leq y \wedge y \leq z \Rightarrow x \leq z$

Example of taking subsets and ordering them by inclusion. This is a typical partial ordering that is not total. The real numbers are a total ordering.

Definition 1.4 (Well-founded order)**.** Given a p.o.set (partially ordered set) $(S, \leq)$ and $U \subseteq S$, we say that $u \in U$ is a minimal element of U if $\forall v < u,\ v \notin U$. There can be infinitely many minimal elements, or none.

Example 1.5

Negative integers has no minimal element

Non negative integers in which 0 is the smallest

Strictly positive rational numbers has no minimal element

A partially ordered set $(S, \leq)$ is said to be well-founded if every non-empty subset U has (powerful word, an existential qualifier) a minimal element.

Example 1.6

Examples - Non-negative integers These are well founded. - The positive rational numbers These are not well founded - Pairs $N \times N$ where $(m, n) \leq (m', n')$ if $m < m' \vee (m = m \wedge n \leq n')$. Between (1, 17) and (2, 5) there are infinitely many elements, but do not be fooled there is still a minimal element.

Fact 1.7. An order is well founded if and only if there are no infinite strictly decreasing sequences (chain) $x_1 > x_2 > x_3 > \cdots$.
An order that is both totally ordered and well-founded is called a well order.

Theorem 1.8 (Zermelo's Theorem)

Every set can be given a well order assuming the axiom of choice. People don't want to believe this because no one can figure out how to write a well ordering of the real numbers.

Note 1.9. Zermelo's well ordering principle, axiom of choice, and zorn's lemma are equivalent. Zorn's lemma is something people don't want to give up.

§1.3 Principle of Induction

Definition 1.10 (Predicate)**.** "predicate". you're in the set if you satisfy property of predicate. "predicate is a statement that may be true or false depending on the values of its variables". "The set defined by P(x) is written as x — P(x), and is the set of objects for which P is true."

$(S, \leq)$ is inductive if $\forall P,\ \forall x \in S,\ \forall y < x, P(y) \Rightarrow P(x)$
Not every order is inductive. Minimal element takes care of the base case. Which orders are inductive, and which are not?

Theorem 1.11
An order is inductive if and only if it is well founded.

Proof.

1. Assume $V \coloneqq \{s \in S \mid \neg P(s)\}$ is not empty. Which would contradict the principle of induction. Then $\forall y < v_0,\ y \notin V$ and $P(y)$. But then this implies $P(v_0)$. So then $V = \varnothing$. i.e. $\forall x\ P(x)$.

2. Ind $\Rightarrow$ Well founded. Assume $U \subseteq S$ has no minimal element. $P(x) \coloneqq x \notin U$. $\forall x(\forall y < xP(y)) \Rightarrow P(x)$. We know this because if all y is less than x, and $y \notin U$ by the predicate, then $x \notin U$ or else it would be a minimal element of U.

   Induction says that $\forall x P(x) \Rightarrow U = \varnothing$ and hence $(S, \leq)$ is well founded because all sets have a minimal element.

□

** Emmy Noether $\Sigma = \{a, b\}.\Sigma^* = \{\epsilon, a, b, aa, ab, \cdots\}$
Under lexicographic ordering this is <u>not</u> well founded.

§1.4 Strings

A finite set is called an alphabet. Σ^* is a set of <u>finite</u> sequences. The number of such sequences is infinite. ϵ is the symbol for the empty word. A subset of the set of sequences is called a language.

Definition 1.12 (monoid)**.** A <u>moinoid</u> is a set S with a binary operation $\cdot$ and a unit e.

1. $\forall x, y, z \in S,\ x \cdot (y \cdot z) = (x\dot{y}) \cdot z$
2. $\forall x \in S,\ x \cdot e = e \cdot x = x$
3. Monoids are not necessarily commutative. If we have that $\forall x, y,\ x \cdot y = y \cdot x$ then we get a special kind of monoid called a commutative monoid.
4. If there is an inverse operation, it is a group. Monoid's are a general form of group.

Example 1.13

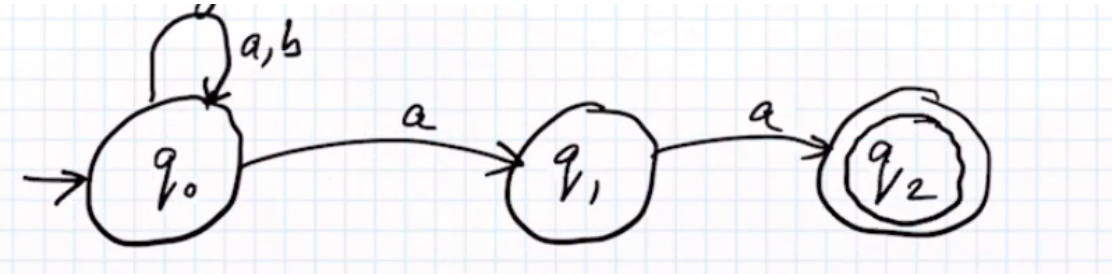
1. Σ^* with concatenation and ϵ is a monoid, but it is not commutative. $aab \cdot ba = aabba$.
2. $\forall x, y, z$ if $xy = xz \Rightarrow y = z$ is a cancellative monoid.

§2 05-05

§2.1 Quantifiers

Can think of $\exists$ as Eloise and $\forall$ as Abeland. Think of statements as a game between these two. The statement is true if Eloise has a winning strategy, and false otherwise (i.e. Abeland has a winning strategy).

§2.2 Design a machine that accepts string ending in "aa".



§2.3 NFA vs DFA

1. There can be multiple next states or no next state. NFA's can make choices. There are multiple computation paths corresponding to different choices. A word is accepted if there exists a path leading to an accept state.

If a machine jams, that is equivalent to rejection. Therefore only sequences that end with "aa" can end up at the accept state.

> **Theorem 2.1**
> The language accepted by any NFA is a regular language. i.e. could also have been accepted by a DFA.

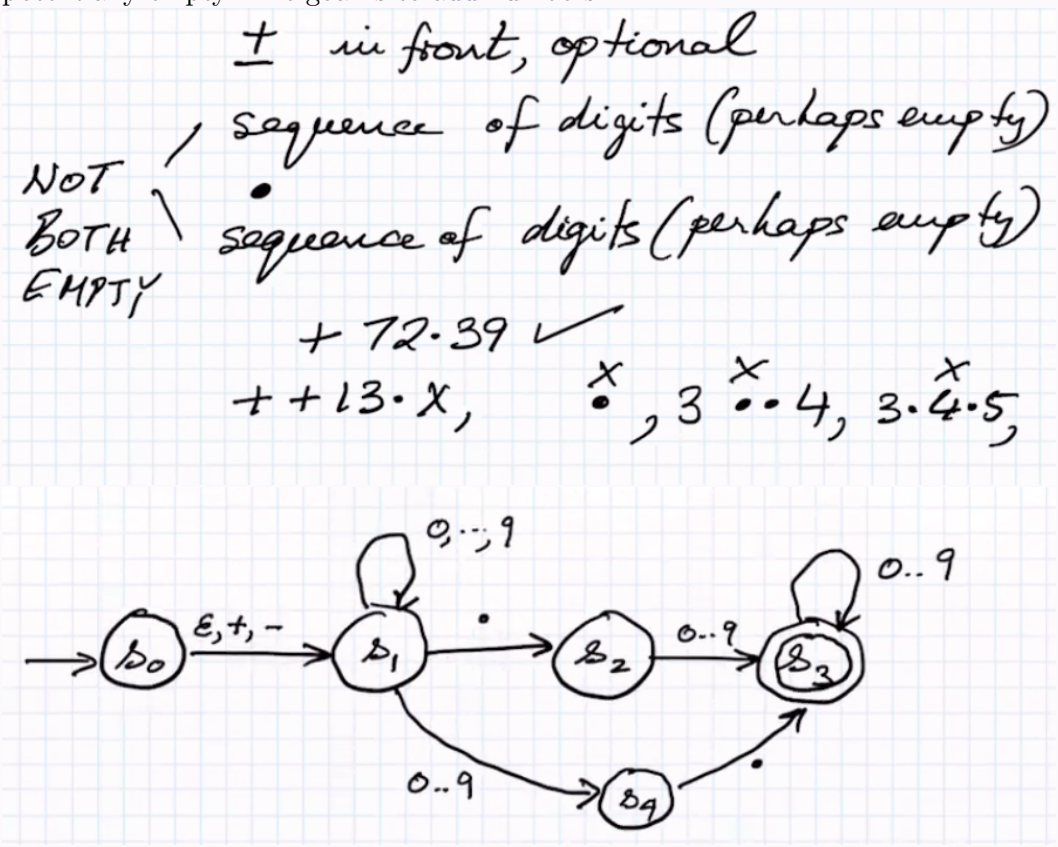
Remark 2.2. There is an algorithm for converting an NFA into an equivalent DFA. So you can go crazy with an NFA and know that it is still a regular language.

Note 2.3. Another extension: NFA + ϵ This machine can make jumps wihtout reading any input.

$$DFA \equiv NFA \equiv NFA + \epsilon \equiv \text{Regular Languages}$$

Example 2.4 (Decimal Numbers)

$\pm$ in front, optional. Two sequences of digits with one or the other (but not both) potentially empty. The goal is to add numbers.



1. Exactly plus, or exactly minus, or nothing. Otherwise the machine will jam.
2. ϵ is not just "hey I can jump to anywhere at any time"; it is part of the design of the machine.
3. There should be at least one thing after the decimal point.
4. No transition for another dot or plus or minus so any of those would cause the input to get rejected.
5. Allow for the possibility of ending with decimal point.
6. Need to two path to acceptance to distinguish between either before the decimal or after the decimal being empty, but not both.

This thing is doing a lot of guessing.

Note 2.5. In order to accept the language, it most both accept the right things and reject the wrong things. Cannot just accept everything.

Definition 2.6 (Formal definition of NFA)**.** An NFA is a 4-tuple:

1. Q: a finite set of states
2. $Q_0 \subseteq Q$: a (finite) set of start states

3. $F \subseteq Q$: a (finite) set of accept states

4. $\Delta : Q \times \Sigma \to 2^Q$.
   Or with ϵ. $\Delta : Q \times (\Sigma \cup \{\epsilon\}) \to 2^Q$

$$\Delta^* : Q \times \Sigma^* \to 2^Q$$
$$\Delta^*(q, \epsilon) = \{q\}$$
$$\Delta^*(q, w \cdot a) = \cup_{q' \in \Delta^*(q,w)} \Delta(q', a)$$

5. The accept state

$$L(N) = \{w \in \Sigma^* \mid \exists q \in Q_0, \Delta^*(q, w) \cap F \neq \varnothing\}$$

Theorem 2.7
Given an NFA $N = (Q, Q_0, F, \Delta)$ there is a DFA $M = (S, s_0, F', \delta)$ such that $L(N) = L(M)$.

Proof. Keep track of all the places where the m/c (machine) could be.

$$S = 2^q$$
$$s_0 = Q_0$$
$$F' = \{S \subseteq Q \mid S \cap F \neq \varnothing\}$$

Take the union of all the places where it could go.

$$\delta(s, a) = \cup_{q \in s} \Delta(q, a)$$

It is easy to see that this works (favorite line of math professors) as advertised. □

Remark 2.8. It is possible to have an exponential blow up in the number of states.

Note 2.9. $S = 2^Q$ refers to the set of all subsets of Q (power set). The power set of S can be identified with the set of all functions from S to a given set of two elements 2^S.

This can be understood where you represent each of the elements in the set as a digit in a binary number. Then, in order to find all possible sets, you would find all possible binary numbers with $|S|$ digits, which gives $2^{|S|}$.

For $NFA + \epsilon$ we define ϵ-closure of a set to be the places one can get to with an ϵ-move.

Example 2.10

Let L be a regular language. Let $\frac{1}{2}L = \{w \in \Sigma^* \mid \exists x \in \Sigma^*, wx \in L \ \wedge \ |w| = |x|\}$.

