COMP2008 Logic

Robin Hirsch

November 3, 2009

What are we going to do?

- Formal Logic

- First-order, predicate logic
  - syntax
  - semantics — sets, relations, functions
  - proofs — Hilbert systems, tableau

- Main results
  - Soundness
  - Completeness
  - Compactness
  - Gödel's incompleteness theorem for arithmetic

- Applications

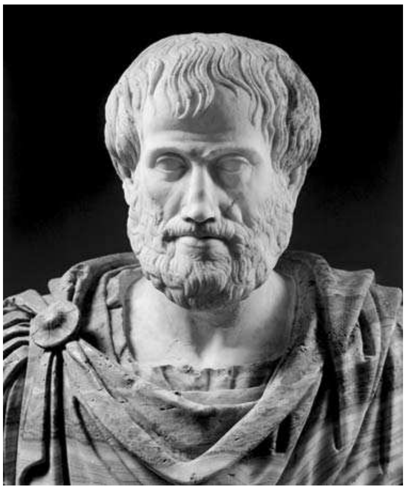
- Other logics — modal, temporal, etc.

- Models of Computation:
  - Finite State Machines
  - Regular Languages
  - Non-determinism
  - Kleene's Theorem

Reading

- "Logic: an introduction to elementary logic" by W Hodges, Penguin, 1977.
- "A friendly introduction to mathematical logic" by C Leary, Prentice-Hall, 2000.
- "Logic for Computer Scientists" by S Reeves and M Clarke, Addison-Wesley, 1999.
- See lecture notes for more reading.

Aristotle's syllogism, 4th century BC.



Identity "A is A"

Non-contradiction "You can't have both A and $\neg A$"

Excluded middle "You must have either A or $\neg A$".

Hilbert's Programme (1900)

Formalise mathematics.

1. Find a set of axioms for mathematics and show that the axioms do not contradict themselves (consistency)

2. Show that these axioms are complete, i.e. they prove all the true statements of mathematics

3. Find an algorithm that determines whether a formula is true or not.

Logic in Computer Science

- For programme specification, e.g. Z.
- for programming languages, e.g. PROLOG.
- for programme verification
- for programme design
- very important in Artificial Intelligence
- knowledge representation
- databases (SQL)

Criticisms of formal logic

- Is it that clear cut?
- Reductionism/Logical Atomism.
- Doesn't model human thought very well.

Formal Logic

Three parts.

- A language. Syntax. Grammar.
- Meaning. Interpreting the language. Semantics.
- Deduction. Proofs. A syntactic devise for proving true statements.

Propositional Logic

Syntax

$$fm ::= prop|\neg fm|(fm \circ fm)$$

where $\circ$ is $\wedge, \vee$ or $\rightarrow$.

$$prop ::= p|q|r|\ldots$$

Semantics A valuation v maps propositions to $\{\top, \bot\}$. v extends to a unique truth-functions (also called v) satisfying

$$\begin{aligned}
v(\neg\phi) = \top &\iff v(\phi) = \bot \\
v(\phi \wedge \psi) = \top &\iff v(\phi) = v(\psi) = \top \\
v(\phi \vee \psi) = \top &\iff v(\phi) = \top \text{ or } v(\psi) = \top \\
v(\phi \rightarrow \psi) = \top &\iff v(\psi) = \top \text{ or } v(\phi) = \bot
\end{aligned}$$

Validity, Satisfiability, Equivalence

- ϕ is *valid* if $v(\phi) = \top$ for all possible valuations v.

- ϕ is *satisfiable* if $v(\phi) = \top$ for at least one valuation v.

- Use truth table to find out.

- $\phi \equiv \psi$ means $v(\phi) = v(\psi)$, for all v.

Expressive Power

Propositional Logic is computationally quite good — it is decidable. But its expressive power is very limited.

- "We are all miserable"

$$\begin{aligned} p &= \text{"Robin is miserable"} \\ q &= \text{"Denise is miserable"} \\ r &= \ldots \\ &\vdots \end{aligned}$$

$$p \wedge q \wedge r \ldots$$

  Very long-winded!

- "Every even number bigger than 2 is not prime"

- "Robin is the brother of Rachel"

Sets, Relations, Functions

Unary Relations A *unary relation* (or *unary predicate*) is a property true of some things but not of others. E.g.

- *Even* is true of 6 but not of 3.
- *Male* is true of S. Stalone but not of Madonna.

Equivalently, a unary relation is just a set (the set of even number, the set of males, etc.)
Write "even(x)" to denote that x is even.
More generally "$P^1(x)$"

Binary Relations A binary relation (or predicate) is true for some *pairs* but not for others.
E.g. "less than" is true of $(3,4)$ but not of $(9,2)$.

A binary relation is a set of pairs (the set of pairs for which the relation is true). E.g.

$$\begin{aligned} \text{Domain} &= \{a, b, c, d\} \\ R &= \{(a, a), (a, b), (b, d), (d, c)\} \end{aligned}$$

Then a is related to b by R — written aRb or $R(a, b)$ or $(a, b) \in R$.

Binary Relations and Graphs

Domain D. Binary relation

$$R \subseteq D \times D$$

Can represent R as a directed graph, with nodes D and (a, b) is an edge iff $(a, b) \in R$.

Properties of Binary Relations

$$R \subseteq D \times D$$

Reflexive

$$(d, d) \in R$$

for all $d \in D$

Symmetric

$$(d, e) \in R \iff (e, d) \in R$$

Transitive

$$(a, b) \in R \wedge (b, c) \in R \Rightarrow (a, c) \in R$$

n-ary relations

An n-ary relation (or predicate) is true of some n-tuples but not others. Write $P^n(x_1, \ldots, x_n)$ to mean P^n is true of $(x_1, \ldots, x_n)$.
Equivalently, P^n is a *set* of n-tuples over some domain D.

$$P^n \subseteq D \times D \times \ldots \times D = D^n$$

Example B^3 is "in between", $D = \{0, 1, 2\}$.

$$B^3 = \{(1, 0, 2), (1, 2, 0)\}$$

Functions

n-**ary function** f over D takes n values $d_1, \ldots, d_n$ and returns a unique value $f^n(d_1, \ldots, d_n) \in D$.

$$f^n : D^n \to D$$

Example $D = \mathbb{Z}$ (integers). Binary function "addition" $+^2$.

$$+^2(3, 4) = 7$$

etc.
An n-ary function is a special kind of $(n+1)$-ary relation (always defined, single valued).

First Order Logic

(or predicate logic).
Main differences with propositional logic

- Includes *relations* between things (also functions).
- Includes *variables* to denote unknown individuals.
- Also includes *quantifiers* $\forall, \exists$.

Much more expressive.

First Order Logic, Syntax

There are many first-order languages. A language $\mathcal{L}$ depends on your choice of

Constant Symbols C,

Function Symbols F, and

Predicate Symbols P.

We write

$$\mathcal{L} = \mathcal{L}(C, F, P)$$

A function symbol $f \in F$ has an *arity*. If the arity is n we may write f^n, meaning that f expects exactly n arguments.
A predicate symbol $r \in P$ also has an arity $n \in \mathbb{N}$. Write r^n.

Syntax Definition

First we must define *terms*. These are names of individuals.

$tm ::= var | c \; : c \in C | f^n(tm, tm, \ldots, tm) \; : f^n \in F$

E.g.

$$0, \; x, \; \times^2(x, x), \; \times^2(x, \times^2(0, y))$$

are all terms, if 0 is a constant, x, y are variables and $\times^2$ is a binary function.

Atomic Formulas

Let r^n be an n-ary predicate symbol.
Let $t_1, \ldots, t_n$ be any terms.
Then

$$r^n(t_1, \ldots, t_n)$$

is an atomic formula. (Plays the role of propositional letter in PL.)

<u>**Example**</u>

$$<^2 (0,0), \;\; <^2 (x,y), \;\; <^2 (0, \times^2(x,x)),$$

$$<^2 (y, \times^2(x, \times^2(y,z)))$$

are all atomic formulas (here $<^2$ is a binary predicate symbol).

Syntax of Formulas

$$fm ::= Atom | \neg fm | (fm \circ fm) | \exists var fm | \forall var fm$$

where $\circ$ is either $\vee, \wedge$ or $\rightarrow$ (or $\leftrightarrow$ if you like), var is any variable.
$\exists$ is a new symbol, the "existential quantifier".
$\forall$ is a new symbol, the "universal quantifier".

Example

$$\begin{aligned}
C &= \{\text{Jane}, \text{Sarah}\} \\
F &= \{\text{Mother}^1\} \\
P &= \{\text{Younger}^2, \text{Birthday}^1, \text{Sibling}^2\}
\end{aligned}$$

Example formulas.

$$\text{Sibling}^2(\text{Mother}^1(\text{Jane}),\ \text{Sarah}),$$
$$\exists x\ \text{Birthday}^1(x),$$
$$\forall x \forall y \forall z((\text{Younger}^2(x,y) \wedge$$
$$\text{Younger}^2(y,z)) \rightarrow \text{Younger}^2(x,z))$$

Order of Quantifiers

- "Everyone votes for someone".

- $\exists y \forall x V(x, y)$?

- $\forall x \exists y V(x, y)$?

English $\rightarrow$ FOL

- “Everyone loves someone”
- “Nobody is loved by everyone”
- “There is no biggest number”
- “Zero is the smallest number”
- “For every number, there is a bigger one” .

Subformulas

A *subformula* ϕ of the formula ψ is a substring of ψ that forms a formula.
E.g. let

$$\psi = \forall x(<^2 (0, x) \rightarrow \exists y <^2 (x, y))$$

Subformulas of ϕ are

$$\exists y <^2 (x, y), \quad \ldots$$

How many subformulas altogether?

