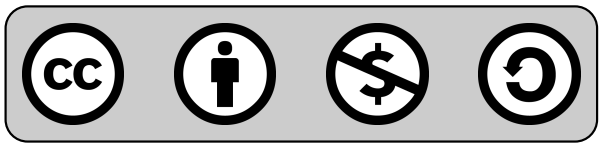
IN112 Mathematical Logic

Christophe Garion
DMIA – ISAE

License CC BY-NC-SA 3.0



Quotes...

It is reasonable to hope that the relationship between computation and mathematical logic will be as fruitful in the next century as that between analysis and physics in the last. The development of this relationship demands a concern for both applications and mathematical elegance.

John Mac Carthy, 1963

"Contrariwise," continued Tweedledee, "if it was so, it might be; and if it were so, it would be; but as it isn't, it ain't. That's logic."

Lewis Carroll, *Alice's Adventures in Wonderland*

Humans make illogical decisions.

Mr. Spock, *Star Treck*

Outline

1 - Introduction

1. **History of logic**
2. **A first guided tour on mathematical logic**
3. **A short bibliography**
4. **Agenda**

What is mathematical logic?

Informal definition

Mathematical logic is the study of the validity of an **argument** as a mathematical object.

First question: what is an argument?

An argument is composed of:

- a set of **declarative** sentences called **premises**
- a word, **therefore**
- a **declarative** sentence called **conclusion**

What is mathematical logic?

Informal definition

Mathematical logic is the study of the **validity** of an argument as a mathematical object.

Second question: what is validity?

Validity of an argument can be defined:

- in **model theory**: is the conclusion true when premises are?
- in **proof theory**: does the argument respect some rules?

A multi-disciplinary field

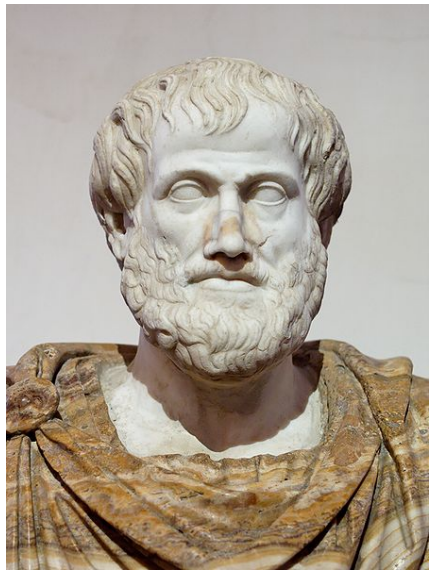
- ~~what is true?~~
- ~~what is false?~~

- what is a proof ?
- what mathematical structures do we need to define a proof?
- is this proof correct?

- is this program correct?
- can I automatically produce code that respect those specifications?
- can we prove automatically that this theorem is true?

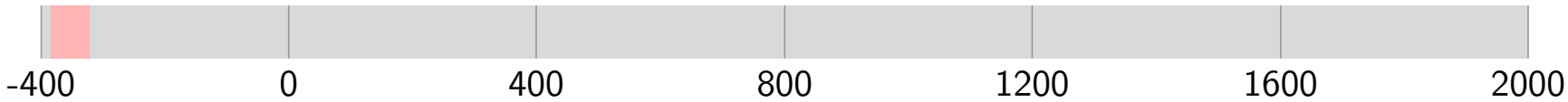
Outline of part 1 - Introduction

Aristotle (384 BC - 322 BC)



- the first work on logic as an autonomous discipline
- reasoning tool: syllogism

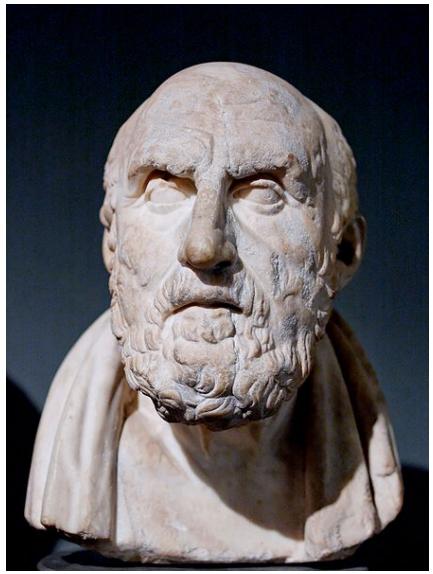
  every man is mortal
  every Greek is a man
  every Greek is mortal
- reasoning does not depend on the **content** of propositions but on their **form**
- notion of **validity**
- notion of **proof by contradiction**

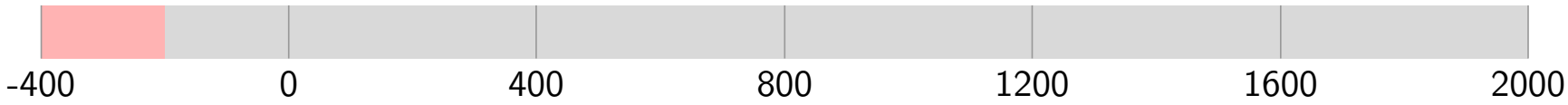


From Antiquity to 18th century

Megarians/Stoics school (500 BC - 200 BC)

- a logic of propositions: **atomic** propositions and **composed** propositions
- **implication**, **disjunction**, **conjunction**
- meaning of sentences (*lektón*)
- modalities

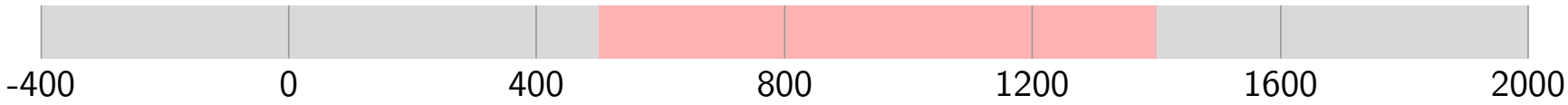




From Antiquity to 18th century

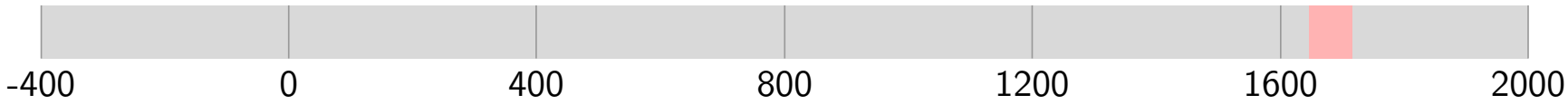
Medieval/scholastic logic (500 AD - 1400 AD)

- teached in faculty: trivium (logic + grammar + rethorics)
- mainly works on Aristotle logic and syllogisms



Gottfried Wilhelm Leibniz (1646 - 1716)

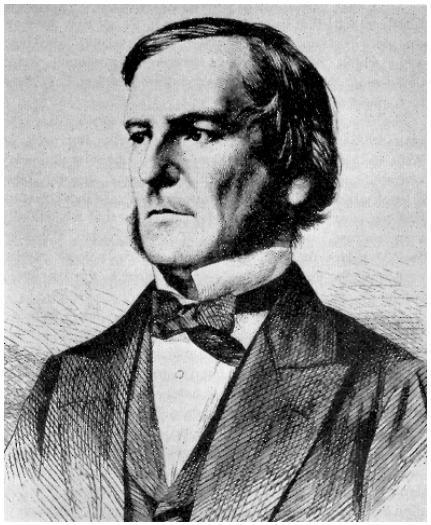
- a visionary in philosophy and mathematics
- *de arte combinatoria*: a first step for showing isomorphism between propositional calculus and classes calculus
- *lingua characteristica universalis*: a **written** artificial language (a sign $\equiv$ an idea) based on prime numbers
- *calculus ratiocinator* (unachieved work): logic as calculus on signs, same idea as reasoning on equations

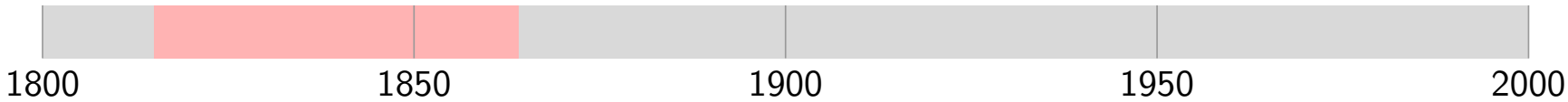


The arising of modern logic

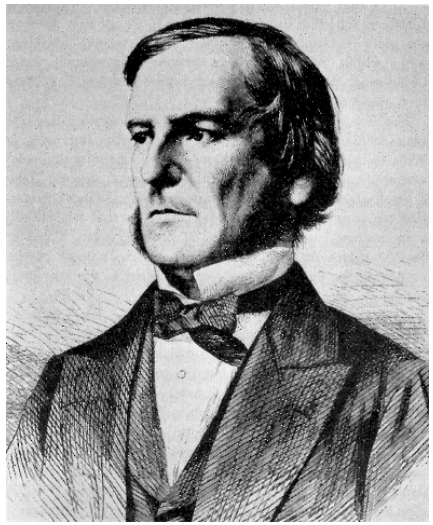
George Boole (1815 - 1864)

- english algebraic school
- same idea as Leibniz's: logic viewed as a calculus
- the first real mathematization of logic
- An Investigation of the Laws of Thought (1854)

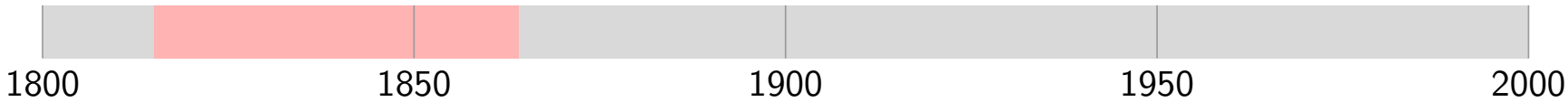




George Boole (1815 - 1864)



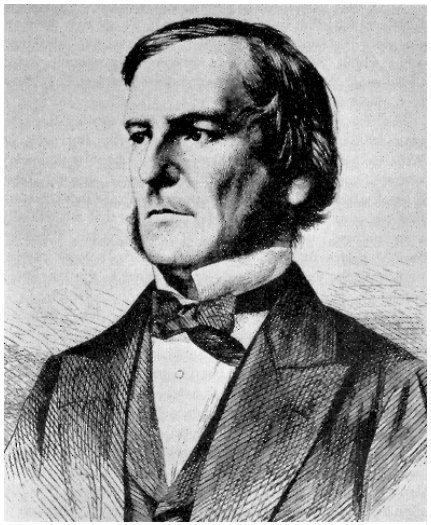
- based on **class algebra**
- x, y, z : **propositions** representing concepts
- $+$, $-$, $\times$: **exclusive disjunction**, **negation**, **conjunction**
- idempotency: p and p is equivalent to p
- that is why you can choose 1 and 0 as they are the solution to the equation $x^2 = x$

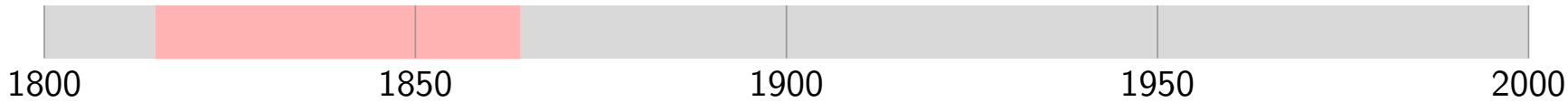


The arising of modern logic

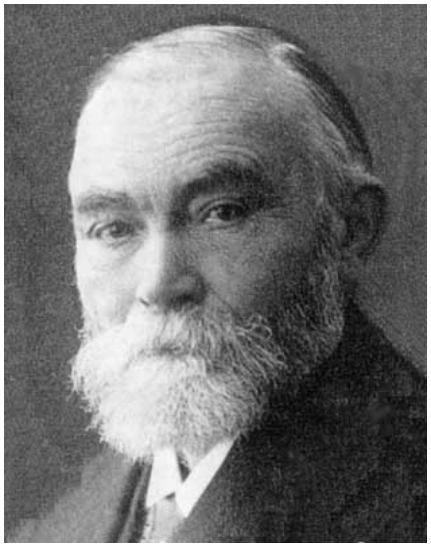
George Boole (1815 - 1864)

- $\neq$ is not algebraic, how to express "some x are y"?
- algebra can be a tool for logic, but logic is about **deductions/inferences**, not **equations**

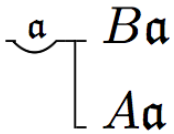




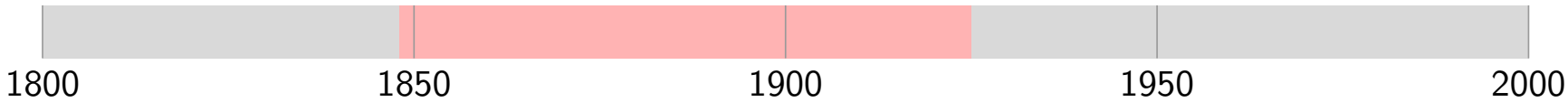
Gottlob Frege (1848 - 1925)



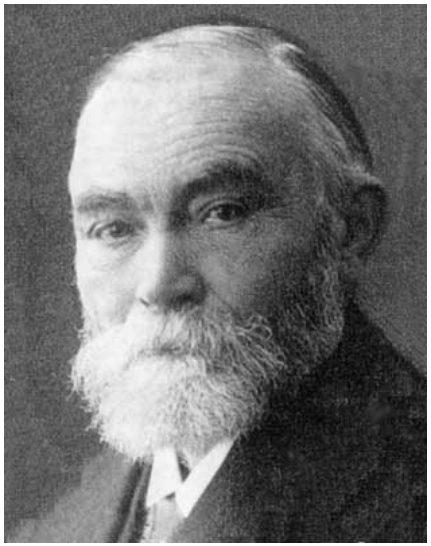
- a formal and symbolic language *Begriffsschrift* (1879)



- worked on foundations of arithmetics as a branch of logic
- *Grundgesetze der Arithmetik*

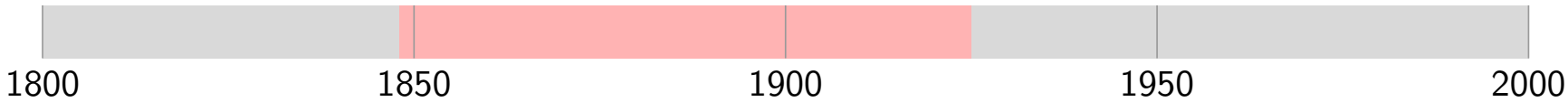


Gottlob Frege (1848 - 1925)



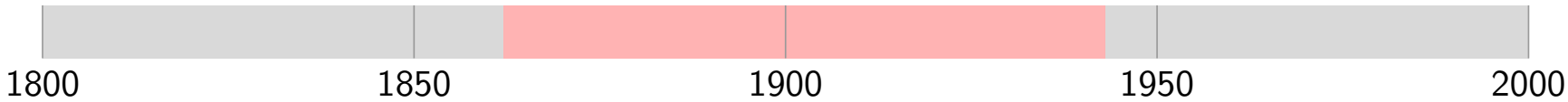
- but there is a paradox, discovered by Russel

  Hardly anything more unfortunate can befall a scientific writer than to have one of the foundations of his edifice shaken after the work is finished. This was the position I was placed in by a letter of Mr. Bertrand Russell, just when the printing of this volume was nearing its completion.

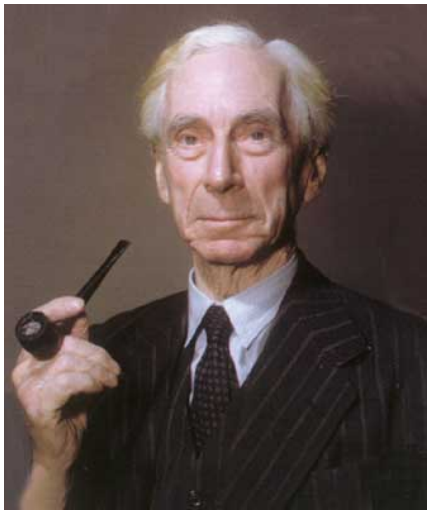


David Hilbert (1862 - 1943)

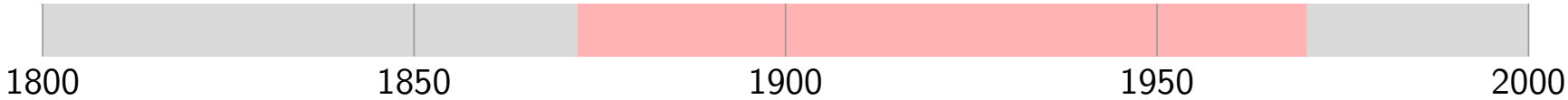
- *Grundlagen der Geometrie* (1899): axiomization of geometry
- the famous 23 problems for the 20^{th} century
- Hilbert's program (1920)
  - all mathematics follows from a set of axioms
  - this set of axioms can be proved to be consistent
- *Grundlagen der Mathematik* (1934/1939)
- a **formal system for deduction**



Bertrand Russel (1872 - 1970)

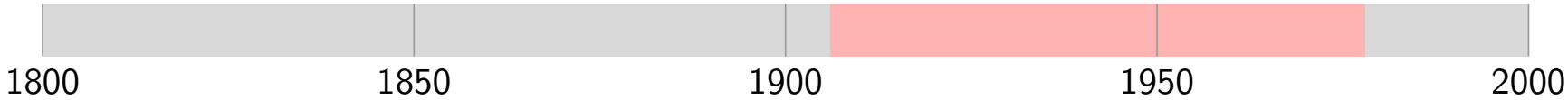


- goal: find solid foundations for mathematics
- *Principles of mathematics* (1903)
  - destroying naive set theory: $E = \{x | x \notin x\}$
  - definition of **type theory**
- *Principia mathematica* (1910)
  - derive all mathematical truths from axioms + inference rules in symbolic logic
  - demonstration of $1 + 1 = 2$ on page 379!



Kurt Gödel (1906 - 1978)

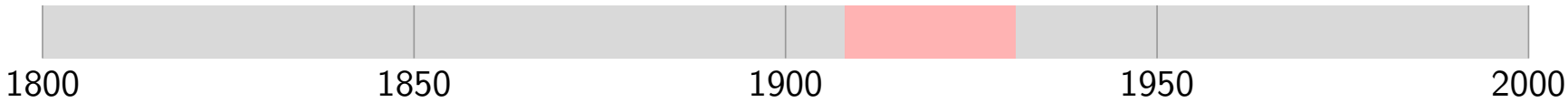
- *Die Vollständigkeit des Axiome des logischen Funktionenkalküls* (1930): completness theorem
- *Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme* (1931): incompletness theorems
  - if arithmetics is consistent, there are statements such that they cannot be proved in arithmetics nor their negations.
  - consistency of arithmetics cannot be proved in arithmetics



The arising of modern logic

Jacques Herbrand (1908 - 1931)

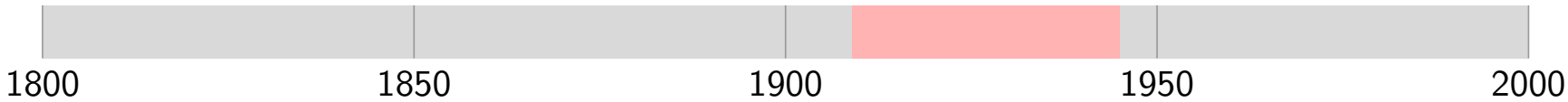
- a fundamental result in logic: reduction of first-order logic to propositional logic
- the first step to automated deduction



The arising of modern logic

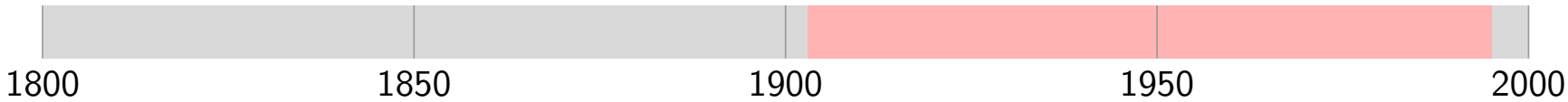
Gerhard Gentzen (1909 - 1945)

- natural deduction
- sequent calculus
- *Hauptsatz* or cut-elimination theorem



Alonzo Church (1903 - 1995)

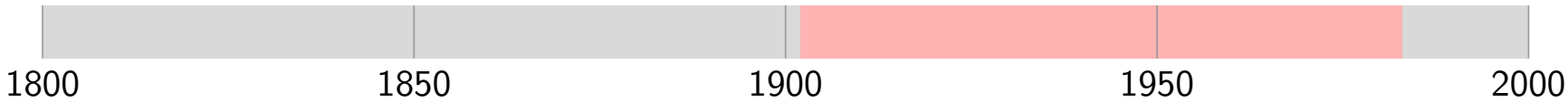
- *Entscheidungsproblem*
- father of the **lambda calculus**
- the Church-Turing thesis
- Church-Rosser theorem



The arising of modern logic

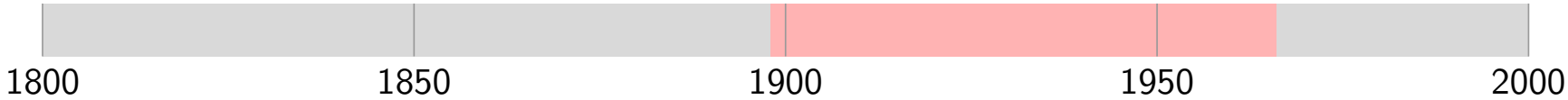
Alfred Tarski (1902 - 1983)

- works on **model theory**: semantics
- the notion of **logical consequence**
- metamathematics as "ordinary" mathematics
- decidability results



Luitzen Brouwer (1898 - 1966)

- a **constructivist** view of proofs
  - a proof must **build** truth, not only **discover** it
  - problem with *tertium non datur* (law of excluded middle)
  - example: prove that there is two irrational numbers a and b such that a^b is rational
- very important for Computer Science



Outline of part 1 - Introduction

Outline of part 1 - Introduction

Argument as a mathematical object?

When verifying mathematically if an argument is valid, two steps are needed:

1. define a **mathematical language** to represent the sentences involved in the argument
   - should not be difficult, as the sentences are **declarative** ...
   - ... but sometimes not so easy (sentences involving time, modalities etc.)

   We then work with:
   - Σ: a set of formulas representing the premises
   - φ: a formula representing the conclusion

Argument as a mathematical object?

When verifying mathematically if an argument is valid, two steps are needed:

2. define a **mathematical relation** $\mathcal{R}$ such that $\Sigma \; \mathcal{R} \; \varphi$ holds iff the argument "Σ therefore φ" is valid.

   Two possible relations:

   - $\models$ called **logical consequence** in model theory
   - $\vdash$ called **deduction** in proof theory

Validity of an argument in model theory

In model theory, verifying that an argument is correct is proving that $\Sigma \models \varphi$ holds.

Principle (validity in model theory)

An argument is valid iff **when premises are true**, then **conclusion is also true**.

Problem: what does mean "a statement is true" ?

Remember that we are working with **declarative** statements, so we can speak about the **truth value** of a statement, e.g.:

sentence	**truth value**
1. this will be the most interesting lecture I've ever had	`true`
2. screw this lecture, I prefer reading the newspaper	`false`
3. will I take pizza at lunch?	✘

Some assumptions on truth values

We will use **classical logic**:

- there are only two truth values: true/false, T/F, 1/0...
- the truth value of a composed statement depends only of the truth values of the statements which compose it

  E.g., the truth value of "A and B" depends on the truth value of A and the truth value of B.

The truth value of a statement is established in an **interpretration** in which we set the truth values of the basic assertions of our language.

For instance, in our world/interpretation:

- "students are intelligent human beings" is true;
- "students have fins" is false.

But we can imagine (particularly in mathematics) an interpretation in which students have fins (or are not intelligent human beings)...

