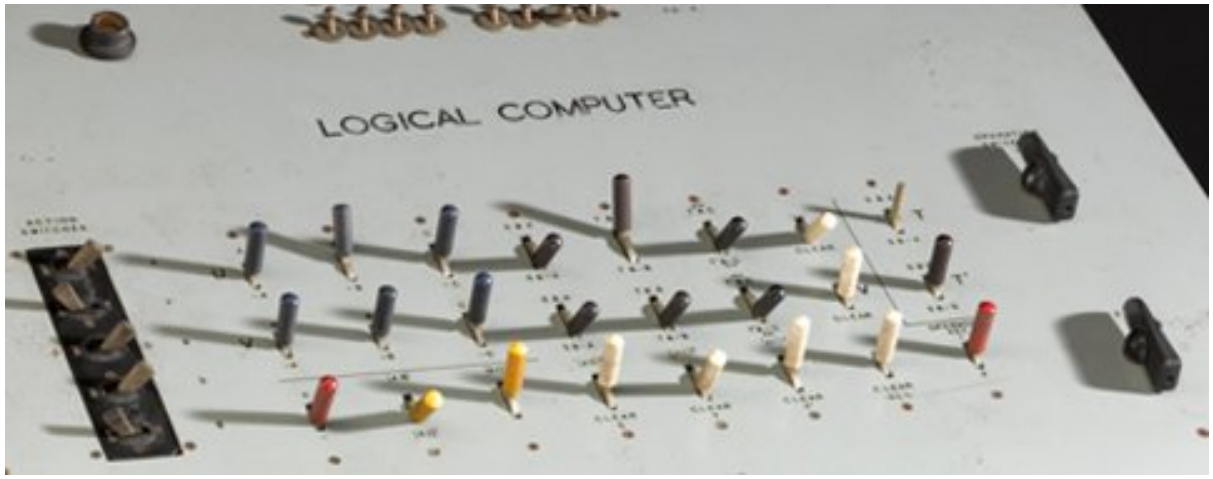
INF551

Computational Logic:

Artificial Intelligence in Mathematical Reasoning



Stéphane Graham-Lengrand

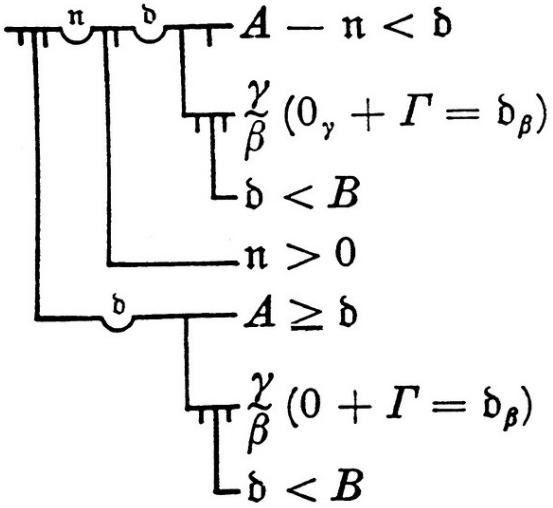
Stephane.Lengrand@Polytechnique.edu

Lecture I

What do you remember from your undergraduate programme?

Contents

I. Predicate logic



Stating the obvious

Logic is about statements, statements require language.

As usual, language = syntax + semantics

Predicate logic on two levels:

- *terms* (whose semantics are the mathematical objects you want to talk about)
- *formulae* (the aforementioned statements)

The rest of this section presents chapter 5 of INF412 course notes.

More generally, today's slides indicate section numbers in course notes:

Section in INF412	Slide title	Section in INF551

http://www.enseignement.polytechnique.fr/informatique/INF412/i.php/Main/Poly

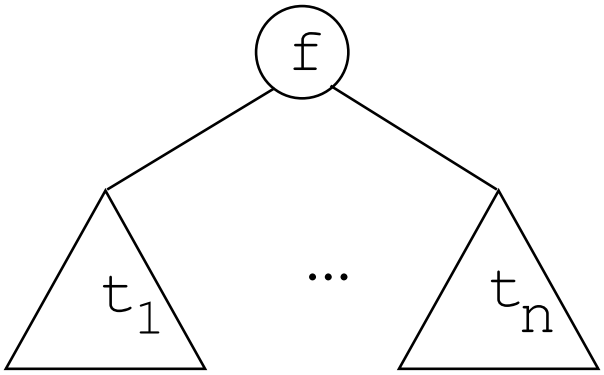
Syntax of terms depends on:

- a *term signature*, i.e. a set of elements called *term symbols*, equiped with a function mapping term symbols to natural numbers called *their arities*
- a denumerable set of elements called *variables*

Terms are defined *inductively* as follows:

- a variable is a term
- for every term symbol f of arity n in the signature, if t_1,...,t_n are terms, then $f(t_1, \ldots, t_n)$ is a term

Do not see $f(t_1, \ldots, t_n)$ as a string with parentheses and commas: even though I use this as *concrete syntax* on my slides, what I am talking about is really a tree, with the root labelled with symbol f and n direct sub-trees t_1,...,t_n:



That was long and tedious. Here's a more synthetic presentation of the same thing:

Let Σ be a term signature. The set of terms is defined by

$$t, t_1, t_2, \ldots ::= x \mid f(t_1, \ldots, t_n)$$

with x ranging over variables and f/n ranging over Σ

Similarly:

predicate signature = set of elements called *predicate symbols*, equiped with function mapping predicate symbols to natural numbers called *their arities*

Let Ψ be a predicate signature. The set of (pre-)formulae is defined by

$$\begin{aligned} A, B, C, \ldots ::= \quad & p(t_1, \ldots, t_n) \\ | \quad & \top \mid \bot \mid \neg A \\ | \quad & A \wedge B \mid A \vee B \mid A \Rightarrow B \\ | \quad & \forall x A \mid \exists x A \end{aligned}$$

with x ranging over variables and p/n ranging over Ψ

Again, we are talking about trees

But this is not finished!

Formulae are not *exactly* trees, because

$\forall x\ p(x)$ is *the same formula* as $\forall y\ p(y)$ (though technically $\neq$ trees)

x is a *bound* variable, as is x in $\int_0^1 f(x)dx$

Set of formulae = set of such trees *quotiented* by the renaming of bound variables

The next two slides formalise this.

Bound = not free

Free variables of terms, defined by induction on terms:

$$\begin{aligned} \mathsf{FV}(x) &:= x \\ \mathsf{FV}(f(t_1, \ldots, t_n)) &:= \mathsf{FV}(t_1) \cup \cdots \cup \mathsf{FV}(t_i) \end{aligned}$$

Free variables of formulae, defined by induction on formulae:

$$\begin{aligned} \mathsf{FV}(p(t_1, \ldots, t_n)) &:= \mathsf{FV}(t_1) \cup \cdots \cup \mathsf{FV}(t_n) \\ \mathsf{FV}(\top) = \mathsf{FV}(\bot) &:= \emptyset \\ \mathsf{FV}(\neg A) &:= \mathsf{FV}(A) \\ \mathsf{FV}(A \wedge B) = \mathsf{FV}(A \vee B) = \mathsf{FV}(A \Rightarrow B) &:= \mathsf{FV}(A) \cup \mathsf{FV}(B) \\ \mathsf{FV}(\forall x A) = \mathsf{FV}(\exists x A) &:= \mathsf{FV}(A) \backslash \{x\} \end{aligned}$$

t (resp. A) is closed if $\mathsf{FV}(t) = \emptyset$ (resp. $\mathsf{FV}(A) = \emptyset$)

Define the swap of 2 variables in (all terms, and then) all formulae P : $(xy)P$

everywhere where x is written (free or bound), write y, and vice versa

(easy definition by induction on terms and formulae)

$\forall x P$ is identified with $\forall y\ (xy)P$ if $y \notin \mathrm{FV}(P)$

$\exists x P$ is identified with $\exists y\ (xy)P$ if $y \notin \mathrm{FV}(P)$

α-equivalence := smallest equivalence relation on trees identifying the above at the root of trees or in their sub-trees

Example: $p(w) \wedge \forall x\ p(x + w)$ and $p(w) \wedge \forall y\ p(y + w)$ are α-equivalent

Why "if $y \notin \mathrm{FV}(P)$" (i.e. y is a fresh variable)?

$(y = -1) \wedge \exists x(x \times x = y)$ not the same as $(y = -1) \wedge \exists y(y \times y = x)$

Formulae = equivalence classes of trees (modulo α-equivalence)

Given a *language* made of a term signature Σ and a predicate signature Ψ,

a (bivaluated) *model structure* of that language is:

- a non-empty set $\mathcal{M}$
- for all f/n in Σ, a function $\hat{f}$ from $\mathcal{M}^n$ to $\mathcal{M}$
- for all p/n in Ψ, a function $\hat{p}$ from $\mathcal{M}^n$ to $\{0,1\}$

Example:

A model structure for $\Sigma = \{0/0, S/1, +/2, \times/2\}$ and $\Psi = \{=/2\}$ can be

$$\mathbb{N} := (\mathbb{N}, 0, n \mapsto n+1, \quad (n,m) \mapsto n+m,$$
$$(n,m) \mapsto n \times m,$$
$$(n,m) \mapsto 1 \text{if } n = m \text{ or } 0 \text{ if not})$$

What is the semantics of $x + x$? Depends on the value of x!