Scope of variables

If $\exists x \phi$ is a subformula of ψ then the *scope* of $\exists x$ is ϕ.
Be careful about multiply quantified variables, e.g.

$$\forall x \exists y (\text{Sibling}^2(x,y) \wedge \exists x \text{Child}^2(y,x))$$

What does it mean?

$$\forall x \forall y (<^2 (x,y) \vee <^2 (y,x)) \vee =^2 (x,y)$$

From now on, use infix notation for $<, =$.

$$\exists x \exists y (x < y \wedge \exists x y < x)$$

is equivalent to

$$\exists x \exists y (x < y \wedge \exists w y < w)$$

Exercise: Using only two variables, write a formula that expresses "there is an ordered sequence of at least n points".

Bound and Free Variables

- If x in scope of $\forall x$ or $\exists x$ then x *is bound*.

- If x is not in the scope of any quantifier then x *is free*.

- If all variables in formula ϕ are bound then ϕ *is a sentence*.

Semantics

$\mathcal{L} = \mathcal{L}(C, F, P)$.
An *$\mathcal{L}$-structure* $S = (D, I)$ gives meaning to symbols in C, F and D.

- D is a set — the domain of the structure S.

- I interprets the constant function and predicate symbols. So I has three parts $I = (I_c, I_f, I_p)$.

  - $I_c : C \to D$

  - For each $f^n \in F$, $I_f(f^n)$ is an n-ary function over D,

    $$I_f(f^n) : D^n \to D$$

  - I_p interprets predicate symbols as relations. For each $r^n \in P$, $I_p(r^n)$ is an n-ary relation over D,

    $$I_p(r^n) \subseteq D^n$$

Equality

$=$ is a very special binary predicate symbol. If '$=$' is a predicate of our language then we always insist that

$$I_p(=) = \{(d, d) : d \in D\}$$

Example

$\mathcal{L} = \mathcal{L}(C, F, P)$.
$C = \{r, s, j, \text{me}\}$
$F = \emptyset$
$P = \{\text{male}^1,\ \text{husb}^2,\ \text{single}^1,\ \text{sibling}^2\}$.

$\mathcal{L}$-structure $S = (D, I)$.
$D = \{\text{Robin}, \text{Soraya}, \text{John}, \text{Mary}, \text{Anne}\}$.
Constants

$$
\begin{aligned}
I_c(r) &= \text{Robin} \\
I_c(s) &= \text{Soraya} \\
I_c(j) &= \text{John} \\
I_c(\text{me}) &= \text{Robin}
\end{aligned}
$$

Predicates

$$
\begin{aligned}
I_p(\text{male}^1) &= \{\text{Robin}, \text{John}\} \\
I_p(\text{husb}^2) &= \{(\text{Robin}, \text{Soraya}), (\text{John}, \text{Mary})\} \\
I_p(\text{single}^1) &= \{\text{Anne}\} \\
I_p(\text{sibling}^2) &= \{(\text{Robin}, \text{Mary}), (\text{Mary}, \text{Robin}), \\
&\quad (\text{Soraya}, \text{Anne}), (\text{Anne}, \text{Soraya})\}
\end{aligned}
$$

Questions

In that structure, which of these hold?

$$
\begin{aligned}
S &\models \mathsf{husb}^2(r, s) \\
S &\models \neg\mathsf{single}^1(j) \\
S &\models (\mathsf{single}^1(r) \vee \mathsf{husb}^2(r, s)) \\
S &\models \exists x \mathsf{single}^1(x)
\end{aligned}
$$

Evaluating Formulas

What about

$$
\begin{aligned}
S &\models \mathsf{male}^1(x)?\\
S &\models \exists x \neg \mathsf{male}^1(x)?\\
S &\models \forall x(\mathsf{male}^1(x) \rightarrow (\mathsf{single}^1(x) \vee \exists y \mathsf{husb}^2(x,y)))?
\end{aligned}
$$

Variable Assignments

$\mathcal{L}$-structure $S = (D, I)$.
An *assignment* A maps variables to elements of the domain

$$A : vars \rightarrow D$$

Now we can interpret all terms and formulas of $\mathcal{L}$.

Terms

$$\begin{aligned} [c]^{S,A} &= I_c(c) \\ [v]^{S,A} &= A(v) \\ [f^n(t_1, \ldots, t_n)]^{S,A} &= I_f(f^n)([t_1]^{S,A}, \ldots, [t_n]^{S,A}) \end{aligned}$$

Atomic Formulas

$r^n \in P$.
$t_1, \ldots, t_n$ are terms.
Write

$$S, A \models r^n(t_1, \ldots, t_n)$$

if

$$([t_1]^{S,A}, \ldots, [t_n]^{S,A}) \in I_p(r^n)$$

(and say "$r^n(t_1, \ldots, t_n)$ is true in S under A").

Truth of Formulas

$$S, A \models r^n(t_1, \ldots, r_n)$$

Done. Let ϕ, ψ be any formulas. Suppose (inductively) we know how to work out whether or not $S, A \models \phi$ and $S, A \models \psi$, for any variable assignment A.

$$\begin{array}{rcl}
S, A \models (\phi \wedge \psi) & \iff & S, A \models \phi \text{ and } S, A \models \psi \\
S, A \models (\phi \rightarrow \psi) & \iff & \text{fill this in} \\
S, A \models \neg\phi & \iff & \text{and this} \\
S, A \models \exists x \phi & \iff & \text{any ideas?} \\
S, A \models \forall x \phi & \iff & ?
\end{array}$$

x-variants

Let A, B be two variable assignments and let x be any variable. We say that A is an x-*variant* of B if

$$A(y) = B(y)$$

for all variables y except perhaps x. We write

$$A \equiv_x B$$

in this case.

Semantics of Quantifiers

$$S, A \models \exists x \phi(x) \iff S, A^* \models \phi(x)$$

for *some* x-variant A^* of A

$$S, A \models \forall x \phi(x) \iff S, A^* \models \phi(x)$$

for *all* x-variants A^* of A

And that's it!

Summary of First Order Logic

Syntax	**Semantics**
$L(C, F, P)$	$S = (D, I),\ I = (I_c, I_f, I_p)$
Variables	$A : V \to D$
Term ::= $var \mid const \mid f^n(t_1, \ldots, t_n)$	$[t]^{S,A} \in D$
Atomic fmla ::= $r^n(t_1, \ldots, t_n)$	$S, A \models r^n(t_1, \ldots, t_n) \iff ([t_1]^{S,A}, \ldots, [t_n]^{S,A}) \in I_p(r^n)$
fmla ::= atom $\mid \neg$fmla$\mid$ (fmla $\vee$ fmla)$\mid$ $\exists$var fmla$\mid$ $\forall$var fmla	$S, A \models \exists x \phi \iff S, A^* \models \phi$ for some $A^* \equiv_x A$

Example

$\mathcal{L}(C, F, P)$.

$$\begin{array}{rcl} C & = & \{\text{zero}\} \\ F & = & \{\text{suc}^1, \text{add}^2, \text{times}^2\} \\ P & = & \{\text{even}^1, \text{less}^2, =\} \end{array}$$

$N = (D, I), \quad D = \{0, 1, 2, \ldots\}$

$$\begin{array}{rcl} I_c(\text{zero}) & = & 0 \\ I_f(\text{suc}^1) & : & n \mapsto n + 1 \\ I_f(\text{add}^2) & : & (m, n) \mapsto m + n \\ I_f(\text{times}^2) & : & (m, n) \mapsto m \times n \\ I_p(\text{even}^1) & = & \{0, 2, 4, \ldots\} \\ I_p(\text{less}^2) & = & \{(m, n) : m < n\} \\ I_p(=) & = & \{(0, 0), (1, 1), (2, 2), \ldots\} \end{array}$$

Variable assignments

$$A: \begin{array}{rcll} x & \mapsto & 5 & \\ y & \mapsto & 7 & \\ w & \mapsto & 3 & (\text{ all } w \neq x, y) \end{array}$$

$$B: \begin{array}{rcll} x & \mapsto & 5 & \\ y & \mapsto & 0 & \\ w & \mapsto & 3 & (\text{ all } w \neq x, y) \end{array}$$

$$\begin{array}{rcl} \phi & = & \exists x (x < y) \\ \psi & = & \forall x \exists y (x < y) \end{array}$$

Question: which of these are true?

- $N, A \models \phi$?
- $N, A \models \psi$?
- $N, B \models \phi$?
- $N, B \models \psi$?

Validity and Satisfiability

Unlike propositional logic, there are two types of validity and two types of satisfiability for predicate logic.

Validity

1. ϕ is *valid in the structure* S if $S, A \models \phi$ for *all* variable assignments A. Write

$$S \models \phi$$

2. ϕ is *valid* if it is valid in all possible structures S. Written

   $\models \phi$ ($\iff$ for every S we have $S \models \phi$)

Satisfiability

1. ϕ is *satisfiable in* S if

   $S, A \models \phi$ for *some* assignment A

2. ϕ is *satisfiable* if it is satisfiable in at least one structure S.

$$\begin{aligned}\phi &= \exists x(x < y)\\ \psi &= \forall x \exists y(x < y)\end{aligned}$$

Recall the structure N based on $\{0, 1, 2, \ldots\}$. Which of these is true?

$$N \models \phi? \qquad\qquad N \models \psi?$$

New structure $Z = (\mathbb{Z}, I^2)$ for same language $\mathcal{L}$. Domain is the set of all integers $\mathbb{Z}$. Interpretation is similar to the one we had,

$$I_p^2(\text{less}^2) \;=\; \{(m,n) : m < n \ (\in \mathbb{Z})\}$$

Which of these is true?