So, how to prove the validity of an argument in MT?

Easy:

1. among **all the possible interpretations**, find those in which all the premises are true (let us call this set of interpretations I)
2. for each interpretation i in I, verify that the conclusion is true in i

. . . but not very efficient for "real-world" problems (we will discuss about this during the lecture).

Validity of an argument in proof theory

OK for model theory, but when you have to prove a lemma/theorem during a math exam, you write all the necessary steps to go from the hypotheses to the conclusion.

The examiner verify that **each step** of your argument uses a **correct reasoning rule**.

This is in fact **proof theory**, and if your argument is valid, it will be denoted by $\Sigma \vdash \varphi$ (φ can be **deduced** from Σ).

In proof theory, we consider:

- some rewritting rules which transform statements into other statements, the **inference rules**
- some assertions called **axioms** which are true "by nature"

Principle (validity in proof theory)

An argument is valid iff **conclusion is obtained from premises and axioms by using inference rules**.

Inference rules: examples

Let us suppose that you have some hypotheses Σ about a function f on $\mathbb{R}$ and you want to prove from Σ that f is $\mathcal{C}^1$.
Thus you should prove that f is continuous and that its derivative exists and is also continuous.

You know that if f is differentiable then f is continuous. So if you prove from Σ that f is differentiable, then f is continuous.

Doing this, you use a **logical inference rule** called Modus Ponens (MP): "from A implies B and A you can deduce B".

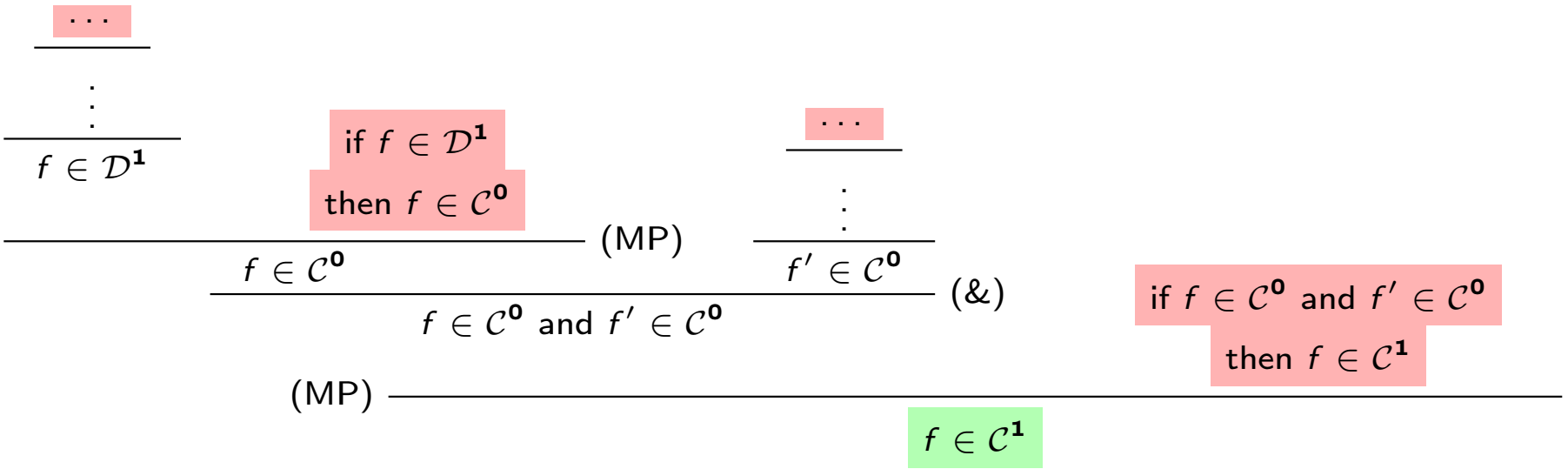
Notice that this rule can be applied to every argument of the same form!

Suppose that you can now prove from Σ that f' is continuous. Then you can use a logical inference rule saying that
"if you prove A and you prove B, then you prove A and B"
to prove that f is $\mathcal{C}^1$.

Our argument in proof theory

An argument in proof theory can be represented by a **tree** whose

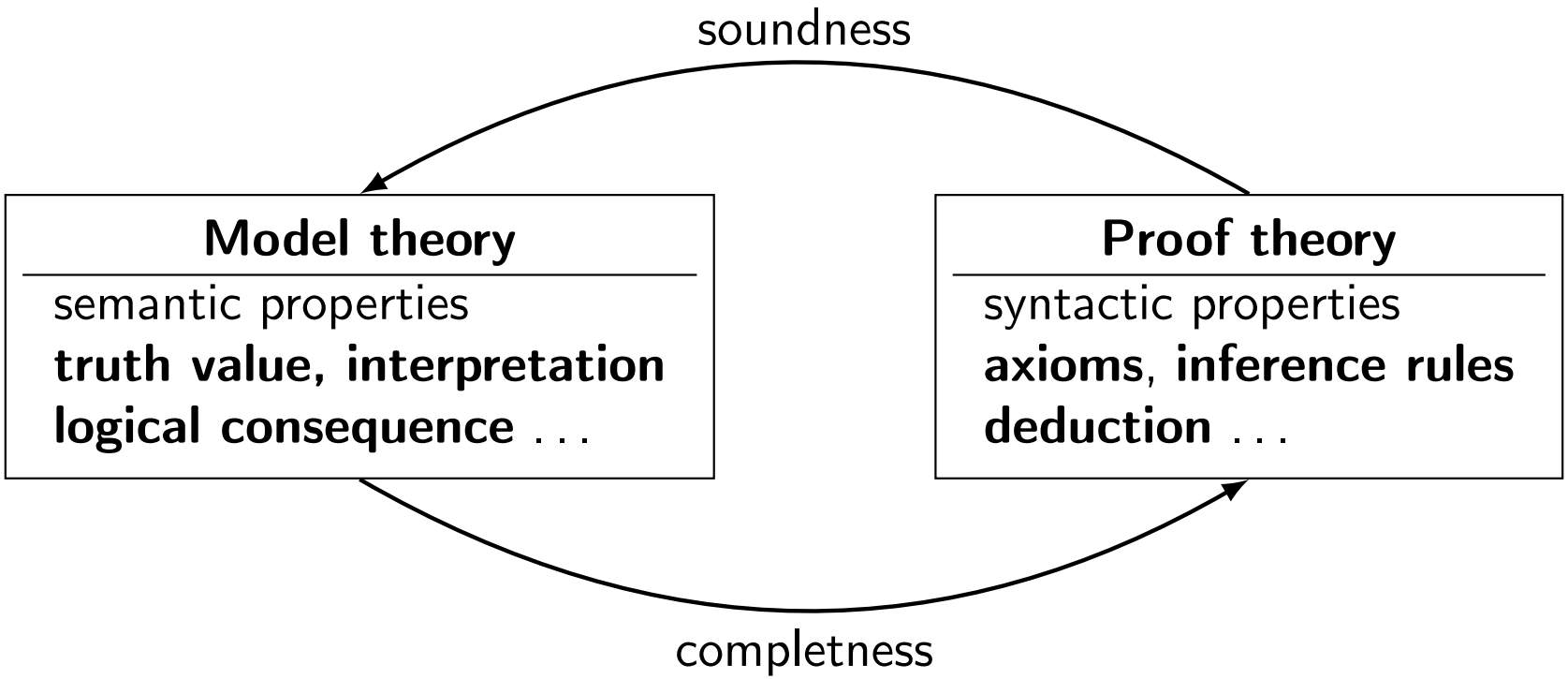
- root is the conclusion of the argument
- leaves are hypotheses or axioms
- "childrening process" is inference rule applying



Links between model and proof theory

Theorem (soundness and completness)

An argument is valid in proof theory iff it is valid in model theory.



Links between model and proof theory

The soudness and completness theorem is the mathematical version of the following statement:

What is provable is true and what is true is provable.

You can use those two tools depending on what you want to show:

what to show	**model theory**	**proof theory**
argument is correct	prove that for every interpretation conclusion is true	find **one** possible proof using inference rules
argument is not correct	exhibit **one** counterexample in which conclusion is false	prove that this is not possible to build a proof using inference rules

Deduction, abduction, induction

Let us define:

- $\mathcal{L}$: a set of formulas representing **rules**
- $\mathcal{H}$: a set of formulas representing **facts**
- φ: a formula representing the **conclusion** of an argument

Definition (deduction)

Find/verify that φ s.t. $\mathcal{L} \cup \mathcal{H} \; \mathcal{R} \; \varphi$.

Definition (abduction)

Find $\mathcal{H}$ s.t. $\mathcal{L} \cup \mathcal{H} \; \mathcal{R} \; \varphi$.

Definition (induction)

Find $\mathcal{L}$ s.t. $\mathcal{L} \cup \mathcal{H} \; \mathcal{R} \; \varphi$.

Outline of part 1 - Introduction

Argument: a first example

Let us consider the following argument:

> *John has travelled by bus or by train. If he has travelled by bus or by car, he has been late and has missed the meeting. He was not late. Therefore he has travelled by train.*

Argument: a first example

Let us consider the following argument:

> *John has travelled by bus or by train. If he has travelled by bus or by car, he has been late and has missed the meeting. He was not late. Therefore he has travelled by train.*

We can represent this argument by using a **propositional language** in which variables represent assertions:

$$\Sigma = \left\{ \begin{array}{l} b \vee t, \\ b \vee c \rightarrow l \wedge m, \\ \neg l \end{array} \right\}$$

$$\varphi \equiv t$$

Model theory: a first example

To prove that the previous argument is correct in model theory, we have to check that in every possible interpretation, when the formulas in Σ are all true then φ is also true.

As we have 5 variables, we have **32 cases** to examine:

t	b	c	l	m	$b \vee t$	$b \vee c \rightarrow l \wedge m$	$\neg l$
T	T	T	T	T	T	T	F
T	T	T	T	F	T	F	F
T	T	T	F	T	T	F	T
T	T	T	F	F	T	F	T
T	T	F	T	T	T	T	F
T	T	F	T	F	T	F	F
T	T	F	F	T	T	F	T
T	T	F	F	F	T	F	T
T	F	T	T	T	T	T	F
T	F	T	T	F	T	F	F
T	F	T	F	T	T	F	T
T	F	T	F	F	T	F	T
T	F	F	T	T	T	T	F
T	F	F	T	F	T	T	F
T	F	F	F	T	T	T	T
T	F	F	F	F	T	T	T
F	T	T	T	T	T	T	F
F	T	T	T	F	T	F	F
F	T	T	F	T	T	F	T
F	T	T	F	F	T	F	T

...

Model theory: a first automated example

OK, we can prove "by hand" that φ is true in every interpretation in which Σ is true, but this is fastidious (and we are doing Computer Science because we are lazy).

We can use a **SAT solver**: a SAT solver is a software which given a set of propositional formulas checks if there is **an** interpretation in which all the formula are true.

Een, Niklas and Niklas Sörensson (2013).
The MiniSat page.
http://minisat.se/.

Model theory: a first automated example

OK, we can prove "by hand" that φ is true in every interpretation in which Σ is true, but this is fastidious (and we are doing Computer Science because we are lazy).

We can use a **SAT solver**: a SAT solver is a software which given a set of propositional formulas checks if there is **an** interpretation in which all the formula are true.

Example with $\{a, a \to b\}$:

minisat-ex.sat

```
p cnf 2 2
1 0
-1 2 0
```

produces the expected result.

Model theory: a first automated example

OK, we can prove "by hand" that φ is true in every interpretation in which Σ is true, but this is fastidious (and we are doing Computer Science because we are lazy).

We can use a **SAT solver**: a SAT solver is a software which given a set of propositional formulas checks if there is **an** interpretation in which all the formula are true.

Example with $\{a, a \to b, \neg b\}$:

minisat-ex-false.sat

```
p cnf 2 3
1 0
-1 2 0
-2 0
```

produces the expected result.

Using SAT on our example

We want to prove that t is true in **all** interpretations in which formulas of Σ are all true. Not the same problem...

Let us look at how **you** can prove t from Σ.

Using SAT on our example

We want to prove that t is true in **all** interpretations in which formulas of Σ are all true. Not the same problem...

Let us look at how **you** can prove t from Σ.

Using SAT on our example

We want to prove that t is true in **all** interpretations in which formulas of Σ are all true. Not the same problem...

Let us look at how **you** can prove t from Σ.

First, let us look at the classical tactics used by students to prove something:

"trivial"	✘ (reserved to prof.)
direct proof	✘ (too painful for their brain)
induction	✘
contraposition	✘
proof by contradiction	hmmm, let's try

Using SAT on our example

We want to prove that t is true in **all** interpretations in which formulas of Σ are all true. Not the same problem...

Let us look at how **you** can prove t from Σ.

Let us suppo

Ah,
that John has taken the bus.
If he has taken the bus, then he has been late. Thus John has been late.
But he has not been late, thus there is a contradiction.
Therefore John has taken the train.

Q.E.D. Bingo!

Back to our SAT problem

We can use the "proof by contradiction" principle with our SAT solver. Let us consider the set

$$\{b \vee t, b \vee c \rightarrow l \wedge m, \neg l, \neg t\}$$

and ask MiniSat if it can find an interpretation for those formulas.

john.sat

```
p cnf 5 7
1 5 0
-1 3 0
-1 4 0
-2 3 0
-2 4 0
-3 0
-5 0
```

And the answer is. . . UNSATISFIABLE!

Now, let's do the "real" proof

Using SAT and model theory, we know that the argument is correct. But can we write a proof that can be accepted as correct by our beloved teacher?

Unfortunately, in most case proof finding is not "mechanical" (otherwise, math. teachers will be unemployed).

But fortunately, in the propositional case, there are systems that can handle the proof, e.g. Gentzen formal system we will see in the lecture.

Unfortunately (again), the form of the proof is not the one you will write by hand.

But if you write the proof in one of those system, a machine can **check** that your proof is correct!

Building and verifying the proof

To verify and build a proof, you can use an **interactive theorem prover** like Coq to prove the theorem.

INRIA (2013a).
The Coq proof assistant.
http://coq.inria.fr/.

Coq allows you to write proofs using tactics and when the proof is established it can be verified. You "just have" to trust the Coq kernel.

Building and verifying the proof

john.v

```
Variable T : Prop.
Variable B : Prop.
Variable C : Prop.
Variable L : Prop.
Variable M : Prop.

Proposition john_train : (B \/ T) /\ (B \/ C -> L /\ M) /\ (~ L) -> T.
Proof.
  intro Hyp_gen.
  destruct Hyp_gen as [Hbus_or_train H1].
  destruct H1 as [Hbus_or_car_implies Hnot_late].
  destruct Hbus_or_train as [ Hbus | Htrain ].
    destruct Hbus_or_car_implies as [ Hlate Hmiss].
      left. exact Hbus.
      absurd L.
        exact Hnot_late.
        exact Hlate.
  exact Htrain.
Qed.
```

Nice, isn't it?

An automatic proof!

Question: is there an algorithm to prove that there is a proof of our assertion?

This field of study is called **automated theorem proving** and is strongly related to SAT. Some theorem provers:

- The E Theorem Prover (`http://www.eprover.org`)
- Vampire (`http://www.vprover.org`)
- SPASS (`http://www.spass-prover.org`)

An automatic proof!

Let us try SPASS on our problem.

The SPASS team (2014).
SPASS: An Automated Theorem Prover for First-Order Logic with Equality.
http://www.spass-prover.org.

An automatic proof!

Let us try SPASS on our problem.

john.spass

```
begin_problem(john_train).

list_of_descriptions.
  name({*The %!?## problem of John and his train*}).
  author({*Christophe Garion*}).
  status(satisfiable).
  description({*Prove (B \/ T) /\ (B \/ C -> L /\ M) /\ (~ L) -> T...*}).
end_of_list.

list_of_symbols.
  predicates[(B,0), (C,0), (L,0), (M,0), (T,0)].
end_of_list.

list_of_formulae(conjectures).
  formula(implies(and(and(or(B, T), implies(or(B, C), and(L, M))),
                  not(L)), T)).
end_of_list.

end_problem.
```

A last remark: formal logic?

The proof we have done here is a **formal** proof, i.e. it does not depend on the "natural language semantics" of the initial sentences.
For instance, you will can reuse **exactly the same proof** for the following argument:

> *John has drunk orange juice or tequila at the party. If he has drunk tequila or whiskey at the party, he has missed the logic lecture and he has lost his ID card. He has not missed the logic lecture. Therefore John has drunk orange juice at the party.*

Argument: a second example

Let us consider the following argument:

> *Let $\mathcal{G} = \{E, \times\}$ be a group. If each element x of the group is its own inverse, then $\mathcal{G}$ is commutative.*

Let us try to model it with propositional logic:

$$\Sigma = \left\{ \begin{array}{ll} g, & (\mathcal{G} \text{ is a group}) \\ i & (\text{inverse property}) \end{array} \right\}$$

$$\varphi \equiv com \quad (\text{commutativity property})$$

Argument: a second example

Let us consider the following argument:

> *Let $\mathcal{G} = \{E, \times\}$ be a group. If each element x of the group is its own inverse, then $\mathcal{G}$ is commutative.*

OK, maybe it is because there are implicit hypotheses:

$$\Sigma = \left\{ \begin{array}{ll} c, & \text{(closure of } \mathcal{G}) \\ a, & \text{(associativity of } \times) \\ id, & \text{(identity)} \\ in, & \text{(invertibility)} \\ i & \text{(inverse property)} \end{array} \right\}$$

$$\varphi \equiv com \quad \text{(commutativity property)}$$

But we cannot prove this in propositional logic, as there is no link between the variables!

A more expressive language

In fact, we need **a more expressive language**. For instance, in "classical" maths, you will express the closure property as follows:

$$\forall (x, y) \in E^2,\ x \times y \in E$$

You need **quantifiers** ($\forall$, $\exists$), **predicates** representing properties (x belongs to E, equality, ...).

This is in fact **First-Order Logic** (First-Order as we can quantify on variables).

A more expressive language

In fact, we need **a more expressive language**. For instance, in "classical" maths, you will express the closure property as follows:

$$\forall (x, y) \in E^2,\ x \times y \in E$$

The formalization of the argument in FOL is the following:

$$\Sigma = \left\{ \begin{array}{l} \forall x \forall y\ E(x) \wedge E(y) \rightarrow E(x \times y) \\ \forall x \forall y \forall z\ E(x) \wedge E(y) \wedge E(y) \rightarrow x \times (y \times z) = (x \times y) \times z \\ \exists e\ E(e) \wedge (\forall x\ E(x) \rightarrow (x \times e = e) \wedge (e \times x = e)) \\ \forall x\ E(x) \rightarrow (\exists y \exists z\ E(y) \wedge E(z) \wedge x \times y = e \wedge z \times x = e) \\ \forall x\ E(x) \rightarrow x \times x = e \end{array} \right\}$$

$$\varphi \equiv \forall x \forall y\ E(x) \wedge E(y) \rightarrow x \times y = y \times x$$

Notice that we will not use "typed" logic, thus we cannot use the $\in$ symbol and use a E predicate to represent it.
Notice also that you have to define laws for equality...

Model theory on our example

We cannot use SAT solvers on our argument as we have quantifiers and predicates. But we can try using a **SMT** solver to verify if the argument is correct.

Definition (informal)

A **Satisfiability Modulo Theory** (**SMT**) problem is a decision problem based on SAT where the interpretation of some symbols is constrained by a background theory.

Model theory on our example

We cannot use SAT solvers on our argument as we have quantifiers and predicates. But we can try using a **SMT** solver to verify if the argument is correct.

Barrett, Clark et al. (2009).
"Satisfiability Modulo Theories".
In:
Handbook of Satisfiability.
Ed. by Armin Biere et al.
Vol. 185.
Frontiers in Artificial Intelligence and Applications.
IOS Press.
Chap. 26, pp. 825–885.
ISBN: 978-1-58603-929-5.

Model theory on our example

We cannot use SAT solvers on our argument as we have quantifiers and predicates. But we can try using a **SMT** solver to verify if the argument is correct.

Let us try with the Alt-Ergo SMT solver.

Conchon, Sylvain and Evelyne Contejean (2013).
Alt-Ergo, an OCaml SMT-solver for software verification.
http://alt-ergo.lri.fr/.

Model theory on our example

We cannot use SAT solvers on our argument as we have quantifiers and predicates. But we can try using a **SMT** solver to verify if the argument is correct.

group.mlw

```
type E

logic times: E, E -> E
logic e: E
logic inv: E -> E

axiom neutral: forall x:E. times(x, e) = e and times(e, x) = e
axiom associativity:
      forall x,y,z:E. times(x, times(y, z)) = times(times(x, y), z)
axiom inverse:
      forall x:E. (times(x, inv(x)) = e) and (times(inv(x), x) = e)
axiom own_inverse: forall x:E. inv(x) = x

goal commutativity: forall x,y:E. times(x, y) = times(y, x)
```

And the answer is. . .

Our theorem: not provable by a computer?

We cannot use a SMT automated prover to prove our theorem, but we can use an interactive theorem prover like Coq to prove the theorem.

We will not use Coq during the lecture and only present automatic reasoning, but you can check the following reference.

Johanssons, Mikael (2007).
Coq and simple group theory.
http://blog.mikael.johanssons.org/archive/2007/08/coq-and-simple-group-theory/.

An automatic proof, again with SPASS!

group.spass (not complete!)

```
begin_problem(group).

list_of_descriptions.
  name({*The problem of group inverses*}).
  author({*Christophe Garion*}).
  status(satisfiable).
  description({*Prove that a group such that every element of the group
                is its own inverse is commutative...*}).
end_of_list.

list_of_symbols.
  functions[(e,0), (times,2), (inv,1)].
  sorts[E].
end_of_list.

list_of_formulae(axioms).
  formula(forall([E(x), E(y)], E(times(x, y)))).
  formula(forall([E(x), E(y), E(z)], equal(times(x, times(y, z)),
                                           times(times(x, y), z)))).
```

So, can we prove difficult theorems using logic?

Theorem (four color)

Given any separation of plan intro contiguous regions (a figure called a map), no more than four colors are required to color the regions of the map so that no two adjacent regions have the same color.

The first theorem to be proved using a computer in 1986 by Appel and Hakken: 1200 hours of calculation!

Appel, K. and W. Haken (1989).
"Every planar map is four colourable".
In: **Contemporary Mathematics** 98.

Not accepted by some mathematicians: no human can verify the proof by hand...

So, can we prove difficult theorems using logic?

Theorem (four color)

Given any separation of plan intro contiguous regions (a figure called a map), no more than four colors are required to color the regions of the map so that no two adjacent regions have the same color.

. . . but an accepted proof has been formalized by Georges Gonthier and Benjamin Werner in 2005 in Coq.

Gonthier, Georges (2008).
"Formal proof – the four-color theorem".
In: **Notices of the American Mathematical Society** 55.11.

Another theorem verified by Coq

The Feit-Thompson theorem is a complicated theorem about groups. Its original proof is a **book of 250 pages**!

Even if mathematicians are strongly confident in the validity of the proof, how can a human check that there is no error?

Again, Georges Gonthier and his team have proven the theorem in Coq in 2012 (see have `http://www.msr-inria.fr/news/the-formalization-of-the-odd-order-theorem-has-been-completed-the`

- 6 years of work
- 170000 lines of code
- 4200 definitions
- 15000 theorems

Outline of part 1 - Introduction

Decision problem

Automated reasoning: find algorithms which allow to **verify** if a statement can be deduced from premises.

We will show that this question can be reduced to the problem of verifying the **validity of a formula**, i.e. find if a **formula is always true**.

The problem is thus to be able to verify if a given formula is in the set of valid formulas.

Definition (decision problem)

A **decision problem** is a question on a formal system whose possible answers are yes and no.

Consequence

The problem of finding if an element is in a set or not is a decision problem.