Look at left half of word and guess that there is something you can tack on to the right such that the entire word is something the original machine would recognize.

Assume DFA (S, s_0, F, δ) recognizes L. We will construct an NFA for $\frac{1}{2}L$.

$$Q = S \times S \times 2^S$$

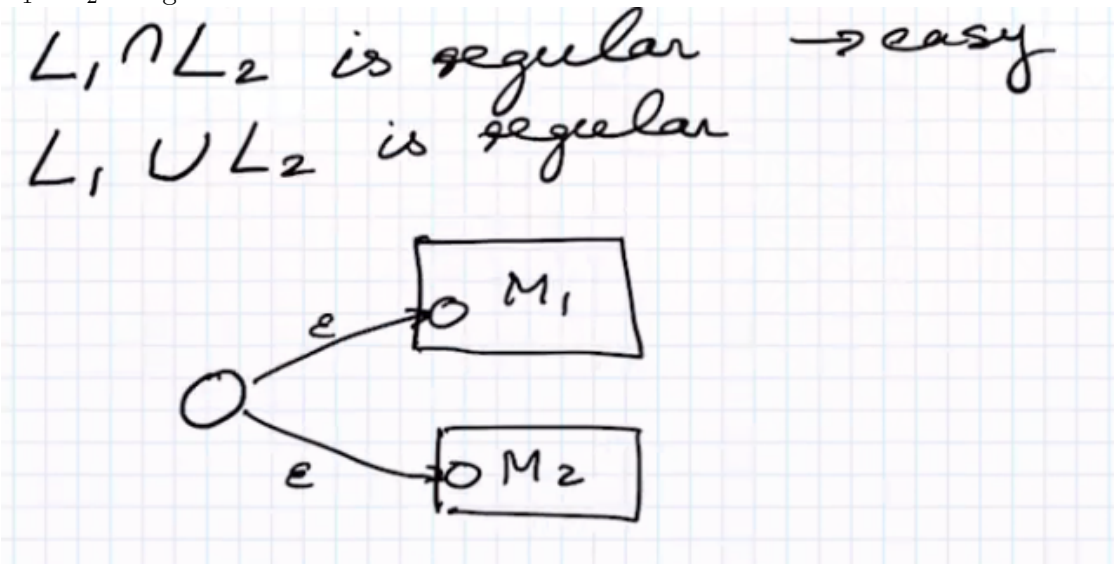
First index for . Second index for guessing where DFA will be when w ends. Third index is to try and track x the second half of the word.

$$Q_0 = \{(s_0, s, \{s\})\}$$
$$F' = \{(s, s, X) \mid X \cap F \neq \varnothing\}$$
$$\Delta((s, \overline{s}, X), a) = \{(s', \overline{s}, X') \mid \delta(s, a) = s'\}$$

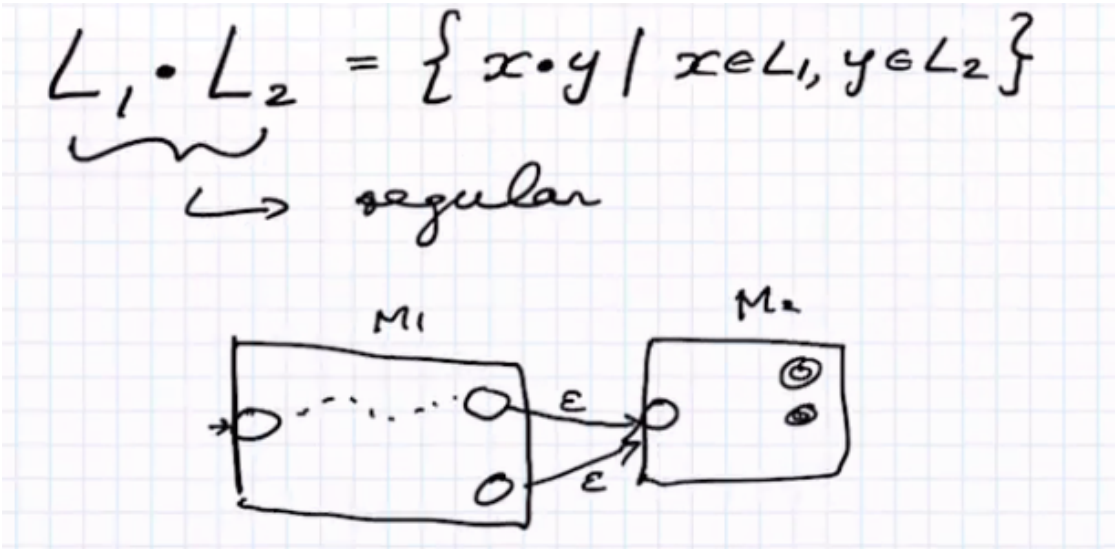
where X' is the set of states DFA can reach from x reading any symbol.

Closure properties of regular languages.

1. $L_1 \cap L_2$ is regular. Define a machine for L_1 and L_2. You create a state space with the cartesian product of states for L_1 and L_2. Then you go through in parallel and if it's in the accept state of both then it's accepted.
2. $L_1 \cup L_2$ is regular.



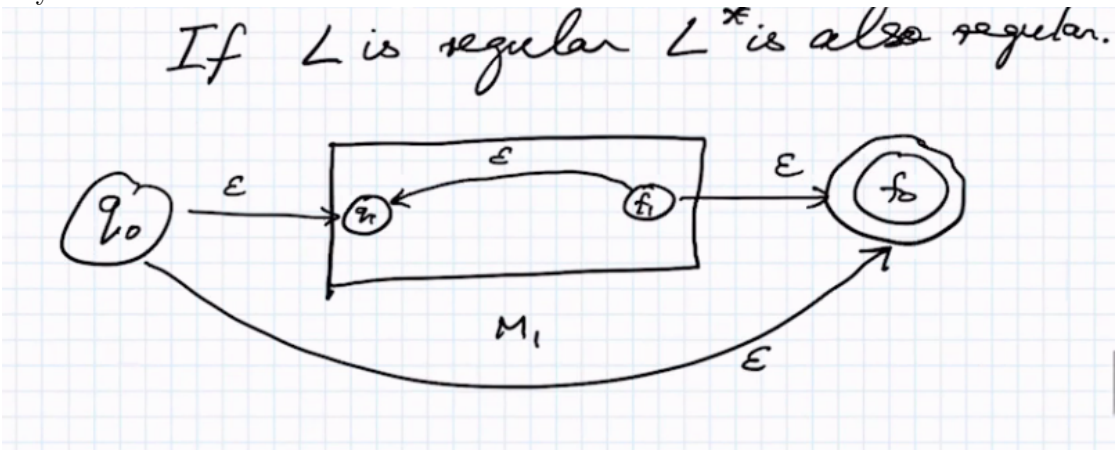
3. Concatenate language. $L_1 \cdot L_2 = \{x \cdot y \mid x \in L_1, y \in L_2\}$.



4. If L is a language, L^* is a new language where

$$L^* = \{w_1 \cdots w_k \mid w_i \in L\} \cup \{\epsilon\}$$

Basically take as many words as you like and glue them all together. Called the kleene (clay-knee) star operator. If L is regular $\Rightarrow L^*$ is regular but not the other way around.



5. Shuffle.

$$L_1 // L_2 = \{x_1 y_1 x_2 y_2 \cdots x_k y_k \mid x_1 x_2 \cdots x_k \in L_1,\ y_1 y_2 \cdots y_k \in L_2\}$$

§2.4 Regular Languages as an algebra

Definition 2.11 (Regular expressions)**.** Notation for describing regular languages in a machine independent fashion.

1. If R_1, R_2 are regular expressions, then $R_1 + R_2,\ R_1 \cdots R_2$ are regular expressions.

   If R is a regular expression, R^* is a regular expression.

A regular expression denotes a language.

$$[\![\varnothing]\!] = \varnothing,$$
$$[\![\epsilon]\!] = \{\epsilon\},$$
$$[\![R_1 \cdot R_2]\!] = [\![R_1]\!] \cdot [\![R_2]\!],$$
$$[\![R^*]\!] = ([\![R]\!])^* / [\![\overline{R}]\!] = \overline{([\![R]\!])}$$

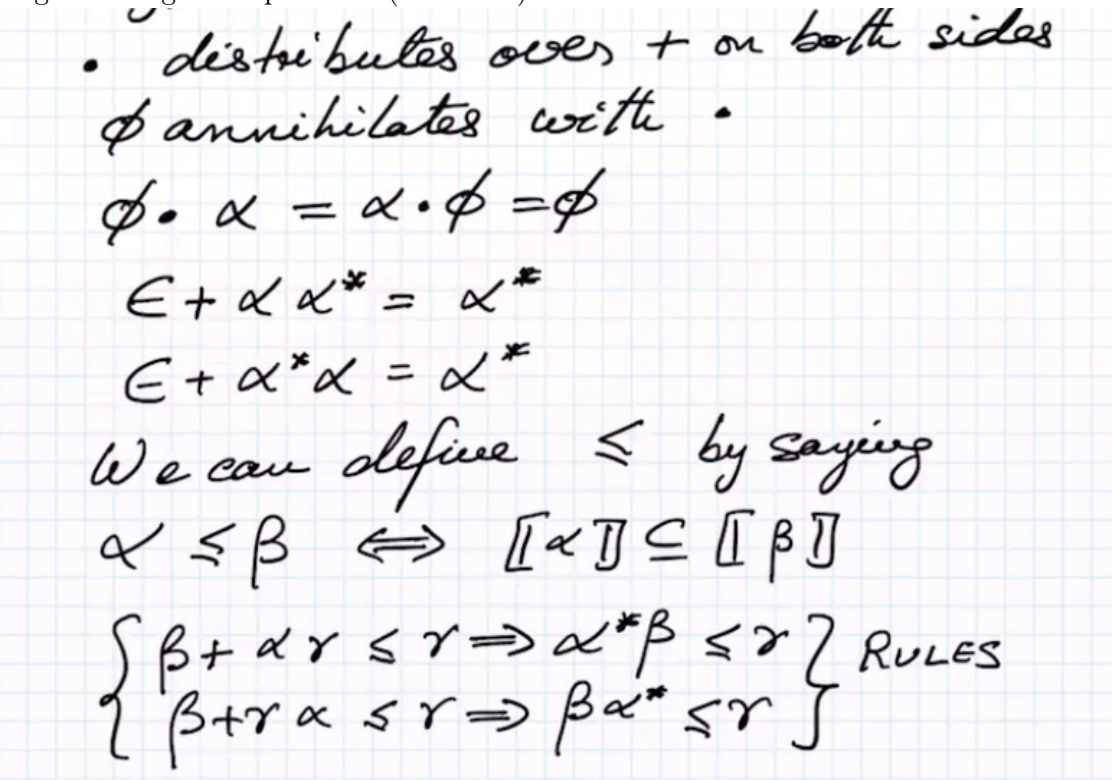
Example 2.12
$(aa + b)^* = \{aa, \epsilon, b, aab, bbaab, \cdots\}$

Theorem 2.13 (Kleene Theorem)
A language is regular iff it can be described by a regular expression.