- $Z \models \phi$
- $Z \models \psi$
- $\models \phi$
- $\models \psi$

Evaluating Formulas

Recall N is $\mathcal{L}$-structure based on natural numbers $\mathbb{N}$. Which of these are true?

$$
\begin{aligned}
N &\models \mathsf{less}^2(\mathsf{zero}, \mathsf{suc}^1(\mathsf{zero}))\\
N &\models \mathsf{even}^1(\mathsf{zero})\\
N &\models \forall x \forall y\ (\mathsf{times}^2(x, y) = \mathsf{times}^2(y, x)))\\
N &\models \forall x \mathsf{even}^1(\mathsf{times}^2(x, \mathsf{suc}^1(\mathsf{suc}^1(\mathsf{zero}))))\\
N &\models \neg\exists x(x = \mathsf{suc}^1(x))\\
N &\models \forall x \exists y(y = \mathsf{suc}^1(x))\\
N &\models \forall x \exists y(x = \mathsf{suc}^1(y))\\
N &\models \exists y \forall x(y = \mathsf{suc}^1(x))
\end{aligned}
$$

Theorem

Let ϕ be an $\mathcal{L}$-formula and let S be an $\mathcal{L}$-structure.

1. ϕ is not valid iff $\neg\phi$ is satisfiable.

2. ϕ is not valid in S iff $\neg\phi$ is satisfiable in S.

Note

If ϕ is a *sentence* (no free variables) then

$$\phi \text{ is satisfiable in } S \iff \phi \text{ is valid in } S$$

Valid Formulas

Which of these are valid?

- $\forall x \forall y(x < y \vee x = y \vee y < x)$

- $\forall x \exists y(x = y)$

- $\neg \forall x p^1(x) \leftrightarrow \exists x \neg p^1(x)$

- $\neg \exists x p^1(x) \leftrightarrow \forall x \neg p^1(x)$

- Any propositional tautology, e. g. $\phi \rightarrow \phi$.

Equivalent formulas

Two formulas ϕ, ψ are *equivalent* if for all structures S and all assignments A we have

$$S, A \models \phi \iff S, A \models \psi$$

Write

$$\phi \equiv \psi$$

Substitution

Write $\phi(x, y)$ if free variables of ϕ are $\{x, y\}$.
Write $\phi(t/x)$ for formula obtained from $\phi(x)$ by replacing all *free* occurrences of x by term t.

Example

Let

$$\begin{aligned}\phi(x) &= \exists y(y < x \wedge \forall x(x < y \rightarrow x = 0)) \\ t &= z + 1\end{aligned}$$

Then

$$\phi(t/x) = \exists y(y < z + 1 \wedge \forall x(x < y \rightarrow x = 0))$$

Clash of variables

Let $\phi(x)$ be the formula

$$\exists y(y > x)$$

and substitute the term $y + 1$ for x.

$$\phi(y + 1/x) = \exists y(y > y + 1)$$

Bad!

Substitutable Terms

Let $\phi(x)$ be a formula and let t be any term. We say that t *is substitutable for* x *in* ϕ if for each variable y occurring in t, no free occurrence of x occurs in the scope of $\forall y$ or $\exists y$ in ϕ.

Useful equivalences

$$
\begin{aligned}
\forall x \neg \phi &\equiv \neg \exists x \phi \\
\exists x \neg \phi &\equiv \neg \forall x \phi \\
\forall x(\phi \wedge \psi) &\equiv \forall x \phi \wedge \forall x \psi \\
\exists x(\phi \vee \psi) &\equiv \exists x \phi \vee \exists x \psi \\
\forall x \phi(x) &\equiv \forall y \phi(y/x) \\
& \text{(replace all free } x\text{'s in } \phi \text{ by } y \\
& \text{provided not in scope of} \\
& \; y\text{-quantifier in } \phi) \\
\forall x \phi \vee \psi &\equiv \forall x(\phi \vee \psi) \\
& \text{(if no free } x\text{'s in } \psi) \\
\exists x \phi \wedge \psi &\equiv \exists x(\phi \wedge \psi) \\
& \text{(if no free } x\text{'s in } \psi)
\end{aligned}
$$

But

$$
\begin{aligned}
\forall x(\phi \vee \psi) &\not\equiv \forall x\phi \vee \forall x\psi \\
\exists x(\phi \wedge \psi) &\not\equiv \exists x\phi \wedge \exists x\psi
\end{aligned}
$$

Why not?

Only free variables matter

Structure $S = (D, I)$.
Assignments $A, B : var \to D$.
Formulas ϕ.

$$FVAR(\phi) = \{\text{free vars. of } \phi\}$$

Theorem 1 *If for each $v \in FVAR(\phi)$ we have*

$$A(v) = B(v)$$

then

$$S, A \models \phi \iff S, B \models \phi \qquad (1)$$

Corollary 2 *If ϕ is a sentence ($FVAR(\phi) = \emptyset$) then for* any *assignments A, B (1) holds.*

Proof of Theorem

Terms Let t be a term. If for each $v \in t$ we have $A(v) = B(v)$ then

$$[t]^{S,A} = [t]^{S,B}$$

Proved by structured term induction.

Base Cases:

$t = c$ ($\in C$).

$$[c]^{S,A} = I_c(c) = [c]^{S,B}$$

$t = v$.

$$[v]^{S,A} = A(v) = B(v) = [v]^{S,B}$$

Inductive Hypotheses.

$$[t_1]^{S,A} = [t_1]^{S,B},\ [t_2]^{S,A} = [t_2]^{S,B},\ \ldots, [t_n]^{S,A} = [t_n]^{S,B}$$

Induction step.

$$\begin{aligned} & [f^n(t_1, \ldots, t_n)]^{S,A} \\ = \ & I_f(f^n)([t_1]^{S,A}, \ldots, [t_n]^{S,A}) \\ = \ & I_f(f^n)([t_1]^{S,B}, \ldots, [t_n]^{S,B}) \ \text{(by I.H)} \\ = \ & [f^n(t_1, \ldots, t_n)]^{S,B} \end{aligned}$$

Formulas By structured formula induction.

Base Case: ϕ is atomic $r^n(t_1, \ldots, t_n)$.

$$
\begin{aligned}
& S, A \models r^n(t_1, \ldots, t_n) \\
\iff & \quad ([t_1]^{S,A}, \ldots, [t_n]^{S,A}) \in I_p(r^n) \\
\iff & \quad ([t_1]^{S,B}, \ldots, [t_n]^{S,B}) \in I_p(r^n) \text{ (by prev. part)} \\
\iff & \quad S, B \models r^n(t_1, \ldots, t_n)
\end{aligned}
$$

Inductive Hypotheses (ϕ, ψ):

Assume that if A and B agree on $FVAR(\phi)$ then

$$S, A \models \phi \iff S, B \models \phi$$

and if A and B agree on $FVAR(\psi)$ then

$$S, A \models \psi \iff S, B \models \psi$$

Inductive Steps:

$\neg\phi$ Suppose A, B agree on $FVAR(\neg\phi) = FVAR(\phi)$. Then

$$
\begin{aligned}
S, A \models \neg\phi \quad & \iff \quad S, A \not\models \phi \\
& \iff \quad S, B \not\models \phi \text{ (by I.H.)} \\
& \iff \quad S, B \models \neg\phi
\end{aligned}
$$

$(\phi \wedge \psi)$ Suppose A and B agree on $FVAR(\phi\wedge\psi)$ $(= FVAR(\phi)\cup FVAR(\psi))$. Then

$$\begin{array}{llll} S, A \models (\phi \wedge \psi) & \Longleftrightarrow & & S, A \models \phi \text{ and } S, A \models \psi \\ & \Longleftrightarrow & \text{I.H.} & S, B \models \phi \text{ and } S, B \models \psi \\ & \Longleftrightarrow & & S, B \models (\phi \wedge \psi) \end{array}$$

$\exists x\phi$ Suppose A, B agree on $FVAR(\exists x\phi)$.

Case 1: $x \notin FVAR(\phi)$.

Then $FVAR(\phi) = FVAR(\exists x\phi)$.

$$\begin{array}{llll} S, A \models \exists x\phi & \Longleftrightarrow & S, A^* \models \phi & \text{for some } A^* \equiv_x A \\ & \Longleftrightarrow & S, A \models \phi & \text{by I.H. and case} \\ & \Longleftrightarrow & S, B \models \phi & \text{by I.H.} \\ & \Rightarrow & S, B \models \exists x\phi & \end{array}$$

Case 2: $x \in FVAR(\phi)$.

$$\begin{array}{llll} S, A \models \exists x\phi(x) & \Longleftrightarrow & S, A^* \models \phi(x) & \text{some } A^* \equiv \\ & \Rightarrow & S, B^* \models \phi(x) & \text{by I.H., wh} \\ & \Rightarrow & S, B \models \exists x\phi(x) & \end{array}$$

where

$$B^*(v) = \begin{cases} B(v) & \text{if } v \neq x \\ A^*(x) & \text{otherwise} \end{cases}$$

Note: $B^* \equiv_x B$ and B^*, A^* agree on $FVAR(\phi)$.

Deduction: Hilbert Systems

Propositional Axioms:

1. $(A \to (B \to A))$

2. $((A \to (B \to C)) \to ((A \to B) \to (A \to C)))$

3. $((A \to B) \leftrightarrow (\neg B \to \neg A))$

4. $(A \leftrightarrow \neg\neg A)$.

Quantifier Axioms:

5. $(\forall x \neg A \leftrightarrow \neg \exists x A)$

6. $(\forall x A(x) \to A(t/x))$ if t is *substitutable* for x in A.

7. $(\forall x(A \to B) \to (\forall x A \to \forall x B))$.