Decidability

Definition (decidability)

A decision problem is **decidable** iff there is an **algorithm** which can anwser the decision problem in a **finite number of steps**.

Definition (semidecidability)

A decision problem is **semidecidable** iff there is an algorithm which:

- halts and returns yes if the answer to the problem is yes
- halts and returns no or **never halts** if the answer to the problem is no

And for our logic languages?

Theorem (decidability of propositional logic)

The problem of finding if a propositional formula is valid or not is decidable.

Theorem (semidecidability of first-order logic)

The problem of finding if a first-order logic formula is valid or not is semidecidable.

Decidability: some examples

Let us consider the following sets:

- arithmetics expressions
- the valid formulas in the FO theory of the integers with $=$, $+$
- the valid formulas in the FO theory of the integers with $=$, $+$, $\times$
- the valid formulas in the FO theory of the real closed fields
- the valid formulas in the FO theory of the groups

Which ones are decidable?

Decidability: some examples

Let us consider the following sets:

- arithmetics expressions ✔ (trivial)
- the valid formulas in the FO theory of the integers with $=$, $+$ ✔ (Pressburger)
- the valid formulas in the FO theory of the integers with $=$, $+$, $\times$ ✘ (Tarski)
- the valid formulas in the FO theory of the real closed fields ✔ (Tarski)
- the valid formulas in the FO theory of the groups ✘ (Tarski)

Which ones are decidable?

Outline of part 1 - Introduction

Mathematical logic for Computer Science?

Foundations of Computer Science:

- links between lambda-calculus and proofs (Curry-Howard theorem)
- algorithms and problems complexity

Programming languages:

- formal semantics, type theory
- logic programming, relational algebra and SQL

Software engineering:

- formal specifications
- verification and validation

Artificial intelligence:

- automatic deduction
- modelling of notions such as belief, knowledge, intention,...

Mathematical logic in CS: example

Exercise

Prove that the following C program is correct.

foo.c

```
void foo(int *a, int *b) {
  int tmp = *a ;
  *a = *b ;
  *b = tmp ;
  return ;
}
```

Questions

- what is **correctness** for computer programs?
- is there a **logic for programs**?

Mathematical logic in CS: example

Exercise

Prove that the following C program is correct.

foo.c

```
void foo(int *a, int *b) {
  int tmp = *a ;
  *a = *b ;
  *b = tmp ;
  return ;
}
```

Hoare, C. A. R. (1969).
"An axiomatic basis for computer programming".
In: **Communications of the ACM** 12.10,
Pp. 576–580.

Mathematical logic in CS: example

OK, so you have proved your C program using Floyd-Hoare logic and its extensions...
...but how can you be sure that your favorite compiler (gcc) will produce an executable file with the same semantics? ☹

INRIA (2013b).
CompCert.
http://compcert.inria.fr/.

Proven using the Coq proof assistant.

Are those methods really useful?

Gouw, Stijn de (2015).
Proving that Android's, Java's and Python's sorting algorithm is broken (and showing how to fix it).
http://envisage-project.eu/proving-android-java-and-python-sorting-algorithm-is-broken-and-how-to-fix-it/.

When trying to prove TimSort, the authors found an important bug.
In practise, there are small "chances" to have a dataset conforming to the bug, but the bug can be exploited in an attack.

Outline of part 1 - Introduction

References

Huth, Michael and Mark Ryan (2004).
Logic in Computer Science – Modelling and reasoning about systems.
Cambridge University Press.

Kleene, Stephen Cole (1967).
Mathematical logic.
Dover Publications.

Chang, Chin-Liang and Richard Char-Tung Lee (1973).
Symbolic logic and mechanical theorem proving.
Academic Press.

Cerrito, Serenella (2008).
Logique pour l'informatique – Introduction à la déduction automatique.
In French.
Vuibert.

Outline of part 1 - Introduction

Objectives of the course

Objective 1

Understand mathematical logic as a tool for verifying validity of arguments.
Understand both model and proof theories.

Objective 2

Present applications for mathematical logic: program verification, formal number theory etc.

Objective 3

Discover a new programming paradigm, logic programming.

Objective 4

Have fun with maths ☺

Agenda and evaluation

1	introduction - general concepts	CG
2-3	propositional logic semantics	CG
4-5	formal systems for PL	CG
6	resolution strategies for PL	CG
7-8	FOL language and semantics	CG
9-10	formal systems for FOL	CG
11	formal number theory	CG
	program verification	
12	logic programming	CG
13	lab session on Prolog	CG
14-15	mini-project on Prolog	CG
16	exam	

Evaluation: 1h15 exam (75%) + mini-project (25%)

All material are available on
`http://www.tofgarion.net/lectures/IN112`

2 - Propositional logic language and semantics

5. **Propositional language $\mathcal{L}_{PL}$**
6. **Classical propositional logic semantics**
7. **Technics and algorithms for validity**

Outline of part 2 - PL language and semantics

5. **Propositional language $\mathcal{L}_{PL}$**
   - Defining a formal language
   - Alphabet of $\mathcal{L}_{PL}$
   - Definition of $\mathcal{L}_{PL}$
6. **Classical propositional logic semantics**
7. **Technics and algorithms for validity**

Outline of part 2 - PL language and semantics

How to define a formal language?

We want to define formally formulas of $\mathcal{L}_{PL}$, but we have to **define formulas using formulas**.

For instance, if φ and ψ are formulas, then $\varphi \wedge \psi$ is a formula.

In mathematics, in such case we use an **inductive definition**.

Definition (inductive or recursive definition)

An **inductive definition** of a set E is composed of:

- a **base case** of the definition which defines elementary elements of E
- an **inductive clause** of the definition which defines elements of E using other elements of E defined with a **finite number of steps** n and **operations**
- an **extremal clause** that says that E is the **smallest set** built using the base case and the inductive clause.

Some definitions by induction

Exercise

Define $\mathbb{N}$ by induction.

Exercise

Define binary trees by induction.

Structural induction

Given a set E defined inductively, we can prove properties on elements of E using **structural induction**.

Definition (structural induction)

Let E be a set defined inductively and $\mathcal{P}$ a property on elements of E to be proved. If:

- $\mathcal{P}$ can be proved to be true on each base case
- if we suppose that $\mathcal{P}$ is true on elements built with n steps then $\mathcal{P}$ is true on elements that can be built with $n+1$ steps

then $\mathcal{P}$ is true for every element of E.

A proof by strucural induction

Exercise

Prove the following property of binary trees: "the number n of nodes in a binary tree of height h is at least $n = h$ and at most $n = 2^h - 1$ where h is the depth of the tree".

Outline of part 2 - PL language and semantics

Alphabet

The language of propositional logic is denoted here by $\mathcal{L}_{PL}$.

As for every language, we must first define an **alphabet** for $\mathcal{L}_{PL}$.

Alphabet

The language of propositional logic is denoted here by $\mathcal{L}_{PL}$.

Definition (alphabet of $\mathcal{L}_{PL}$)

The alphabet of $\mathcal{L}_{PL}$ is composed of:

- an infinite and enumerable set of **propositional variables** noted $Var = \{p, q, r, \ldots\}$
- **logical connectors**:

	meaning	**arity**
$\top$	top/true	0
$\bot$	bottom/false	0
$\neg$	negation	1
$\vee$	or/disjunction	2
$\wedge$	and/conjunction	2
$\rightarrow$	implication	2
$\leftrightarrow$	logical equivalence	2

- **parentheses** ()

Signature of a propositional language

In the alphabet, $\top$, $\bot$, $\neg$, $\vee$, $\rightarrow$, $\leftrightarrow$, (and) are called **logical symbols** because their logical meaning is already defined.

On the contrary, *Var* depends on the problem to be modelled and thus the propositional variables are called **non-logical symbols**. It is also called the **signature** of the language.

So, when you want to model a problem using $\mathcal{L}_{PL}$, you first have to define the signature of your language, i.e. the propositional variables you will use.

Outline of part 2 - PL language and semantics

Definition of $\mathcal{L}_{PL}$

$\mathcal{L}_{PL}$ is defined **inductively** with **well formed formulas** (wffs).

Definition (well formed formulas)

- if p is a propositional variable, then p is a wff. p is an **atomic formula** or **atom**.
- $\top$ and $\bot$ are wff.
- if φ is a wff, alors $(\neg\varphi)$ is a wff.
- if φ and ψ are wff, then $(\varphi \vee \psi)$, $(\varphi \wedge \psi)$, $(\varphi \rightarrow \psi)$ and $(\varphi \leftrightarrow \psi)$ are wff.

Modelling exercise

Exercise

Use propositional language to model the following declarative sentences.

1. it is raining and it is cold.
2. if he eats too much, he will be sick.
3. it is sunny but it is cold.
4. if it is cold, I take my jacket.
5. I take either a jacket, either an umbrella.
6. it is not raining.
7. in autumn, if it is cold then I take a jacket.
8. in winter, I take a jacket only if it is cold.
9. if Peter does not forget to book tickets, we will go to theater.
10. if Peter does not forget to book tickets and if we find a baby-sitter, we will go to theater.
11. he went, although it was very hot, but he forgot his water bottle.
12. when I am nervous, I practise yoga or relaxation. Someone practising yoga also practises relaxation. So when I do not practise relaxation, I am calm.
13. my sister wants a black and white cat.

Syntax tree of PL wff

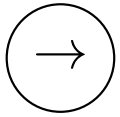
Every wff can be represented by a **syntax tree**, which is a **finite binary tree**.

Definition (syntax tree)

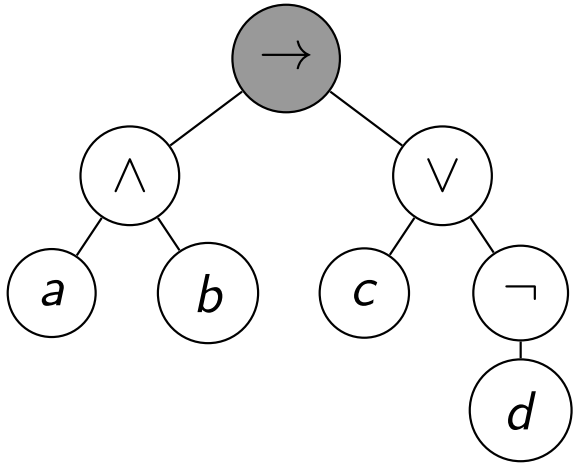
Let φ be a wff. The **syntax tree** of φ is defined inductively as follows:

- if φ is an atomic formula, then $\mathcal{ST}(\varphi) = \langle \emptyset, \varphi, \emptyset \rangle$
- if $\varphi \equiv \neg\varphi_1$, then $\mathcal{ST}(\varphi) = \langle \mathcal{ST}(\varphi_1), \neg, \emptyset \rangle$
- if $\varphi \equiv \varphi_1 * \varphi_2$ where $*$ is a binary connector, then $\mathcal{ST}(\varphi) = \langle \mathcal{ST}(\varphi_1), *, \mathcal{ST}(\varphi_2) \rangle$

Example for $((a \wedge b) \rightarrow (c \vee \neg d))$:

The **main connector**: 

The tree **structure is determined by parenthesis** in the wff.



Some conventions

To simplify the writing, some conventions can be used:

- removing of external parentheses: $(a \wedge b) \rightsquigarrow a \wedge b$
- $\neg$ is written without parentheses: $(\neg a) \rightsquigarrow \neg a$
- connectors are associative from left to right: $((a \wedge b) \wedge c) \rightsquigarrow a \wedge b \wedge c$

Connectors can be ordered by growing priority:

$$\leftrightarrow \quad \rightarrow \quad \vee \quad \wedge \quad \neg$$

For instance, we will write $p \wedge q \rightarrow r \vee s$ instead of $(p \wedge q) \rightarrow (r \vee s)$.

Some definitions for later

Definition (literal)

- if $p \in Var$ then p is a literal
- if $p \in Var$ then $\neg p$ is a literal

Definition (subformulas)

The subformulas set of φ denoted by $sf(\varphi)$ is defined inductively as follows:

- if φ is an atomic formula p, then $sf(\varphi) = \{p\}$
- $sf(\neg\varphi) = \{\neg\varphi\} \cup sf(\varphi)$
- if *conn* is a binary connector,
  $sf(\varphi_1 \ conn \ \varphi_2) = \{\varphi_1 \ conn \ \varphi_2\} \cup sf(\varphi_1) \cup sf(\varphi_2)$

Notice that in PL, the number of subformulas of a wff φ is **finite**.

Outline of part 2 - PL language and semantics

5. Propositional language $\mathcal{L}_{PL}$

6. **Classical propositional logic semantics**
   - Boolean functions
   - Interpretations
   - Satisfiability and logical consequence
   - Useful lemmas and theorems

7. Technics and algorithms for validity

Outline of part 2 - PL language and semantics

Boolean functions

Let $\mathcal{B}$ be the set $\{T, F\}$.

Definition (boolean function)

A **boolean function** with n variables is a function $f : \mathcal{B}^n \to \mathcal{B}$.

The unary boolean function $f_\neg$ is defined as follows:

- $f_\neg(T) = F$
- $f_\neg(F) = T$

The binary boolean function $f_\vee$ is defined as follows:

- $f_\vee(T, T) = T$
- $f_\vee(F, T) = T$
- $f_\vee(T, F) = T$
- $f_\vee(F, F) = F$

Boolean functions

The binary boolean function $f_\wedge$ is defined as follows:

- $f_\wedge(T, T) = T$
- $f_\wedge(F, T) = F$
- $f_\wedge(T, F) = F$
- $f_\wedge(F, F) = F$

The binary boolean function $f_\rightarrow$ is defined as follows:

- $f_\rightarrow(T, T) = T$
- $f_\rightarrow(F, T) = T$
- $f_\rightarrow(T, F) = F$
- $f_\rightarrow(F, F) = T$

The binary boolean function $f_\leftrightarrow$ is defined as follows:

- $f_\leftrightarrow(T, T) = T$
- $f_\leftrightarrow(F, T) = F$
- $f_\leftrightarrow(T, F) = F$
- $f_\leftrightarrow(F, F) = T$

Minimal connectors set

There are in fact **16 boolean functions** corresponding to connectors and some of them are related.

For instance, you can see that $f_{\rightarrow}(x, y) = f_{\vee}(f_{\neg}(x), y)$, i.e. that $a \rightarrow b$ is logically equivalent to $\neg a \vee b$.

Thus, we can define boolean functions only for a **minimal connectors set** and define the other connectors from them.

For instance, the following sets are minimal for classical propositional logic:

$$\{\neg, \vee\}, \{\rightarrow, \bot\}, \{\uparrow\}, \{\wedge, \leftrightarrow, \bot\}, \ldots$$

$\{\neg, \vee\}$ and $\{\rightarrow, \bot\}$ are often chosen.

Outline of part 2 - PL language and semantics

Interpretation and truth value

Definition (propositional interpretation)

An **interpretation** for a set of propositional variables Var is an always defined function $\mathcal{I} : Var \to \{T, F\}$.

Definition (truth value)

The **truth value** of a wff φ under an interpretation $\mathcal{I}$, denoted by $[\![\varphi]\!]_{\mathcal{I}}$ is defined by:

- $[\![p]\!]_{\mathcal{I}} = \mathcal{I}(p)$ iff $p \in Var$
- $[\![\top]\!]_{\mathcal{I}} = T$ and $[\![\bot]\!]_{\mathcal{I}} = F$
- $[\![\neg\varphi]\!]_{\mathcal{I}} = f_{\neg}([\![\varphi]\!]_{\mathcal{I}})$
- $[\![\varphi \vee \psi]\!]_{\mathcal{I}} = f_{\vee}([\![\varphi]\!]_{\mathcal{I}}, [\![\psi]\!]_{\mathcal{I}})$
- $[\![\varphi \wedge \psi]\!]_{\mathcal{I}} = f_{\wedge}([\![\varphi]\!]_{\mathcal{I}}, [\![\psi]\!]_{\mathcal{I}})$
- $[\![\varphi \to \psi]\!]_{\mathcal{I}} = f_{\to}([\![\varphi]\!]_{\mathcal{I}}, [\![\psi]\!]_{\mathcal{I}})$
- $[\![\varphi \leftrightarrow \psi]\!]_{\mathcal{I}} = f_{\leftrightarrow}([\![\varphi]\!]_{\mathcal{I}}, [\![\psi]\!]_{\mathcal{I}})$

Interpretation: example

Let p be a propositional variable representing the statement "John is physically attending the lecture".

Let i be a propositional variable representing the statement "John is intellectually attending the lecture".

1. 10/11/13 at 16:00, I notice that John is here and is asking interesting questions.

$$\mathcal{I}_1 : \quad \begin{array}{rl} p & \mapsto T \\ i & \mapsto T \end{array} \qquad [\![p \wedge i]\!]_{\mathcal{I}_1} = T$$

2. 10/15/13 at 08:00, I notice that John is here and is sleeping.

$$\mathcal{I}_2 : \quad \begin{array}{rl} p & \mapsto T \\ i & \mapsto F \end{array} \qquad [\![p \wedge i]\!]_{\mathcal{I}_2} = F$$

Outline of part 2 - PL language and semantics

Satisfiability: definitions

Definition (satisfiability)

An intepretation $\mathcal{I}$ **satisfies** a wff φ iff $[\![\varphi]\!]_{\mathcal{I}} = T$.

$\mathcal{I}$ is said to be a **model** of φ denoted by note $\models_{\mathcal{I}} \varphi$.

Let Σ be a set of propositional formulas. $\mathcal{I}$ is a model of Σ iff $\mathcal{I}$ is a model for each formula in Σ.

- a wff φ is **satisfiable** iff there is an interpretation $\mathcal{I}$ such that $\models_{\mathcal{I}} \varphi$.
- a wff φ is a **tautology** iff for every interpretation $\mathcal{I}$, $\models_{\mathcal{I}} \varphi$.
  φ is also said to be **valid**.
  This is noted $\models \varphi$.
- a wff φ is an **antilogy** or **contradiction** iff for every interpretation $\mathcal{I}$, $\not\models_{\mathcal{I}} \varphi$ (i.e. $[\![\varphi]\!]_{\mathcal{I}} = F$).
- a wff φ is **neutral** iff it is neither a tautology nor an antilogy.

Some tautologies...

$\models a \to (b \to a)$
$\models (a \to (b \to c)) \to ((a \to b) \to (a \to c))$
$\models (\neg b \to \neg a) \to ((\neg b \to a) \to b)$
$\models a \to a$
$\models \neg a \to (a \to b)$
$\models \neg\neg a \to a$
$\models (a \to \neg b) \to (b \to \neg a)$
$\models (a \to b) \to (\neg b \to \neg a)$
$\models (a \wedge b) \to a$
$\models a \to (a \vee b)$

Logical consequence

Definition (logical consequence)

Let φ and ψ be two wffs. ψ is a **logical consequence** of φ, denoted by $\varphi \models \psi$, iff for every interpretation $\mathcal{I}$ such that $[\![\varphi]\!]_{\mathcal{I}} = T$, then $[\![\psi]\!]_{\mathcal{I}} = T$.

Definition (logical consequence of a set)

Let $n \in \mathbb{N}^*$, $\Sigma = \{\varphi_1, \ldots, \varphi_n\}$ be a set of wffs and ψ be a wff. ψ is a **logical consequence** of Σ iff for every interpretation $\mathcal{I}$ such that $[\![\varphi_1 \wedge \ldots \wedge \varphi_n]\!]_{\mathcal{I}} = T$, then $[\![\psi]\!]_{\mathcal{I}} = T$.

This noted $\Sigma \models \varphi$.

Logical consequence

N.B. (important)

Verifying that "from Σ we can deduce φ" is proving $\Sigma \models \varphi$ in model theory!

Outline of part 2 - PL language and semantics

Replacement theorem

Theorem (replacement)

Let φ be a wff written with the variables $p_1, \ldots, p_n$. Let φ' be the wff obtained by ***replacing variables*** *$p_1, \ldots, p_n$ in φ by the wffs $\varphi_1, \ldots, \varphi_n$. Then $\models \varphi \Rightarrow \models \varphi'$.*

Example: $\models p \vee \neg p$ implies $\models ((p \wedge q) \rightarrow r) \vee \neg((p \wedge q) \rightarrow r)$

Consequence

This allows us to generate an infinite set of tautologies from the previous ones. . .

Substitution theorem

Definition (substitution)

Let ψ be a subformula of a wff φ. Let ψ' be a wff. Then $\varphi[\psi/\psi']$ denotes the formula obtained by replacing the occurrences of ψ in φ by ψ'.

Theorem (substitution)

Le ψ be a subformula of a wff φ. Let ψ' be a formula. If $\models \psi \leftrightarrow \psi'$, then $\models \varphi \leftrightarrow \varphi[\psi/\psi']$.

Thus logical equivalence preserves validity.

Lemmas

Lemma (monotonicity of $\models$ for PL)

If $\Sigma \models \psi$, then for every wff φ $\Sigma \cup \{\varphi\} \models \psi$.

Lemma

If $\models \varphi$, then for every set of wffs Σ $\Sigma \models \varphi$.

Lemma (ex falso quodlibet)

ψ is a contradiction iff for every wff φ, $\psi \models \varphi$.

Lemma (important for Resolution principle!)

If $\Sigma \models \varphi$, then $\Sigma \cup \{\neg\varphi\}$ is a contradiction.

Compactness of propositional logic

Theorem (compactness of PL)

Let Σ be a (possibly infinite) set of wffs. Then Σ is satisfiable iff every ***finite*** *subset of Σ is satisfiable.*

This is useful if you want to prove that a set Σ is **unsatisfiable**: it is sufficient to find a finite subset of Σ that is unsatisfiable.

We will use this theorem with the Resolution formal system.

Deduction theorem

Theorem (deduction)

$\varphi \models \psi$ *iff* $\models \varphi \to \psi$

$\varphi_1, \ldots, \varphi_n \models \psi$ *iff* $\varphi_1, \ldots, \varphi_{n-1} \models \varphi_n \to \psi$

N.B.

This theorem is **important**: it allows to transform the "validity of an argument" problem to the "validity of a formula" problem.
Verifying that "from $\{\varphi_1, \ldots, \varphi_n\}$ we can deduce ψ" is proving $\models \varphi_1 \to (\ldots\ldots(\varphi_n \to \psi)\ldots)$.

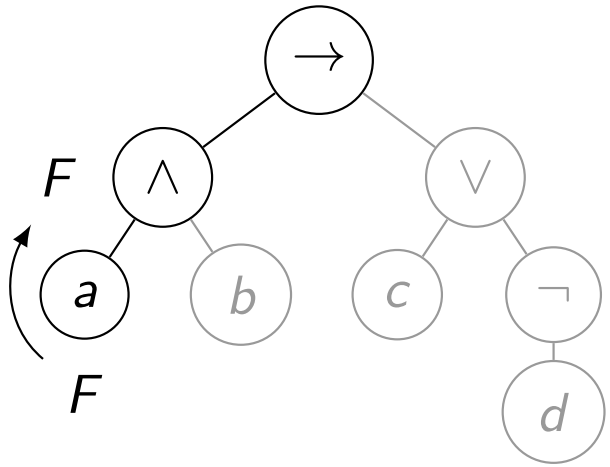
Outline of part 2 - PL language and semantics

Outline of part 2 - PL language and semantics

Formula evaluation using syntax tree

When evaluating a formula in an interpretation $\mathcal{I}$, you can apply strategies to prune the evaluation tree.
For instance, if the first parameter of $f_\wedge$ is F, then it is not necessary to evaluate the second parameter: the result is F.

For instance, compute $[\![a \wedge b \to c \vee \neg d]\!]_\mathcal{I}$ in an interpretation $\mathcal{I}$ s.t. $\mathcal{I}(a) = F$.