Proof. Easy to see that reg exp implies regular. We already showed relevant closure properties. Reverse direction later if even. □

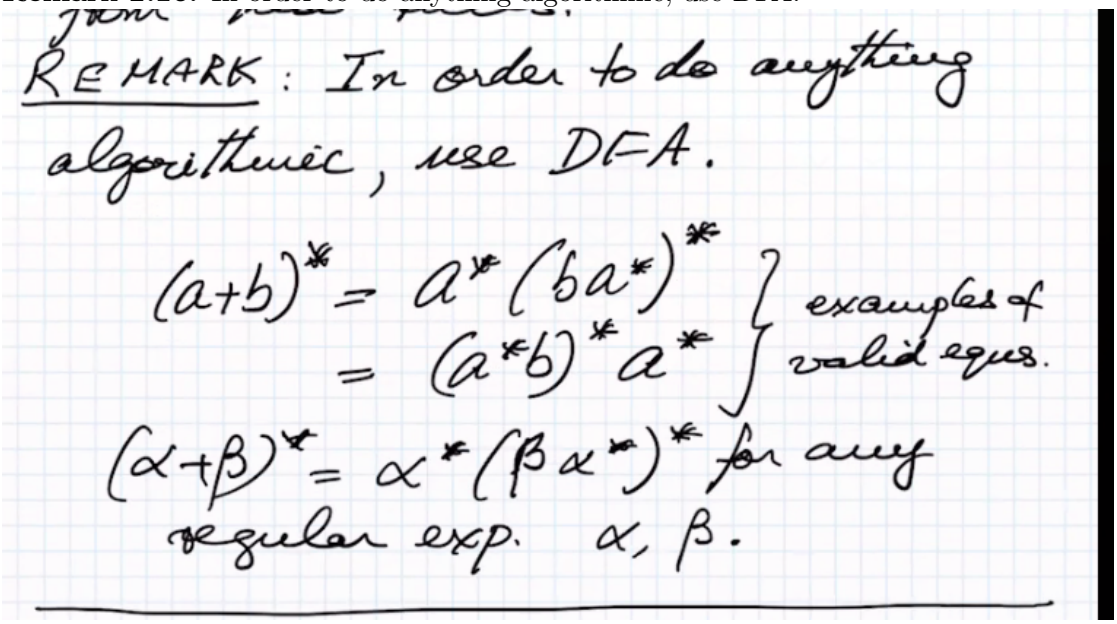
Note 2.14. "+" is notation for "or".

We have a set of equational axioms for reasoning about equality of regular expressions. Reg set of regular expressions (without -).



Theorem 2.15
These rules are complete. i.e. every valid equation can be derived from these rules.

Remark 2.16. In order to do anything algorithmic, use DFA.



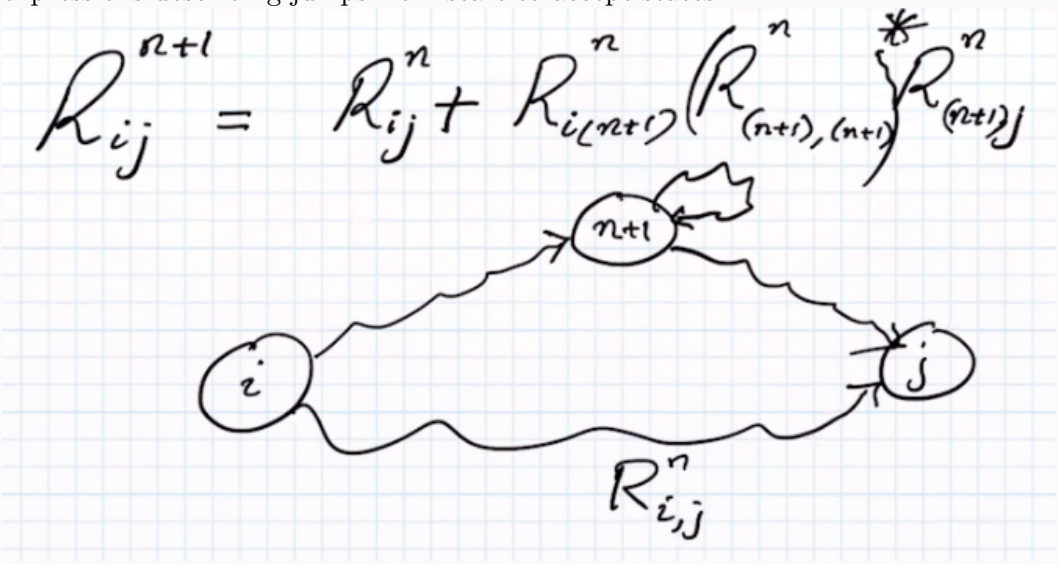
§2.5 From DFA to reg exp

Enumerate the states of a DFA $1, 2, \ldots, k$. For every pair of states i, j we define R_{ij}^n as a regular expression describing all strings that take the DFA from i to j only traversing states $1, \ldots, n$ along the way.

Example 2.17

R_{ij}^0 would represent only direct jumps from i to j. Either the empty word, or a single symbol.

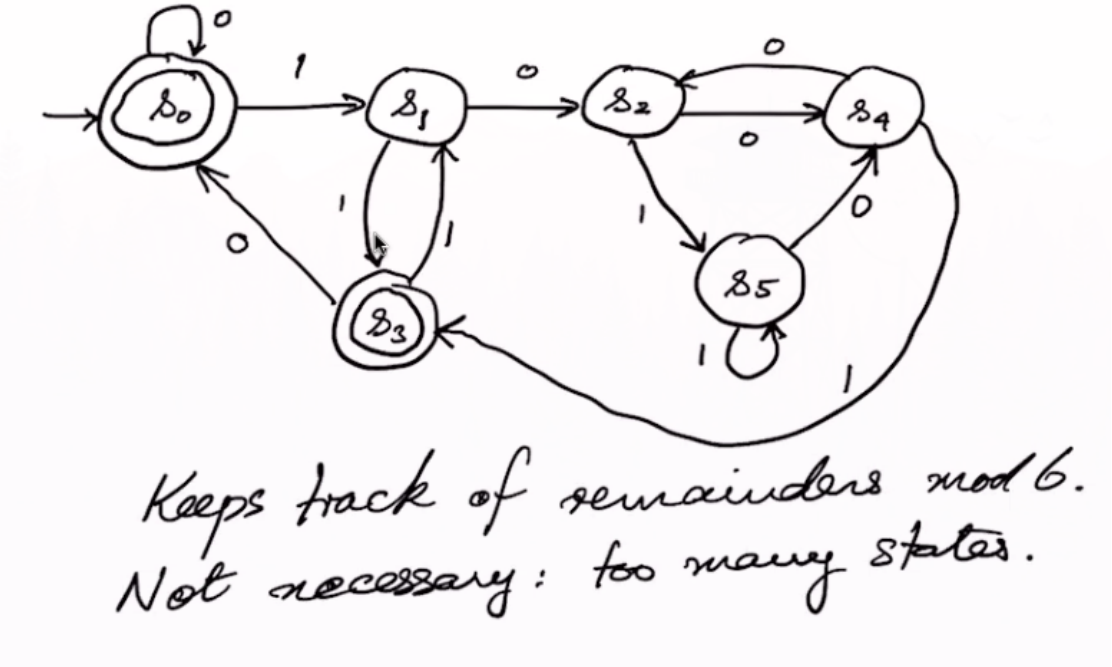
We can compute R_{ij}^{n+1} from R_{ij}^n. Then you can build the DFA by union all regular expressions describing jumps from start to accept states.



§3 05-06

§3.1 Minimization of DFA

Example of machine that checks divisibility by 3. But we already saw a three state machine that does the same thing.



$$\delta(s_0, 0) = s_0 = \delta(s_3, 0)\delta(s_1, 1) = s_1 = \delta(s_3, 1)$$

These states are doing the same thing so there's no reason to keep them separate.

Note 3.1. Once you get to a state, it doesn't matter how you got there. What happens in the future only depends on what you read in the future.

This is the fundamental meaning of the word state. It encodes the information to predict the future.

Definition 3.2. Given a DFA $M = (S, s_0, \delta, F)$, Σ, p, q are equivalent, $p \approx q$ if

$$\forall x \in \Sigma^*,\ \delta^*(p, x) \in F \Leftrightarrow \delta^*(q, x) \in F$$

So the languages defined by their starting states are the same.

Remark 3.3. $p \not\approx q$ means $\exists x \in \Sigma^*$ such that

$$\begin{gathered}(\delta^*(p, x) \in F \ \wedge \ \delta^*(q, x) \notin F) \\ \vee \\ (\delta^*(p, x) \notin F \ \wedge \ \delta^*(q, x) \in F)\end{gathered}$$

Observe that $\approx$ is an equivalence relation.

Lemma 3.4

$p \approx q \Rightarrow \forall a \in \Sigma\ \delta(p,a) \approx \delta(q,a)$. But not necessarily $\delta(p,a) = \delta(q,a)$.

$$[p] \coloneqq \{q \mid p \approx q\},\ p \approx q \Leftrightarrow [p] = [q]$$

This means that $[p] = [q] \Rightarrow [\delta(p,a)] = [\delta(q,a)]$.

We define a new machine $M' = (S', s_0', \delta', F')$.

$$S' = S/\approx,\ s_0' = [s_0],\ F'\{[s]\ \}$$

This is a simpler draft

Lemma 3.5

$$p \in F\ \wedge p \approx q \Rightarrow q \in F$$

This is trivial when considering $\epsilon \in \Sigma^*$

Lemma 3.6

$\forall w \in \Sigma^*,\ \delta'^*([p], w) = [\delta^*(p, w)]$

Proof. By induction on the length w. □

Theorem 3.7

$L(M) = L(M')$

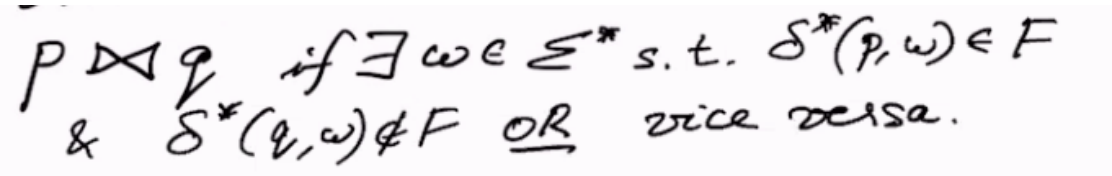
Proof.

$$x \in L(M') \Leftrightarrow \delta'^*([s_0], x) \in F'$$
$$\Leftrightarrow [\delta^*(s_0, x)] \in F' \Leftrightarrow \delta^*(s_0, x) \in F$$
$$\Leftrightarrow x \in L(M)$$

□

Quotient construction. Might be that every equivalence class contains element, but maybe each one contains several states. But definitely didn't get a bigger machine. This process is called collapsing a machine.

§3.2 Splitting Algorithm



Fact 3.8. If $\exists a \in \Sigma$ such that $\delta(p,a) bowtie \delta(q,a)$, then $p bowtie q$. The contrapositive of the above.

Matrices of booleans. You can multiply matrices of booleans by using "and" and "or". The collection of $n \times n$ matrices over a ring is always a ring.

Algorithm

1. For every (p,q) such that $p \in F \wedge q \notin F$, put 0 in the (p,q) cell of the matrix.
2. Repeat until no more changes. For each pair (p,q) that is not already marked with a 0, check if $\exists a \in \Sigma$ such that $(\delta(p,a), \delta(q,a))$ is marked with 0. If so mark (p,q) 0.

This algorithm correctly computes the equivalence classes. The time complexity is $O(n^4)$ in the naive case. $O(n^2 k)$ using data structures. $O(n \log n)$ Hopcraft method.

Theorem 3.9

If at the end two states are not marked 0 then they are equivalent.

Theorem 3.10

There is a unique minimal automaton to recognize a regular language.

This is how you prove the equivalence of regular expressions.

Definition 3.11 (Right invariant)**.** An equivalence relation R on Σ^* is said to be right-invariant if

$$\forall x, y \in \Sigma^* \text{ such that } xRy \Rightarrow \forall z \in \Sigma^*,\ azRyz$$

Example 3.12

DFA $M = (Q, \Sigma, q_0, \delta, F)$,

$$xR_m y \Leftrightarrow \delta^*(q_0, x) = \delta^*(q_0, y)$$

Example 3.13

Given $L \subseteq \Sigma^*$, not necessarily regular, we define R_L on Σ^*

$$xR_L y \Leftrightarrow xz \in L \Leftrightarrow yz \in L$$

Note 3.14. The index of an equivalence relation is the number of equivalence classes. Infinite equivalency classes means infinite index.