Valuation = function from (finite sets of) variables to $\mathcal{M}$

Given a model structure as on previous slide,

Semantics of terms: given term t & valuation ϕ covering FV(t),

the semantics of t according to ϕ, denoted $[\![t]\!]_\phi$ & defined in $\mathcal{M}$ by induction on t:

$$
\begin{aligned}
[\![x]\!]_\phi &:= \phi(x) \\
[\![f(t_1,\ldots,t_n)]\!]_\phi &:= \hat{f}([\![t_1]\!]_\phi,\ldots,[\![t_n]\!]_\phi)
\end{aligned}
$$

Semantics of formulae:

given formula A & valuation ϕ covering FV(A), the semantics of A according to ϕ, denoted $[\![A]\!]_\phi$ & defined in $\{0,1\}$ by induction on A:

$$
\begin{aligned}
[\![p(t_1,\ldots,t_n)]\!]_\phi &:= \hat{p}([\![t_1]\!]_\phi,\ldots,[\![t_n]\!]_\phi) \\
[\![A \wedge B]\!]_\phi &:= \hat{\wedge}([\![A]\!]_\phi,[\![B]\!]_\phi) \\
&\ldots \\
[\![\forall x A]\!]_\phi &:= 1 \text{ if for all } a \in \mathcal{M}, [\![A]\!]_{\phi,x\mapsto a} = 1 \\
&:= 0 \text{ if not} \\
[\![\exists x A]\!]_\phi &:= 1 \text{ if there is } a \in \mathcal{M} \text{ with } [\![A]\!]_{\phi,x\mapsto a} = 1 \\
&:= 0 \text{ if not}
\end{aligned}
$$

. . . with $\hat{\top} = 1, \hat{\bot} = 0, \hat{\neg}, \hat{\wedge}, \hat{\vee}, \hat{\Rightarrow}$ . . . given by your favourite truth tables

Notation To specify the model structure, say $\mathbb{M}$, used to compute the semantics of term t (resp. formula A), we sometimes write $[\![t]\!]^{\mathbb{M}}_{\phi}$ (resp. $[\![A]\!]^{\mathbb{M}}_{\phi}$)

. . . but often omitted if the model structure is clear (simply writing $[\![t]\!]_{\phi}$ or $[\![A]\!]_{\phi}$)

Remark When A is closed, no need for valuation:

$[\![A]\!]_{\phi}$ independent from ϕ (in which case we may simply write $[\![A]\!]$)

. . . but valuation useful for sub-formulae

Terminology A *theory* is a set of closed formulae

A model structure is said to be a *model* of a closed formula A (resp. a theory $\mathcal{T}$) if $[\![A]\!] = 1$ (resp. for every A in $\mathcal{T}$, $[\![A]\!] = 1$)

A closed formula A is *valid*, denoted $\models A$, if every model structure is a model of A

A closed formula A is a *semantical consequence* of a theory $\mathcal{T}$, denoted $\mathcal{T} \models A$, if every model of $\mathcal{T}$ is a model of A

We're supposed to build mathematics from scratch, and we haven't got to natural numbers yet, but so far you have used

- characters
- trees
- sets!

II. Meta-mathematics

Where the snake seems to bite its tail

Mathematics = the art of formally studying concepts

Logic until INF412 (and until Frege) = the informal way you did mathematics

But logic itself can be formalised as an object of mathematical study

Example:

- Define the set of formulae
- Define the set of valid formulae

- But to do so requires a few mathematical concepts

  Examples : numbers, trees, sets ; how are these defined ?
- if Logic itself is our object of study, which logic do we use to reason about it?

Impossible to do any mathematics ex-nihilo

Meta-mathematics

To address this, Hilbert introduced the idea that the snake bites **another** snake's tail:

Distinguish the object level and the meta level:

- objet level = the logic / the theory / the rules. . . that we define / study
- meta level = the logic / the theory / the rules. . . that we use to reason about object level

  Remark: the two snakes have no reason to be of the same species

  i.e. object-level and meta-level have no reason to be the same logic

- Meta-level is usually implicit (just like when you studied geometry at school)

  If one day you formalise the meta-level, you will informally place yourself in a meta-meta-level, and so on.

  To start off doing *anything*, we must informally agree on some basic concepts like trees
- *OR* we can use a machine as a *reference* for the meta-level (cf. practicals)

  but you still have to trust the implementation

A stupid remark

Look again at the definition of $[\![A]\!]$

We think we do semantics, giving a "meaning" to formulae...

...and "$[\![A]\!] = 1$" is a statement of the meta-level

But in fact, we are just translating A from the object-level to the meta-level:

$[\![A \wedge B]\!] = 1$ if $[\![A]\!] = 1$ and $[\![B]\!] = 1$

$[\![\forall x A]\!] = 1$ if for all $a \in \ldots$, $[\![A]\!]_{x \mapsto a} = 1$

... $\wedge$ is translated as "and", $\forall$ is translated as "for all",...