◯ unnecessary computations

Algorithm for evaluation: main part

Function evaluate(t, $\mathcal{I}$)

Input: a formula φ to evaluate represented by its syntax tree $t = \langle t_1, c, t_2 \rangle$ and an interpretation $\mathcal{I}$

Output: the boolean value T (if $\models_{\mathcal{I}} \varphi$) or F (if $\not\models_{\mathcal{I}} \varphi$)

```
1  if t1 = ∅ and t2 = ∅ then
2  |   return I(c) ;
3  end
4  switch c do
5  |   case ¬: return evaluateNot(t1, I) ;
6  |   ;
7  |   case ∨: return evaluateOr(t1, t2, I) ;
8  |   ;
9  |   case ∧: return evaluateAnd(t1, t2, I) ;
10 |   ;
11 |   case →: return evaluateImplies(t1, t2, I) ;
12 |   ;
13 |   case ↔: return evaluateEquiv(t1, t2, I) ;
14 |   ;
```

Algorithm for evaluation: $\neg$ and $\vee$

Function evaluateNot$(t, \mathcal{I})$

```
1 return f_¬(evaluate(t, I)) ;
```

Function evaluateOr$(t_1, t_2, \mathcal{I})$

```
1 v_1 ← evaluate(t_1, I) ;
2 if v_1 = T then
3 |   return T ;
4 else
5 |   return evaluate(t_2, I) ;
6 end
```

Algorithm for evaluation: $\wedge$ and $\rightarrow$

Function evaluateAnd($t_1, t_2, \mathcal{I}$)

```
1 v1 ← evaluate(t1, I) ;
2 if v1 = F then
3 |   return F ;
4 else
5 |   return evaluate(t2, I) ;
6 end
```

Function evaluateImplies($t_1, t_2, \mathcal{I}$)

```
1 v1 ← evaluate(t1, I) ;
2 if v1 = F then
3 |   return T ;
4 else
5 |   return evaluate(t2, I) ;
6 end
```

Outline of part 2 - PL language and semantics

Truth table

Definition (truth table)

A **truth table** for a formula φ using $\{a_1, \ldots a_n\}$ as propositional variables is a table with $n+1$ columns and 2^n rows, such that:

- each case of the table contains the value T or the value F
- each row represents a **different interpretation** for $\{a_1, \ldots a_n\}$. This interpretation is represented by the values appearing in the n first cases of the row
- the last column of a row representing the interpretation $\mathcal{I}$ contains the value of $[\![\varphi]\!]_{\mathcal{I}}$.

Truth table: example and conclusion

Example for $a \wedge b$:

a	b	$a \wedge b$
F	F	F
F	T	F
T	F	F
T	T	T

So, if we want to prove the validity of φ, build a truth table for φ and verify that the truth value of φ in each row is T...

...but in $O(2^n)$ where n is the number of propositional variables appearing in φ!

Complexity of truth tables

Let us consider a language with **300 propositional variables** and a wff φ using those variables.

- **space complexity**

  The number of lines of a truth table for φ is $2^{300} \approx 10^{90}$

  But the number of atoms in the Universe is approximately 10^{80}!

- **time complexity**

  Let's use the **Titan** supercomputer of the Oak Ridge National Laboratory (20 Pflops)...

  Supposing that Titan computes 10^{16} rows per seconds, it takes $10^{90}/(10^{16} \times 365 \times 24 \times 3600 \approx 10^{66}$ years.

  But the age of the Universe is approximately 13.7 billions years!

The yoga problem

Exercise

Show that the formula representing the yoga problem is valid using truth tables.

Outline of part 2 - PL language and semantics

Equivalent formulas

Definition (equivalent formulas)

Two wffs φ and ψ are equivalent iff for every interpretation $\mathcal{I}$, $[\![\varphi]\!]_{\mathcal{I}} = [\![\psi]\!]_{\mathcal{I}}$.

This is denoted by $\varphi \equiv \psi$.

Idea

If we want to prove that φ is a tautology, we can show that $\varphi \equiv \top$.

Some equivalent formulas...

Definition of $\leftrightarrow$ and $\rightarrow$:

$a \leftrightarrow b \equiv (a \rightarrow b) \wedge (b \rightarrow a)$
$a \rightarrow b \equiv \neg a \vee b$

Commutativity of $\wedge$ and $\vee$:

$a \wedge b \equiv b \wedge a$
$a \vee b \equiv b \vee a$

Associativity of $\wedge$ and $\vee$:

$(a \wedge b) \wedge c \equiv a \wedge (b \wedge c)$
$(a \vee b) \vee c \equiv a \vee (b \vee c)$

Distributivity of $\wedge$ and $\vee$:

$a \vee (b \wedge c) \equiv (a \vee b) \wedge (a \vee c)$
$a \wedge (b \vee c) \equiv (a \wedge b) \vee (a \wedge c)$

Some equivalent formulas...

Using $\top$ and $\bot$:

$$a \vee \top \equiv \top$$
$$a \vee \bot \equiv a$$
$$a \wedge \top \equiv a$$
$$a \wedge \bot \equiv \bot$$
$$a \vee \neg a \equiv \top$$
$$a \wedge \neg a \equiv \bot$$
$$\neg\neg a \equiv a$$

Idempotency:

$$a \vee a \equiv a$$
$$a \wedge a \equiv a$$

De Morgan's laws:

$$\neg(a \vee b) \equiv \neg a \wedge \neg b$$
$$\neg(a \wedge b) \equiv \neg a \vee \neg b$$
$$a \vee (\neg a \wedge b) \equiv a \vee b$$
$$a \wedge (\neg a \vee b) \equiv a \wedge b$$
$$a \vee (a \wedge b) \equiv a$$
$$a \wedge (a \vee b) \equiv a$$

The yoga problem

Exercise

Show that the formula representing the yoga problem is valid using equivalent formulas.

Outline of part 2 - PL language and semantics

Conjunctive normal form

A particular form of equivalent formula is **conjunctive normal form**.

Definition (conjunctive normal form)

- a **literal** is a propositional variable or the negation of a propositional variable
- a **clause** is a unordered disjunction of literals
- a wff φ is in **conjunctive normal form** (CNF) iff $\varphi = \varphi_1 \wedge \ldots \wedge \varphi_n$ where $\forall i \in \{1, \ldots, n\}$ φ_i is a clause

Theorem (existence of a conjunctive normal form)

Every wff φ can be rewritten into a wff $CNF(\varphi)$ in CNF such that φ and $CNF(\varphi)$ are logically equivalent.

What is so interesting about CNF?

We know that if $CNF(\varphi)$ is valid, then φ is valid. Is it easy to check if $CNF(\varphi)$ is valid?

YES, because remember that

$$CNF(\varphi) \equiv \bigwedge_{i=1}^{n} \bigvee_{j=1}^{n_i} L_{i,j} \text{ where } L_{i,j} \text{ is a literal}$$

So, to verify that $CNF(\varphi)$ is valid, check that in every clause of $CNF(\varphi)$ there are a literal and its negation.

Theorem (validity of a disjunction of literal)

A disjunction of literals $_1 \vee \ldots \vee L_n$ is valid iff there are $1 \leq i < j \leq n$ s.t. $L_i \equiv \neg L_j$.

How to compute a CNF

We can establish some rewritting rules to translate a formula φ into an equivalent CNF.
For instance, removing implication can be achieved using the following equivalence/rewritting rule:

$$\varphi \to \psi \equiv \neg\varphi \vee \psi$$

and you can write other rules to perform the translation.

But we want a translation **algorithm** that is:

- determinist (the same input will produce the same result)
- efficient (in term of conjunctions number for instance)

Translation algorithm: the main idea

Here are the big steps of the translation algorithms (details in the next slides):

1. eliminate all $\rightarrow$ and $\leftrightarrow$ symbols using the following rules:

$$\varphi \leftrightarrow \psi \equiv (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$$
$$\varphi \rightarrow \psi \equiv \neg\varphi \vee \psi$$

After such a preprocessing, we will obtain a formula in which:

- double negations could appear
- negation could appear in front of non atomic formulas

Translation algorithm: the main idea

Here are the big steps of the translation algorithms (details in the next slides):

2. translate the previously obtained formula in NNF.

Definition (negative normal form)

A wff φ is said to be in negative normal form (NNF) iff the negations appearing in φ only concern atoms.

After such a preprocessing, we will obtain a formula

- containing only $\vee$, $\wedge$ and $\neg$ connectors
- in which $\neg$ only concerns atoms

Translation algorithm: the main idea

Here are the big steps of the translation algorithms (details in the next slides):

3. translate the previously obtained formula $NNF(\varphi)$ in NNF to CNF with a recursive algorithm:
   - if $NNF(\varphi)$ is a literal: OK
   - if $NNF(\varphi) = \psi_1 \wedge \psi_2$: easy, compute the CNF of ψ_1 and ψ_2 and you are done
   - if $NNF(\varphi) = \psi_1 \vee \psi_2$: more difficult...

The remove-imp function

Function remove-imp(φ)

Input: a wff φ
Output: a wff WI(φ) without $\rightarrow$ equivalent to φ

```
1 switch φ do
2 |   case φ is a literal: return φ ;
3 |   ;
4 |   case φ is φ1 ∧ φ2: return remove-imp(φ1) ∧ remove-imp(φ2) ;
5 |   ;
6 |   case φ is φ1 ∨ φ2: return remove-imp(φ1) ∨ remove-imp(φ2) ;
7 |   ;
8 |   case φ is φ1 → φ2: return ¬ remove-imp(φ1) ∨ remove-imp(φ2) ;;
9 endsw
```

The NNF function

Function NNF(φ)

Input: a wff φ without $\rightarrow$ symbols

Output: a wff NNF(φ) equivalent to φ in which all negations are in front of atoms

```
1  switch φ do
2      case φ is a literal: return φ ;
3      ;
4      case φ is ¬¬φ1: return φ1 ;
5      ;
6      case φ is φ1 ∧ φ2: return NNF(φ1) ∧ NNF(φ2) ;
7      ;
8      case φ is φ1 ∨ φ2: return NNF(φ1) ∨ NNF(φ2) ;
9      ;
10     case φ is ¬(φ1 ∧ φ2): return NNF(¬φ1) ∨ NNF(¬φ2) ;
11     ;
12     case φ is ¬(φ1 ∨ φ2): return NNF(¬φ1) ∧ NNF(¬φ2) ;
13     ;
14 endsw
```

The CNF function

Function CNF(φ)

Input: a wff φ in NNF
Output: a wff CNF(φ) equivalent to φ in conjunctive normal form

```
1 switch φ do
2     case φ is a literal: return φ ;
3     ;
4     case φ is φ1 ∧ φ2: return CNF(φ1) ∧ CNF(φ2) ;
5     ;
6     case φ is φ1 ∨ φ2: return DISTR(CNF(φ1), CNF(φ2)) ;;
7 endsw
```

The disjunction case has to be held by another function.

The DISTR function

Function DISTR(φ_1, φ_2)

Input: 2 wffs φ_1 and φ_2 in CNF
Output: a wff DISTR(φ_1, φ_2) in CNF equivalent to $\varphi_1 \vee \varphi_2$

```
1  switch φ do
2  |  case φ1 is φ11 ∧ φ12:
3  |  |  return DISTR(φ11, φ2)∧ DISTR(φ12, φ2) ;
4  |  end
5  |  case φ2 is φ21 ∧ φ22:
6  |  |  return DISTR(φ1, φ21)∧ DISTR(φ1, φ22) ;
7  |  end
8  |  otherwise
9  |  |  return φ1 ∨ φ2 ;
10 |  end
11 endsw
```

Rewritting rules used

You have the algorithms, but you can also use your brain with the following rewritting rules obtain from equivalent formulas:

step 1 (remove impl.)
$$\begin{cases} \varphi \leftrightarrow \psi & \equiv (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi) \\ \varphi \rightarrow \psi & \equiv \neg\varphi \vee \psi \end{cases}$$

step 2 (NNF)
$$\begin{cases} \neg(\neg\varphi) & \equiv \varphi \\ \neg(\varphi \wedge \psi) & \equiv \neg\varphi \vee \neg\psi \\ \neg(\varphi \vee \psi) & \equiv \neg\varphi \wedge \neg\psi \end{cases}$$

step 3 (CNF)
$$\begin{cases} \varphi \vee (\psi \wedge \gamma) & \equiv (\varphi \vee \psi) \wedge (\varphi \vee \gamma) \\ (\psi \wedge \gamma) \vee \varphi & \equiv (\psi \vee \varphi) \wedge (\gamma \vee \varphi) \end{cases}$$

The yoga problem

Exercise

Show that the formula representing the yoga problem is valid by translating it into CNF.

Outline of part 2 - PL language and semantics

Why another algorithm?

There is a simple algorithm to verify if a CNF is valid, but it can take too much time: for a clause with n literals, checking that there are a literal and its negation in the clause is $O(n^2)$.

Davis and Putnam developed in 1960 a more efficient algoritm to check the validity of a propositional formula.

Davis, Martin and Hilary Putnam (1960).
"A Computing Procedure for Quantification Theory".
In: **Journal of the ACM** 7.3,
Pp. 201–215.
ISSN: 0004-5411.
DOI: 10.1145/321033.321034.
http://doi.acm.org/10.1145/321033.321034.

The main principle of the DP algorithm

The main idea of the DP algorithm is to use the fact that if φ is valid, then $\neg\varphi$ is not satisfiable. Moreover, it uses a set of clauses representing $CNF(\varphi)$.

Definition (set of clauses representing a formula)

Let φ be a wff. $CL(\varphi)$ is the **set** of **clauses** obtained from $CNF(\varphi)$ by removing the conjunction connector in $t(\varphi)$.

If $CNF(\varphi) = \bigwedge_{i=1}^{n} C_i$ then $CL(\varphi) = \bigcup_{i=1}^{n} \{C_i\}$.

Theorem (equivalence between φ and $CL(\varphi)$)

Let φ be a wff. φ is unsatisfiable iff $CL(\varphi)$ is unsatisfiable.

The rules used in DP procedure

taut eliminate every tautology from S and obtain a set S'.
S is unsatisfiable iff S' is.

one if there is an **unit** clause in S (a clause is unit if it containes only one literal), compute S' by eliminating every occurrence of L in S.
If $S' = \emptyset$, then S is satisfiable.
Otherwise, build S'' by eliminating $\neg L$ from S'.
S is unsatisfiable iff S'' is.

pure a literal L is **pure** in S if $\neg L$ does not appear in S. If L is pure, obtain S' from S by eliminating L.
S is unsatisfiable iff S' is.

split if $S = \{A_1 \vee L, \ldots, A_n \vee L, B_1 \vee \neg L, B_m \vee \neg L, R\}$ with L and $\neg L$ not appearing neither in A_i, B_j nor R, then S is unsatisfiable iff $\{A_1, \ldots, A_n, R\}$ and $\{B_1, \ldots, B_m, R\}$ are.

Davis and Putnam procedure: example

Let us consider $\varphi = p \vee q \to (\neg p \vee q \to (\neg q \vee r \vee t \to (s \vee \neg q \to s)))$.
We have first to compute $\neg\varphi$ and put it in CNF (easy here ☺).

We apply the DP procedure:

$\{p \vee q, \neg p \vee q, \neg q \vee r \vee t, s \vee \neg q, \neg s\}$	
$\{q, \neg q \vee r \vee t, s \vee \neg q, \neg s\}$	split with p
$\{r \vee t, s, \neg s\}$	one with q
$\{s, \neg s\}$	pure with r and t
$\emptyset$	one with s

Therefore $\{p \vee q, \neg p \vee q, \neg q \vee r \vee t, s \vee \neg q, \neg s\}$ is unsatisfiable, thus φ is valid.

We will see that the Resolution formal system has an inference rule that is similar to the split rule.

Outline of part 2 - PL language and semantics

The SAT problem

The SAT problem is the propositional satisfiability problem, i.e. the problem of determining if a wff is satisfiable or not.

Theorem (Cook, 1971)

*The SAT problem is **NP-complete**.*

- ➥ SAT is difficult to solve, but there are instances that can be solved efficiently
- ➥ **proving that a wff is valid** is Co-NP-complete!

The SAT solvers use formulas in CNF and the most widely used algorithm is a refinement of the DP algorithm.

The SAT problem is used in various industrial problems: electronics, verification of microprocessors, planning etc.

Some SAT solvers you can use

Online SAT solvers:

bool SAT	`http://www.boolsat.com/`
MiniSat	`http://www.msoos.org/2013/09/minisat-in-your-browser/`

Offline SAT solvers:

MiniSat	`http://minisat.se/`
Sat4j	`http://www.sat4j.org/`
CrypotMiniSat2	`http://www.msoos.org/cryptominisat2/`

3 - Formal systems for propositional logic

Outline of part 3 - Formal systems for propositional logic

What is a formal system?

Definition (formal system)

A formal system is composed of two elements:

- a **formal language** (grammar) defining a set of expressions E
- a **deductive system** or **deductive apparatus** on E

Deductive system

Definition (deductive system)

A **deduction system** (or **inference system**) on a set E is composed of a set of rules used to derive elements of E from other elements of E. They are called **inference rules**.

If an inference rule allows to derive e_{n+1} (conclusion) from $\mathcal{P} = \{e_1, \ldots, e_n\}$ (premises), it will be noted as follows:

$$\frac{e_1 \;\; e_2 \;\; \ldots \;\; e_n}{e_{n+1}}$$

When an inference rule is such that $\mathcal{P} = \emptyset$ it is called an **axiom**.

If e_1 is an axiom, it is either noted $\frac{}{e_1}$ or simply e_1.

Deductive system

Intuition

A rule $\dfrac{e_1 \;\; e_2}{e_3}$ means:

- from e_1 and e_2 you can deduce e_3
- to prove e_3, it is sufficient to prove e_1 and to prove e_2

Rules schemata

How to define an inference rule?

- ➥ it is in fact an inductive relation...
- ➥ for instance, the Modus Ponens inference rule (from "A" and "A implies B" deduce "B") has an infinite number of instances:

$$\frac{p \quad p \to q}{q} \qquad \frac{p \vee q \quad p \vee q \to r \wedge t}{r \wedge t} \qquad \ldots$$

We will use **rules schemata** to represent infinite number of rule instances.

Definition (rule schema)

A **rule schema** is a notation representing **all instances** of an inference rule by using **metavariables**. We will denote metavariables with **uppercase latin letters**.
A rule schema is **instanciated** by replacing every metavariable in the schema by an element of E.

Deduction as a sequence

Definition (deduction as a sequence)

Let $\mathcal{F}$ be a formal system on E. A **deduction** of e in $\mathcal{F}$ from the hypotheses $\mathcal{H} \subset E$ is a **finite** sequence of elements of E $e_1, \ldots, e_n$ such that $e_n = e$ and for all $i \in \{1, \ldots, n-1\}$:

- either e_i is an instance of an axiom of $\mathcal{F}$
- either $e_i \in \mathcal{H}$
- either e_i is deduced from $e_j, \ldots, e_{j+k}$ such that $j + k < i$ by using an instance

$$\frac{e_j \quad \ldots \quad e_{j+k}}{e_i}$$

of an inference rule of $\mathcal{F}$

Deduction as a sequence

Example on PL ("bad" formal system!):

Definition of $\mathcal{F}$

$$\frac{}{A \vee \neg A}(A_1) \qquad \frac{A \quad A \to B}{B}(MP)$$

Deduction of $p \vee r$ from $q \vee \neg q \to p \vee r$:

1. $q \vee \neg q$ $(A_1[A|q])$
2. $q \vee \neg q \to p \vee r$ $(\in \mathcal{H})$
3. $p \vee r$ $(MP[A|q \vee \neg q, B|p \vee r])$

using axiom A_1 by replacing A by q

Deduction as a sequence

Example on PL ("bad" formal system!):

Definition of $\mathcal{F}$

$$\frac{}{A \vee \neg A}(A_1) \qquad \frac{A \quad A \to B}{B}(MP)$$

Deduction of $p \vee r$ from $q \vee \neg q \to p \vee r$:

1. $q \vee \neg q$ $(A_1[A|q])$ using an hypothesis
2. $q \vee \neg q \to p \vee r$ $(\in \mathcal{H})$
3. $p \vee r$ $(MP[A|q \vee \neg q, B|p \vee r])$

Deduction as a sequence

Example on PL ("bad" formal system!):

Definition of $\mathcal{F}$

$$\frac{}{A \vee \neg A}(A_1) \qquad \frac{A \quad A \to B}{B}(MP)$$

Deduction of $p \vee r$ from $q \vee \neg q \to p \vee r$:

1. $q \vee \neg q$ $(A_1[A|q])$
2. $q \vee \neg q \to p \vee r$ $(\in \mathcal{H})$
3. $p \vee r$ $(MP[A|q \vee \neg q, B|p \vee r])$

using inference rule MP by replacing A by $q \vee \neg q$ and B by $p \vee r$

Deduction as a sequence: the classical way

Deduction as a sequence is in fact what you are doing when writing a proof in maths.

Let $\mathcal{G} = \{E, \times\}$ be a group. Prove that if each element x of the group is its own inverse, then $\mathcal{G}$ is commutative.

$$\begin{aligned} x \times (y \times (x^{-1} \times y^{-1})) &= x \times (y \times (x \times y)) && \text{hyp. + rules} \\ &= (x \times y) \times (x \times y) && \text{hyp. + rules} \\ &= e && \text{hyp. + rules} \end{aligned}$$

and then multiply both sides by $y \times x$ using again hypotheses and rules.

Deduction as a tree

Definition (deduction as a tree)

Let $\mathcal{F}$ be a formal system on E. The set $\mathcal{T}_D(\mathcal{H}, \mathcal{F})$ of deduction trees from $\mathcal{H}$ in $\mathcal{F}$ is defined inductively as follows:

- a tree with only one node such that this node is an instance of an axiom of $\mathcal{F}$ or an element of $\mathcal{H}$ is an element of $\mathcal{T}_D(\mathcal{H}, \mathcal{F})$
- if $t_1, \ldots, t_n$ are elements of $\mathcal{T}_D(\mathcal{H}, \mathcal{F})$ such that for every i in $\{1, \ldots, n\}$ the root of t_i is an element e_i of E and $\dfrac{e_1 \quad \ldots \quad e_n}{e_{n+1}}$ is an instance of a rule of $\mathcal{F}$, then the tree whose root is e_{n+1} and whose subtrees of e_{n+1} are $t_1, \ldots, t_n$ is an element of $\mathcal{T}_D(\mathcal{H}, \mathcal{F})$.

A finite tree of $\mathcal{T}_D(\mathcal{H}, \mathcal{F})$ whose root is an element e of E is called a **deduction** of e from $\mathcal{H}$ in $\mathcal{F}$. This is noted $\mathcal{H} \vdash_{\mathcal{F}} e$.

Deduction as a tree

Example on PL ("bad" formal system!):

Definition of $\mathcal{F}$

$$\frac{}{A \vee \neg A}(A_1) \qquad \frac{A \quad A \to B}{B}(MP)$$

Deduction of $p \vee r$ from $q \vee \neg q \to p \vee r$:

$$\dfrac{\dfrac{}{q \vee \neg q}\,(A_1) \qquad q \vee \neg q \to p \vee r}{p \vee r}\,(MP)$$

Proofs and theorems

Definition (proof)

A deduction tree $t \in \mathcal{T}_D(\emptyset, \mathcal{F})$ of e is called a **proof** of e.

e is called a **theorem** of $\mathcal{F}$.

This is noted $\vdash_{\mathcal{F}} e$.

The same definition as in maths: a theorem is a formula that can be deduced only from axioms (and other theorems).

What is expected from a formal system?

Effectiveness

- explicitness: no ambiguities
- mechanical: steps are determinist, no choice
- finite: it will stop

Beware, in most formal systems:

- deduction/proof **finding** is not effective (vs. truth tables for instance)
- deduction/proof **verification** is effective

What is expected from a formal system?

Completeness

Every tautology is a theorem (wrt a particular semantics).

Soundness

Every theorem is a tautology (wrt a particular semantics).

Consistency

$\varphi \wedge \neg\varphi$ **cannot** be proved.