Theorem 3.15 (Myhill-nerode 1957)
The following are equivalent

1. L is regular
2. L is the union of some of the equivalence classes of a right-invariant equivalence relation R of finite index.
3. Any equivalence relation R with the properties described in (2) has to refine R_L.

Note 3.16. To say that an equivalence relation refines another means that it partitions all the equivalence relations.

Full proof is in the notes.

§3.3 Brozozowski's Algorithm

Will convert some DFA to NFA.

1. Reverse all the arrows change the initial states to an accept state, and make all the accept states start states.
2. Determinize it. This may cause exponential blow up because you are looking at the set of all subsets. Convert it from NFA to DFA.
3. Remove unreachable states.
4. Repeat 1.
5. Repeat 2. Might expect double exponential blow up, but instead there is a huge collapse to the minimal version of the original machine.
6. Repeat 3.

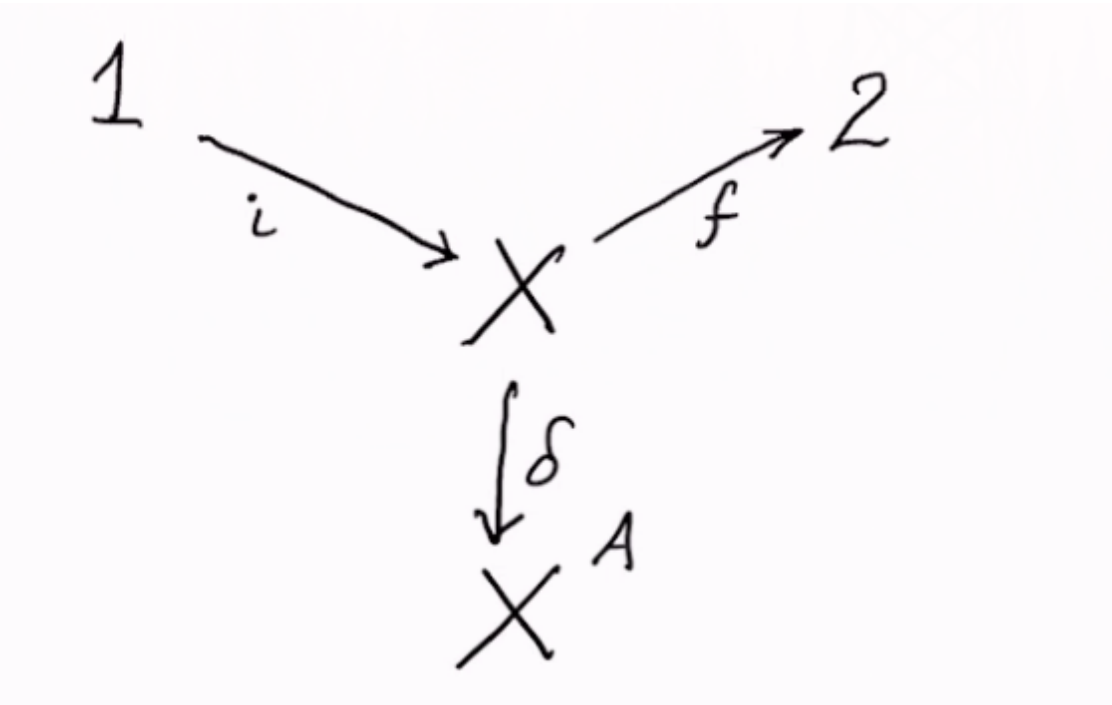
The result is the minimal machine.

Can't explain the whole thing. The point is to get interested and excited about this topic.

Note 3.17. X^A means the set of all functions from A to X. $\delta(x) : X \to X^A$.

$$X^A \approx [A \to X]$$

We can either write $\delta(x)(a)$ in which case $\delta : X \to X^A$ or $\delta(x, a)$ where $\delta : X \times A \to X$. This process is called currying.



§3.4 Infinite state automata

A^* is set of all words. Every word is a state. Initiale state is empty word.

States are words. Transition is stick the new letter at the end of the word.

There exists a unique map $r : A^* \to X$ such that the diagram above commutes.

Whatever path you follow that leads to the same end state results in the same outcome. Same as moving a different path with different endpoints.

Note 3.18. Give $f : V \to W$, $f^A : V^A \to W^A$. $\varphi \in V^A$, then $f^A(\varphi)(a) = f(\varphi(v))$. This is countably infinite.

Definition 3.19 (Reachability)**.** r maps the empty string to the . r tracks every word through the automaton with δ^*. r is the reachability map. It tells you which states can be reached. An automaton is reachable if every state can be reached.

§4 05-07

Lemma 4.1 (Pumping Lemma)

$$L = \{a^n b^n \mid n \geq 0\}$$

Not possible to recognize this language with finite automaton because it would require unbounded memory. Non regular languages could recognize this.

Suppose we have putative (generally considered or reputed to be) DFA that recognizes L. Then we can pick a word with a block of a with length greater that the number of states in the machine. This means that the machine will have to repeat a state. This rests on the pigeon hole principle.

The idea is that now you can exploit this loop as many times as you'd like by inserting the string that brings you about the loop.

Now for the lemma. Let L be a regular language. Then

$$\exists p > 0 \text{ such that } \forall w \in \Sigma^* \mid w \in L \wedge |w| \geq p$$
$$\exists x, y, z \in \Sigma^* \text{ such that } w = xyz,\ |xy| \leq p,\ |y| > 0$$
$$\forall i \in \mathbb{N},\ xy^i z \in L$$

An intuition for this is that y is the word that takes you through the loop, so you can repeat it as many times as you'd like.
Given a DFA for L choose p to be strictly greater than the number of states in the DFA. If $|w| \geq p \wedge w \in L$ then as the automaton changes state it must hit the same state twice while reading the first p letters, when p is greater than the number of states in the machine. □

Definition 4.2 (Finite Language)**.** A finite language is one containing a finite number of words.

Fact 4.3. Every finite language is regular. The p that you choose is longer than any word in the language. So then a finite language would have no words of length greater than p.

Fact 4.4. L regular $\Rightarrow$ L can be pumped. Contrapositive is that L cannot be pumped $\Rightarrow$ L not regular.

Note that it is not true that L can be pumped $\Rightarrow$ L regular.

Lemma 4.5 (Pumping Lemma Contrapositive)

$$L \subseteq \Sigma^* \ s.t. \ \forall p > 0$$
$$\exists w \in L \ with \ |w| \geq p \ s.t.$$
$$\forall x, y, z \in \Sigma^* \ with \ w = xyz, \ |xy| \leq p \wedge |y| > 0$$
$$\exists i \in \mathbb{N} \ s.t. \ xy^i z \notin L$$
$$\Rightarrow L \text{ is not regular}$$

Use games to deconstruct this statement. You and the devil. You play the $\exists$ quantifiers, and the devil plays the $\forall$ quantifiers. You must come up with a strategy to win every game.

The obvious first move is represented by a symbolic p. Your first move is explicit. The devil's move in step 3 must be analyzed by an exhaustive case analysis. Your last move must specify a response for all cases.

Example 4.6

$L = \{a^n b^n \mid n \geq 0\}$

1. Demon choses $p > 0$
2. You chose $w = a^p b^p$
3. The devil is constrained by $|xy| \leq p$ to choose y to consist exlucisvely of a's. So $y = a^k$ for some $0 < k \leq p$.
4. I choose $i = 2$. Didn't quite catch the rest

Thus L is not regular.

Example 4.7

$L = \{a^q \mid q \text{ a prime number}\}$

Demon picks $p > 0$. I pick a^n where $n > p$, n is a prime. Demon has to pick $y = a^k$ where $0 < k \leq p$. I pick $i > 1$, deferring the exact choice. New string $xy^i z$ is $a^{n+(i-1)k}$. Choose $i = n + 1$. Then $a^{n+nk} = a^{n(1+k)}$ which is not a prime number so L is not regular.

Example 4.8

$L = \{a^n b^m \mid n \neq m\}$

Wants you have a stock of languages that you know are not regular, you don't always have to do pumping. This example is hard to do directly with the pumping lemma.

$\overline{L}$ (L complement) is a big mess. But $\overline{L} \cap a^*b^* = \{a^n b^n \mid n \geq 0\}$. This is not a regular language to L is not regular.

Example 4.9

$L = \{a^i b^i \mid i > j\}$.

Example 4.10

$L = \{x + y = z \mid xyz \in \{0,1\}^* \wedge \text{the equation is valid}\}$

Demon picks p. I pick

$$\underbrace{111\cdots1}_{p}$$

Definition 4.11. If $S \subseteq \mathbb{N}$, define $unary(S) = \{1^n \mid n \in S\}$. $binary(S) = \{w \in \{0,1\}^* \mid w\text{read as a binary number is in}S\}\}$

If $binary(S)$ is regular does that mean $unary(S)$ is regular? No. Consider $S = \{2^n \mid n \geq 1\}$. Then binary is regular because 100^* is clearly regular. $unary(S) = \{1^{2^p}\}$ is not regular because you can pump to a non power of 2.

§4.1 Kleene Algebras

Definition 4.12 (Semi-ring)**.** A set with 2 operations. S: the carrier of the semi-ring. $+ : S \times S \to S$. $\times : S \times S \to S$. $0 : S$, $1 : S$. $(S, +, 0)$ forms a commutative monoid. $(S, x, 1)$ forms a monoid. $\times$ distributes over $+$. 0 annihilates with x i.e. $a \times 0 = 0 \times a = 0$.

Semi-ring is similar to a ring, but doesn't require that each element has an additive inverse. i.e. if $(S, +, 0)$ forms a group then it produces a ring instead of a semi-ring.

Example 4.13

$(\mathbb{N}, 0, 1, +, \times)$. This forms a semi ring, because we don't have the negative integers for the additive inverses. $(\mathbb{Z}, 0, 1, +, \times)$ would form a ring.

$\mathbb{Z} + i\mathbb{Z} = \{a + ib \mid a, b \in \mathbb{Z},\ i^2 = -1\}$ is the ring of Gaussian integers.

Example 4.14

$n \times n$ matrices over $\mathbb{N}$. Multiply by matrix multiplication and add componentwise.

Example 4.15

An idempotent semiring J is a semi ring such that $\forall x \in J, x^2 = x$. $(T, F, \vee, \wedge)$.

idempotent is an element which is unchanged in value when multiplied or otherwise operated on by itself.

Example 4.16

If we have any semiring, the set of $n \times n$ matrices with entries in this semiring form a semi-ring.

Definition 4.17 (Kleene Algebras)**.** $K = (S, +, \cdot, 0, 1, *)$.

1. $(S, +, 0)$: commutative monoid
2. $(S, \cdot, 1)$: monoid
3. $(S, +, \cdot, 0, 1)$ forms an idempotent semiring.
4. $$1 + aa^* = a^*$$ $$a + a^*a = a^*$$
5. We introduce a partial order $a \leq b \coloneqq a + b = b$ (check that this is really a partial order). 2 rules.
$$b + ac \leq c \Rightarrow a * b \leq c$$ $$b + ca \leq c \Rightarrow ba^* \leq c$$

Example 4.18

Let Σ be a finite alphabet $S =$ regular languages $\subseteq \Sigma^*$. $+$ is union. $\cdot$ concatenation. $*$ is kleene star.

Example 4.19

S any set and R the collection of binary relations on S. A binary relation $r \subseteq S \times S$. $+$ is union. $\cdot$ is relational composition. xry means $(x, y) \in r$. $x(r \cdot s)y = \exists z s.t. xrz \wedge zsy$. $0 \coloneqq \varnothing$. $1 \coloneqq \{(s, s) \mid s \in S\}$. r^* is the reflexive, transitive closure of r.

graphs are a nice way of picturing relations. r^* is reflexive, transitive closure of r. transitive closure let's you take the paths of the graph. And everything is related to itself. Directed graph because not necessarily symmetric. xr^*y if $\exists n \in \mathbb{N}, z_1, \cdots, z_n s.t. xrz_1 \wedge z_1 r z_2 \cdots z_n ry$. i.e. there exists a path from x to y.