Equality Axioms:

8. $(x = x)$

9. $((x = y) \to (t(x) = t(y/x)))$

10. $((x = y) \to (A(x) \to A(y/x)))$ if y is substitutable for x in A.

An *instance* of any of the axioms above is obtained by replacing A, B, C etc. by arbitrary formulas.

Inference Rules

Modus Ponens

$$\frac{A, (A \to B)}{B}$$

Universal Generalisation

$$\frac{A(x)}{\forall x A(x)}$$

(N.B. $A(x) \to \forall x A(x)$ is not an axiom, as it is not valid. Universal generalisation says that if $A(x)$ is valid then $\forall x A(x)$ is also valid. This rule is sound.)

Proofs

A proof of ϕ is a finite sequence

$$\phi_0, \phi_1, \phi_2, \ldots, \phi_n = \phi$$

such that, for each $i \leq n$, either

- ϕ_i is an instance of one of the axioms or
- ϕ_i is obtained from ϕ_j (and maybe ϕ_k) where $j, k < i$, by an inference rule.

Write

$$\vdash \phi$$

in this case.

Proving from hypotheses

So far, this is all to do with validity over arbitrary models. If you want to find validities in a particular model, or a particularly type of model, then you can add hypotheses.
These hypotheses are formulas which are valid in the type of formula you want, and they define it.

E.g. Linearly Ordered Models

Hypotheses:

$$\forall x \forall y (x < y \vee y < x \vee x = y)$$
$$\forall x \neg (x < x)$$
$$\forall x \forall y \forall z ((x < y \wedge y < z) \rightarrow x < z)$$

Proofs with hypotheses

Let Γ be a set of hypotheses. Write

$$\Gamma \vdash \phi$$

if there is a sequence

$$\phi_0, \phi_1, \ldots, \phi_n = \phi$$

such that for each $i \leq n$ either

- ϕ_i is an axiom,
- ϕ_i is obtained from ϕ_j (ϕ_k) (some $j, k < i$) by an inference rule, or
- $\phi_i \in \Gamma$.

Example Proof using Hypotheses

$$\text{Linear Order} \vdash \forall x \forall y \neg (x < y \land y < x)$$

Proof

- $\forall x \forall y \forall z ((x < y \land y < z) \rightarrow x < z)$ (Hypothesis)

- $\forall x \forall y \forall z ((x < y \land y < z) \rightarrow (x < z)) \rightarrow$
  $((x < y \land y < x) \rightarrow x < x)$ (Ax. 6)

- $((x < y \land y < x) \rightarrow x < x)$ (Modus Ponens)

- $\forall x \neg (x < x)$ (Hypothesis)

- $((x < y \land y < x) \rightarrow x < x) \rightarrow (\neg (x < x) \rightarrow$
  $\neg (x < y \land y < x))$ (Ax. 3)

- $(\neg (x < x) \rightarrow \neg (x < y \land y < x))$ (Modus Ponens)

- $\neg (x < x)$ (Ax. 6)

- $\neg (x < y \land y < x)$ (Modus Ponens)

- $\forall x \forall y \neg (x < y \land y < x)$ (Universal Generalisation)

Entailment

Let Γ be a set of formulas and let S be an $\mathcal{L}$-structure. Write

$$S \models \Gamma$$

if $S \models \phi$ for each $\phi \in \Gamma$ (say "S is a model of Γ) and

$$\Gamma \models \phi$$

if every model of Γ is a model of ϕ (i.e. $S \models \Gamma \Rightarrow S \models \phi$).

Strong Completeness

$$\Gamma \vdash \phi \iff \Gamma \models \phi$$

Corollary 3 *There is an enumeration of the valid formulas.*

First Order Tableaus

literals	$r^n(t_1, \ldots, t_n), \neg r^n(t_1, \ldots, t_n)$
α-fmlas	$(\phi \wedge \psi), \neg(\phi \vee \psi), \neg(\phi \rightarrow \psi), \neg\neg\phi$
β-fmla	$(\phi \vee \psi), \neg(\phi \wedge \psi), (\phi \rightarrow \psi)$
γ-fmla	$\forall x \phi, \neg\exists x \phi$
δ-fmla	$\exists x \phi, \neg\forall x \phi$

Is every formula of first-order logic a literal, an α, β, γ or δ-formula?

Expansion Rules

α **formulas** Add both formulas in one branch at each leaf below current node. Tick current node.

<table>
<tr><td>$(\phi \wedge \psi)$
$\mid$
ϕ
ψ</td><td>$\neg(\phi \vee \psi)$
$\mid$
$\neg\phi$
$\neg\psi$</td></tr>
<tr><td>$\neg(\phi \rightarrow \psi)$
$\mid$
ϕ
$\neg\psi$</td><td>$\neg\neg\phi$
$\mid$
ϕ</td></tr>
</table>

β **formulas** Make two separate branches (one with each formula) at each leaf below current node. Tick current node.

<table>
<tr><td>$\begin{array}{ccc} & (\phi \vee \psi) & \\ \swarrow & & \searrow \\ \phi & & \psi \end{array}$</td><td>$\begin{array}{ccc} & (\phi \rightarrow \psi) & \\ \swarrow & & \searrow \\ \neg\phi & & \psi \end{array}$</td><td>$\begin{array}{ccc} & \neg(\phi \wedge \psi) & \\ \swarrow & & \searrow \\ \neg\phi & & \neg\psi \end{array}$</td></tr>
</table>

δ **formulas** Choose new constant p (not included in tableau so far). Add formula at each leaf below current node. Tick current node.

$\exists x \phi(x)$	$\neg \forall x \phi(x)$
$\vert$	$\vert$
$\phi(p/x)$	$\neg \phi(p/x)$

γ **formulas** Pick any closed term t. Add formula at each leaf below current node. Do *not* tick node.

$\forall x \phi(x)$	$\neg \exists x \phi(x)$
$\vert$	$\vert$
$\phi(t/x)$	$\neg \phi(t/x)$

Tableau Example

Is $(\forall x \neg p(x) \rightarrow \neg \exists y\, p(y))$ valid? Make tableau for negated formula.

$$
\begin{array}{c}
\neg(\forall x \neg p(x) \rightarrow \neg \exists y\, p(y)) \quad (1) \\
|\alpha(1) \\
\forall x\, \neg p(x) \quad (2) \\
\neg\neg \exists y\, p(y) \quad (3) \\
|\alpha(3) \\
\exists y\, p(y) \quad (4) \\
|\delta(4, c) \\
p(c) \quad (5) \\
|\gamma(2, c) \\
\neg p(c) \\
\hline
\end{array}
$$

Closed tableau.

So $(\forall x \neg p(x) \rightarrow \neg \exists y p(y))$ is valid.

Second tableau example

Is $\forall x \neg q(x) \vee \exists x \forall y \neg(x < y)$ valid? Make tableau for negated formula.

$\neg(\forall x \neg q(x) \vee \exists x \forall y \neg(x < y))$ (1)
$|\alpha(1)$
$\neg\forall x \neg q(x)$ (2)
$\neg\exists x \forall y \neg(x < y)$ (3)
$|\delta(2, c)$
$\neg\neg q(c)$ (4)
$|\alpha(4)$
$q(c)$
$|\gamma(3, c)$
$\neg\forall y \neg(c < y)$ (5)
$|\delta(5, d)$
$\neg\neg(c < d)$ (6)
$|\alpha(6)$
$c < d$
$|\gamma(3, d)$
$\neg\forall y \neg(d < y)$ (7)
$|\delta(7, e)$
$\neg\neg(d < e)$
$|\alpha$
$d < e$
$|$

Herbrand Structures

A *closed term* t is built up from constants and function symbols only — no variables.
A *Herbrand structure* $H = (D, I)$ has

Domain

$$D = \{\text{closed terms}\}$$

Interpretation $I = (I_c, I_f, I_p)$.

$$\begin{aligned} I_c(c) &= c \\ I_f(f^n) : (d_1, \ldots, d_n) &\mapsto f^n(d_1, \ldots, d_n) \end{aligned}$$

I_p can be chosen freely.

It follows, for any closed term t, that

$$[t]^{H,A} = t$$

Herbrand Theorem

Let L be a language with ∞ many constant symbols (and no equality predicate in this version of the theorem).
If ϕ is satisfiable (i.e. $S, A \models \phi$, some S some A) then ϕ is satisfiable in a Herbrand model H, i.e. $H, A \models \phi$ (some A).

Soundness Theorem

ϕ is satisfiable $\Rightarrow$ tableau for ϕ stays open (forever).

Proof. Let $S, A \models \phi$.
Prove, by induction over number of expansions in construction of T so far, that there is a structure S (and an assignment A) such that T has a branch and every formula on that branch is *true* in S, A.
Base case. After 0 expansions there is only one node (ϕ) and $S, A \models \phi$.
I.H. After n expansions T contains a branch, say Θ, and there is a structure S and an assignment A such that if $\lambda \in \Theta$ then $S, A \models \lambda$.
Induction Step. Expand one node x of T to get T'. Must show that T' still has a suitable branch.

- If $x \notin \Theta$ then Θ is still a branch of T', so result holds by I.H. (use the same structure S, same assignment A).

- If $x \in \Theta$ and x is an α formula, then expansion formulas α_1, α_2 are added at every leaf. Since $S, A \models \alpha$ we have $S, A \models \alpha_1$ and $S, A \models \alpha_2$. So α_1, α_2 get added as extra nodes at the end of branch Θ but both these formulas are true in S, A, so the extended branch still has the property (use the same structure S same assignment A).

- $x \in \Theta$ and x is a β formula, then expansion formulas β_1, β_2 are added as two separate successors to each leaf of T. So there are two extensions to Θ — one has β_1 at the end and the other has β_2 at the end. Since $S, A \models \beta$ either $S, A \models \beta_1$ or $S, A \models \beta_2$. Therefore one of the two extensions of Θ still has the property (same S, same A).