Axioms independence

The axioms of the formal system are independent.

Outline of part 3 - Formal systems for propositional logic

Outline of part 3 - Formal systems for propositional logic

Deductive system: the simpliest one?

Definition (axioms)

A1 $A \to (B \vee A)$
A2 $(A \to B) \to ((A \to (B \to C)) \to (A \to C))$
A3 $(A \to B) \to ((A \to \neg B) \to \neg A)$
A4 $\neg\neg A \to A$

Definition (Modus Ponens or detachment rule)

From A and $A \to B$ deduce B:

$$\frac{A \qquad A \to B}{B}\ (MP)$$

Some remarks on Hilbert system

We only use the $\neg$, $\vee$ and $\rightarrow$ connectors, defining $\wedge$ for instance from those connectors.
You can find other axiomatizations with only $\neg$ and $\rightarrow$ for instance.

The Hilbert system is rather simple:

- axioms and rules are "intuitive"
- few axioms and rules

In fact, the original Hilbert-Ackermann system has lots of axioms, explaining the role of each logical connector in the deduction process.

Outline of part 3 - Formal systems for propositional logic

Soundness and completeness theorem

Theorem (soundness and completeness of $\mathcal{H}$)

- $\mathcal{H}$ is **sound**: for every wff φ, $\vdash_{\mathcal{H}} \varphi \Rightarrow \models \varphi$
- $\mathcal{H}$ is **complete**: for every wff φ, $\models \varphi \Rightarrow \vdash_{\mathcal{H}} \varphi$

Proof of soundness: "trivial" by structural induction.

Proof of completeness: more complicated (model each line of the truth table by the corresponding deduction)...

Theorem (consistency)

$\mathcal{H}$ is consistent: for every wff φ, either $\nvdash_{\mathcal{H}} \varphi$ or $\nvdash_{\mathcal{H}} \neg\varphi$.

Easy to prove using previous theorem...

Let us try to prove something easy

Exercise

Prove that $\vdash_{\mathcal{H}} p \to p$.

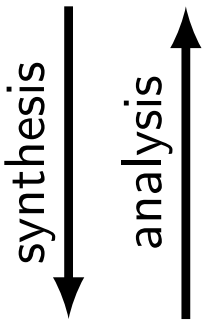
Let us try to prove something easy

Exercise

Prove that $\vdash_{\mathcal{H}} p \to p$.

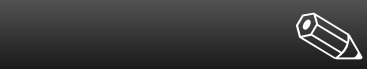
Not so easy...

What are the two classical strategies used in maths for building proofs?



$$\dfrac{\dfrac{\neg\neg p \qquad \neg\neg p \to p}{p} \qquad p \to q \to r}{q \to r}$$

Let us try to prove something easy



Exercise

Prove that $\vdash_{\mathcal{H}} p \to p$.

Synthesis is difficult: how can you choose the axioms to prove $p \to p$ for instance?

In the Hilbert system, analysis is difficult too: given $p \to p$ to prove, you know that as $p \to p$ is not an axiom, the previous step in the proof tree is applying MP:

$$\frac{? \qquad ? \to (p \to p)}{p \to p}$$

How can you choose the previous formula? Some intuition in next slides...

Let us try to prove something easy

Exercise

Prove that $\vdash_{\mathcal{H}} p \to p$.

Finally, the proof:

$$\dfrac{\dfrac{(p \to (p \to p)) \to ((p \to ((p \to p) \to p)) \to (p \to p)) \qquad p \to (p \to p)}{(p \to ((p \to p) \to p)) \to (p \to p)} \qquad p \to ((p \to p) \to p)}{p \to p}$$

Easy ☺

Some theorems of $\mathcal{H}$

Identity	(IDE)	$\vdash_{\mathcal{H}} p \leftrightarrow p$
De Morgan's laws	(DM)	$\vdash_{\mathcal{H}} \neg(p \wedge q) \leftrightarrow (\neg p \vee \neg q)$
		$\vdash_{\mathcal{H}} \neg(p \vee q) \leftrightarrow (\neg p \wedge \neg q)$
Commutativity	(COM)	$\vdash_{\mathcal{H}} (p \vee q) \leftrightarrow (q \vee p)$
		$\vdash_{\mathcal{H}} (p \wedge q) \leftrightarrow (q \wedge p)$
Associativity	(ASSO)	$\vdash_{\mathcal{H}} (p \wedge (q \wedge r)) \leftrightarrow ((p \wedge q) \wedge r)$
		$\vdash_{\mathcal{H}} (p \vee (q \vee r)) \leftrightarrow ((p \vee q) \vee r)$
Distributivity	(DIS)	$\vdash_{\mathcal{H}} (p \wedge (q \vee r)) \leftrightarrow ((p \wedge q) \vee (p \wedge r))$
		$\vdash_{\mathcal{H}} (p \vee (q \wedge r)) \leftrightarrow ((p \vee q) \wedge (p \vee r))$
Contraposition	(CONT)	$\vdash_{\mathcal{H}} (p \rightarrow q) \leftrightarrow (\neg q \rightarrow \neg p)$
Material implication		$\vdash_{\mathcal{H}} (p \rightarrow q) \leftrightarrow (\neg p \vee q) \leftrightarrow \neg(p \wedge \neg q)$
Idempotency	(IDM)	$\vdash_{\mathcal{H}} p \leftrightarrow (p \wedge p)$
		$\vdash_{\mathcal{H}} p \leftrightarrow (p \vee p)$
Exp.-Imp.	(EX/IM)	$\vdash_{\mathcal{H}} ((p \wedge q) \rightarrow r) \leftrightarrow (p \rightarrow (q \rightarrow r))$
Double negation	(DN)	$\vdash_{\mathcal{H}} p \leftrightarrow \neg\neg p$
Absorption		$\vdash_{\mathcal{H}} (p \wedge (p \vee q)) \leftrightarrow p$
		$\vdash_{\mathcal{H}} (p \vee (p \wedge q)) \leftrightarrow p$

Deduction theorem: the algorithm for $\rightarrow$...

Theorem (deduction)

Let $\{A_1, \ldots, A_{n-1}, A\}$ be a set of wffs and B be a wff. Then $\{A_1, \ldots, A_{n-1}, A\} \vdash_{\mathcal{H}} B$ iff $\{A_1, \ldots, A_{n-1}\} \vdash_{\mathcal{H}} A \rightarrow B$.

Proof made by structural induction on proof tree of B which gives you a hint on how to prove a wff in Hilbert system:

$$\dfrac{\dfrac{(A \rightarrow C) \rightarrow ((A \rightarrow (C \rightarrow B)) \rightarrow (A \rightarrow B)) \qquad \dfrac{\vdots}{A \rightarrow C}}{(A \rightarrow (C \rightarrow B)) \rightarrow (A \rightarrow B)} \qquad \dfrac{\vdots}{(A \rightarrow (C \rightarrow B))}}{A \rightarrow B}$$

So now you have an hint for the proof of $p \rightarrow p$.

The train example

Exercise

Prove that the following argument is valid:

> *John has travelled by bus or by train. If he has travelled by bus or by car, he has been late and has missed the meeting. He was not late. Therefore he has travelled by train.*

We have to prove $\{b \vee t, (b \vee c) \rightarrow (l \wedge m), \neg l\} \vdash_{\mathcal{H}} t$

or $\vdash_{\mathcal{H}} ((b \vee t) \wedge ((b \vee c) \rightarrow (l \wedge m)) \wedge \neg l) \rightarrow t$

➥ not so easy...

Outline of part 3 - Formal systems for propositional logic

Outline of part 3 - Formal systems for propositional logic

Introduction

Hilbert system is essentially a **synthetic** method and is very difficult to use: for instance, you have to **choose** an instance of an axiom (the deduction theorem gives us some hints on how to prove formulas like $\varphi \to \psi$).

An **analytic** method is easier to use: we can **transform and decompose** the formula to be proved and thus starting from the formula to prove, we can build the proof tree.

Is there some **analytic** proof methods for PL?

Sequent system $\mathcal{G}$ invented by Gentzen will be presented with a complete algorithm

Formal language: notion of sequent

The base notion of Gentzen's system is the **sequent**.

N.B.

The formal language of the formal system is not $\mathcal{L}_{PL}$!

Definition (sequent)

A **sequent** is defined by two **multisets** of wffs Γ and Δ (resp. called **antecedent** and **consequent** of the sequent) linked by the symbol $\Rightarrow$.

A sequent is noted $\Gamma \Rightarrow \Delta$.

Sequent: intuition

A sequent $\Gamma \Rightarrow \Delta$ represents an **argument** (or a **judgement**): from hypotheses/context Γ we can deduce Δ.

Instead of having a formal system working on formulas, we have a formal system working on **arguments**.

The inference rules will allow us to derive argument from other arguments, which is what you do intuitively.
For instance, if you can prove $a \wedge b$ from Γ, then you can prove a from Γ.

Compare to $\mathcal{H}$ system in which you "forget" your hypothesis during the proof.

Historically the $\mathcal{G}$ system we will use is derived from other systems invented by Gentzen: **natural deduction**, $\mathcal{LK}$ and $\mathcal{LK}$ without the cut rule.

Sequent validity

Intuition

The sequent $\{\varphi_1, \ldots, \varphi_n\} \Rightarrow \{\psi_1, \ldots, \psi_m\}$ represents the wff $(\varphi_1 \wedge \ldots \wedge \varphi_n) \rightarrow (\psi_1 \vee \ldots \vee \psi_m)$.

$\{\} \Rightarrow \{\psi_1, \ldots, \psi_m\}$ represents the wff $\top \rightarrow (\psi_1 \vee \ldots \vee \psi_m)$ i.e. $(\psi_1 \vee \ldots \vee \psi_m)$.

$\{\varphi_1, \ldots, \varphi_n\} \Rightarrow \{\}$ represents the wff $\varphi_1 \wedge \ldots \wedge \varphi_n \rightarrow \bot$ i.e. $(\neg\varphi_1 \vee \ldots \vee \neg\varphi_n)$.

Definition (sequent validity)

$\Gamma \Rightarrow \Delta$ is **valid** if the "wff" $(\Gamma \rightarrow \Delta)$ representing the sequent is also valid. This is noted $\models_{\mathcal{G}} \Gamma \Rightarrow \Delta$.

How to use $\mathcal{G}$?

Let us suppose that we want to prove that a wff φ is valid. Then from the previous slide, it is equivalent to prove that the sequent $\Rightarrow \varphi$ is valid. We can of course make the same reasoning for proving that φ is a contradiction or that φ is a logical consequence of Σ:

$$\begin{aligned} \models \varphi \Leftrightarrow & \quad \models_{\mathcal{G}} \Rightarrow \varphi \\ \models \neg\varphi \Leftrightarrow & \quad \models_{\mathcal{G}} \varphi \Rightarrow \\ \Sigma \models \varphi \Leftrightarrow & \quad \models_{\mathcal{G}} \Sigma \Rightarrow \varphi \end{aligned}$$

Now, we have to define a complete and sound system that allows us to automatically derive a proof of a valid sequent.

Outline of part 3 - Formal systems for propositional logic

Gentzen's deductive system: axioms

Definition (axioms of $\mathcal{G}$)

The axioms of $\mathcal{G}$ are the sequents of the following form:

$$\frac{}{\Gamma, A, \Gamma' \Rightarrow \Delta, A, \Delta'} \qquad \frac{}{\Gamma, \bot, \Gamma' \Rightarrow \Delta} \qquad \frac{}{\Gamma \Rightarrow \Delta, \top, \Delta'}$$

E.g.: $\{a \wedge b, c, d \vee e\} \Rightarrow \{\neg e, c, f\}$, $\{a, b\} \Rightarrow \{b, c\}$.

Gentzen's deductive system: inference rules

Definition (inference rules of $\mathcal{G}$)

$$\frac{\Gamma \Rightarrow \Delta, A}{\Gamma, \neg A \Rightarrow \Delta}\ (\neg l) \qquad \frac{\Gamma, A \Rightarrow \Delta}{\Gamma \Rightarrow \Delta, \neg A}\ (\neg r)$$

$$\frac{\Gamma, A, B \Rightarrow \Delta}{\Gamma, A \wedge B \Rightarrow \Delta}\ (\wedge l) \qquad \frac{\Gamma \Rightarrow \Delta, A \qquad \Gamma \Rightarrow \Delta, B}{\Gamma \Rightarrow \Delta, A \wedge B}\ (\wedge r)$$

$$\frac{\Gamma, A \Rightarrow \Delta \qquad \Gamma, B \Rightarrow \Delta}{\Gamma, A \vee B \Rightarrow \Delta}\ (\vee l) \qquad \frac{\Gamma \Rightarrow \Delta, A, B}{\Gamma \Rightarrow \Delta, A \vee B}\ (\vee r)$$

$$\frac{\Gamma \Rightarrow \Delta, A \qquad \Gamma, B \Rightarrow \Delta}{\Gamma, A \to B \Rightarrow \Delta}\ (\to l) \qquad \frac{\Gamma, A \Rightarrow \Delta, B}{\Gamma \Rightarrow \Delta, A \to B}\ (\to r)$$

A closer look at inference rules

The $\mathcal{G}$ inference rules allow to:

- build argument with more complex wffs from existing arguments. For instance

$$\frac{\Gamma \Rightarrow \Delta, A \qquad \Gamma \Rightarrow \Delta, B}{\Gamma \Rightarrow \Delta, A \wedge B} \ (\wedge r)$$

  says that if you can prove A and Δ from Γ and B and Δ from Γ, then you can prove $A \wedge B$ and Δ from Γ.

- "swap" some wffs between the context/deduction parts of the sequent. For instance

$$\frac{\Gamma, A \Rightarrow \Delta}{\Gamma \Rightarrow \Delta, \neg A} \ (\neg r)$$

  says that if you can prove Δ from Γ and A, then you can prove Δ and $\neg A$ from Γ.

A closer look at inference rules

The $\mathcal{G}$ inference rules allow to:

- build argument with more complex wffs from existing arguments. For instance

$$\frac{\Gamma \Rightarrow \Delta, A \qquad \Gamma \Rightarrow \Delta, B}{\Gamma \Rightarrow \Delta, A \wedge B} \ (\wedge r)$$

  says that if you can prove A and Δ from Γ and B and Δ from Γ, then you can prove $A \wedge B$ and Δ from Γ.

- "swap" some wffs between the context/deduction parts of the sequent. For instance

$$\frac{\Gamma, A \Rightarrow \Delta}{\Gamma \Rightarrow \Delta, \neg A} \ (\neg r)$$

  says that if you can prove Δ from Γ and A, then you can prove Δ and $\neg A$ from Γ.

A closer look at inference rules

The $\mathcal{G}$ inference rules allow to:

- build argument with more complex wffs from existing arguments. For instance

$$\frac{\Gamma \Rightarrow \Delta, A \qquad \Gamma \Rightarrow \Delta, B}{\Gamma \Rightarrow \Delta, A \wedge B} \ (\wedge r)$$

  says that if you can prove A and Δ from Γ and B and Δ from Γ, then you can prove $A \wedge B$ and Δ from Γ.

- "swap" some wffs between the context/deduction parts of the sequent. For instance

$$\frac{\Gamma, A \Rightarrow \Delta}{\Gamma \Rightarrow \Delta, \neg A} \ (\neg r)$$

  says that if you can prove Δ from Γ and A, then you can prove Δ and $\neg A$ from Γ.

Notice that as we are working with multisets, the order of formulas is not important.

Soundness and completness of $\mathcal{G}$

Theorem (soundness and completeness of $\mathcal{G}$)

- $\mathcal{G}$ is **sound**: for every wff φ, $\vdash_{\mathcal{G}} \Rightarrow \varphi$ implies $\models \varphi$
- $\mathcal{G}$ is **complete**: for every wff φ, $\models \varphi$ implies $\vdash_{\mathcal{G}} \Rightarrow \varphi$

Soundness: easy to prove by using structural induction.

Completness: more difficult

- express Hilbert's $\mathcal{H}$ system in $\mathcal{G}$
- use direct proof (what are you doing the next 2 weeks?)

Proving $p \to p$

Exercise

Prove that $p \to p$ is a valid using $\mathcal{G}$.

OK, proving $p \to p$ is proving the sequent $\Rightarrow p \to p$.

$$\dfrac{\dfrac{}{\mathsf{p} \Rightarrow \mathsf{p}}}{\Rightarrow p \to p}\ (\to r)$$

Easy, hu?

This is the deduction theorem representation for $\mathcal{G}$: from p you can deduce p, therefore $p \to p$ is a theorem...

A more difficult proof

Exercise

Prove $(p \to q) \to (\neg q \to \neg p)$ in $\mathcal{G}$.

OK, that is more difficult:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\neg q, \mathrm{p} \Rightarrow \mathrm{p}}}{\neg q \Rightarrow p, \neg p}(\neg r) \quad \dfrac{\dfrac{}{\mathrm{q} \Rightarrow p, \mathrm{q}}}{q, \neg q \Rightarrow p}(\neg l)}{p \to q, \neg q \Rightarrow \neg p}(\to l)}{p \to q \Rightarrow \neg q \to \neg p}(\to r)}{\Rightarrow (p \to q) \to (\neg q \to \neg p)}(\to r)$$

So, can we achieve this proof **automatically**?

Outline of part 3 - Formal systems for propositional logic

Goal of automatic proof building

Can a program (or you) automatically build a proof tree for a given sequent?

Remember that we have two methods for building the proof tree:

- **synthesis**: difficult to use, you have to choose axioms to start for instance
- **analysis**: consider the inference rules "backwards" as tactics to build the tree starting from the wff/sequent you want to prove.

  For instance, $\mathcal{H}$ is not a good system due to MP:

$$\frac{? \qquad ? \to \varphi}{\varphi}\ (MP)$$

Does $\mathcal{G}$ have good properties that allow to build the proof tree?

Two important properties of $\mathcal{G}$

Theorem (reversibility of rules of $\mathcal{G}$)

*Every rule of $\mathcal{G}$ is such that the wff represented by the conclusion sequent of the rule is **logically equivalent** to the conjunction of the wff represented by the premises sequents of the rule.*

Theorem (subformula property)

Every formula appearing in a sequent premise of $\mathcal{G}$ inference rule is a subformula of a formula appearing in the sequent conclusion of the rule.

Not the case for MP for instance.

Canonical deduction tree

Definition (canonical deduction tree)

A **canonical deduction tree** of a sequent $\Gamma \Rightarrow \Delta$ is a tree built using systematically inference rules of $\mathcal{G}$. A branch building is stopped if:

- either an axiom has been produced
- either no inference rule can be applied

It is easy to prove that a canonical deduction tree is finite using finiteness of wff and the subformula property.

Theorem (validity of a sequent)

A sequent S is valid iff a canonical deduction tree of S such that every leaf of the tree is an axiom of $\mathcal{G}$ can be built.

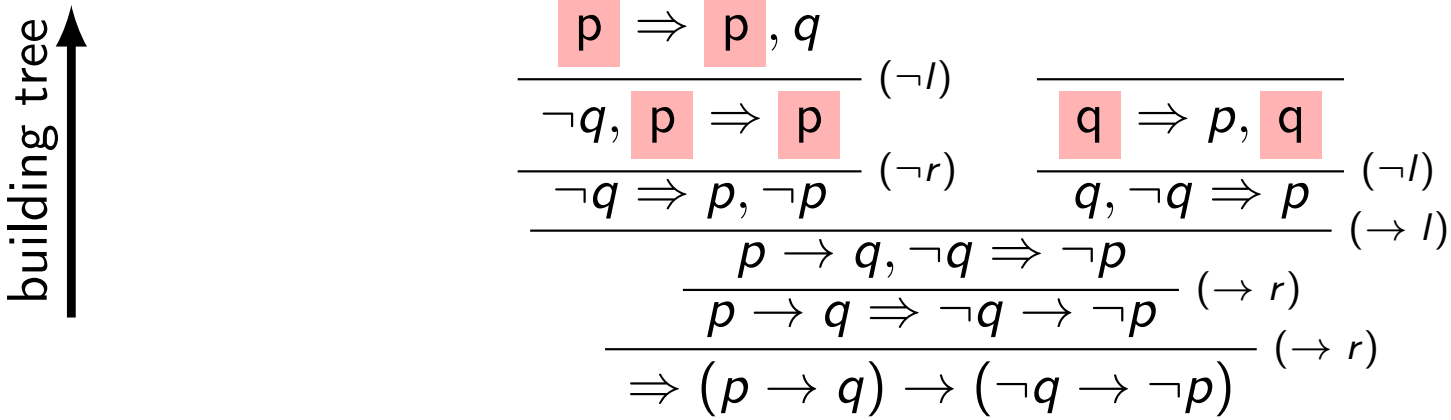
Base idea of automatization

"Algorithm":

- to prove or refute the validity of an argument "from Σ we can deduce φ", represent the argument by the sequent $\Sigma \Rightarrow \varphi$ which will be the root of the tree
- use the inference rules "backwards" from the sequent to build the tree until no rule can be applied
- if every leaf of the tree is an axiom, the argument is valid
- if there is one leaf that is not axiom, the argument is not valid and this leaf shows a counterexample for the validity of the sequent

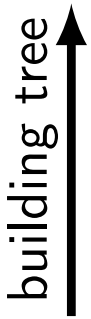
Example of canonical deduction tree: valid wff

Is $(p \to q) \to (\neg q \to \neg p)$ valid?



Example of canonical deduction tree: non valid wff

Is $(p \to q) \to (\neg p \to \neg q)$ valid?



$$\dfrac{\dfrac{\dfrac{\dfrac{q, q \Rightarrow p}{q \Rightarrow \neg q, p}\,(\neg r)}{q, \neg p \Rightarrow \neg q}\,(\neg l) \qquad \dfrac{\dfrac{q \Rightarrow p, p}{\Rightarrow p, p, \neg q}\,(\neg r)}{\neg p \Rightarrow p, \neg q}\,(\neg l)}{p \to q, \neg p \Rightarrow \neg q}\,(\to l)}{p \to q \Rightarrow \neg p \to \neg q}\,(\to r)}{\Rightarrow (p \to q) \to (\neg p \to \neg q)}\,(\to r)$$

The sequent is not valid. You can find a counterexample by building an interpretation $\mathcal{I}$ falsifying one the non axiom leaf:

$$\mathcal{I}(p) = F$$
$$\mathcal{I}(q) = T$$

Tactics to build the proof tree

When building the proof tree, you have to **choose rules**. Is it important?

Theorem

Let $\mathcal{T}$ be a proof tree for $\Gamma \Rightarrow \Delta$. Let φ be the main formula in Γ or Δ used in last rule in $\mathcal{T}$. Then for every formula ψ in Γ or Δ, there is a proof tree $\mathcal{T}'$ of $\Gamma \Rightarrow \Delta$ such that ψ is the main formula used in the last rule in $\mathcal{T}'$.

So you can choose whatever formula you want to apply backwards the rules, it will work!

But of course, the height/number of branches of the tree will depend on the rule you choose...

➥ try to choose rules that do not "branch" the tree!

You can also verify that if a sequent is an axiom, not matter the rule you apply on it backwards you will obtain an axiom.

Let us write some proofs!

Exercise

Verify in $\mathcal{G}$ the yoga argument.

Exercise

Verify in $\mathcal{G}$ the train argument (or the tequila argument, as you want).

Outline of part 3 - Formal systems for propositional logic

Outline of part 3 - Formal systems for propositional logic

Conjunctive normal form: reminder

Definition (conjunctive normal form)

- a **literal** is a propositional variable or the negation of a propositional variable
- a **clause** is a unordered disjunction of literals
- a wff φ is in **conjunctive normal form** (CNF) iff $\varphi = \varphi_1 \wedge \ldots \wedge \varphi_n$ where $\forall i \in \{1, \ldots, n\}$ φ_i is a clause

Theorem (existence of a conjunctive normal form)

Every wff φ can be rewritten into a wff $t(\varphi)$ in CNF such that φ and $t(\varphi)$ are logically equivalent.