Solving for x. $ax + b = x$. a^*b is the smallest solution. $aa * b + b = (aa * +1)b = a^*b$. Smallest solution means that if x is another solution, then $a^*b \leq x$.

Example 4.20

If K is any kleene algebra, $M_n(K)$ is $n \times n$ matrices with entries in K. Do operations as you would expect about matrices. What the heck is star though? 0 is 0. 1 is 1 along diagonal and 0 everywhere else.

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^* := \begin{bmatrix} (a+bd^*c)^* & (a+bd^*c)^*bd^* \\ (d+ca^*b)^*ca^* & (d+ca^*b)^* \end{bmatrix}$$

Now we can solve matrix vector equations.

Dexter Kozen has developed and extended the theory tremendously including probabilistic extensions.

§5 05-11

Σ corresponds to the alphabet. Σ^* represents finite words. Σ^ω is the set of infinite words over Σ.

If $\Sigma = \{a, b\}$, Σ^ω is uncountable. $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$. A string in Σ^ω is countable, so positions in the string can be indexed by $\mathbb{N}$ i.e. countable.

There is a natural partial order on Σ^∞. $\alpha \leq \beta$ if α is a prefix of β. Infinite strings cannot be compared.
First ordinal. Reflexivity is part of a partial order.

Remark 5.1. We can define a metric on Σ^* by letting $d(x, y) = 2^{-n}$ where n is the first position where $x[n]! = y[n]$. The longer the shared prefix the closer two words are. This is an ultrametric which means that it satisfies the following strengthened version of the triangle inequality

$$d(x, z) \leq \max(d(x, y), d(y, z))$$

Σ^∞ is the cauchy completion of Σ^* and the metric naturally extends to Σ^*.

Infinite strings are important for the verification of reactive systems. They are designed to run forever so they have to talk about infinite computations.

§5.1 ω regular expressions.

Let E be a regular expression and $\epsilon \notin L(E)$.
E^ω is an ω-regular expression.

$$L_\omega(E^\omega) := \{w_1 w_2 w_3 \cdots \mid w_i \in L(E)\}$$

where $L_\omega(F)$ is the ω-language defined by the ω regular expression F.
Let E_i be regular xpressions and F_i be regular expressions such that $\epsilon \notin L(F_i)$.

$$E_1(F_1)^\omega + \cdots + E_n(F_n)^\omega$$

is a general ω-regular expression.
$E^\omega = EE^\omega$ so E^ω is also an ω-regular expression.

When concatenating where $\alpha, \beta \in \Sigma^\omega$, $\alpha\beta = \alpha$.

Definition 5.2. An ω-language ($\subseteq \Sigma^\omega$) is ω-regluar if defined by some ω regular expression. This ω-regular expression may not be unique.

Note 5.3. $\cdot$ is concatenation. $+$ is union. $(.)^\omega$ is infinite repetition.

§5.2 ω-automata

Specifically non deterministic biichi automata.

$$A = (Q, \Sigma, \delta, Q_0, F)$$
$$\delta : Q \times \Sigma \to 2^Q$$
$$s' \in \delta(s, a) \equiv s \to_a s'$$

Given a word $\sigma \in \Sigma^\omega$, a run of ω in A is an infinite sequence of states $q_0, q_1, \cdots$ such that $q_0 \in Q_0$, $\forall i\ q_i \to_{\sigma[i]} q_{i+1}$.

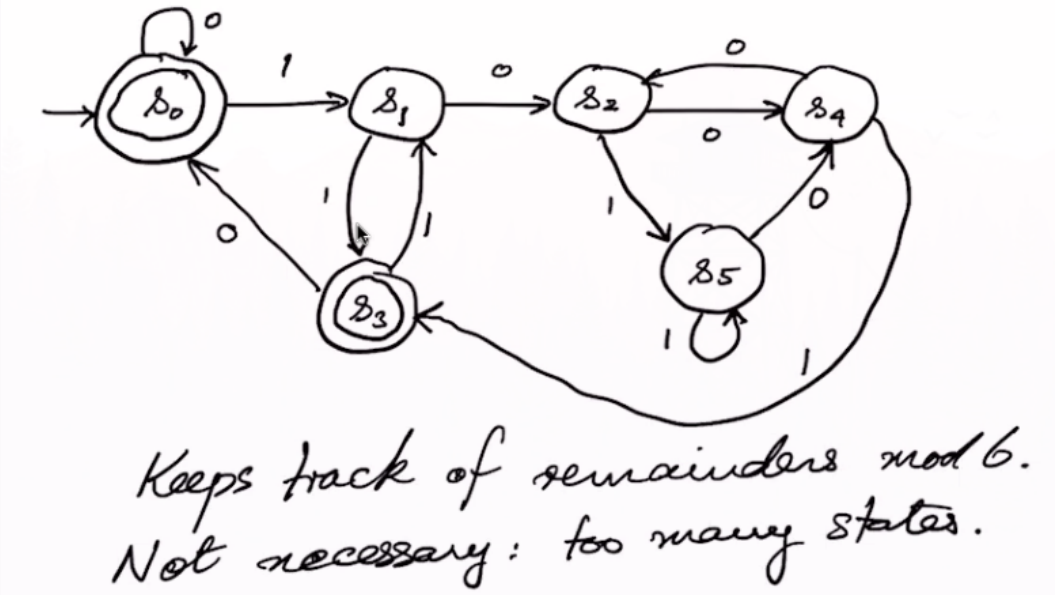
$$inf(r) = \{q \mid q \text{ occurs infinitely often in} r\}$$

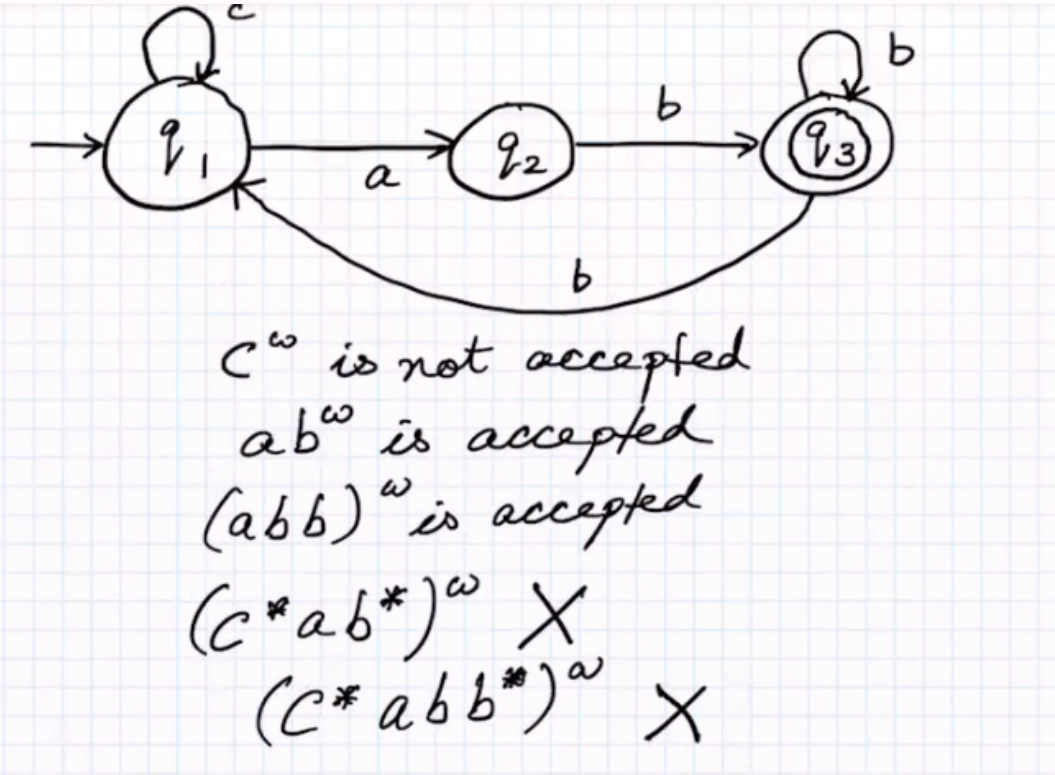
A run is accepting if it hits an accept state infinitely often. There are only finitely many accept states so one of them has to be hit infinitely many times.

$$inf(r) \cap F \neq \varnothing$$

A word is accepted by A if that word has some accepting run.

Example 5.4





$$L_\omega(A) = c^*ab(b^+ + bc^*ab)^\omega$$

We see that it doesn't necessarily have to be periodic, but perhaps a periodic aspect to it. Get it to the start state first, then think of a pattern.

Theorem 5.5

NBA's accept exactly the ω-regular languages.

Proof. Show that any ω-regular language can be captured by an NBA.

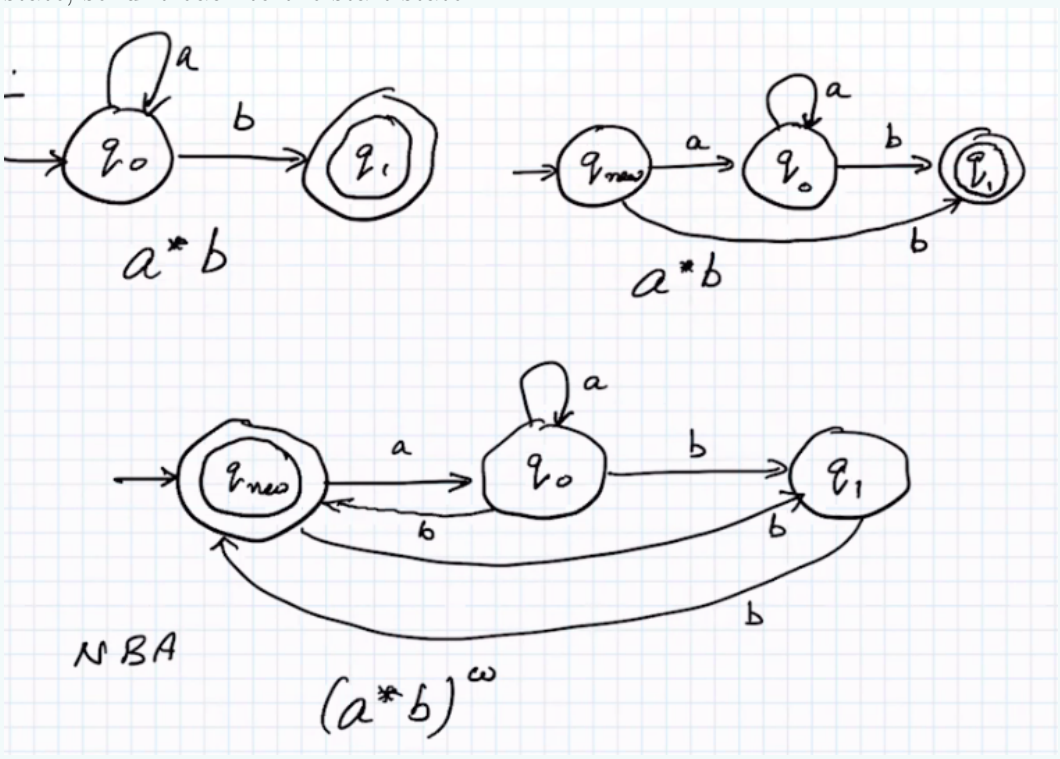
Given $A_i = (Q_i, \Sigma, \delta_i, Q_{0,i}, F_i)$, $i = 1, 2$ with $Q_1 \cap Q_2 = \varnothing$. Then $A_1 + A_2 = (Q_1 \cup Q_2, \Sigma, \delta, Q_{0,1} \cup Q_{0,2}, F_1 \cup F_2)$. You can start either in start states of A_1 or A_2.