Semantics = Syntax of the meta-level

In other words, everything is syntax

Hilbert's program (1920)

Hilbert: that hierarchy of snakes/levels (object, meta, meta-meta,...) is acceptable if the same language and logic can be used at all levels.

His 1920 program:

Find a language to express all mathematics.

Find a logic X made of finitely many axioms + a notion of proof $\vdash_X$ and,

writing X for X used as the meta-logic and X for X used as the object logic,

- **Completeness**:

  Prove in X that all "true" mathematical statements can be proved in X

  i.e. Prove in X the theorem "If $[\![A]\!] = 1$ then $\vdash_X A$"

- **Consistency**: Prove in X that no contradiction can be proved in X

  i.e. Prove in X the theorem "$\nvdash_X \bot$"

- **Decidability**:

  Have an algorithm that terminates on all inputs A, outputting whether $\vdash_X A$ or $\nvdash_X A$

- X should be based on arithmetic (the only thing that indeniably exists),

  but all the mathematics we know should be derived from it

Hilbert's program after the 30s

- Language to express all mathematics: that of predicate logic 

- Logic X, axioms to derive all mathematics: DEPENDS

Equality+ Arithmetic itself ($\mathcal{PA}$) : **weak**
Zermelo-Fraenkel's Set theory (cf. Lect. 2 & 3) : **ok**


In both cases: "few" axioms but with an axiom schema

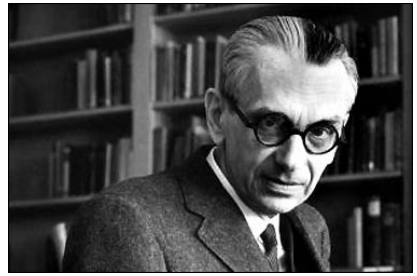
- Logic X, notion of proof: predicate logic 

- Logic X, completeness: DEPENDS

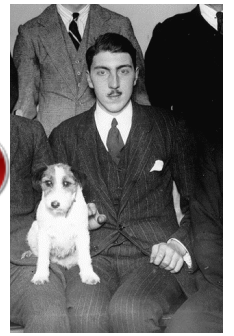
$[\![A]\!] = 1$ ambiguous, as it depends on model structure!

If $[\![A]\!] = 1$ is meant "in all model structures", then 

If $[\![A]\!] = 1$ is meant "in $\mathbb{N}$" (written $[\![A]\!]^{\mathbb{N}} = 1$) for $\mathcal{PA}$, then 



- Logic X, consistency:




- Logic X, decidability (Entscheidungsproblem):

III. Some theories

Let Σ be a term signature and Ψ a predicate signature containing at least $= /2$

$$\forall x, x = x$$

$$\forall x_1 \ldots x_i x_i' x_{i+1} \ldots x_n,$$
$$x_i = x_i' \Rightarrow$$
$$f(x_1, \ldots, x_i, \ldots, x_n) = f(x_1, \ldots, x_i', \ldots, x_n) \quad \text{for each } f/n \in \Sigma$$

$$\forall x_1 \ldots x_i x_i' x_{i+1} \ldots x_n,$$
$$x_i = x_i' \Rightarrow$$
$$p(x_1, \ldots, x_i', \ldots, x_n) \Rightarrow p(x_1, \ldots, x_i, \ldots, x_n) \quad \text{for each } p/n \in \Psi$$

$\Sigma = \{0/0, S/1, +/2, \times/2\}$, and $\Psi = \{= /2\}$

Equality axioms, plus:

$\forall y\,(0 + y = y)$

$\forall x\,\forall y\,(S(x) + y = S(x + y))$

$\forall y\,(0 \times y = 0)$

$\forall x\,\forall y\,(S(x) \times y = (x \times y) + y)$

$\forall x\,\forall y\,(S(x) = S(y) \Rightarrow x = y)$

$\forall x\,\neg(0 = S(x))$

$\forall x_1 \ldots x_n\,(\{^0/_z\}P \Rightarrow \forall x\,(\{^x/_z\}P \Rightarrow \{^{S(x)}/_z\}P) \Rightarrow \forall y\,\{^y/_z\}P)$

for all formulae P with $\mathrm{FV}(P) = \{x_1, \ldots, x_n, z\}$

Last line is an *axiom schema* (infinitely many axioms)!

Remark: $\mathbb{N}$ is a model of Peano's arithmetic

. . . is what you imagine it is.

Careful though when defining $\{{}^{t}/_{x}\}(\forall y P)$ and $\{{}^{t}/_{x}\}(\exists y P)$!!!

What if $y \in \mathrm{FV}(t)$?