Translation into CNF: rewritting rules

A formula can be translated into a formula in CNF using the following rules:

step 1 (remove impl.)
$$\begin{cases} \varphi \leftrightarrow \psi & \equiv (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi) \\ \varphi \rightarrow \psi & \equiv \neg\varphi \vee \psi \end{cases}$$

step 2 (NNF)
$$\begin{cases} \neg(\neg\varphi) & \equiv \varphi \\ \neg(\varphi \wedge \psi) & \equiv \neg\varphi \vee \neg\psi \\ \neg(\varphi \vee \psi) & \equiv \neg\varphi \wedge \neg\psi \end{cases}$$

step 3 (CNF)
$$\begin{cases} \varphi \vee (\psi \wedge \gamma) & \equiv (\varphi \vee \psi) \wedge (\varphi \vee \gamma) \\ (\psi \wedge \gamma) \vee \varphi & \equiv (\psi \vee \varphi) \wedge (\gamma \vee \varphi) \end{cases}$$

Set of clauses

Definition (set of clauses representing a formula)

Let φ be a wff. $CL(\varphi)$ is the **set** of **clauses** obtained from $CNF(\varphi)$ by removing the conjunction connector in $t(\varphi)$.

If $CNF(\varphi) = \bigwedge_{i=1}^{n} C_i$ then $CL(\varphi) = \bigcup_{i=1}^{n} \{C_i\}$.

Definition

Let $\Sigma = \{\varphi_1, \ldots, \varphi_n\}$ a set of wffs. Then $cl(\Sigma) = \bigcup_{i \in \{1,\ldots,n\}} CL(\varphi_i)$.

Theorem (equivalence between Σ and $CL(\Sigma)$)

Let Σ be a set of wffs. Σ is satisfiable iff $cl(\Sigma)$ is satisfiable (and Σ is unsatisfiable iff $cl(\Sigma)$ is unsatisfiable).

Outline of part 3 - Formal systems for propositional logic

Deductive system of $\mathcal{R}$

Definition (factorisation rule F)

$$\frac{A \vee A \vee B_1 \vee \ldots \vee B_n}{A \vee B_1 \vee \ldots \vee B_n} \ (F)$$

such that $n \geq 0$ and A and all B_i are literals.

Definition (resolution rule R)

$$\frac{A \vee B_1 \vee \ldots \vee B_n \qquad \neg A \vee C_1 \vee \ldots \vee C_m}{B_1 \vee \ldots \vee B_n \vee C_1 \vee \ldots \vee C_m} \ (R)$$

such that $n \geq 0$, $m \geq 0$ and A, all B_i and all C_i are literals.

Some remarks on $\mathcal{R}$

There is no axiom in $\mathcal{R}$!

- ➥ so we cannot deduce theorems using $\mathcal{R}$ (we will see how to do that)
- ➥ but we can use $\mathcal{R}$ to deduce φ from Σ

The rule (R) can be viewed as a generalization of classical rules, e.g. MP and transitivity.

$$\frac{A \qquad A \to B}{B}\ (MP) \qquad\qquad \frac{A \to B \qquad B \to C}{A \to C}\ (trans)$$

Soundness of $\mathcal{R}$

Theorem (soundness of $\mathcal{R}$ rules)

The rules of $\mathcal{R}$ are valid.

Easy to prove.

We have here a limited soundness for $\mathcal{R}$ due to the lack of axioms, but we have the following result:

Theorem

Let Γ be a finite set of clauses and c a clause. If $\Gamma \vdash_{\mathcal{R}} c$ then $\Gamma \models c$.

Find John's eyes color

Exercise

John has blue eyes or green eyes and black hair or brown hair. He does not have black hair if he has green eyes. He has blue eyes if he has brown hair.
Modelize the previous sentences using a propositional language and answer the following questions using Resolution formal system:

- what color are John's eyes?
- what color are John's hair?

Outline of part 3 - Formal systems for propositional logic

Completeness of $\mathcal{R}$

Rules (R) and (F) define a new formal system $\mathcal{R}$. Is $\mathcal{R}$ complete?

What is the relation between:

$$\Sigma \models \varphi \quad \not\Leftrightarrow \quad CL(\Sigma) \vdash_{\mathcal{R}} CNF(\varphi)$$

It seems that the argument validity problem for PL can be reduced to a deduction problem into the new formal system...

...but this is not true!

Find a counterexample.

A new decision problem

Translate argument validity problem into another problem:

$$\begin{aligned} \Sigma \models \varphi &\Leftrightarrow \Sigma \cup \{\neg\varphi\} \text{ is unsatisfiable} \\ &\Leftrightarrow CL(\Sigma \cup \{\neg\varphi\}) \text{ is unsatisfiable} \end{aligned}$$

Conclusion

Showing that an argument is valid can be reduced to showing that a set of clauses is unsatisfiable.

Due to the compactness theorem, we just have to show that a finite subset of $CL(\Sigma \cup \{\neg\varphi\})$ is unsatisfiable.

A special case of deduction in $\mathcal{R}$: refutation

There is a particular case in the (R) rule in which you obtain the **empty clause**:

$$\frac{A \qquad \neg A}{\square}\ (R)$$

By convention, $\square$ is the symbol for **unsatisfiability** ($\bot$ can also be used), as the set of clauses used in R is clearly unsatisfiable.

Definition (refutation)

A refutation of a set of clauses Γ is a deduction of $\square$ in $\mathcal{R}$.

Theorem (soundness of $\mathcal{R}$ for refutation)

Let Γ be a set of clauses. If $\Gamma \vdash_{\mathcal{R}} \square$ then Γ is unsatisfiable.

Using refutation

Exercise

Show that the following set of wff is unsatisfiable:

$$\{p, p \to q \vee r, q \to r, \neg r, s \vee t\}$$

Completeness of $\mathcal{R}$

Finally:

Theorem (completeness of Resolution)

Let Σ be a set of wffs. If Σ is unsatisfiable, then $CL(\Sigma) \vdash_{\mathcal{R}} \square$.

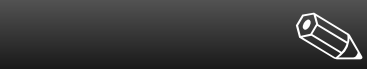
This can be proved by reducing the completeness problem to the completeness problem of semantic trees (not presented in lecture).

How to use Resolution?

What to prove	**How to prove it**
Σ is unsatisfiable	$CL(\Sigma) \vdash_{\mathcal{R}} \square$
$\Sigma \models \varphi$	$CL(\Sigma \cup \{\neg\varphi\}) \vdash_{\mathcal{R}} \square$
$\models \varphi$	$CL(\neg\varphi) \vdash_{\mathcal{R}} \square$

Remember that you have only to prove that a **subset** of Γ is unsatisfiable!

I promise, last time with the train problem...



Exercise

Verify in $\mathcal{R}$ the train argument (or the tequila argument, as you want).

Outline of part 3 - Formal systems for propositional logic

Resolvent and factor

In order to simplify the system, we can combine the two previous rules into a single rule.
First, we have to define **factors** and **binary resolvents** (easy).

Definition (factor)

Let C be a clause. Then a clause obtain by applying (F) on C is called **factor of** C.

Definition (binary resolvent)

Let C_1 and C_2 be two clauses. A clause obtained by applying (R) (if possible) on C_1 and C_2 is called **binary resolvent of** C_1 **and** C_2.

Resolvent: combining the two notions

We can now combine the two notions:

Definition (resolvent)

Let C_1 and C_2 be two clauses. A **resolvent** is a factor of a clause which can be:

- either a binary resolvent of C_1 and C_2
- either a binary resolvent of a factor of C_1 and C_2
- either a binary resolvent of C_1 and a factor of C_2
- either a binary resolvent of a factor of C_1 and a factor of C_2

A new formal system with a single rule!

Definition (resolution rule R)

$$\frac{C_1 \qquad C_2}{R(C_1, C_2)}\ (R)$$

where $R(C_1, C_2)$ is a resolvent of C_1 and C_2.

Of course, all previous properties apply to this new system (particularly completeness).

4 - First-order logic language and semantics

PL is not very expressive.

Example

Define a propositional language and express formulae to represent and to solve a 4x4 Sudoku game, for instance:

1			
	2		
			4
		3	

What do we need?

In order to have a more compact notation we need:

predicates

Predicates (or relations)

Predicates represent **properties** verified by **objects**.

For instance, $P(1, 2, 4)$ could mean "the case at column 1 and row 2 has 4 for value"

A predicate use a **singular term** and a **general term** to build a statement.

This statement will be "true" if the property represented by the general term is verified by the object represented by the singular term.

What do we need?

In order to have a more compact notation we need:

predicates

Particular predicates

We need sometimes to define particular predicates.

For instance, we need an **equality** predicate for Sudoku.

In this case, we need to characterize the properties of the given predicate, they are not given!

What do we need?

In order to have a more compact notation we need:

predicates + quantifiers

Quantifiers

Universal and **existential** quantifiers can be used on **variables**.

For instance, we could write:

$$\forall x \forall y \forall z \forall z' \; P(x, y, z) \rightarrow (\neg(z' = z) \rightarrow \neg P(x, y, z'))$$

Notice that there is no "," used to separate $\forall$ quantifiers.

What do we need?

In order to have a more compact notation we need:

predicates + quantifiers

Quantifiers: examples

- there is a man who is wise
  $\exists x\ (M(x) \wedge W(x))$
- every man is wise
  $\equiv$ it is wrong that there is a man who is not wise
  $\neg(\exists x\ (M(x) \wedge \neg W(x))$
- by convention $\neg\exists x\ \varphi(x)$ will be denoted by $\forall x\ \neg\varphi(x)$.

What do we need?

In order to have a more compact notation we need:

predicates + quantifiers

Quantifiers: beware of scope

- somebody has come and is gone
  $\exists x\ C(x) \wedge G(x)$
- somebody has come and somebody is gone
  $(\exists x\ C(x)) \wedge (\exists y\ G(y))$
- everybody is a male or a female
  $\forall x\ M(x) \vee F(x)$
- everybody is a male or everybody is a female
  $(\forall x\ M(x)) \vee (\forall x\ F(x))$

What do we need?

In order to have a more compact notation we need:

predicates + quantifiers + functions

Functions

Sometimes, we need to build **objects** from **other objects**.

For instance, when dealing with integers, the $+$ **function** allows to build an integer from two other integers.

$1 + 2$ is not a property but a new integer!

What do we need?

In order to have a more compact notation we need:

predicates + quantifiers + functions

Functions: example

"$\underbrace{\text{John's father}}_{\text{new individual}}$ plays soccer"

➥ $Plays(father(john), soccer)$

Remark: in another language

$\exists x\ F(x, john) \wedge Plays(x, soccer)$

Outline of part 4 - FOL language and semantics

Outline of part 4 - FOL language and semantics

Alphabet

Definition (alphabet)

The alphabet of $\mathcal{L}_{FOL}$ is composed of:

- logical symbols
  - an inifinite and enumerable set $\mathcal{V}$ of individual variables $x, y, \ldots$
  - connectors: $\top, \bot, \neg, \rightarrow, \wedge, \vee, \leftrightarrow$
  - quantifiers: $\exists$, $\forall$
  - , ( )
- non-logical symbols
  - an enumerable set $\mathcal{P}$ of predicate symbols $P, Q, R, \ldots$
  - an enumerable set $\mathcal{F}$ of functions $f, g, h, \ldots$
  - an enumerable set $\mathcal{C}$ of individual constants $a, b, c, \ldots$

Signature of a first-order language

Like in the propositional case, $\mathcal{V}$, $\top$, $\bot$, $\neg$, $\vee$, $\rightarrow$, $\leftrightarrow$, (,) and are called **logical symbols** because their logical meaning is already defined.

On the contrary, $\mathcal{P}$, $\mathcal{F}$ and $\mathcal{C}$ depend on the problem to be modelled and thus the predicate, function and constant symbols are called **non-logical symbols**. It is also called the **signature** $\mathcal{S}$ of the language.

So, when you want to model a problem using $\mathcal{L}_{FOL}$, you first have to define the signature of your language, i.e. $\mathcal{S} = \langle \mathcal{P}, \mathcal{F}, \mathcal{C} \rangle$.

When defining predicates and functions, the arity is often denoted using the / notation:

$P/2$	a predicate P of arity 2
$f/3$	a function f of arity 3

Outline of part 4 - FOL language and semantics

$\mathcal{L}_{FOL}$ terms

An expression is a sequence of symbols.

Some expressions, called **terms**, represents **objects**.

ex: Socrates, John's father, 3+(2+5), ...

Definition (term)

The set of terms of $\mathcal{L}_{FOL}$ is defined inductively by:

- a variable is a term
- a constant is a term
- if f is a function symbol with arity m and if $t_1, \ldots, t_m$ are terms, then $f(t_1, \ldots, t_m)$ is a term

Well-formed formulas

Some expressions are interpreted as **assertions**. Those expressions are **well formed formulas** (wffs).

Definition (atomic formula)

If P is a predicate symbol with arity n and if $t_1, \ldots, t_n$ are terms, then $P(t_1, \ldots, t_n)$ is an **atomic formula** of $\mathcal{L}_{FOL}$.

Definition (well formed formula)

The set of wff of $\mathcal{L}_{FOL}$ is defined inductively as follows:

- an atomic formula is a wff
- $\top$ and $\bot$ are wffs
- if φ and ψ are wffs, then $(\neg\varphi)$, $(\varphi \vee \psi)$, $(\varphi \wedge \psi)$, $(\varphi \to \psi)$ and $(\varphi \leftrightarrow \psi)$ are wffs
- if φ is a wff and x is a variable, then $(Qx\ \varphi)$ where $Q \in \{\forall, \exists\}$ is a wff
  φ is called the **scope of** Qx (cf. later).

Some conventions (as in the PL case)

To simplify the writing, some conventions can be used:

- removing of external parentheses: $(a \wedge b) \rightsquigarrow a \wedge b$
- $\neg$ is written without parentheses: $(\neg a) \rightsquigarrow \neg a$
- connectors are associative from left to right: $((a \wedge b) \wedge c) \rightsquigarrow a \wedge b \wedge c$
- quantifiers sequences can be simplified: $Q_1 x (Q_2 y\ \varphi) \rightsquigarrow Q_1 x Q_2 y\ \varphi$

Connectors and quantifiers can be ordered by growing priority like in the PL case:

$$\forall \quad \exists \quad \leftrightarrow \quad \rightarrow \quad \vee \quad \wedge \quad \neg$$

Some remarks on $\mathcal{L}_{FOL}$

Constants can also be viewed as **0-ary functions**, i.e. functions that does not take parameters. We use a distinct set $\mathcal{C}$ to simplify the presentation of FOL semantics.

If you consider a FO language whose signature is the following:

- $\mathcal{C} = \emptyset$
- $\mathcal{F} = \emptyset$
- every predicate symbol P in $\mathcal{P}$ is a 0-ary symbol, i.e. it does not take parameters

then you obtain **propositional logic**. Thus, PL is a subset of FOL.

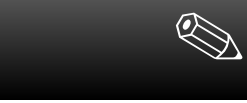
Modelling exercise

Exercise

Use fist-order logic to model the following declarative sentences. In each case, state what predicates, constants and functions mean.

1. every rose is a flower
2. no rose is a flower
3. some roses are flowers
4. some roses are not flowers
5. George is french, John is english and they are friends
6. all rats and all mice are gray
7. all giraffes are taller than all rats
8. cats and dogs are mammals
9. romans and greeks were enemies
10. who likes David likes also Tom
11. David only likes one person
12. everybody has a father and a mother

Modelling exercise: maths, again...



Exercise

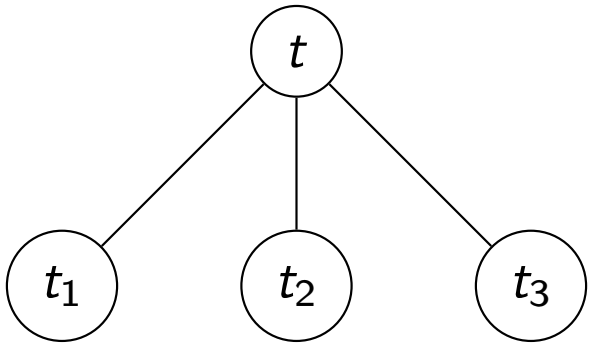
Let E be a set. Model the following mathematical notions using a first-order language. Define precisely the signature of the language.

- $=$ define the "classical" equality relation on E (not easy!)
- $\leq$ is a preorder on E
- $(E, .)$ is a monoid

Syntax tree of a term

As in the PL case, every wff can be represented by a **syntax tree**. We have first to define how to build the syntax tree for terms.

In the FO case, the tree is not necessary a binary tree. We use the notation $\langle t, t_1, t_2, t_3 \rangle$ to represent for instance the following tree:



$\langle t, \emptyset \rangle$ will represent the tree consisting of only one node containing t without subtrees.

Syntax tree of a term

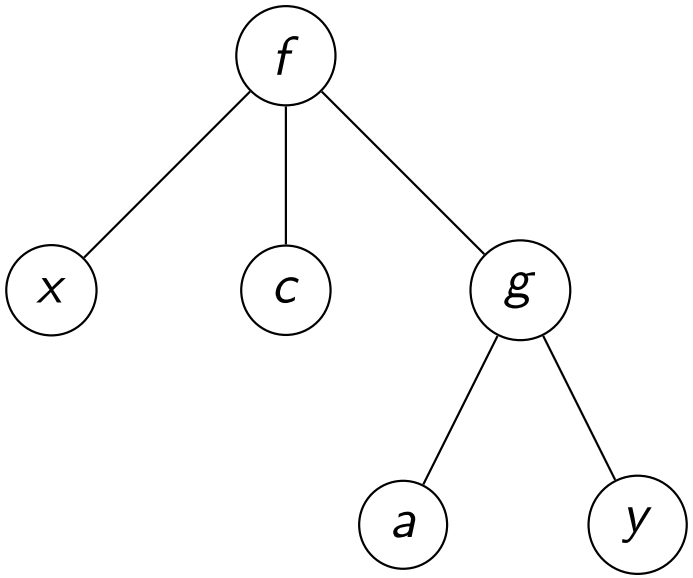
Definition (syntax tree for terms)

Let t be a term. The **syntax tree** of t is defined inductively as follows:

- if t is a variable, then $\mathcal{ST}(t) = \langle t, \emptyset \rangle$
- if t is a constant, then $\mathcal{ST}(t) = \langle t, \emptyset \rangle$
- if t is a n-ary function applied to terms $t_1, \ldots, t_n$, then $\mathcal{ST}(t) = \langle f, \mathcal{ST}(t_1), \ldots, \mathcal{ST}(t_n) \rangle$

Syntax tree of a term

For instance, the term $f(x, c, g(a, y))$ will be represented by the tree



Syntax tree of a FOL wff

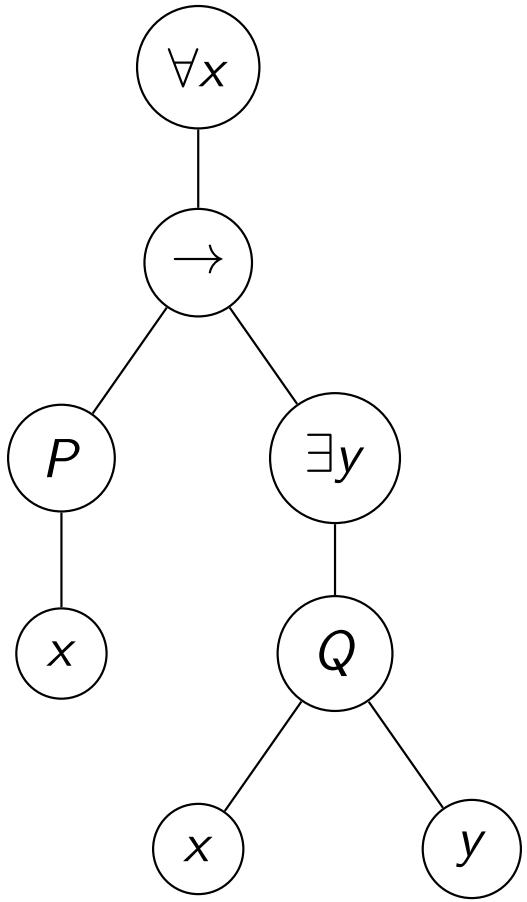
Definition (syntax tree)

Let φ be a wff. The **syntax tree** of φ is defined inductively as follows:

- if φ is an atomic formula $P(t_1, \ldots, t_n)$, then $\mathcal{ST}(\varphi) = \langle P, \mathcal{ST}(t_1), \ldots, \mathcal{ST}(t_n) \rangle$
- if $\varphi \equiv \neg\varphi_1$, then $\mathcal{ST}(\varphi) = \langle \neg, \mathcal{ST}(\varphi_1) \rangle$
- if $\varphi \equiv \varphi_1 * \varphi_2$ where $*$ is a binary connector, then $\mathcal{ST}(\varphi) = \langle *, \mathcal{ST}(\varphi_1), \mathcal{ST}(\varphi_2) \rangle$
- if $\varphi \equiv \mathcal{Q}x\ \varphi_1$ where $\mathcal{Q} \in \{\forall, \exists\}$ and $x \in \mathcal{V}$ then $\mathcal{ST}(\varphi) = \langle \mathcal{Q}x, \mathcal{ST}(\varphi_1) \rangle$

Syntax tree of a FOL wff

For instance, the formula $\forall x\ P(x) \to \exists y\ Q(x, y)$ will be represented by the following tree:



Outline of part 4 - FOL language and semantics

A correct definition of scope

We have defined the **scope** of a formula $Qx\ \varphi$ to be φ, but is it really the case?

Consider for instance $\forall x\ (P(x) \to (\exists x\ Q(x)))$. If the intuitive meaning of the scope of $\forall x$ is to define the formula in which you can replace x by "what you want", it is false.

Using the syntax tree, we can define scope in a better way:

Definition (scope)

Let $Qx\ \varphi$ be a wff with $Q \in \{\forall, \exists\}$. The **scope** of Qx in $Qx\ \varphi$ is the subtree of Qx in $\mathcal{ST}(Qx\ \varphi)$ minus the subtrees in $\mathcal{ST}(Qx\ \varphi)$ reintroducing a new quantifier for x.

With this definition the scope of $\forall x$ in $\forall x\ (P(x) \to (\exists x\ Q(x)))$ is only $P(x)$.

A correct definition of scope

We have defined the **scope** of a formula $Qx\ \varphi$ to be φ, but is it really the case?

Consider for instance $\forall x\ (P(x) \to (\exists x\ Q(x)))$. If the intuitive meaning of the scope of $\forall x$ is to define the formula in which you can replace x by "what you want", it is false.

N.B. (important)

Avoid reintroducing new quantifiers for a previously quantified variable in wff!

For instance, rewrite the previous formula as $\forall x\ (P(x) \to (\exists y\ Q(y)))$ which is unambiguous.

Free and bound variables

Definition (free and bound variables)

The set BV of **bound variables** and FV of **free variables** of a wff φ are defined inductively as follows:

- if φ is an atomic formula $P(t_1, \ldots, t_n)$, then $BV(\varphi) = \emptyset$ and $FV(\varphi) = \{t_i | i \in \{1, \ldots, n\}$ and t_i is a variable$\}$
- if $\varphi \equiv \neg\varphi_1$ then $BV(\varphi) = BV(\varphi_1)$ and $FV(\varphi) = FV(\varphi_1)$
- if $\varphi \equiv \varphi_1$ conn φ_2 where conn $\in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$ then $BV(\varphi) = BV(\varphi_1) \cup BV(\varphi_2)$ and $FV(\varphi) = FV(\varphi_1) \cup FV(\varphi_2)$
- if $\varphi \equiv Qx\ \varphi_1$ where $Q \in \{\forall, \exists\}$, then $BV(\varphi) = BV(\varphi_1) \cup \{x\}$ and $FV(\varphi) = FV(\varphi_1) - \{x\}$

Definition (closed formula)

A **closed** formula is a formula φ such that $FV(\varphi) = \emptyset$.