- $x \in \Theta$ and x is a δ, e.g. $\exists y \phi(y)$, then $\phi(c/y)$ gets added to the end of every leaf, where c is a new constant. Since $S, A \models \exists y \phi(y)$ there is $A^* \equiv_y A$ and $S, A^* \models \phi(y)$. Define structure S' — same as S except $I_c(c) = A^*(y)$ in S'. Then $S', A \models \phi(c/y)$. Any other formula ρ on the branch does not involve the constant c (that's why you have to choose a new constant) so $S', A \models \rho$. So property is still true (in S' under A). Case $\neg\forall y A(y)$ is similar.

- $x \in \Theta$ and x is a γ formula, e.g. $\forall y \phi(y)$. Expand with closed term t. Then $\phi(t/y)$ is added to end of Θ. Since $S, A \models \forall y \phi(y)$ we have $S, A \models A(t/y)$, so property still holds (use same S, same A).

By induction, for any number of expansions, T contains a branch Θ and there is a structure S and an assignment A such that $x \in \Theta \Rightarrow S, A \models x$. It follows that Θ is an open branch. $\blacksquare$

Fairness

Suppose you have several (countably many) processes $P_1, P_2, \ldots, P_k, \ldots$ and each of them is waiting for some input. It might be that when you give P_i some input, it creates a new process P_{k+1}, so the list can grow, but it will always be countable. In what order should you supply inputs to the various processes?

You could simply supply input to P_1 again and again, but that would be *unfair* to all the other processes. In a *fair* schedule, if any process P_i is waiting for input at time t then eventually (at some time $t' > t$) P_i will get some input.

If processes are always waiting for input, then each process will get input infinitely often.

Since the total number of requests for input is countable, it is possible to find a fair schedule.

Completeness Theorem

Let ϕ be a sentence.
If tableau for ϕ is expanded with a *fair* system and stays open forever then ϕ is satisfiable.

Let T be the 'limit' tableau (open). Let Θ be open branch. Define structure $H = (D, I)$.

$$
\begin{aligned}
D &= \{\text{closed terms}\} \\
I_c(c) &= c \\
I_f(f^n) &: (t_1, \ldots, t_n) \mapsto f^n(t_1, \ldots, t_n) \\
I_p(r^n) &= \{(t_1, \ldots, t_n) : r^n(t_1, \ldots, t_n) \in \Theta\}
\end{aligned}
$$

Claim

Any sentence λ

$$\lambda \in \Theta \Rightarrow H \models \lambda$$

and

$$\neg\lambda \in \Theta \Rightarrow H \models \neg\lambda$$

Proof of claim By structured induction over λ.

Base case: λ is atomic $r^n(t_1, \ldots, t_n)$. Since λ is a sentence, each t_i is a closed term. So if $r^n(t_1, \ldots, t_n) \in \Theta$ then by definition of H, $([t_1]^H, \ldots, [t_n]^H) = (t_1, \ldots, t_n) \in I_p(r^n)$, so $H \models r^n(t_1, \ldots, t_n)$.

$\neg\lambda$ case is similar.

I.H. λ, μ arbitrary formulas. t an arbitrary closed term.

$$\begin{aligned}
\lambda(t) \in \Theta &\Rightarrow H \models \lambda(t) \\
\neg\lambda(t) \in \Theta &\Rightarrow H \models \neg\lambda(t) \\
\mu(t) \in \Theta &\Rightarrow H \models \mu(t) \\
\neg\mu(t) \in \Theta &\Rightarrow H \models \neg\mu(t)
\end{aligned}$$

Ind. Steps ($\neg, \wedge, \exists$)

Propositional cases $((\lambda \wedge \mu), \neg(\lambda \wedge \mu), \neg\lambda, \neg\neg\lambda)$ — covered by COMP1002.

$\exists y \lambda(y) \in \Theta$. Then since expansion sequence is fair, eventually this δ sentence must have been expanded and $\lambda(c/y)$ must have be included in Θ, for some constant c. By I.H. since $\lambda(c/y) \in \Theta$ and c is a closed term we have $H \models \lambda(c/y)$. Therefore $H \models \exists y \lambda(y)$, as required.

$\neg\exists y \lambda(y) \in \Theta$. Then, since expansion sequence is fair, for every closed term t we have $\neg\lambda(t/y) \in \Theta$. By I.H. for every closed term t we have $H \models \neg\lambda(t/y)$. Hence $H \models \forall y \neg\lambda(y)$, i.e. $H \models \neg\exists y \lambda(y)$.

Completeness

So, by induction, if $\lambda \in \Theta$ then $H \models \lambda$. It follows that $H \models \phi$, where ϕ is the formula at the root. This proves that ϕ is satisfiable, as required.

Tableau Summary

- Tableau method is sound and complete for first order logic (this is Gödel's completeness theorem).

- If ϕ is not satisfiable its tableau will close finitely, if fair sequence is used (completeness).

- If ϕ is satisfiable its tableau will never close (soundness).

- But a tableau construction may never terminate.

Recursive Languages

A language L is just a set of strings over some finite alphabet Σ.
L is *recursive* if there is a computer program that takes an arbitrary string $s \in \Sigma^*$ as an input and outputs

$$\begin{cases} \text{"yes"} & \text{if } s \in L \\ \text{"no"} & \text{otherwise} \end{cases}$$

The program must be guaranteed to terminate, for any $s \in \Sigma^*$.

The set of all formulas of first order logic is a recursive set (a parsing program decides if a string is a well formed formula).

The valid statements of first order logic form a language, but this language is not recursive (not decidable).

Recursively Enumerable Languages

A language L is *recursively enumerable* (r.e.) if there is a computer program that outputs strings from L, only strings from L, and will eventually output any given string from L.

The valid statements of first-order logic form a recursively enumerable language.

First Order Logic is r.e.

Let $\phi_0, \phi_1, \ldots$ be an enumeration of all formulas.

For $(i = 0, i{+}{+}, \text{forever})$
    { Start new tableau T_i with $\neg\phi_i$ at root;
    For each $j < i$
        { expand T_j once, using a fair schedule;
        If T_j becomes closed, output "ϕ_j is valid";
        }
    }

Note: for any formula ϕ_k, if ϕ_k is valid then eventually T_k will close and the program will output ϕ_k (completeness, though you do not know how long this will take).
If ϕ_k is not valid then T_k will never close (soundness).
So the program only outputs valid formulas, and any given valid formula will eventually get output.

Recursive and r.e. languages

The set of formulas of first-order logic is a recursive set.
The set of valid formulas of FOL is not recursive, but it is r.e.
The set of true statements of arithmetic is not even r.e.
This last statement is Gödel's incompleteness theorem.

Proving from Assumptions

Suppose you want to prove that ϕ is valid in a particular model, or type of model (e. g. linear order).

Write down assumptions Σ that define this type of model. E.g.

$$\Sigma = \left\{ \begin{array}{l} \forall x \forall y (x = y \lor x < y \lor y < x), \\ \forall x \forall y \forall z ((x < y \land y < z) \to x < z), \\ \forall x \neg (x < x) \end{array} \right\}$$

New rule: you can add any assumption in Σ at leaf of tableau at any time. A *proof* of ϕ using Σ is a closed tableau for $\neg\phi$, but you can use assumptions to help you close the tableau. Write

$$\Sigma \vdash \phi$$

in this case.

Insert one slide here

Entailment and Strong Completeness

Recall

$$\Sigma \models \phi \text{ means } S \models \Sigma \Rightarrow S \models \phi$$

i.e. every model of Σ is also a model of ϕ.

Strong Completeness

$$\Sigma \models \phi \iff \Sigma \vdash \phi$$

Compactness

If

$$\Sigma \vdash \phi$$

then

$$\Sigma_0 \vdash \phi$$

for some finite subset Σ_0 of Σ.

Inconsistency, Compactness, Completeness.

Σ is *inconsistent* if

$$\Sigma \vdash (p \wedge \neg p)$$

If Σ is inconsistent then, by compactness, Σ_0 is inconsistent, for some finite subset Σ_0 of Σ.
By strong completeness theorem, every consistent set has a model.
Hence, compactness says that if every finite subset of Σ has a model then there is a model for the whole of Σ.

Compactness theorem and non-standard analysis

Let

$$\Sigma = \{\text{all valid statements about } \mathbb{N}\}$$

in a language with constants $0, 1, 2, \ldots$ functions $+, \times$ and predicate $=$.
E.g. $2 + 2 = 4 \in \Sigma$.
Also $\forall x \forall y (x \times y = y \times x) \quad \in \Sigma$.
Let ω be another constant symbol.
Every finite subset of

$$\Sigma^{+} = \Sigma \cup \{\omega > 0, \omega > 1, \omega > 2, \ldots, \omega > n, \ldots\}$$

has a model (what model?).
Therefore Σ^{+} has a model.

Non-standard real analysis

Let L be similar but with a constant for every real number. Let

$$\Sigma = \{\text{all valid statements about } \mathbb{R}\}$$

and

$$\Sigma^+ = \Sigma \cup \{\alpha > r : r \in \mathbb{R}\}$$

Every finite subset of Σ^+ has a model (just interpret α as a sufficiently big real number), therefore Σ^+ has a model M. Then $[\alpha]^M$ is an "infinitely big" real number and $[\frac{1}{\alpha}]^M$ is and "infinitesimally small" positive real number.