Hint:

$$\{{}^{t}/_{x}\}(\forall y P) = \forall y \{{}^{t}/_{x}\} P$$
$$\{{}^{t}/_{x}\}(\exists y P) = \exists y \{{}^{t}/_{x}\} P$$

if $x \neq y$ and $y \notin \mathrm{FV}(t)$

otherwise, rename y by swapping it with a fresh variable z: $(yz)P$

Many notions stacked on each other:

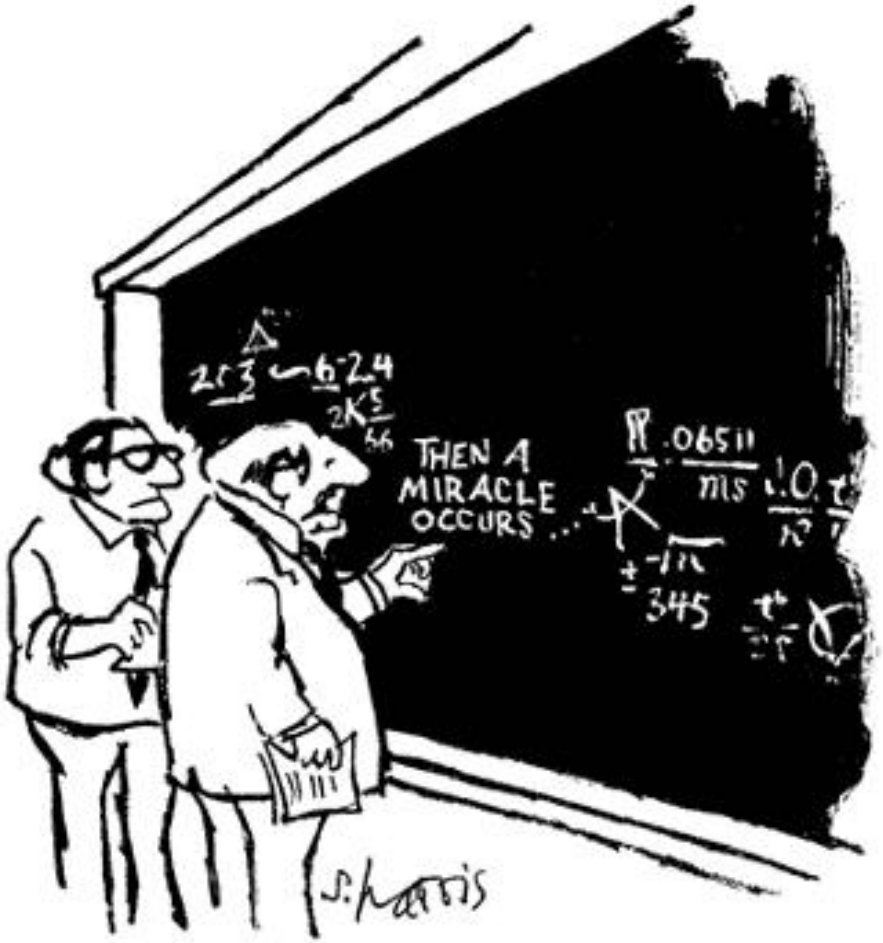
variable swapping . α-equivalence . quotienting trees . substitution

Many mistakes in books

Many mistakes in symbolic computation systems

(programming languages, proof systems, . . .)

IV. The notion of proof & the soundness/completeness theorem



"I think you should be more explicit here in step two."

The definitions of validity & semantical consequence...

- ...are annoying, as they heavily rely on meta-level
- ...are nowhere near offering a algorithmic way of checking

  (at least in predicate logic)

  even checking that a structure is a model of a formula may require infinitely many checks ($\forall \ldots$)

Obsession of mathematicians:

justify, by an argument of finite size, statements about infinitely many objects

In that respect, semantics = useless (translates object-level $\forall$ to meta-level "for all")

Looking for more syntactic notion, with a *finite* object that

- justifies validity
- can be communicated and checked by someone else (or a computer)

$\Rightarrow$ **The notion of proof**

Many different formalisms

In all of them (except exceptions) proofs are finite objects, with a notion of size

(you can prove a property of all proofs by induction on their sizes)

Frege-Hilbert proofs:

Proof of $\mathcal{T} \vdash A$ = sequence of formulae such that:

- the last one is A
- every formula is either
  - a consequence of the previous ones by one of those two inference rules:

$$\frac{A \Rightarrow B \quad A}{B} \qquad \frac{A}{\forall x A}$$

  - or a formula in $\mathcal{T}$
  - or one of the “logical axioms”

    See INF412 course notes for their list, or practical

By looking, in that sequence of formulae, at how they depend on each other by the two inference rules, you can also see a Frege-Hilbert proof as a tree