Free and bound variables: examples

free bound

$(\exists x\ P(x)) \wedge (\forall y\ \neg Q(y)) \wedge R(z)$

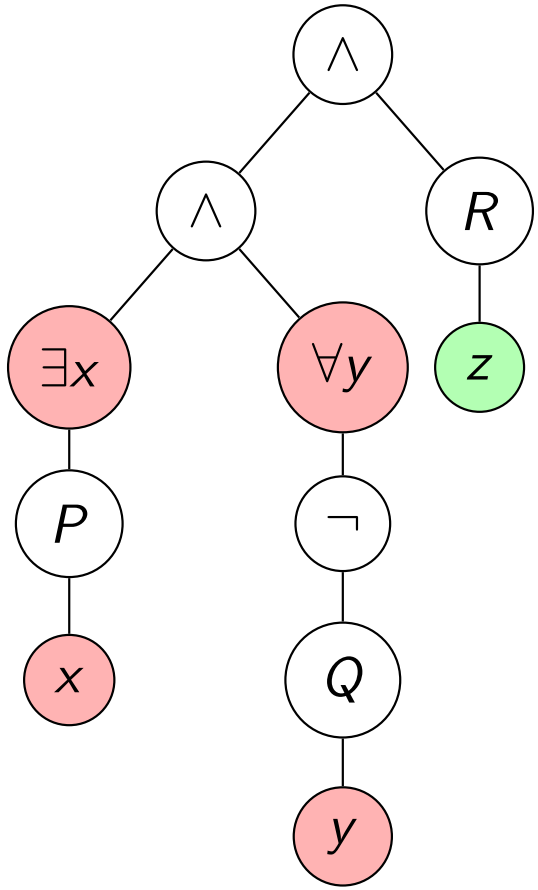
$(\exists x\ P(x)) \wedge Q(x)$

N.B.

When modelling "real" notions, it is very difficult to use open formulas (i.e. non closed formulas).

Free and bound variables: using the syntax tree

It is easy to use the syntax tree of a formula to find if a given variable x is bound or free: if there is an ancestor of the node of x of the form Qx, then x is bound.



Outline of part 4 - FOL language and semantics

Substitutions

As variables are placeholders, we should be able to **replace** them with concrete (or not) **terms**.

Definition (substitution)

Let φ be a wff, x a variable and t a term. $\varphi[x/t]$ denotes the formula obtained by replacing all **free occurrences** of x in φ by t.

You will sometimes find the "contrary" in some textbook, i.e. $[t/x]$ meaning "replace x by t".

Examples:

$P(x)[x/y] \equiv P(y)$
$P(x)[x/x] \equiv P(x)$
$(P(x) \to \forall x\ P(x))[x/y] \equiv (P(y) \to \forall x\ P(x))$

Using the syntax tree of φ, it means replacing all x nodes by the syntax tree of t.

Free substitutions

Substitution should preserve validity in semantics.

Let us consider $\exists y\ P(x, y)$. Can x be substituted by y in this formula?

➥ no, as you change the meaning of the formula!

Definition (free substitution)

A term t is **freely substitutable** to x in φ if

- φ is an atomic formula
- $\varphi \equiv \neg\varphi_1$ and t is freely substituable to x in φ_1
- $\varphi \equiv \varphi_1$ conn φ_2 where conn $\in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$ and t is freely substituable to x in φ_1 and φ_2
- $\varphi \equiv Qy\ \varphi_1$ where $Q \in \{\forall, \exists\}$ and
  - x and y are the same variable
  - y is not a variable of t and t is freely substitutable for x in φ_1

Outline of part 4 - FOL language and semantics

Subformulas

Like in the propositional case, we can define the notion of **subformulas** of a wff.

Definition (subformulas)

The subformulas set of φ denoted by $sf(\varphi)$ is defined inductively as follows:

- if φ is an atomic formula $P(t_1, \ldots, t_n)$, then $sf(\varphi) = \{P(t_1, \ldots, t_n)\}$
- $sf(\neg\varphi) = \{\neg\varphi\} \cup sf(\varphi)$
- if *conn* is a binary connector, $sf(\varphi_1 \ conn \ \varphi_2) = \{\varphi_1 \ conn \ \varphi_2\} \cup sf(\varphi_1) \cup sf(\varphi_2)$
- if $Q \in \{\forall, \exists\}$, $sf(Qx \ \varphi) =$ $\{Qx \ \varphi\} \cup \{\varphi[x/t] \mid$ forall $x \in \mathcal{V}$ and t freely subs. for x in $\varphi\}$

Subformulas: example

Let us consider a FO language whose signature is $\langle \{P/1\}, \{f/1\}, \emptyset$.

The subformulas set of $\forall x\ P(x)$ is

$$\{\forall x\ P(x), P(x), P(f(x)), P(f(f(x))), \ldots\}$$

Thus, if you consider a language with an non-empty set of functions, the subformulas set of a wff can be **infinite**!

Outline of part 4 - FOL language and semantics

Outline of part 4 - FOL language and semantics

Semantics: introduction

A statement in FOL L is **neither true nor false**, because it has **no interpretation**.

➥ e.g. is $\forall x\ M(x) \to \exists y\ F(x, y)$ true or false?

An interpretation must be given to **non-logical** symbols (M and F here).

Examples:

$M \equiv$ ". is a man"
$F \equiv$ ". is the father of ."

The sentence signifies "every man has a father".

$M \equiv$ ". is a country"
$F \equiv$ ". is president of ."

The sentence signifies "every country has a president".

Semantics: interpretation domain

A FOL language is based on **predicates** which express properties verified by **objects**.

➥ we need a set of objects to work on

Can the set of constants defined in the signature be sufficient?

➥ no, because they may be functions!

Moreover, we need to fix the **possible range of (all) the variables**.

Consequence

When wanting to evaluate a formula, we need:

- a set of objects, called **domain of discourse** or **interpretation domain**
- a **mapping** from terms to objets of the interpretation domain

Semantics: evaluating formulas

When trying to evaluate the "truth value" of $P(a)$, we want to verifiy if the object represented by a verifies the property represented by P.

➥ we need to explicitely define all the objects verifying the property represented by P

Thus, the interpretation of P will be a **subset of the interpretation domain**.

What happens when dealing with variables?

- $P(x)$: using the mapping from terms to interpretation domain, verify that the object represented by x verifies P
- $\forall x\ P(x)$: verify that **each element of interpretation domain** verifies P

For composed formulas, the same rules as in PL will be used.

Outline of part 4 - FOL language and semantics

Interpretation

Definition (interpretation)

An interpretation I for FOL is a pair $\langle D_I, \mathcal{I} \rangle$ such that:

- D_I is a non empty set of objects (or individuals) called **interpretation domain**.
- $\mathcal{I}$ is an **interpretation function** s.t.:
  - every predicate symbol P/n is associated to a subset of D_i^n
  - every constant symbol a is associated to $\mathcal{I}(a) \in D_i$
  - every function symbol f/n is associated to a function from D_I^n in D_I

Interpretation: a first example

The two following examples are taken from Huth and Ryan 2004.

Huth, Michael and Mark Ryan (2004).
Logic in Computer Science – Modelling and reasoning about systems.
Cambridge University Press.

Interpretation: a first example

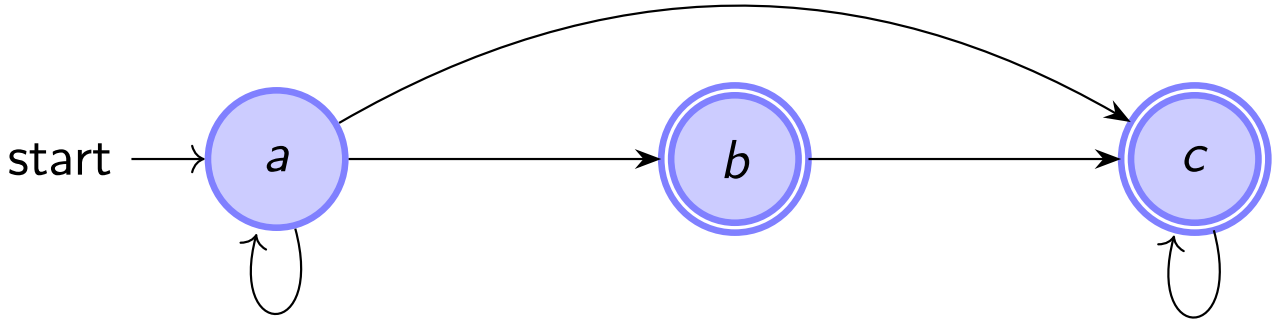
Let us consider a FO language with the following signature: $\langle \{F/1, R/2\}, \emptyset, \{i\} \rangle$.

We will use this language to represent a transition system:

$F(x)$	x is a final state
$R(x, y)$	there is a transition between x and y
i	a constant representing the initial state

Interpretation: a first example

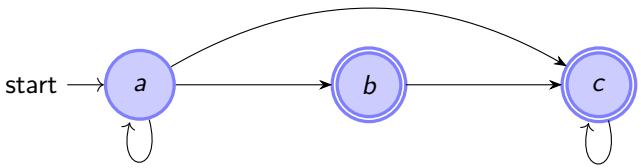
Let us consider the following transition system:



An intepretation for representing this system can be the following:

$$
\begin{aligned}
D_I &= \{a, b, c\} \\
\mathcal{I}(i) &= a \\
\mathcal{I}(F) &= \{b, c\} \\
\mathcal{I}(R) &= \{(a, a), (a, b), (a, c), (b, c), (c, c)\}
\end{aligned}
$$

Interpretation: a first example

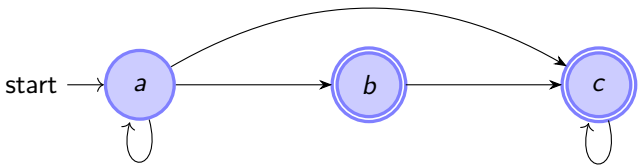


$$D_I = \{a, b, c\}$$
$$\mathcal{I}(i) = a$$
$$\mathcal{I}(F) = \{b, c\}$$
$$\mathcal{I}(R) = \{(a, a), (a, b), (a, c), (b, c), (c, c)\}$$

We can try to "explain" intuitively some formulas on this interpretation:

$$\exists y \ R(i, y)$$

Interpretation: a first example



$$D_I = \{a, b, c\}$$
$$\mathcal{I}(i) = a$$
$$\mathcal{I}(F) = \{b, c\}$$
$$\mathcal{I}(R) = \{(a, a), (a, b), (a, c), (b, c), (c, c)\}$$

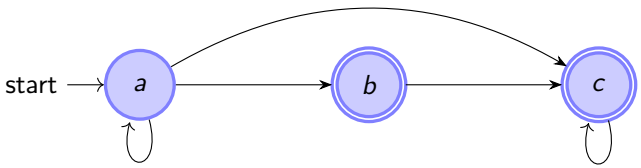
We can try to "explain" intuitively some formulas on this interpretation:

$$\exists y \ R(i, y)$$

"there a transition from the initial state to some state"

➥ true, see from example that $(a, b) \in \mathcal{I}(R)$

Interpretation: a first example

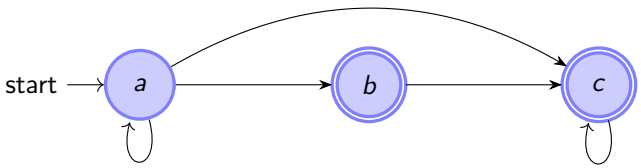


$$D_I = \{a, b, c\}$$
$$\mathcal{I}(i) = a$$
$$\mathcal{I}(F) = \{b, c\}$$
$$\mathcal{I}(R) = \{(a, a), (a, b), (a, c), (b, c), (c, c)\}$$

We can try to "explain" intuitively some formulas on this interpretation:

$$\neg F(i)$$

Interpretation: a first example



$$D_I = \{a, b, c\}$$
$$\mathcal{I}(i) = a$$
$$\mathcal{I}(F) = \{b, c\}$$
$$\mathcal{I}(R) = \{(a, a), (a, b), (a, c), (b, c), (c, c)\}$$

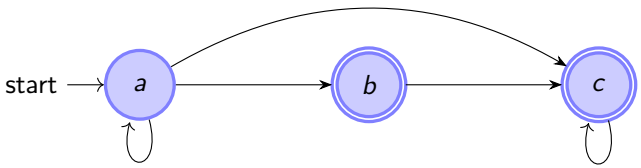
We can try to "explain" intuitively some formulas on this interpretation:

$$\neg F(i)$$

"the initial state is not a final state"

➥ true, see $\mathcal{I}(F)$

Interpretation: a first example

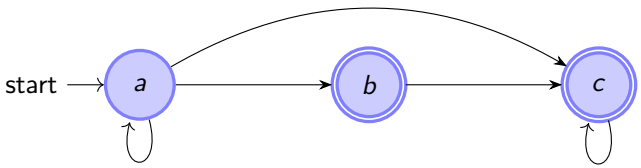


$$D_I = \{a, b, c\}$$
$$\mathcal{I}(i) = a$$
$$\mathcal{I}(F) = \{b, c\}$$
$$\mathcal{I}(R) = \{(a, a), (a, b), (a, c), (b, c), (c, c)\}$$

We can try to "explain" intuitively some formulas on this interpretation:

$$\forall x \forall y \forall z \ (R(x, y) \wedge R(x, z) \rightarrow y = z)$$

Interpretation: a first example



$$D_I = \{a, b, c\}$$
$$\mathcal{I}(i) = a$$
$$\mathcal{I}(F) = \{b, c\}$$
$$\mathcal{I}(R) = \{(a, a), (a, b), (a, c), (b, c), (c, c)\}$$

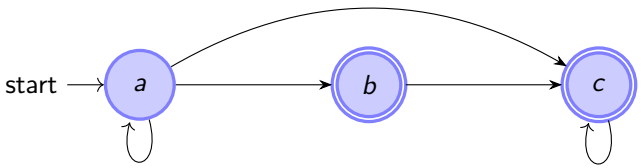
We can try to "explain" intuitively some formulas on this interpretation:

$$\forall x \forall y \forall z \ (R(x, y) \wedge R(x, z) \rightarrow y = z)$$

"the transition relation is deterministe"

➥ false, $(a, a) \in \mathcal{I}(R)$, $(a, b) \in \mathcal{I}(R)$ and $(a, c) \in \mathcal{I}(R)$

Interpretation: a first example

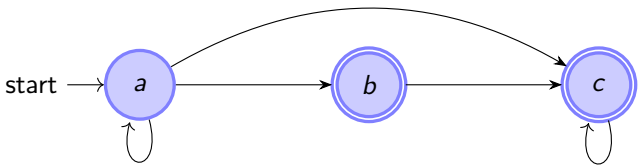


$$D_I = \{a, b, c\}$$
$$\mathcal{I}(i) = a$$
$$\mathcal{I}(F) = \{b, c\}$$
$$\mathcal{I}(R) = \{(a, a), (a, b), (a, c), (b, c), (c, c)\}$$

We can try to "explain" intuitively some formulas on this interpretation:

$$\forall x \exists y \ R(x, y)$$

Interpretation: a first example



$$D_I = \{a, b, c\}$$
$$\mathcal{I}(i) = a$$
$$\mathcal{I}(F) = \{b, c\}$$
$$\mathcal{I}(R) = \{(a, a), (a, b), (a, c), (b, c), (c, c)\}$$

We can try to "explain" intuitively some formulas on this interpretation:

$$\forall x \exists y \ R(x, y)$$

"there is no deadlock state"

➥ true, see $\mathcal{I}(R)$

Interpretation: a second example (more complex)

Let us consider a FO language with the following signature:
$\langle \{\leq /2\}, \{\bullet/2\}, \{e\} \rangle$ ($\leq$ and $\bullet$ will be used in infix notation).

We will use this language to represent an order on **binary strings**:

e	a constant representing the empty word
$x \bullet y$	represents the concatenation of x and y
$x \leq y$	x is a prefix of y

Interpretation: a second example (more complex)

Let us consider a FO language with the following signature:
$\langle \{\leq /2\}, \{\bullet/2\}, \{e\} \rangle$ ($\leq$ and $\bullet$ will be used in infix notation).

We will use this language to represent an order on **binary strings**:

e	a constant representing the empty word
$x \bullet y$	represents the concatenation of x and y
$x \leq y$	x is a prefix of y

An interpretation for representing the order on binary strings can be the following:

$$
\begin{aligned}
D_I &= \text{the set of finite words on the alphabet } \{0,1\} \\
\mathcal{I}(e) &= \epsilon \\
\mathcal{I}(\bullet) &= \text{string concatenation over } D_I \\
\mathcal{I}(\leq) &= \text{string natural prefix ordering over } D_I
\end{aligned}
$$

Formula evaluation: introduction

Let us consider the previous example on transition systems:

$$
\begin{aligned}
D_I &= \{a, b, c\} \\
\mathcal{I}(i) &= a \\
\mathcal{I}(F) &= \{b, c\} \\
\mathcal{I}(P) &= \{(a, a), (a, b), (a, c), (b, c), (c, c)\}
\end{aligned}
$$

Intuitively, $\exists x\ F(x)$ holds in I: for instance $\langle b \rangle \in \mathcal{I}(F)$.

In other words, there is an **assignment of variables** in I denoted by σ s.t. "formula $F(x)$ is true in I given σ"

This will be noted: there is σ s.t. $\models_{I,\sigma} F(x)$.

We will then deduce $\models_I \exists x\ F(x)$.

Outline of part 4 - FOL language and semantics

Terms evaluation

Definition (assignment)

An assignment σ of I is a function from the set of variables symbols to D_I.
For $x \in \mathcal{V}$ and $a \in D_I$, we will denote by $\sigma[x \mapsto a]$ the assignment in which x is assigned to a and for all $y \in \mathcal{V}$ different from x, $\sigma[x \mapsto a](y) = \sigma(y)$.

Definition (terms evaluation)

The evaluation of a term under an interpretation I and an assignment σ is defined inductively as follows:

- $val_{I,\sigma}(a) = \mathcal{I}(a)$ if a is a constant symbol
- $val_{I,\sigma}(x) = \sigma(x)$ if x is a variable symbol
- $val_{I,\sigma}(f(t_1, \ldots, t_n)) = \mathcal{I}(f)(val_{I,\sigma}(t_1), \ldots, val_{I,\sigma}(t_n))$

Formula evaluation: truth value

Definition (truth value)

The truth value of a wff φ under an interpretation I and an assignment σ, denoted by $[\![\varphi]\!]_{I,\sigma}$ is defined by:

- if $P(t_1, \ldots, t_n)$ is an atomic wff, $[\![P(t_1, \ldots, t_n)]\!]_{I,\sigma} = T$ iff $\langle val_{I,\sigma}(t_1), \ldots, val_{I,\sigma}(t_n)\rangle \in \mathcal{I}(P)$
- $[\![\top]\!]_{I,\sigma} = T$ and $[\![\bot]\!]_{I,\sigma} = F$
- $[\![\neg\varphi]\!]_{I,\sigma} = f_\neg([\![\varphi]\!]_{I,\sigma})$
- $[\![\varphi \vee \psi]\!]_{I,\sigma} = f_\vee([\![\varphi]\!]_{I,\sigma}, [\![\psi]\!]_{I,\sigma})$
- $[\![\varphi \wedge \psi]\!]_{I,\sigma} = f_\wedge([\![\varphi]\!]_{I,\sigma}, [\![\psi]\!]_{I,\sigma})$
- $[\![\varphi \to \psi]\!]_{I,\sigma} = f_\to([\![\varphi]\!]_{I,\sigma}, [\![\psi]\!]_{I,\sigma})$
- $[\![\varphi \leftrightarrow \psi]\!]_{I,\sigma} = f_\leftrightarrow([\![\varphi]\!]_{I,\sigma}, [\![\psi]\!]_{I,\sigma})$
- $[\![\forall x\ \varphi(x)]\!]_{I,\sigma} = T$ iff for all $d \in D_I$ $[\![\varphi(x)]\!]_{I,\sigma[x\mapsto d]} = T$

Outline of part 4 - FOL language and semantics

Satisfaction, validity, models

We use the same definition as in the PL case:

Definition

- φ is **satisfiable in** I **under** σ $[\![\varphi(x)]\!]_{I,\sigma} = T$.
  This is noted $\models_{I,\sigma} \varphi$.
- φ is **satisfiable in** I (i.e. I satisfies φ) iff for every assignment σ in I, $\models_{I,\sigma} \varphi$ holds.
  This is noted $\models_I \varphi$ and I is said to be a **model** of φ.
- φ is **satisfiable** iff φ has a model
- φ is **valid** iff every interpretation I is a model of φ
- φ is **contradictory** iff for all interpretation I $\models_I \neg\varphi$.
- let Σ be a set of formulas. I is a model of Σ iff I is a model of every formula in Σ.

NB: there is a formal difference between tautologies and valid wff in FOL.

Logical consequence

The same as in PL:

Definition (logical consequence)

Let φ and ψ be two wffs. φ is a **logical consequence** of ψ, denoted by $\psi \models \varphi$, iff for every interpretation $\mathcal{I}$ and for every assignement σ, $\models_{I,\sigma} \psi$ implies $\models_{I,\sigma} \varphi$.

Definition (logical consequence of a set)

Let $n \in \mathbb{N}^*$, $\Sigma = \{\psi_1, \ldots, \psi_n\}$ be a set of wffs and φ be a wff. φ is a **logical consequence** of Σ iff for every interpretation $\mathcal{I}$, for every assignement σ and for all $\psi \in \{\psi_1, \ldots, \psi_n\}$, $\models_{I,\sigma} \psi$ implies $\models_{I,\sigma} \varphi$.

This noted $\Sigma \models \varphi$.

Logical consequence: some remarks

The definition of logical consequence seems more complicated than the "intuitive" one you may think about. Why do we not use the following definition: "$\psi \models \varphi$ iff for all interpretation I, if $\models_I \psi$ then $\models_I \varphi$"?

The main problem is the eventual use of **free variables** in formulas.

Think about the following logical consequence: $P(x) \models \forall x\ P(x)$. Does it have to hold?

You can convince yourself by using $\mathbb{N}$ as interpretation domain and P as a predicate signifying "to be even".

Notice that if you use only sentences (i.e. closed formulas), then the two definitions are equivalent.

Outline of part 4 - FOL language and semantics

Useful theorems, like in the PL case

Theorem (deduction)

$\Sigma, \varphi \models \psi$ *iff* $\Sigma \models \varphi \rightarrow \psi$

This theorem allows to reduce the "validity of an argument" problem to the "validity of a wff" problem (but this problem is more complicated in the FO case!).

Theorem

$\Sigma \models \varphi$ *iff* $\Sigma \cup \{\neg\varphi\}$ *is not satisfiable.*

Again, this is kind of a "reductio ad absurdum" theorem for FOL that will be useful for the Resolution formal system.

Compactness of FOL: a first (and important) drawback

Theorem (compactness of FOL)

*Let Σ be a set of **closed** FOL wff. Then Σ is satisfiable iff every finite subset of Σ is satisfiable.*

This theorem is really important to understand the limitations of FOL concerning expressiveness.

Think also about this equivalent formulation of the theorem: Σ is unsatisfiable iff there is a finite subset of Σ that is unsatisfiable.

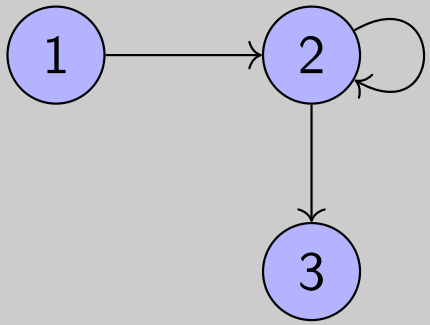
Compactness of FOL: an example

We want to define **reachability in a graph** using FOL. Suppose that we define a language with signature $\langle \{G/2, R/2\}, \emptyset, \emptyset \rangle$ where:

- $G(x, y)$ means that there is a direct edge from x to y
- $R(x, y)$ means that y is reachable from x, i.e. that there is a **finite** path from x to y

Can we really define $R(x, y)$?