$$\delta(q, a) = \delta_i(q, a) \text{ if } q \in Q_i$$
$$L_\omega(A_1 + A_2) = L_\omega(A_1) \cup L_\omega(A_2)$$

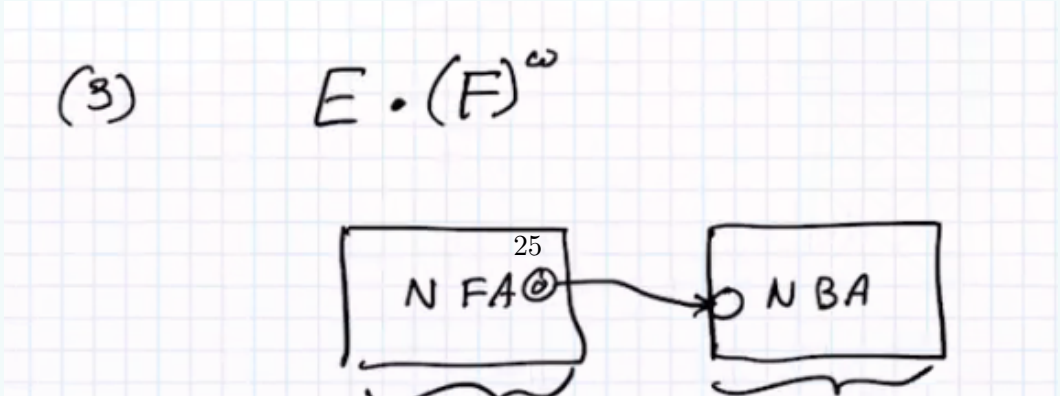
We are now considering F^ω. We have an NFA for F and assume that all the inital states are non accepting. $Q_0 \cap F = \varnothing$ and there are no incoming transition to any state in Q_0. You may worry that this will be too restricting on applicable NFAs. But if your NFA does not satisfy these assumptions we can modify it so that it does and defines the same language.

Idea for doing this: Brand new unique start state. And then leave everything afterwards untouched.

Given $A = (Q, \Sigma, \delta, Q_0, F)$. NBA $A' = (Q, \Sigma, \delta', Q_1, F')$. $L_\omega(A') = (L(A))^\omega$. The goal is to keep recognizing infinitely many words. So each time you hit an accept state, send it back to the start state.



Case three. NFA . NBA.



Theorem 5.6

Other direction

Suppose $A = (Q, \Sigma, \delta, Q_0, F)$ NBA. Fix any 2 states. $A_{qp} \coloneqq (Q, \Sigma, \delta, \{q\}, \{p\})$. This converts it to an NFA. It is not meant to read omega words.

$$L(A_{qp}) = L_{qp} = \{w \in \Sigma^* \mid \delta^*(q, w) \ni p\}$$

These are words that take you from q to p.

Now consider an accepted word σ of $L_\omega(A)$ with an accepting run. It must hit sum $q \in F$ that appears infinitely often. Therefore

$$\sigma = \underbrace{w_0}_{L_{q_0 q}} \underbrace{w_1}_{L_{qq}} \underbrace{w_1}_{L_{qq}} \cdots$$

where w_i are non empty. Any σ the can be expressed like this has an accepting run. Therefore

$$L_\omega(A) = \cup_{q_0 \in Q_0} \cup_{q \in F} L_{q_0 q} \cdot (L_{qq} \setminus \{\epsilon\})^\omega$$

Therefore anything recognized by an NBA is described by an ω-regular expression. Kleene's theorem for ω-languages.

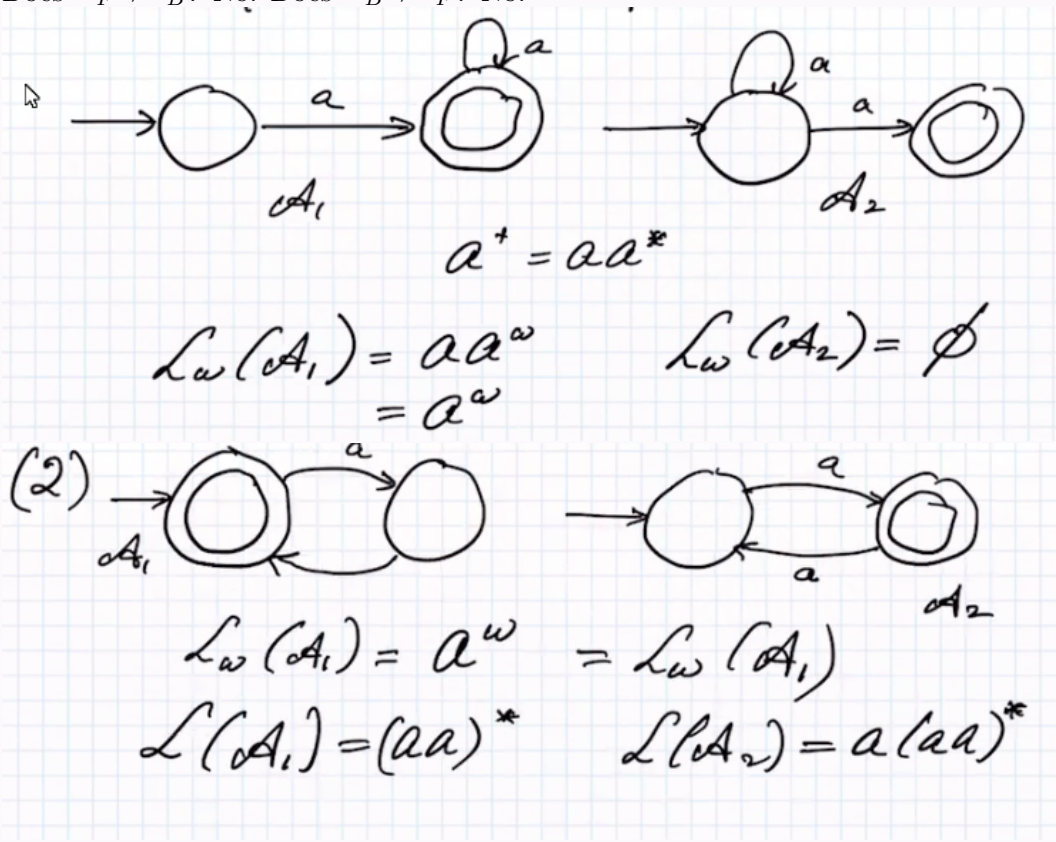
Lemma 5.7

$L_\omega(A) \neq \varnothing$ iff $\exists$ a reachable accept state that belongs to a cycle.

$A_1 \equiv_F A_2$ they are the same as NFA. $L(A_1) = L(A_2)$. $A_1 \equiv_B A_2$ they are the same as NBA. $L_\omega(A_1) = L_\omega(A_2)$.

Example 5.8

Does $\equiv_F \Rightarrow \equiv_B$? No! Does $\equiv_B \Rightarrow \equiv_F$? No!



Idea here is that it distinguishes even length from odd length, but infinity length is both even and odd.

Theorem 5.9

NBAs and DBAs are not equivalent. There is no DBA that can recognize $(a+b)^*b^\omega$. This is what makes omega automata much trickier.

Proof. Suppose we have a DBA for this language. We can define δ^* as usual on the underlying DFA. Consider the word $b^\omega \in (a+b)^*b^\omega$

Since b^ω is accepted, we must hit an accept state at some point. Therefore $\delta^*(q_0, b^{n_1})$ is an accept state. $b^{n_1}ab^\omega \in L$. This has to hit an accept state again after reading a. You can continue this process. Have to hit the same accept state twice. But then $\exists i, j, i < j$ where

$$\delta^*(q_0, b^{n_1}ab^{n_2}\cdots ab^{n_i}) = \delta^*(q_0, b^{n_1}ab^{n_2}\cdots ab^{n_j})$$

But then there is a loop. So you can include infinitely many b in the language. Contradiction. So NBA are strictly more expressive than DBA. □

Let $\alpha \in \Sigma^*\omega$ and define $\alpha|_n \in \Sigma^*$ to be the length of n finite prefix.

Definition 5.10. For $W \subseteq \Sigma^*$ we define $\vec{W} = \{\alpha \in \Sigma^\omega \mid \exists \text{ infinitely many } n \in \mathbb{N} s.t.\ \alpha|_n \in W\}$ Words where you can always make the prefix in W longer. $\vec{W}$ is called a limit language.

Theorem 5.11
An ω-regular language L is recognizable by a deterministic BA iff there is a regular language W such that $L = \vec{W}$.

Proof. DIY □

Theorem 5.12
DBA are closed under complement.

If A is a DBA, $\exists$ an NBA A' such that $L_\omega(A') = \Sigma^*\omega \setminus L(A)$.

$A' = (Q', \Sigma, \delta', Q_0', F').A = (Q, \Sigma, \delta, Q_0, F)$.

$$Q' = (Q \times \{0\}) \cup ((Q \setminus F) \cup \{1\})$$
$$Q_0' = Q_0 \times \{0\}$$

§6 05-12

§6.1 Basic Propositional Logic

Lot's of confusion about what is meant by a true statement vs valid statement vs theorem.

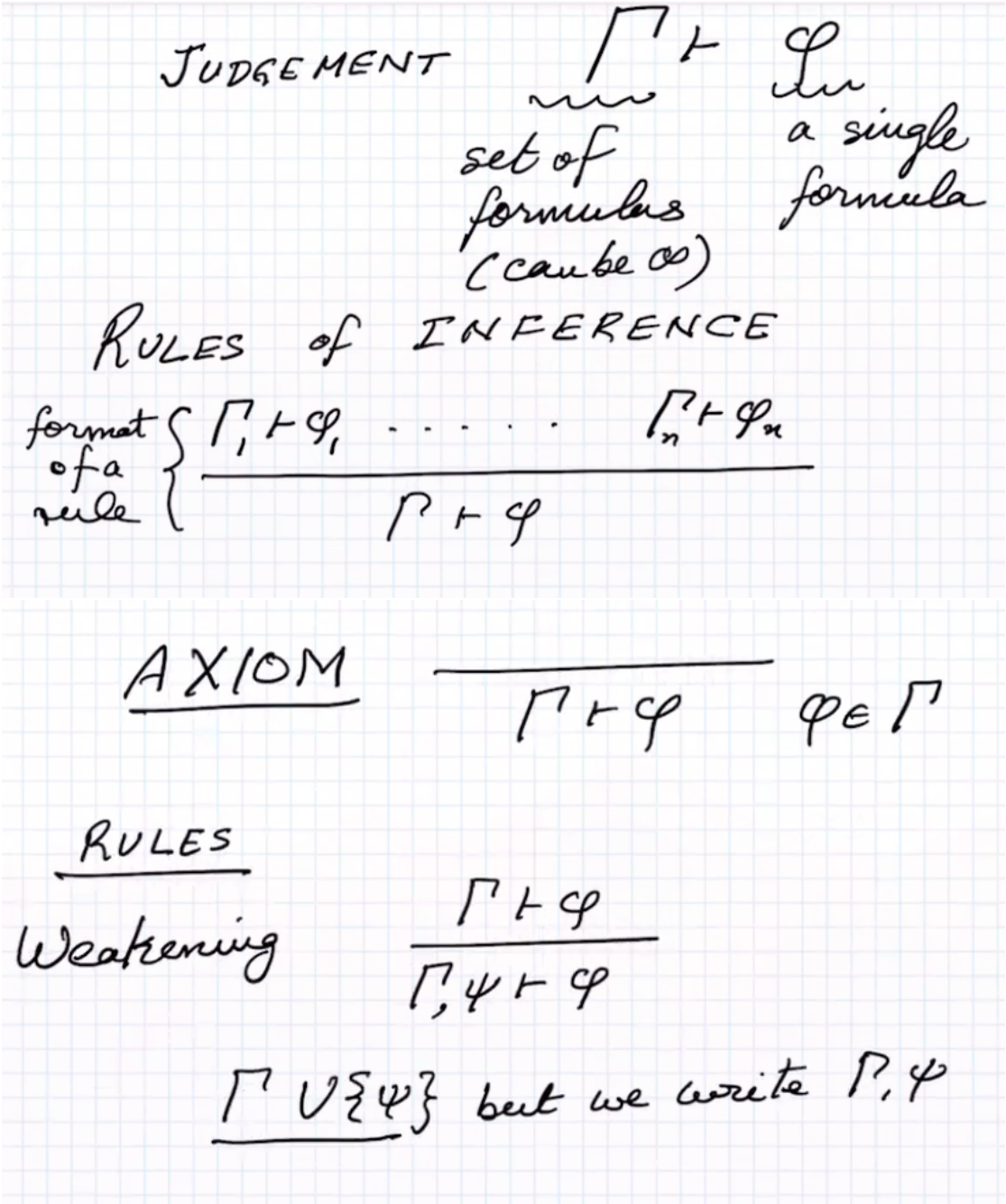
§6.1.1 Syntax

PROP $= \{p, q, r, \cdots\}$ are propositional variables.
φ, ψ are meta-variables for describing generic propositional formulas.
Formulas are defined inductively. Inductive because you have some formulas and you give rules for building new formulas from old ones.
This is syntax. What I'm allowed to write, but explaining anything about what they mean. Explaining what they mean is called semantics.