- whose internal nodes are labelled by instances of the two inference rules
- whose leaves are labelled by logical axioms or axioms from $\mathcal{T}$
- whose conclusion is labelled by A

Example:

$$\dfrac{\dfrac{}{B \Rightarrow C \Rightarrow D} \qquad \dfrac{\dfrac{}{A \Rightarrow B} \qquad \dfrac{}{A}}{B}}{C \Rightarrow D} \qquad \dfrac{\dfrac{}{A \Rightarrow C} \qquad \dfrac{}{A}}{C}$$
$$\dfrac{C \Rightarrow D \qquad\qquad C}{D}$$

is a proof of $A, A \Rightarrow B, A \Rightarrow C, B \Rightarrow C \Rightarrow D \vdash D$

Pushing further the concept of proofs-as-trees, *Natural Deduction* is an alternative formalism for proofs

$$\frac{}{\Gamma, A \vdash A}$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B} \qquad \frac{\Gamma \vdash A \Rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B}$$

$$\frac{\Gamma \vdash P}{\Gamma \vdash \forall x P}\, x \notin \mathsf{FV}(\Gamma) \qquad \frac{\Gamma \vdash \forall x P}{\Gamma \vdash \left\{{}^{t}\!/_{x}\right\} P}$$

$$\frac{\Gamma \vdash \left\{{}^{t}\!/_{x}\right\} P}{\Gamma \vdash \exists x P} \qquad \frac{\Gamma \vdash \exists x P \quad \Gamma, P \vdash Q}{\Gamma \vdash Q}\, x \notin \mathsf{FV}(\Gamma, Q)$$

$$\frac{\Gamma \vdash A_1 \quad \Gamma \vdash A_2}{\Gamma \vdash A_1 \wedge A_2} \qquad \frac{\Gamma \vdash A_1 \wedge A_2}{\Gamma \vdash A_i}\, i \in \{1, 2\}$$

$$\frac{\Gamma \vdash A_i}{\Gamma \vdash A_1 \vee A_2}\, \{i, j\} = \{1, 2\} \qquad \frac{\Gamma \vdash A_1 \vee A_2 \quad \Gamma, A_1 \vdash B \quad \Gamma, A_2 \vdash B}{\Gamma \vdash B}$$

$$\frac{}{\Gamma \vdash \top} \qquad \frac{\Gamma, A \vdash \bot}{\Gamma \vdash \neg A} \qquad \frac{\Gamma \vdash \bot}{\Gamma \vdash A} \qquad \frac{\Gamma \vdash \neg\neg A}{\Gamma \vdash A}$$

Soundness theorem:

$$\text{If } \mathcal{T} \vdash A \text{ then } \mathcal{T} \models A$$

Proof: Easy induction on the size of the proof, using truth tables.

(just mind how you deal with free variables)

Corollary:

Let $\mathbb{M}$ be a model of $\mathcal{T}$

$$\text{If } \mathcal{T} \vdash A \text{ then } \mathbb{M} \text{ is a model of } A$$

By contraposition:

$$\text{If } \mathbb{M} \text{ is not a model of } A, \text{ then } \mathcal{T} \nvdash A$$

Method to prove that A cannot be proved from $\mathcal{T}$:

find a model of $\mathcal{T}$ where $[\![A]\!] = 0$

Example with $A = \bot$: to prove that $\mathcal{T}$ is consistent, it suffices to find a model of $\mathcal{T}$

Another consequence of soundness

Let $\mathcal{T}$ be a theory and A a closed formula.

Assume there is a model of $\mathcal{T}$ where $[\![A]\!] = 1$ and another one where $[\![A]\!] = 0$.

What can you say about $\mathcal{T} \vdash A$? about $\mathcal{T} \vdash \neg A$?

Hilbert seriously hoped that this would not be the case for $\mathcal{T} = \mathcal{PA}$

Completeness theorem:

Assume the term and predicate signatures are denumerable.

$$\text{If } \mathcal{T} \models A \text{ then } \mathcal{T} \vdash A$$

Proof: Suppose $\mathcal{T} \nvdash A$. Cleary, $\mathcal{T}, \neg A \nvdash \bot$. If we could prove that every consistent theory has a model (see next slide), then that model of $\mathcal{T}, \neg A$ would be a model of $\mathcal{T}$ that is not a model of A. Qed.

Corollary: The compactness theorem

If $\mathcal{T} \models A$ then there is a finite subset Γ of $\mathcal{T}$ such that $\Gamma \models A$

Proof: Since proofs are *finite object*, a proof of $\mathcal{T} \vdash A$ only uses finitely many axioms of $\mathcal{T}$; call them Γ; we have $\Gamma \vdash A$ and by soundness $\Gamma \models A$.