Can do calculus perfectly rigorously in this way. Can show that

$$\forall x((|x| < r) \to (x = St(x) + Inf(x)))$$

where r is a constant for any positive real, $St(x)$ is a "standard real" and $Inf(x)$ is an "infinitesimal real". Then let

$$f'(x) = St\left(\frac{f(x+\delta x) - f(x)}{\delta x}\right)$$

where x is any standard real and δx is any infinitesimal, provided this does not depend on the choice of δx.

Gödel's Incompleteness Theorem

Consider true statements of arithmetic.

$$\begin{array}{rcl} C & = & \{0, 1, 2, \ldots\} \\ F & = & \{+, -, \times\} \\ P & = & \{=, <\} \end{array}$$

Theorem 4 (Gödel, 1931) *If S is any r.e. set of L-sentences then either*

- *There is a statement ϕ which is true in arithmetic ($\mathbb{N}$) but $\phi \notin S$ (incompleteness), or*

- *There is a statement ϕ which is false in arithmetic and $\phi \in S$ (inconsistency).*

We will prove a slightly weaker result: if Γ is any finite set of axioms in this language then either $\Gamma \vdash \perp$ (using a tableau), i.e. Γ is inconsistent, or there is a formula ϕ which happens to be true in $\mathbb{N}$ and yet $\Gamma \not\vdash \phi$ (incompleteness).

Proof Sketch

Idea: every formula is a string of characters. So can code a formula as a number.
E.g ASCII code. So "$p(x)$" has code "160 050 170 051".
Also, given a code like this, we can recover syntactic information. E.g. let n be a code. The statement "the last character of the formula with code n is a right bracket" can be expressed by the formula

$$\exists z \;\; n = 1000 \times z + 51$$

This code number is called the "Gödel number" of the formula.
Can convert formula to code and code back to formula.
Can write a first order formula $\phi(n)$ that is true iff n is the Gödel number of a formula.
Similarly, every tableau can be represented as a string, so every tableau has a Gödel number.
Let G, F, T be coding and decoding functions,

so if ϕ is a formula and T is a tableau then $G(\phi)$ and $G(T)$ are their codes numbers. If $n \in \mathbb{N}$ then $F(n)$ is the formula ϕ (if any) such that $G(\phi) = n$ and $T(n)$ is the tableau T such that $G(T) = n$.

Tableau T proves formula ϕ if T is closed and $\neg\phi$ is at root.

Can write formulas

$$\begin{aligned}\rho(n) &= T(n) \text{ is closed tableau} \\ \phi(n,m) &= T(n) \text{ is a proof of } F(m) \\ \lambda(n) &= F(n) \text{ is a formula with one free variable, } x\end{aligned}$$

Let

$$A_0(x), A_1(x), A_2(x), \ldots$$

be an enumeration of all the formulas with one free variable x. If $F(m)$ has one free variable then $F(m) = A_k(x)$ (some k). Can write

$$\phi(n,k,q) = (T(n) \text{ is a proof of } A_k(q))$$

Consider

$$\neg\exists n \phi(n,x,x)$$

This is a formula with one free variable. So there is some n_0 such that

$$A_{n_0}(x) = \neg\exists n \phi(n,x,x)$$

We have $\mathbb{N} \models A_{n_0}(m)$ iff "there is no proof of $A_m(m)$.

Finally, consider

$$A_{n_0}(n_0)$$

We have

$\mathbb{N} \models A_{n_0}(n_0) \iff$ no tableau proves $A_{n_0}(n_0)$!

If $\mathbb{N} \models A_{n_0}(n_0)$ then no tableau proves it (incompleteness).
If $\mathbb{N} \not\models A_{n_0}(n_0)$ then some tableau proves it (inconsistency).

Other Logics

1. Second order, third order, ... higher order logic (no completeness, so not even r.e.).

2. Fixpoint logic.

3. Modal Logic.

4. Temporal Logic.

5. $$\left.\begin{array}{l}\text{Dynamic Logic}\\ \text{Intuitionistic Logic}\\ \text{Linear Logic}\end{array}\right\}\text{funny def. of } \rightarrow$$

6. Algebraic Logic.

7. Horn Clause Logic (Prolog) — satisfiability for propositional case is in **P**.

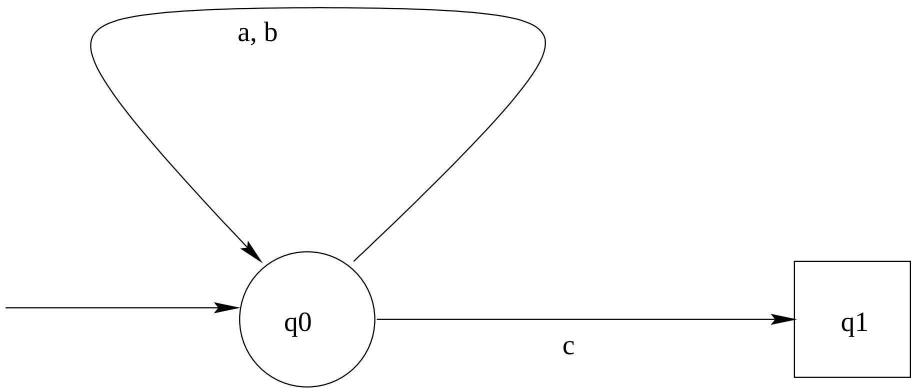
Finite State Machines

$$FSM = (Q, \Sigma, \delta, q_0, F)$$

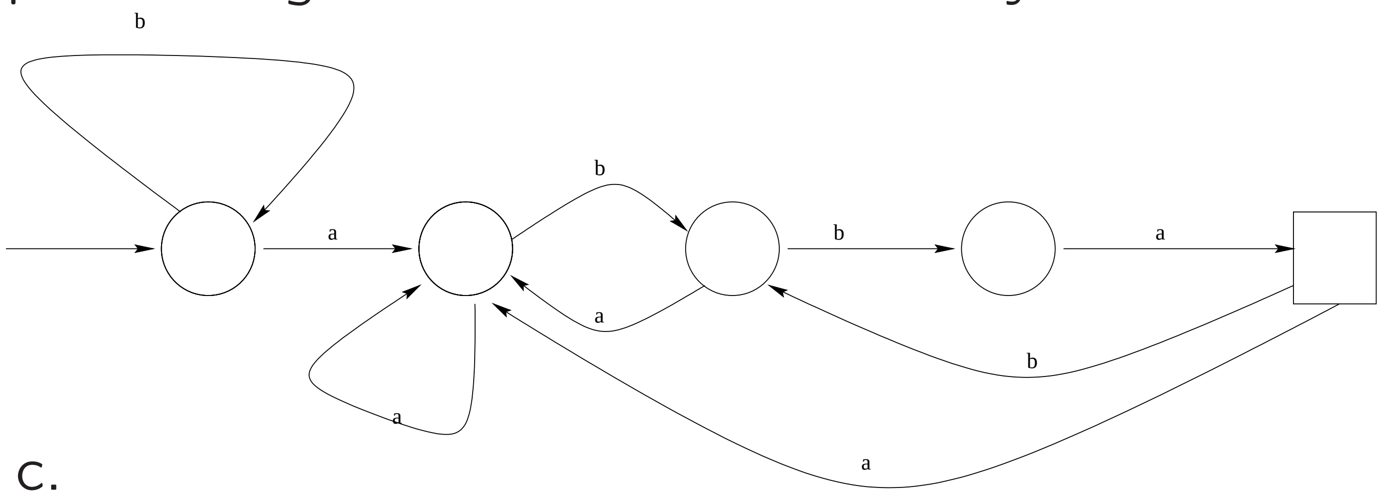
where Q is a finite set of states
Σ is a finite alphabet
$q_0 \in Q$ is the start state
$F \subseteq Q$ is the set of final or halting states, and

$$\delta : Q \times \Sigma \to Q$$

FSM example 1



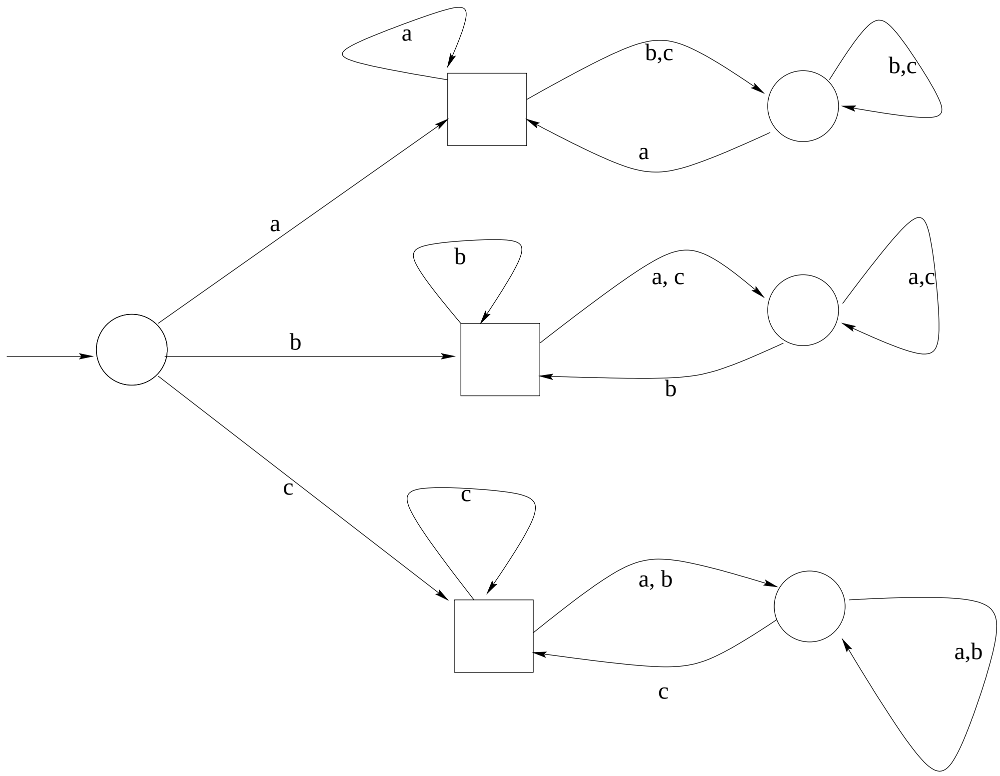
accepts a string of a’s and b’s followed by a



final c.

accepts strings ending with “abba”.