1. T for true is a formula, and $\bot$ for false is a forumla.
2. Any propositional variable is a formula
3. If φ is a formula, so is $\neg\phi$.
4. If φ and ψ are formulas, so are $\varphi \wedge \psi,\ \varphi \vee \psi,\ \varphi \Rightarrow \psi$

§6.1.2 Proof Theory

How to use and manipulate formulas in deduction. Proof theory is not about being true. It's about true.
Gamma is the set of assumptions you are making.
$\gamma \cup \{\psi\}$ but we write γ, ψ



You can do introduction or elimination.
$\neg\varphi$ is a marco for $\varphi \Rightarrow bot$.
A proof is a tree whose leaves are axioms, the nodes are rules instances, and the rute is a single judgement $\gamma \vdash \varphi$.
If the root has the form $\vdash$ we say that φ is a theorem.

§6.1.3 Semantics

Interpret logical formulas as elements of a simple mathematical structure.

A valuation v : PROP $\to \{0, 1\}$. Extend v to a map on formulas by structural in-

duction.

$$v(T) = tt \quad v(\bot) = ff$$
$$v(\varphi) \underbrace{\wedge}_{\text{Boolean algebra}} v(\varphi') = v(\varphi \underbrace{\wedge}_{\text{Syntax}} \varphi')$$

If $\models \varphi$, then $\forall v,\ v(\varphi) = tt$. Such a formula is called a tautology.

$\vdash$ denotes a syntactic implication while $\models$ denotes a semantic implication.

Theorem 6.1 (Soundness)
If $\gamma \vdash \varphi$ then $\gamma \models$ in particular if $\vdash \varphi$ then $\models \varphi$.

Theorem 6.2 (Completeness)
If $\gamma \models \varphi$ then $\gamma \vdash \varphi$.

Suppose that γ has no valuation such that $\forall \varphi \in \gamma,\ v(\varphi) = tt$. Then we say that γ is unsatisfiable.

Theorem 6.3 (Compactness)
If γ is unsatisfiable, then some finite subset of γ is unsatisfiable.

Remark 6.4. Tepmoral logic fails this property.

§6.2 Temporal Logic

as opposed to propositional logic. The basic atomic properties can change their truth values in time.
A mir Puuli showed this is very useful for reasoning about code execution especially concurrent programs.

§6.2.1 Syntax

PROP $= \{p, q, r, \dots\}$. Formulas $\neg, \wedge, \dots$. Two temporal operators O, u. If φ is a formula then 0φ is a formula. If φ, ψ are formulas, then $\varphi u \psi$ is a formula.
$\varphi = true | p | \varphi_1 \wedge \varphi_2 | \neg\varphi | O\varphi | \varphi u \psi$
Intuition. Instead of a valuation we have a linear sequence of valuations. We will call these valuations states. The entire sequence is called a history.
Defined operators.

§6.3 Labelled Transition System

$(S, Act, \to, I, AP, L)$ S states perhaps infinite. Act actions. $\to \subseteq S \times A \times S$. represents function $a\ s \to s'$ $I \subseteq S$. initial states. AP are the set of atomic propositions.

§7 05-13

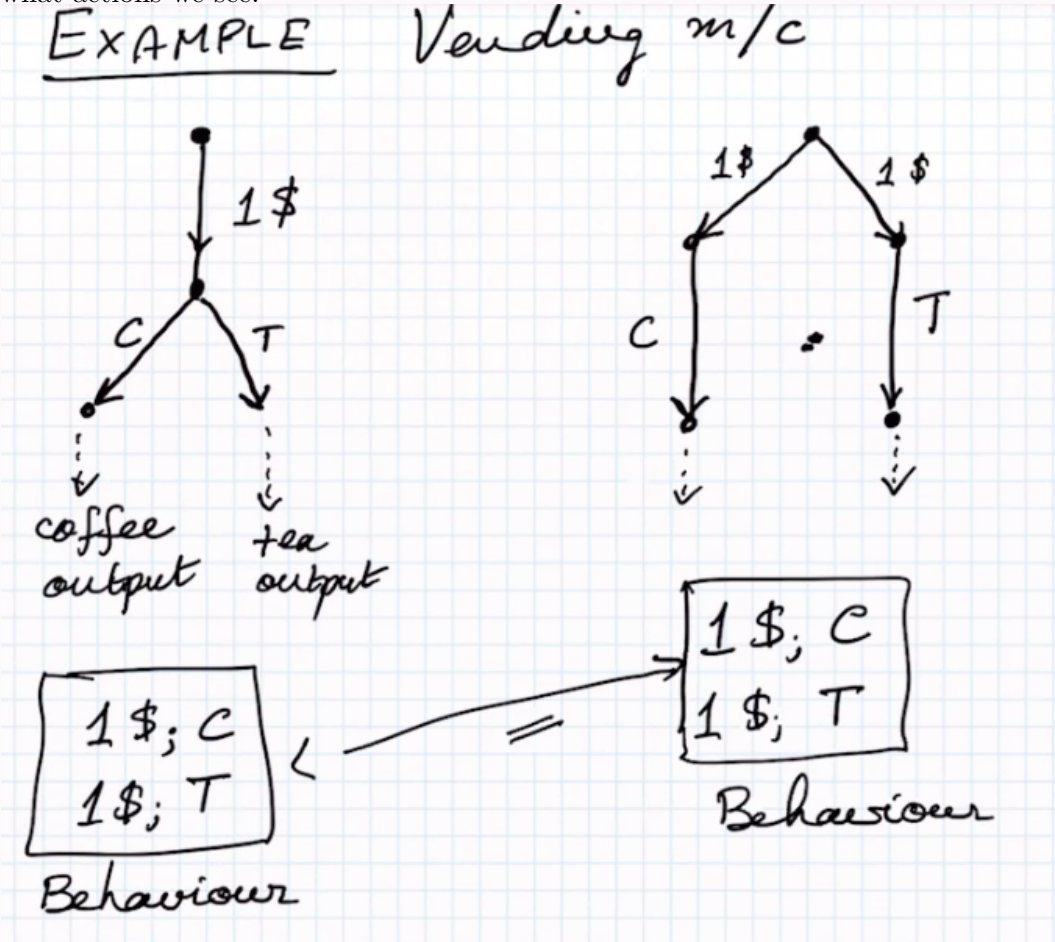
§7.1 Bisimulation

Transition systems as models of software or hardware or embedded systems.

Nondeterminism, need not be finite state.
When are two systems observably the same?

Example 7.1 (Vending Machine)

Difference from DFA, some actions get rejected. The rejection is different from the rejection in an NFA because really that encodes a state where you can't escape, while this is more true rejection. No start states or accept states. Just interested in what actions we see.



Internal vs. external choice is what makes these two machines different. This shows how sequences are inadequate description of the behavior. So how can we actually compare to LTS? With bisimulation.

Definition 7.2 (LTS (Labeled Transition System))**.**

$$S \to \text{ set of states, perhaps infinite}$$
$$A \to \text{ set of actions, finite}$$
$$\to \subseteq S \times A \times S$$
$$(s, a, s') \in \to, \quad s \xrightarrow{a} s'$$

From state s, if action a is performed, you can end up in s'.

Definition 7.3 (Bisimulation (semi-formed))**.** When comparing two systems, we form a binary relation between the states of a simple. This isn't a big deal because we could merge the two systems.

We say $s, t \in S$ are bisimilar (written $s \sim t$) if

$$\forall a \in A \; s \xrightarrow{a} s' \Rightarrow \exists t' s.t. t \xrightarrow{a} t' \text{ and } s' \sim t'$$
$$\forall a \in At \xrightarrow{a} t' \Rightarrow \exists s' s.t. s \xrightarrow{a} s' \text{ and } s' \sim t'$$

If s can do something, t can do the same thing. And they may end up in different states, but those states themselves will be bisimilar. This should make you uneasy because it's an inductive definition with no base case.

There are 4 ways of clarifying this definition.

1. We define an $\mathbb{N}$ indexed family of equivalence relations (infinitely many of them) as follows: $s \sim_0 t$ always. Then

   $$s \sim_{n+1} \; if \; \forall a \in A \; s \xrightarrow{a} s' \Rightarrow \exists t' s.t. t \xrightarrow{a} t' \text{ and } s' \sim_n t'$$
   vice versa

   Now we say that $\sim = \cap_n \sim_n$

   In the vending machine example $s_0 \sim_1 t_0$ but $s_0 \not\sim_2 t_0$ and hence they are certainly not bisimilar systems.

2. R the family of equivalence relations $\subseteq S \times S$ order by inclusion. Smallest is everything is related to itself. Largest is everything related to everything. This forms a complete lattice. $F : R \to R$.

   $$sF(R)t \text{ if } \forall a s \xrightarrow{a} s' \Rightarrow \exists t' s.t. t \xrightarrow{a} t' \text{ and } s' R t'$$
   vice versa

   F is easily seen to be monotone i.e. if $R_1 \subseteq R_2$ then $F(R_1) \subseteq F(R_2)$. It follows that there is a unique greatest fixed point. i.e. a special $\sim \in R$ such that $F(\sim) = \sim$ and if R is any relatoin such that $F(R) = R$ then $R \subseteq \sim$. This is called fixed point bisimilarity.

3. We saw R is a dynamic relation or a bisimulation relation if whenever sRt then .

   $$\forall a \forall s' \; s \xrightarrow{a} s' \Rightarrow \exists t' s.t. t \xrightarrow{a} t' \text{ and } s' R t'$$
   vice versa

   Note that this is not circular. R is given somehow and it may or may not have this property. We say that $s \sim t$ if $\exists R$, a bisimulation relation with sRt.

   Fact 7.4. If R_i is any family of bisimulation relations, then $\cup_i R_i$ is also a bisimulation as is $\cap_i R_i$. $\sim = \cup_{R_a} R$.

   Easy to see that (2) and (3) are equivalent. (1) is not equivalent without a further assumption.

§7.2 Lattice Theory and Fixed Points

Remember that a poset is a partially ordered set. i.e. a set equipped with a partial order.

Given $(S, \leq)$, we say that u is the least upper bound of $X \subseteq S$ if

$$\forall x \in X, \; x \leq u \text{ and } \forall y \in S, if \; \forall x \in X, x \leq y \Rightarrow u \leq y$$

If $X \subseteq S$ we say v is a lower bound of x if $\forall x \in X,\ v \leq x$. If v in addition is greater than any other lower bound, we call it the Infimum.

Definition 7.5. In a lattice there is a supremum and infimum for every pair of elements. They are not necessarily unique. By induction it follows that every finite set of elements has a supremum and infimum. Infinite numbers may not.

Definition 7.6 (Complete Lattice)**.** A lattice is complete if every subset has a least upper bound.