Completion of a theory Lemma:

If a theory $\mathcal{T}$ on a denumerable signature (Σ, Ψ) is consistent, then there is a consistent theory $\mathcal{T}' \supseteq \mathcal{T}$ on denumerable signature (Σ', Ψ) (with $\Sigma' \supseteq \Sigma$) such that

1. for all closed formulae A, either $\mathcal{T}' \vdash A$ or $\mathcal{T}' \vdash \neg A$
2. for all closed formulae $\exists x A$,
   if $\mathcal{T}' \vdash \exists x A$ then $\mathcal{T}' \vdash \left\{{}^{t}\!/_{x}\right\} A$ for some closed term t made from signature Σ'

Proof: see INF412 or INF551 course notes

Now, we define a model structure for the signature (Σ, Ψ):

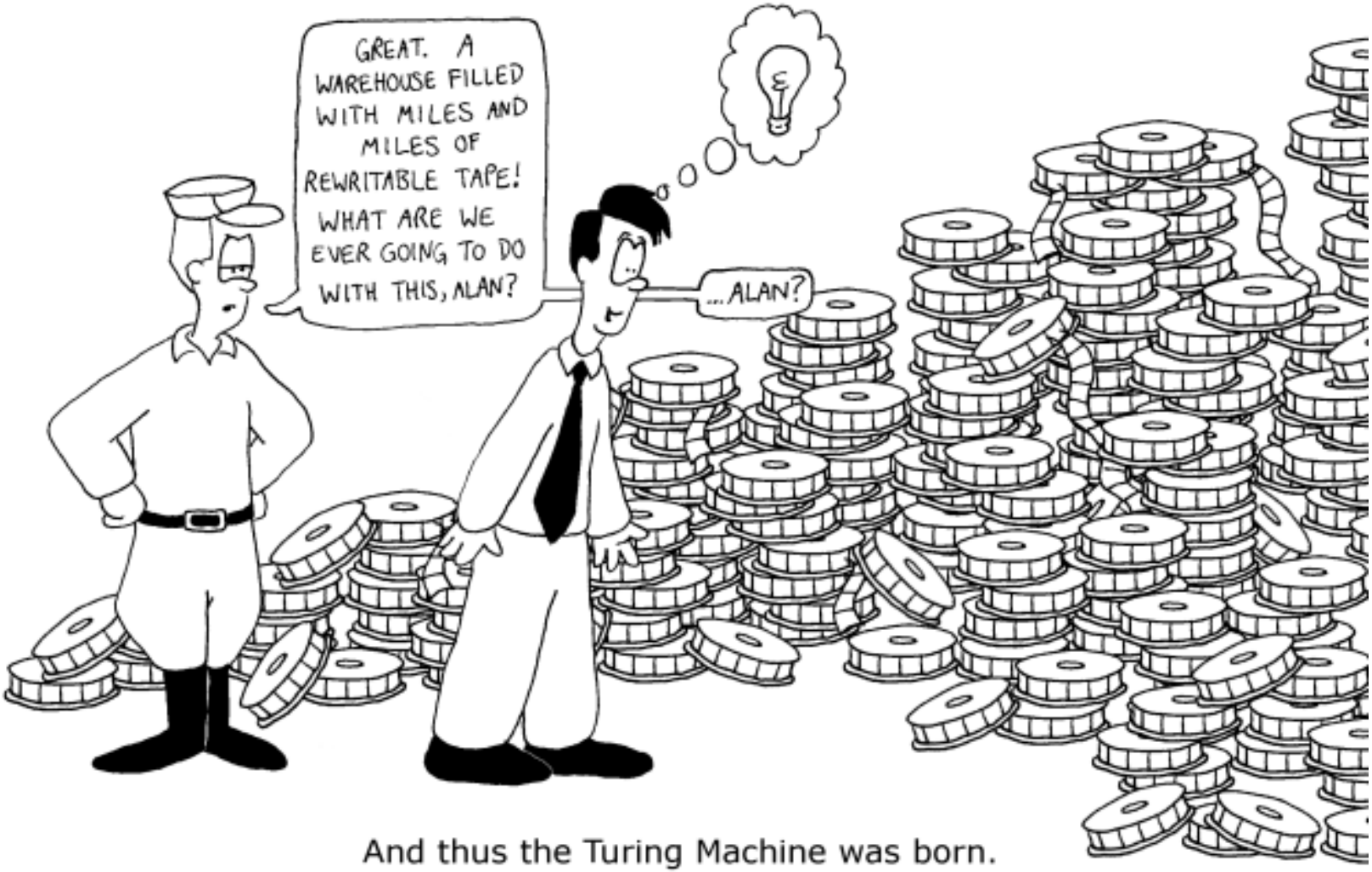
- $\mathcal{M}$ is the set of closed terms on Σ'
- for every f/n in Σ, $\hat{f}$ maps $t_1, \ldots, t_n$ to $f(t_1, \ldots, t_n)$
- for every p/n in Ψ, $\hat{p}$ maps $t_1, \ldots, t_n$ to 1 if $\mathcal{T}' \vdash p(t_1, \ldots, t_n)$ and to 0 if not

Lemma: For all closed formulae A, if $\mathcal{T}' \vdash A$ then $[\![A]\!] = 1$ in that model structure

Proof: Easy induction on the size of A, using 1. and 2.