Example 3 — "finite memory"



The algorithm

Algorithm 1

Input string s

$q = q_0$

$p = 0$

while $p < n$ and $\delta(q, s_p)$ is defined **do**

  $q = \delta(q, s_p)$

  $p{+}{+}$

end while

if $\delta(q, s_p)$ is not defined and $p < n$ **then**

  Reject

end if

if $q \notin F$ **then**

  Reject

end if

if $q \in F$ and $p = n$ **then**

  Accept

end if

The language accepted by a FSM

Definition 5 *Let* $M = (Q, \Sigma, q_0, \delta, F)$ *be a FSM.*

$$L(M) = \{\textit{strings } s \in \Sigma^* : M \textit{ accepts } s\}$$

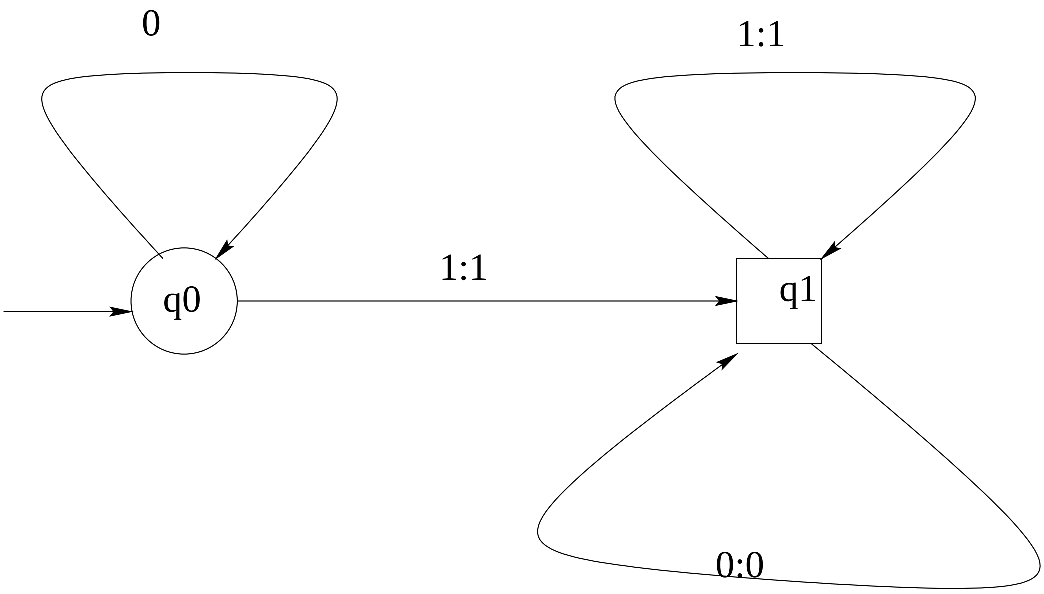
Output "Transducers"

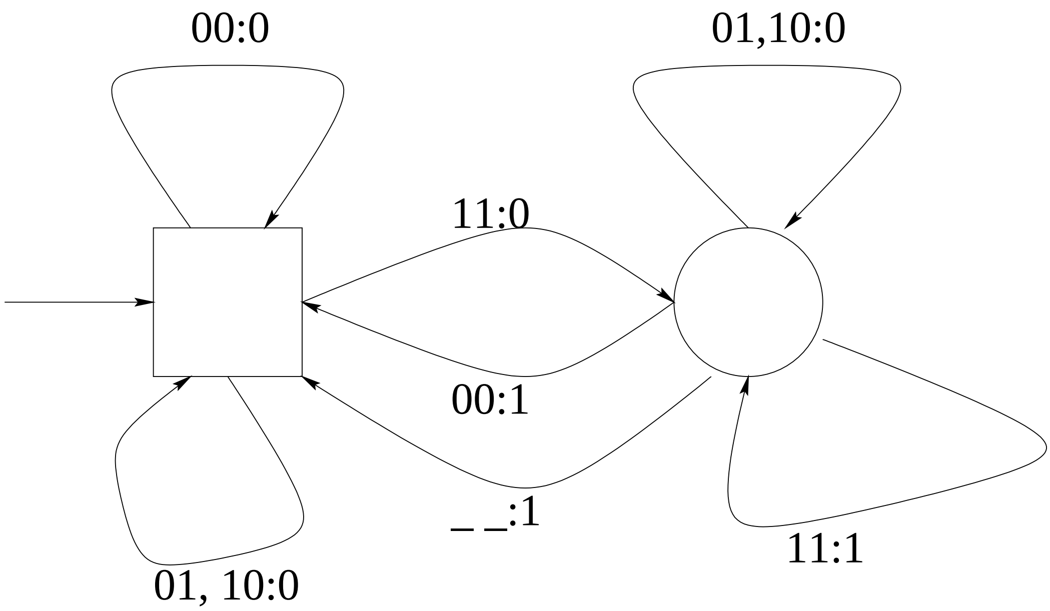
$$(Q, \Sigma, q_0, \delta, \Omega, \omega, F)$$

- Q is finite set of states
- Σ is finite input alphabet
- $q_0 \in Q$ is start state
- $\delta : Q \times \Sigma \to Q$ is "transition function"
- Ω is finite output alphabet
- $\omega : Q \times \Sigma \to \Omega \cup \{\Lambda\}$ is "output function".

Example transducer

What do these transducers do?





Regular Languages

Definition

$$R ::= \emptyset | \{\Lambda\} | \{s\}\ (s \in \Sigma) | R \cup R' | RR' | R^*$$

Regular Languages

Base cases.

$$\emptyset,\ \{\Lambda\},\ \{s\} : s \in \Sigma$$

Recursive cases.

$$\begin{aligned} R \cup S &= \{w \in \Sigma^* : w \in R \text{ or } w \in S\} \\ RS &= \{rs : r \in R,\ s \in S\} \\ R^* &= \{r_0 r_1 \ldots r_{n-1} : n \in \mathbb{N},\ r_i \in R \text{ for each } i < n\} \end{aligned}$$

Note: $\Lambda \in R^*$, always.

Regular Expressions

Shorthand to specify regular languages.

c Matches a single (non-special) character c.

(r) Matches same as r.

r^* Matches zero or more repetitions concatenated of strings matching r.

$r|s$ Matches any string that matches r or s.

rs Matches a word matching r concatenated with a word matching s.

Example

$(aa)^*$ matches strings of 'a's of even length.
$((ab)|c)d$ matches 'abd' and 'cd' only.

Other Regular Expressions

$[s]$ matches any one character from the string s, e.g. $[cat]$ matches c or a or t.

$[n-m]$ matches any one character in range n to m.

$[\sim s]$ matches any one character *not* in string s.

r^+ matches *one* or more repetitions of r.

$\cdot$ matches any one character from Σ.

$\backslash\backslash$ matches '$\backslash$'.

$\backslash *$ matches $*$

$\backslash c$ matches various special characters c

$\backslash t$ matches tab character

$\backslash n$ newline character

$\backslash ddd$ matches character with ASCII code ddd.

Kleene's Theorem

A language can be recognised by a FSM if and only if the language is a regular language.

I.e. If L is a regular language then there is an FSM $\mathcal{M}$ such that $L = L(\mathcal{M})$,
and conversely
if M is a FSM then $L(M)$ is a regular language.

Non-Deterministic FSMs

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

Only difference is

$$\delta : Q \times \Sigma \to \wp(Q)$$

E.g $\delta(q, s) = \{q_1, q_3, q_7\}$.

If $X \subseteq Q$ is a set of states and $s \in \Sigma$ is a character, let

$$\delta(X, s) = \bigcup_{q \in X} \delta(q, s)$$

How a NFSM works

Algorithm 2

$X = \{q_0\}$

for $p = 0,\ p < n,\ p{+}{+}$ **do**

  $X = \delta(X, s_p)$

end for

if $X \cap F = \emptyset$ **then**

  reject

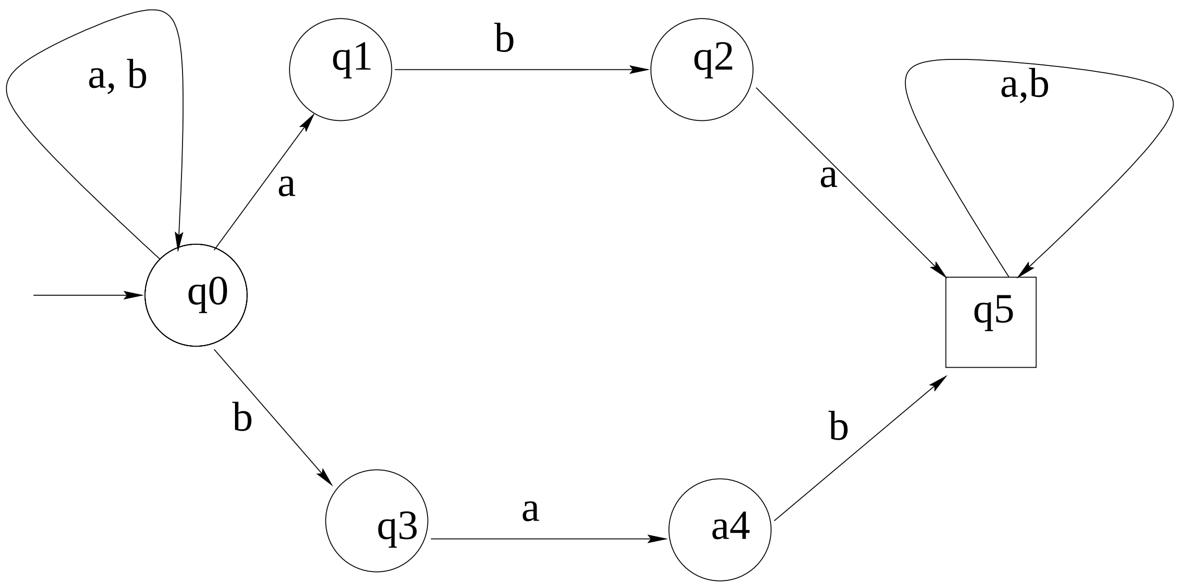
else

  accept

end if

$$L(M) = \{w \in \Sigma^* : \ M \text{ accepts } w\}$$

NFSM Example



Accepts any string of 'a's and 'b's containing either 'aba' or 'bab'.