Fact 7.7. It follows that every set has a greatest lower bound.

The least upper bound of $\varnothing$ is a least element of the whole set.

Proof. Let $X \subseteq L$, where L is a complete lattice. Let $V = \{v \in L \mid \forall x \in X\ v \leq x\}$ be the set of lower bounds of X.

Let $g = \sup(V)$. Claim g is $\inf(X)$.

Note 7.8. $\forall x \in X,\ \forall v \in V,\ v \leq x$. i.e. $\forall x \in X$, x is an upper bound for V so since g is the lesat upper bound for V we have that $\forall x \in X,\ g \leq x$, i.e. $g \in V$.

Since g is an upper bound for V it is greatre than any element of V so it is the glb of X.

□

Theorem 7.9

If $f : L \to L$ is monotone and L is a complete lattice, then the fixed points of f, i.e. $x\ s.t.\ f(x) = x$ forms a complete lattice in its own right. In particular there is a least fixed point and a greatest fixed point. There is a least fixed point and a greatest fixed point.

Proof. Find the upper bound of set of inflationary points and show that it is fixed. □

Note to be a bisimulation relation means $R \subseteq F(R)$. Least upper bound of $\{R \mid R \text{ is a bisimulation}\} = \cup_R R = gfp(F)$ (greatest fixed point). (2) and (3) are really the same. (1) is not the same unless the transition system has a special property $\forall s, a\{t \mid s \to_a t\}$ is finite. This is called image-finiteness.

If TS is image finite then (1) is equivalent to (2) and (3). The definition of bisimulation is an example of a co-inductive definition.

§7.3 Logical Characterization of Bisimulation

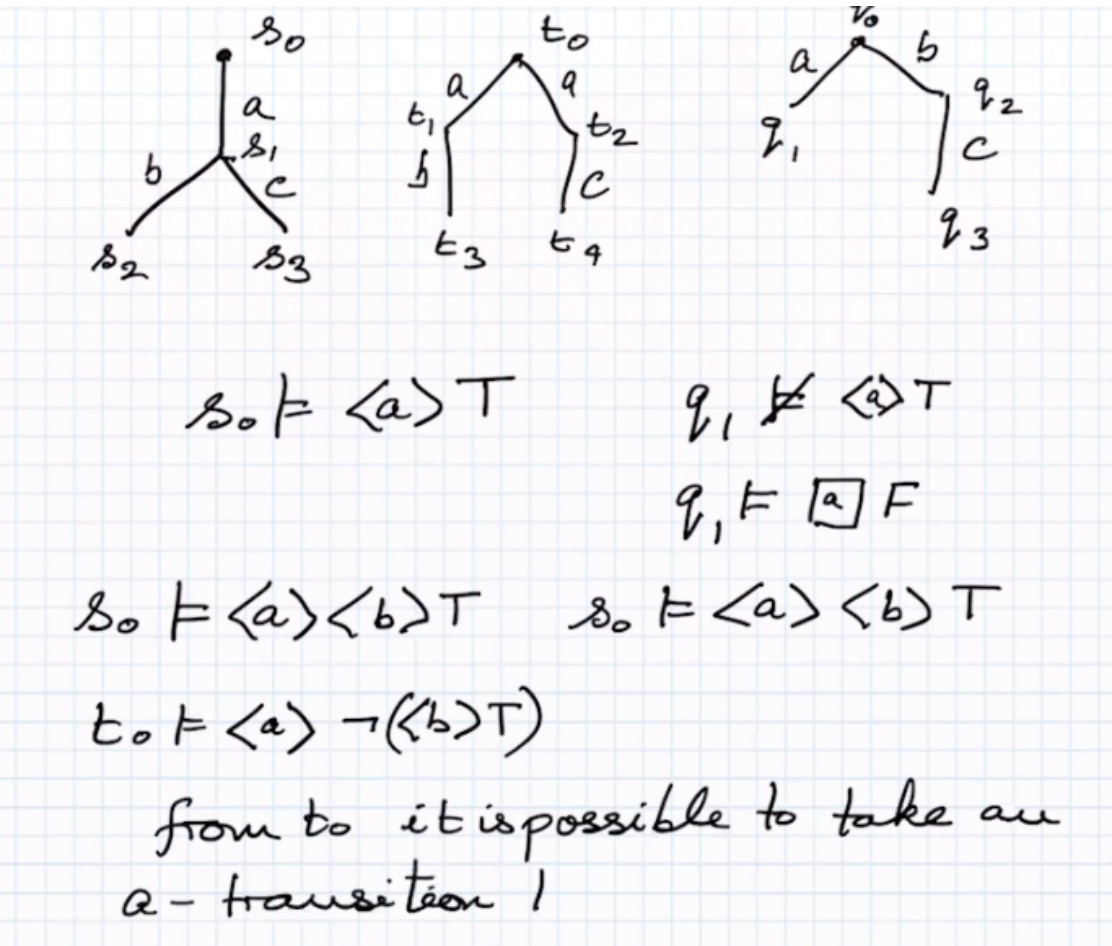
Given an LTS, we define a modal logic called Hennessy-Milner Logic.

$$\varphi ::== T \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \langle a \rangle \varphi$$

$s \models \langle a \rangle \varphi \rangle$ if $\exists t\ s.t.\ s \to_a t$ and $t \models \varphi$. At least one state satisfies.

$[a]\varphi = \neg\langle a \rangle \neg\varphi$

Every state t such that $s \to_a t$ must satisfy φ.



Theorem 7.10

$s \sim t$ if and only if $\forall \varphi s \models \varphi \Leftrightarrow t \models \varphi$ (image finiteness assumed). Don't be afraid of infinite conjunctions.

Proof. We define a new equivalence relation $\approx$. $s \approx t \Leftrightarrow \forall \varphi \; s \models \varphi \Leftrightarrow t \models \varphi$.

Idea is to prove that $\approx$ is a bisim relation. Suppose $s \approx t$ and suppose $s \to_a s'$. We want to show that $\exists t' \; s.t. t \to_a t'$ and $s' \approx t'$.

Assume that s and t are not bisimilar. Then $\forall t' \; s.t. \; t \to_a t', \; s' \not\approx t'$. There are only finitely many such $t' : t'_1, \ldots, t'_n$. □

§7.4 Probabilistic Bisimulation and Logical Characterizaiton

Probabilistic transition systems. $s \models \langle a \rangle_q \varphi$. $a \in Act$, $q \in Q \cap [0,1]$ The probability of winding up in a state sastisfying φ after doing an a action in state $s \geq q$. Larsen and Skou proved logical characterization using

$$\langle a \rangle_q \varphi \text{ and } \neg\varphi \text{ and } \varphi_1 \wedge \varphi_2$$

They also assumed a very strong finite branching property. And probabilities must be integer multiples of some fixed rational number.

AMAZINGLY proved logical characterization with no finite branching assumption and no negation in the logic. We alo got logical characterization of simulation.

§8 05-13

§8.1 Learning Automata

Dana Angluin (1987)
Model. She assumed the existence of a teacher: minimally adequat teacher.
It's always regular languages. Trying to learn a DFA. You can ask "is w in L?".
After collecting some information you propose an answer. A teacher says "Yes" or "No" and here is a counter example: a word in L that your proposed machine rejects or a word not in L which your machine incorrectlys accepts. Teacher's choice not yours.

Theorem 8.1

With polynomially man queries you are guaranteed to learn the correct unique minimal DFA for L (even if teacher has a different more complex version).

Basic data structure: Observation table. $S, E \subseteq \Sigma^*$, both finite. Helpful to think of S as states and E as experiments.

Assume you have a table with certain properties satisfied. We can propose a DFA
An observation table is said to be closed when

$$\forall t \in S \cdot \Sigma,\ \exists s \in S \ s.t.\ row(t) = row(s)$$

An observation table is consistent when

$$\forall s_1, s_2 \in S, row(s_1) = row(s_2) \Rightarrow \forall a \in \Sigma\ row(s, a) = row(s_2, a)$$

An observation table desribes a DFA iff it is closed and consistent.

Algorithm:

1. When it's not closed, you ask for more queries.
2. When it's not consistent, you expand the number of experiments.
3. When it's closed and consistent, you present your DFA. If it's wrong, you use the counter example to expand the table and start over.

Example 8.2

This is an extended example going through the process of finding the DFA. I won't write it down. These notes do a great job

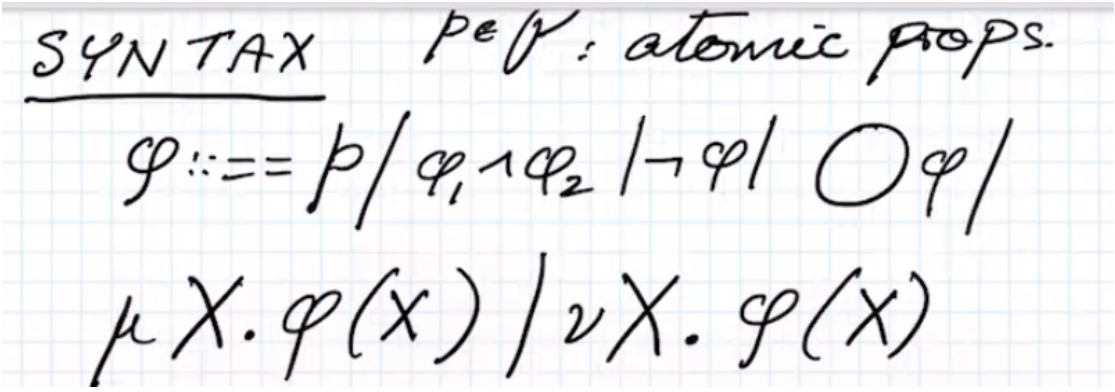
A similar algorithm works for weighted automata.

This is kinda automata theory meets machine learning. Another application is in trying to understand what an RNN does. The question of interpreting what an RNN does once it has been trained. It's important to have explainable AI.

Idea: Use an RNN as the teacher and extract a DFA as the explanation of what the RNN is doing. Gail Weiss. Alika Utepoua and Prakash are exploring this.

§8.2 Fixed Point Operators and LTL

$\Box\varphi = O\varphi \wedge OO\varphi \cdots$
This leads to infinitely many conjunctions. We should be able to prove this with induction.



X is a new syntacting entity. It is a variable that can range over formulas. μX and νX are variable binders. They are called fixed point operators.

Recall 8.3. Every monotone function from a complete lattice to itself has a least fixed point and a greatest fixed point. μ stands for least fixed point. ν stands for greatest fixed point.

We are not allowed to put a fixed point operator binding an X unless X appears in the scope of an even number of negations.

$\mu X.\varphi \wedge \neg X$ and $\mu X.X \Rightarrow \varphi$ are not allowed because they involve negated X. Remember that $x \Rightarrow y$ is the same as $\neg x \vee y$.

We defined the semantics in terms of sequences $\sigma \models \varphi$.

$$[\![\varphi]\!] = \{\sigma \mid \sigma \models \varphi\}$$

gives a definable set. Given a formula with a free variable, we interpret it not as a set but as a function from sets to sets.

$$[\![\varphi(X)]\!] = S \mapsto [\![\varphi[S/X]]\!]$$

$\varphi[S/X]$ notation for substitute the set S regarded as a formula for X in φ.

e.g. $\varphi(X) = \psi \wedge OX$.

$[\![\varphi(X)]\!](S) = \{\sigma \mid \sigma \models \psi \wedge \sigma[1..] \in S\}$

Fact 8.4. With the restriction these functions are always monotone and $P(\Sigma^\omega)$ is a complete lattice.

Now we define

$$[\![\mu X.\varphi(X)]\!] = lfp[\![\varphi(X)]\!]$$
$$[\![\nu X.\varphi(X)]\!] = gfp[\![\varphi(X)]\!]$$

It's a nightmare to interpret nested fixed points in human terms (although it is still well defined).

$\Box\varphi = \nu X.\varphi \wedge OX$