This structure is a model of $\mathcal{T}$ (for all $A \in \mathcal{T}$ we have $A \in \mathcal{T}'$ and therefore $\mathcal{T}' \vdash A$)

V. Computability



And thus the Turing Machine was born.

Define a notion of *programs* (e.g. Turing machines) in denumerable numbers,

each of them *representing* a partial function from $\mathbb{N}$ to $\mathbb{N}$

(i.e. a total function from $\mathbb{N}$ to $\mathbb{N} \cup \{\infty\}$)

We apply programs to numbers by interpreting them as their represented function

(If p is a program and n a number, we can write $p(n)$)

Partial functions from $\mathbb{N}$ to $\mathbb{N}$ (or total functions from $\mathbb{N}$ to $\mathbb{N} \cup \{\infty\}$) that are represented by programs are said to be *computable*

A program p *terminates* on n if $p(n) \neq \infty$

A set of natural numbers A is *decidable* (resp. *semi-decidable*) if the function

$$\begin{array}{rcl} \mathbb{N} & \longrightarrow & \mathbb{N} \cup \{\infty\} \\ n \in A & \mapsto & 1 \\ n \notin A & \mapsto & 0 \end{array} \qquad \left(\text{resp.} \quad \begin{array}{rcl} \mathbb{N} & \longrightarrow & \mathbb{N} \cup \{\infty\} \\ n \in A & \mapsto & 1 \\ n \notin A & \mapsto & \infty \end{array} \right)$$

is computable.

Generalising to other sets $\mathcal{C}$, $\mathcal{D}$,...

Extend this to functions from $\mathcal{C}$ to $\mathcal{D}$ if you have injections into $\mathbb{N}$ ($\ulcorner\cdot\urcorner^{\mathcal{C}}$ and $\ulcorner\cdot\urcorner^{\mathcal{D}}$):

$f : \mathcal{C} \longrightarrow \mathcal{D}$ computable if $\begin{array}{ccc} \mathbb{N} & \longrightarrow & \mathbb{N} \\ \ulcorner x \urcorner^{\mathcal{C}} & \mapsto & \ulcorner f(x) \urcorner^{\mathcal{D}} \end{array}$ computable

Formally, computability depends on injections,

but two injections $\ulcorner\cdot\urcorner, \ulcorner\cdot\urcorner' : \mathcal{C} \longrightarrow \mathbb{N}$ define the same notion of computability for $\mathcal{C}$

if $\ulcorner x \urcorner \mapsto \ulcorner x \urcorner'$ and its inverse (from $\mathbb{N}$ to $\mathbb{N}$) are computable

Hence, 1 notion of computability for

- finite sets
- $\mathbb{N}^2$ such that projections are computable
- $\mathbb{N}^n$ such that projections are computable
- lists such that access to the head and acces to the tail are computable
- trees such that access to sub-trees and node-labels are computable

etc. In practice, no ambiguity

Programs can be represented as numbers.

There is a **universal program**, i.e. a program u such that for any program p and any number n, $u(\ulcorner(\ulcorner p\urcorner, n)\urcorner) = p(n)$

The **halting problem** is undecidable:

The set of all $\ulcorner(\ulcorner p\urcorner, n)\urcorner$ such that p terminates on n is undecidable

The **generalised halting theorem**:

Let A be a decidable set of programs such that every program in A always terminates (i.e. for all $p \in A$ and all $n \in \mathbb{N}$, p terminates on n).

There is a total computable function that is not represented by a program in A

Formulae and proofs can be represented as numbers

Theorem: The set of all A such that $\mathcal{PA} \vdash A$ is semi-decidable:

Proof:

Lemma (Checking a proof is decidable):

The function mapping $\ulcorner(\ulcorner\pi\urcorner, \ulcorner A\urcorner)\urcorner$ to 1 if π is a proof for $\mathcal{PA} \vdash A$, to 0 otherwise, is computable by a program p

Given a closed formula A, enumerate all numbers n until $p(\ulcorner(n, \ulcorner A\urcorner)\urcorner) = 1$.

If $\mathcal{PA} \vdash A$, then the program will find a proof and terminate.

If not it will run forever.

What's next

Practical: try solving propositional problems by a computer

Next time: how people have done it before you

Questions?