Lemma

For every NFSM M there is a FSM M' such that

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

Proof. Let $M = (Q, \Sigma, q_0, \delta, F)$, where $\delta : Q \times \Sigma \to \wp(Q)$. Then let

$$M' = (\wp(Q), \Sigma, \{q_0\}, \delta', F')$$

where

$$\delta'(X, s) = \bigcup_{q \in X} \delta(q, s)$$

and

$$F' = \{X \in \wp(Q) : X \cap F \neq \emptyset\}$$

$\blacksquare$

Null Transitions

Λ-transitions.
Let

$$\delta : Q \times (\Sigma \cup \{\Lambda\}) \rightarrow \wp(Q)$$

If in state $q \in Q$ and there is $q' \in \delta(q, \Lambda)$ then it can move into q' without taking any input.

Eliminating Null Transitions

M is a NFSM-Λ. Define NFSM M'.

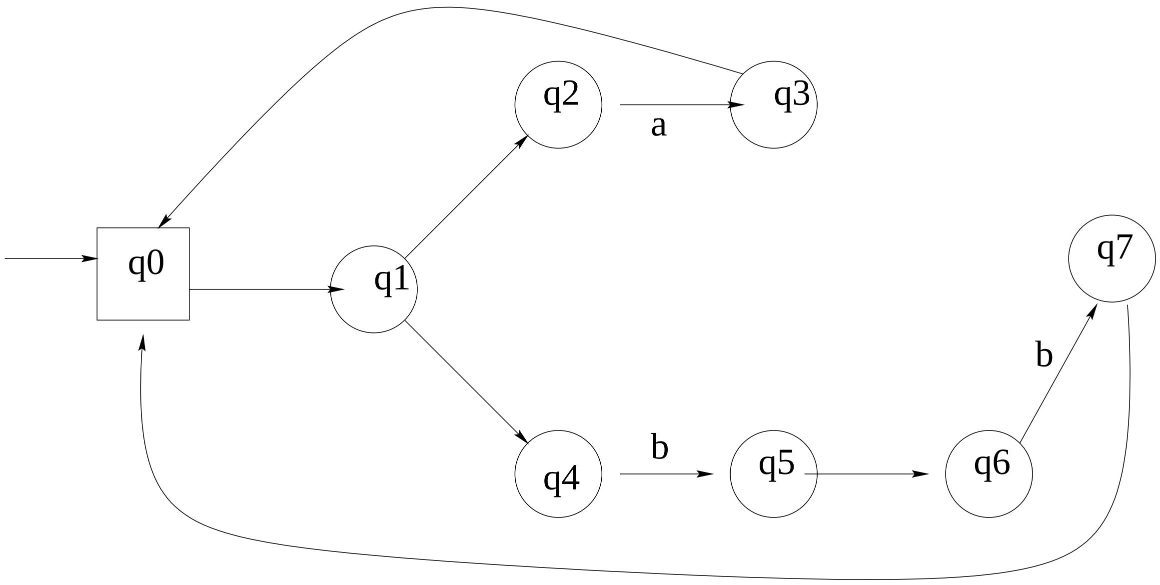
1. M' has same states, same initial state, same alphabet as M.

2. M' has same final states as M, but also q_0 if you can get to final state of M by Λ-transitions.

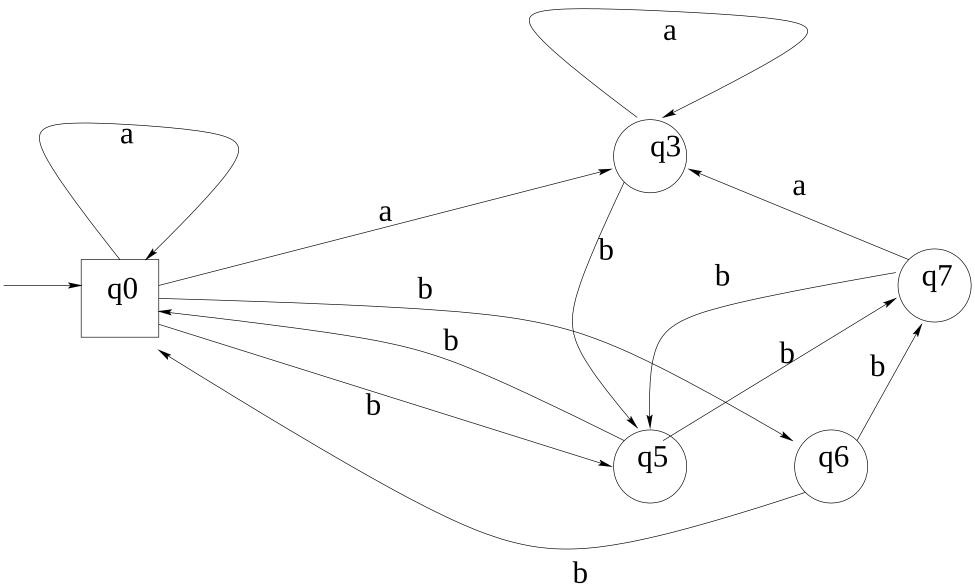
3. For each pair $q, r \in Q$ and each $s \in \Sigma$, if there is a path from q to r containing exactly one arc labelled s and all the rest labelled Λ, then include an arc from q to r labelled s in M'.

4. Remove isolated states.

Example



Eliminating Null Transitions



Summarising what we have done

FSM at most one transition labelled s from q

NFSM maybe more than one transition labelled s from q

NFSM-Λ also allows null transitions.

If M is any NFSM-Λ then there is an NFSM M_1 such that

$$L(M_1) = L(M)$$

and there is a FSM M_2 such that

$$L(M_2) = L(M_1) = L(M)$$

Every regular language is recognised by an FSM

If R is a regular language then there is FSM M with

$$L(M) = R$$

Given regular language R we find NFSM-Λ that recognised it, by structured induction.

Base Cases

$\emptyset$

$\{\Lambda\}$

$\{s\}$ where $s \in \Sigma$.

Exercise: define the appropriate FSMs in these cases.

Inductive Cases

Induction Hypothesis

Let R, S be regular languages and suppose there are NFSM-Λs M and N such that

$$L(M) = R \quad \text{and} \quad L(N) = R$$

Induction Steps

Concatenation Language RS

Union $R \cup S$

Iterative closure

Fill in the NFSM-Λs.

First half of Kleene's Theorem

By induction, for all regular languages R there is a NFSM-Λ M that accepts R. By previous slides, there is a FSM M' such that

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

This proves half of Kleene's Theorem (the hard part).

The other half of Kleene's Theorem

Let M be a FSM. Then $L(M)$ is a regular language.

$$M = (\{0, 1, \ldots, n-1\}, \Sigma, q_0, \delta, F)$$

Define a language

$$L(p, x, q) = \{w \in \Sigma* : \exists \text{path labelled } w \text{ from } p \text{ to } q \text{ with all internal nodes} < x\}$$

where $p, x, q < n$. We prove that $L(p, x, q)$ is regular by weak induction over x.

Base case $x = 0$

There are no states < 0 so a path from p to q must be a direct link, and the edge (p, q) must be labelled by some $s \in \Sigma$.

$$L(p, 0, q) = \{s \in \Sigma : q \in \delta(p, s)\} = \bigcup_{s : q \in \delta(p,s)} L(s)$$

which is a union of regular languages, hence a regular language.

Induction Hypothesis

For any states p, q, suppose $L(p, k, q)$ is regular (some $k \geq 0$).

Induction Step

Want to show that $L(p, k+1, q)$ is regular. Let

$$\begin{aligned} L'(p, k+1, q) &= L(p, k+1, q) \setminus L(p, k, q) \\ &= \{w \in \Sigma^* : \exists \text{path from } p \text{ to } q \\ &\qquad \text{passing through } k \text{ but} \\ &\qquad \text{none higher} \end{aligned}$$

If $w \in L'(p, k+1, q)$ then $w = tuv$ where

$$\begin{array}{rcl} t & \in & L(p,k,k) \\ u & \in & L(k,k+1,k) \\ v & \in & L(k,k,q) \end{array}$$

$$u = \Lambda | u_0 u_1 \ldots u_i$$

where $u_j \in L(k,k,k)$, each $j \leq i$. So

$$u \in L(k,k,k)^*$$

Hence,

$$L(p,k+1,q) = L(p,k,q) \cup L(p,k,k) L(k,k,k)^* L(k,k,q)$$

which is a regular language, using I.H.

By induction, $L(p,x,q)$ is always regular. So

$$L(M) = \bigcup_{q \in F} L(q_0, n, q)$$

is a regular language.