Example



Is ③ reachable from ① ?

The Löwenheim-Skolem theorem

Theorem (Löwenheim-Skolem)

Let $\mathcal{S}$ be a signature, I an interpretation on $\mathcal{S}$ and K an infinite cardinal numbers. Then there is an interpretation I' on $\mathcal{S}$ s.t. $|D_{I'}| = K$ and

- *if $K < |D_I|$ then I' is an elementary submodel of I*
- *if $K > |D_I|$ then I' is an elementary extension of I*

Hu?

It means that you cannot control the cardinalities of models of first-order theories with an infinite model.

So for instance:

- the FO theory of $(\mathbb{N}, +, \times, 0, 1)$ has uncountable models...
- the FO theory of $(\mathbb{R}, +, \times, 0, 1)$ has a countable model...

Outline of part 4 - FOL language and semantics

Undecidability of FOL

Question: is the set of valid wffs in FOL decidable (what Hilbert has called the *Entscheidungsproblem*)?

Unfortunately, the answer is **no** for a first-order language with at least a predicate of arity 2.

Church, Alonzo (1936).
"An unsolvable problem of elementary number theory".
In: **American Journal of Mathematics** 58,
Pp. 345–363.

Turing, Alan (1937).
"On computable numbers, with an application to the Entscheidungsproblem".
In: **Proceedings of the London Mathematical Society** 42,
Pp. 230–265.

Semidecidability of FOL

But we have a semi-decidability result for FOL:

Theorem (semidecidability of first-order logic)

The problem of finding if a first-order logic formula is valid or not is semidecidable.

This means that there is an algorithm which:

- halts and returns yes if the answer to the problem is yes
- halts and returns no or **never halts** if the answer to the problem is no

So, are we doomed?

The semidecidability of FOL is of course a big drawback, but there are tractable fragments of FOL that are interesting:

- propositional logic of course ☺
- monadic predicate logic, i.e. FOL restricted to unary predicate symbols and no function symbols
- description logics, which is a subset of FOL used for knowledge representation (ontologies, Semantic Web, etc.)
- some modal logics
- etc.

Notice of course that verifying that a wff is valid with a "truth table" algorithm is impossible: we have to checked that the formula is true in every possible interpretation and in particular in every possible domain!

But we will see an interesting algorithm in the next slides...

Outline of part 4 - FOL language and semantics

Skolem standard form

We have seen that in PL, CNF of a wff φ that is useful to determine the validity of φ (and is used in the Resolution formal system). Is there a corresponding form in FOL?

➥ yes, this is **Skolem standard form**

Questions:

- is there a Skolem standard form for each wff?
- how to obtain a Skolem standard form for a given wff φ?
- is the Skolem standard form for φ logically equivalent to φ?
- how can we use Skolem standard form to determine the validity of a wff?

We will obtain a Skolem standard form for a given wff using the steps presented in the next slides.

Skolem standard form: conjunctive prenex form

Step 1: obtain a **conjunctive prenex form**

- rename bound variables to avoid using the same variable in different scopes (**important!**)
- transform a closed FOL wff into an logically equivalent wff which is in **prenex** form: this is a wff in which **all** quantifiers are at the beginning of the formula.

$$\varphi \rightsquigarrow \underbrace{Q_1 x_1 \dots Q_n x_n}_{\text{prefix}} \underbrace{\varphi'}_{\text{matrix}}$$

- transform matrix into a formula in CNF

$\varphi \rightsquigarrow Q_1 x_1 \dots Q_n x_n \;\; M[x_1, \dots, x_n]$ where M is a formula in CNF.

Skolem standard form: conjunctive prenex form

Rules that can be applied to obtain a prenex form (the translation of φ' into a CNF has been treated in the PL case):

$$\neg(\forall x\ \varphi) \equiv \exists x\ (\neg\varphi) \tag{1}$$

$$\neg(\exists x\ \varphi) \equiv \forall x\ (\neg\varphi) \tag{2}$$

$$(\forall x\ \varphi_1) \vee \varphi_2 \equiv \forall x\ (\varphi_1 \vee \varphi_2) \quad \text{if } x \notin FV(\varphi_2) \tag{3}$$

$$(\exists x\ \varphi_1) \vee \varphi_2 \equiv \exists x\ (\varphi_1 \vee \varphi_2) \quad \text{if } x \notin FV(\varphi_2) \tag{4}$$

$$(\forall x\ \varphi_1) \wedge \varphi_2 \equiv \forall x\ (\varphi_1 \wedge \varphi_2) \quad \text{if } x \notin FV(\varphi_2) \tag{5}$$

$$(\exists x\ \varphi_1) \wedge \varphi_2 \equiv \exists x\ (\varphi_1 \wedge \varphi_2) \quad \text{if } x \notin FV(\varphi_2) \tag{6}$$

$$\varphi_1 \vee (\forall x\ \varphi_2) \equiv \forall x\ (\varphi_1 \vee \varphi_2) \quad \text{if } x \notin FV(\varphi_1) \tag{7}$$

$$\varphi_1 \vee (\exists x\ \varphi_2) \equiv \exists x\ (\varphi_1 \vee \varphi_2) \quad \text{if } x \notin FV(\varphi_1) \tag{8}$$

$$\varphi_1 \wedge (\forall x\ \varphi_2) \equiv \forall x\ (\varphi_1 \wedge \varphi_2) \quad \text{if } x \notin FV(\varphi_1) \tag{9}$$

$$\varphi_1 \wedge (\exists x\ \varphi_2) \equiv \exists x\ (\varphi_1 \wedge \varphi_2) \quad \text{if } x \notin FV(\varphi_1) \tag{10}$$

Skolem standard form: conjunctive prenex form

Theorem (logical equivalence)

Let φ be a wff, then there is a wff φ_{pr} in conjunctive prenex form obtained using the previous rules that is logically equivalent to φ.

Beware of the $\rightarrow$ connector! For instance, find the prenex form of $(\forall x\ \varphi_1) \rightarrow \varphi_2$.

Skolem standard form: quantifiers elimination

Step 2: eliminate existential then universal quantifiers

- eliminate every symbol $\exists x_i$ in the prefix
- replace in matrix every occurrence of x_i by a symbol $f_i(x_{i_0}, \ldots, x_{i_k})$ such that:
  - f_i is a new function symbol called **Skolem function**
  - $x_{i_0}, \ldots, x_{i_k}$ are variables symbols such that each of these variables is universally quantified and appears before $\exists x_i$ in the prefix
- if there is no universally quantified variable appearing before x_i in the prefix, x_i is replaced by a new constant α_i called **Skolem constant**
- a formula $\forall x_1 \ \ldots \ \forall x_r \ (C_1 \wedge \ldots \wedge C_r)$ s.t. every C_i is a clause is obtained
- $\forall x_i$ are eliminated by convention. $C_1 \wedge \ldots \wedge C_r$ is therefore obtained

Skolem standard form: quantifiers elimination

What about logical equivalence? Unfortunately, logical equivalence between a wff and its Skolem standard form is not true.
For instance, think about $\exists x\ P(x)$ and $P(\alpha)$.

We have a weaker result:

Theorem (satisfiability for Skolem standard form)

Let φ be a wff, then there is a wff φ_{sk} in Skolem standard form using the previous rules such that φ is satisfiable iff φ_{sk} is satisfiable.

Skolem standard form: clausal form and set of wffs

Step 3: obtain a clausal form

- by **convention**, the previous formula is represented by the set of clauses $\{D_1, \ldots, D_r\}$ called the clausal form of the formula
- the clausal form of a set of wffs E is the union of the clausal forms of the wffs appearing in E

Skolem standard form: clausal form and set of wffs

That is the final step of the translation. There is an important result:

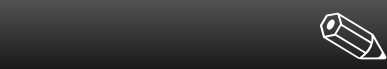
Theorem (satisfiability for clausal form)

Let φ be a wff and $Cl(\varphi)$ the set of clausal forms obtained from φ using the previous procedure. φ is satisfiable iff $Cl(\varphi)$ is satisfiable.

Let E be a set of wffs and $Cl(E)$ the set of clausal forms obtained from E using the previous procedure. E is satisfiable iff $Cl(E)$ is satisfiable.

For the Resolution formal system, think about this alternative formulation of the theorem: E is unsatisfiable iff $Cl(E)$ unsatisfiable...

Skolem standard forms



Exercise

Give a Skolem standard form of the following formulas:

- $\forall x\ (H(x) \to ((\exists y\ F(x,y)) \wedge (\exists z\ M(x,z)))$.
- $(\forall x\ P(x)) \to (\exists x\ Q(x))$
- $\forall x \forall y\ (\exists u\ Q(x,y,u) \vee \neg(\exists z\ P(x,z) \wedge P(y,z)))$

Herbrand universe

The previous theorem is inapplicable: it must be shown that $Cl(E)$ cannot be satisfied by **every possible interpretation**, particularly using **every possible domain**!

We will use a special domain built from constant and function symbols defined in the language. This domain and the algorithm presented in the next slides are the work of Jacques Herbrand.

Informally, the domain we will consider is built from the **ground terms** that can be built from constants and functions appearing in a set of clauses.

Herbrand universe

Definition (Herbrand universe)

Let S be a set of clauses. Let H_0 be the set of constants appearing in S (if no constant appears in S, let us define $H_0 = \{a\}$).
Let $i \in \mathcal{N}$.
$H_{i+1} = H_i \cup \{f(t_1, \ldots, f_{n_f}) \mid f$ appears in S and $\forall j \in \{1, \ldots, n_f\}\ t_j \in H_i\}$.
H_∞ is called the **Herbrand universe of** S.

Example: if $S = \{P(a), Q(f(x), y)\}$, then $H_\infty = \{a, f(a), f(f(a)), \ldots\}$

Definition (Herbrand base)

Let S be a set of clauses. The **Herbrand base of** S is the set of **ground instances** of atomic formulas built from predicate symbols appearing in S.

Herbrand interpretation

Definition (Herbrand interpretation)

Let S be a set of clauses. An FOL interpretation $I_H = \langle D_I, \mathcal{I}_H \rangle$ is an **Herbrand interpretation** on S iff:

- $D_I = H_\infty$
- if a is a constant symbol appearing in S, $\mathcal{I}_H(a) = a$
- let f be a function symbol appearing in S and $h_1, \ldots, h_n$ be elements of H_∞. Then $\mathcal{I}_H(f)$ maps $(h_1, \ldots, h_n)$ to $f(h_1, \ldots, h_n)$

Concerning predicates interpretation, this is often represented by a subset of the Herbrand base.

For instance, considering $S = \{P(a), Q(f(x))\}$,
$\mathcal{I}_H = \{P(a), P(f(a)), Q(f(a))\}$ means that $\mathcal{I}_H(P) = \langle a, f(a) \rangle$ and
$\mathcal{I}_H(Q) = \langle f(a) \rangle$

Herbrand interpretation

Theorem

Let S be a set of clauses. If there is an interpretation $\mathcal{I}$ that satisfies S then there is an Herbrand interpretation that satisfies S.

Herbrand's theorem

Theorem (Herbrand)

*A set of clauses is unsatisfiable iff there is a **finite** subset of **ground clauses** of S instanciated with the Herbrand universe of S that is unsatisfiable.*

This theorem gives us a refutation procedure (remember the link with the validity of an argument problem)!

- build H_1
- build S_1: set of clauses of S instanciated only with H_1 terms
- verify with a **propositional procedure** like Davis and Putnam procedure that S_1 is unsatisfiable
- if not, restart with H_2
- ...

5 - Formal systems for first-order logic

Outline of part 5 - FOL formal systems

Previously on mathematical logic...

A formal system is composed of two elements:

- a **formal language** (alphabet + grammar) defining a set of expressions E
- a **deductive system** or **deductive apparatus** on E

Definition (deductive system)

A **deduction system** (or **inference system**) on a set E is composed of a set of rules used to derive elements of E from other elements of E. They are called **inference rules**.

If an inference rule allows to derive e_{n+1} (conclusion) from $\mathcal{P} = \{e_1, \ldots, e_n\}$ (premises), it will be noted as follows:

$$\frac{e_1 \;\; e_2 \;\; \ldots \;\; e_n}{e_{n+1}}$$

When an inference rule is such that $\mathcal{P} = \emptyset$ it is called an **axiom**.

If e_1 is an axiom, it is either noted $\frac{}{e_1}$ or simply e_1.

A completeness theorem for FOL

An important property for formal systems is the **completeness property**: given a **valid** wff φ (i.e. $\models \varphi$), is there a **deduction** of φ (i.e. $\vdash \varphi$)?

Gödel has shown in his doctoral dissertation that this property is true for FOL.

Gödel, Kurt (1929).
"Über die Vollständigkeit des Logikkalküls".
Doctoral dissertation. University of Vienna.

Hilbert's formal system for FOL: $\mathcal{H}$

Definition (axioms of $\mathcal{H}$)

A1 $P \to (Q \to P)$

A2 $(P \to (Q \to R)) \to ((P \to Q) \to (P \to R))$

A3 $(\neg P \to \neg Q) \to ((\neg P \to Q) \to P)$

A4 $\forall x\ (P \to Q) \to (P \to \forall x\ Q)$ where x is not free in P

A5 $\forall x\ P \to P[x/t]$ where x is freely substituable by t in P

Hilbert's formal system for FOL: $\mathcal{H}$

Definition (Modus Ponens rule (MP))

From A and $A \to B$ infer B:

$$\frac{A \qquad A \to B}{B}\ (MP)$$

Definition (generalization rule ($\forall$))

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

Gentzen's formal system for FOL: $\mathcal{G}$

The same inference rules as in PL are used + 2 rules for $\forall$:

Definition

suppl. inference rules for $\mathcal{G}$

$$\frac{A[x/t], \forall x\ A, \Gamma \Rightarrow \Delta,}{\forall x\ A, \Gamma \Rightarrow \Delta}\ (\forall l) \qquad \frac{\Gamma \Rightarrow A[x/y], \Delta}{\Gamma \Rightarrow \forall x\ A, \Delta}\ (\forall r)$$

where in $(\forall r)$, y is not free in the sequent conclusion and is not a free variable of A.

Examples of proof in $\mathcal{G}$

Prove that $\forall x\ P(x) \to \exists y\ P(y)$, i.e. that $\forall x\ P(x) \Rightarrow \neg\forall y\ \neg P(y)$ is valid:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{P(t_0), \forall x\ P(x), \forall y\ \neg P(y) \Rightarrow P(t_0)}}{P(t_0), \forall x\ P(x), \neg P(t_0), \forall y\ \neg P(y) \Rightarrow}\,(\neg l)}{P(t_0), \forall x\ P(x), \forall y\ \neg P(y) \Rightarrow}\,(\forall l)}{\forall x\ P(x), \forall y\ \neg P(y) \Rightarrow}\,(\forall l)}{\forall x\ P(x) \Rightarrow \neg\forall y\ \neg P(y)}\,(\neg r)$$

But if we always use $(\forall l)$:

$$\dfrac{\dfrac{\dfrac{\vdots}{P(t_0), P(t_1), \forall x\ P(x) \Rightarrow \neg\forall y\ \neg P(y)}\,(\forall l)}{P(t_0), \forall x\ P(x) \Rightarrow \neg\forall y\ \neg P(y)}\,(\forall l)}{\forall x\ P(x) \Rightarrow \neg\forall y\ \neg P(y)}\,(\forall l)$$

Outline of part 5 - FOL formal systems

Outline of part 5 - FOL formal systems

Skolem standard form

The formal language used in the Resolution formal system is based on the Skolem standard form.

Remember that you obtain a Skolem standard form from a wff φ by applying the following procedure:

1. transform φ into a conjunctive prenex form, i.e. a formula $Q_1 x \ldots Q_n x\ \varphi'$ where φ' is a formula without quantifier and in CNF
2. eliminate existential quantifiers by using Skolem constants and functions
3. remove universal quantifiers by convention

You obtain a formula $D_1 \wedge \ldots \wedge D_n$ where each D_i is a clause and where every variable is implicitely universally quantified.

A formula φ and its Skolem standard form φ_{sk} are not necessarily logically equivalent, but φ is satisfiable iff φ_{sk} is satisfiable.

Set of clauses

We will use in fact the **clausal form** of φ, i.e. the set $\{D_1, \ldots, D_n\}$ if $\varphi_{Sk} = D_1 \wedge \ldots \wedge D_n$.
The clausal form of a wff φ is denoted by $CL(\varphi)$. The usual extension to set of wffs can be done.

N.B. (important)

Variables should be renamed between the different clauses (cf. further), otherwise you will not be able to apply Resolution rule in some cases!

Outline of part 5 - FOL formal systems

Introduction and some intuition

In PL, Resolution base idea was to find complementary pairs like $\{A, \neg A\}$ to show that a **set of clauses is unsatisfiable**.

Let us look at some FO clauses and see if the Resolution principle can be applied.

Case 1: all terms are **ground**.

clauses set	deduced wff	OK?
$\{P(a), \neg P(a)\}$	$\square$	✔
$\{P(a,b), \neg P(a,b)\}$	$\square$	✔
$\{P(a) \vee Q(b), \neg P(a) \vee R(c)\}$	$Q(b) \vee R(c)$	✔

Easy, can be reduced to PL case.

In PL, Resolution base idea was to find complementary pairs like $\{A, \neg A\}$ to show that a **set of clauses is unsatisfiable**.

Let us look at some FO clauses and see if the Resolution principle can be applied.

Case 2: there are some variables that are not useful for the Resolution rule.

clauses set	deduced wff	OK?
$\{P(a) \vee Q(x), \neg P(a) \vee R(y)\}$	$Q(x) \vee R(y)$	✔

Again, this is an easy case.

In PL, Resolution base idea was to find complementary pairs like $\{A, \neg A\}$ to show that a **set of clauses is unsatisfiable**.

Let us look at some FO clauses and see if the Resolution principle can be applied.

Case 3: there are some variables that must be used in the Resolution rule with some ground terms.

clauses set	deduced wff	OK?
$\{P(x), \neg P(a)\}$	$\square$	✔
$\{P(x) \vee Q(b), \neg P(a) \vee R(c)\}$	$Q(b) \vee R(c)$	✔
$\{P(x) \vee Q(x), \neg P(a) \vee R(c)\}$	$Q(\,a\,) \vee R(c)$	✔

Remember that **the variables are universally quantified**, so you can **substitute** the variables by some other terms (remember Herbrand theorem!).

Notice that you have to substitute all the concerned variable instances.

In PL, Resolution base idea was to find complementary pairs like $\{A, \neg A\}$ to show that a **set of clauses is unsatisfiable**.

Let us look at some FO clauses and see if the Resolution principle can be applied.

Case 4: there are some variables that must be used in the Resolution rule with some variables.

clauses set	deduced wff	OK?
$\{P(x), \neg P(y)\}$	$\square$	✔
$\{P(x), \neg P(f(y))\}$	$\square$	✔
$\{P(x) \vee Q(x), \neg P(f(y)) \vee R(y)\}$	$Q(\ f(y)\ ) \vee R(y)$	✔

Again, as the variables are universally quantified, it should not be difficult.

In the second and the third cases, we can choose for instance $[x/f(a), y/a]$, but we will choose the **most general substitution**.

Substitution and instance

In order to be able to apply Resolution rule, we have to use **substitution** on variables. We have already seen the definitions:

Definition (substitution)

A **substitution** is defined by a set $[\ldots, v_i/t_i, \ldots]$ where each v_i is a variable and each t_i a term different from v_i.

Definition (instance)

Let $\theta = [\ldots, v_i/t_i, \ldots]$ be a substitution. Let φ be a FOL wff. Then $\theta(\varphi)$ is the formula obtained from φ by replacing each free instance of v_i in φ by t_i. $\theta(\varphi)$ is called **instance** of φ.

Example:

$\theta = [x/a, y/f(b), z/c]$
$\varphi = P(x, y, z)$
$\theta(\varphi) = P(a, f(b), c)$

Unification

What we want in the Resolution principle is substitutions that "make the formulas be indentical". This is defined by the notion of **unifiers**.

Definition (unifier)

A substitution θ is an **unifier** for the set $\{\varphi_1, \ldots, \varphi_n\}$ iff $\theta(\varphi_1) = \ldots = \theta(\varphi_n)$. $\{\varphi_1, \ldots, \varphi_n\}$ is said to be **unifiable**.

Example: $\{P(x), P(a)\}$ is unifiable using $\theta = \{x/a\}$

Definition (most general unifier)

An unifier σ for a set $\{\varphi_1, \ldots, \varphi_n\}$ is called **most general unifier** iff for all unifier θ there is a substitution λ such that $\theta = \sigma \circ \lambda$.

Example: the most general unifier of $\{P(x), P(f(y))\}$ is $[x/f(y)]$

Disagreement set

In order to unify expressions, we must find the "disagreements" between them and variables substitutions will solve those disagreements.

Can we find automatically the correct substitutions, i.e. the most general unifiers?

Definition (disagreement set)

The disagreement set of a set of expressions W is obtained by the following procedure:

- find the position of the first **symbol** (starting from left) on which the expressions disagree
- extract for each expression the sub-expression beginning with the symbol at the previously found position

Disagreement set

In order to unify expressions, we must find the "disagreements" between them and variables substitutions will solve those disagreements.

Can we find automatically the correct substitutions, i.e. the most general unifiers?

For instance, for $\{P(x, f(y, z)), P(x, a), P(x, g(h(k(x))))\}$:

Disagreement set

In order to unify expressions, we must find the "disagreements" between them and variables substitutions will solve those disagreements.

Can we find automatically the correct substitutions, i.e. the most general unifiers?

For instance, for $\{P(x, \underline{f(y,z)}), P(x, \underline{a}), P(x, \underline{g(h(k(x)))})\}$:

- the disagreement symbol is in position 4
- the disagreement set is $\{f(y,z), a, g(h(k(x)))\}$

Unification algorithm

Function unify(W)

Input: a set of clauses W

Output: a most general unifier σ if it exists, NO else

```
1  k ← 0 ;
2  W_0 ← W ;
3  σ_k ← ϵ ;
4  while |W_k| ≠ 1 do
5  |   D_k ← disagreement set of W_k ;
6  |   if there is v_k, t_k in D_k s.t. v_k ∉ t_k then
7  |   |   σ_{k+1} = σ_k ∘ [v_k/t_k] ;
8  |   |   W_{k+1} ← W_k{v_k/t_k} ;
9  |   |   k ← k + 1 ;
10 |   else
11 |   |   return NO ;
12 |   end
13 end
14 return σ_k ;
```

Unification algorithm: example

Is there a most general unifier of $\{P(a, x, f(g(y))), P(z, f(z), f(u))\}$?

Unification algorithm: example

Is there a most general unifier of $\{P(a, x, f(g(y))), P(z, f(z), f(u))\}$?

1. $\sigma_0 = \epsilon$ and $W_0 = \{P(a, x, f(g(y))), P(z, f(z), f(u))\}$
   - $D_0 = \{a, z\}$

Unification algorithm: example

Is there a most general unifier of $\{P(a, x, f(g(y))), P(z, f(z), f(u))\}$?

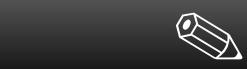
1. $\sigma_0 = \epsilon$ and $W_0 = \{P(a, x, f(g(y))), P(z, f(z), f(u))\}$
   - $D_0 = \{a, z\}$
   - $v_0 = z$ and $t_0 = a$

Unification algorithm: example

Is there a most general unifier of $\{P(a, x, f(g(y))), P(z, f(z), f(u))\}$?

1. $\sigma_0 = \epsilon$ and $W_0 = \{P(a, x, f(g(y))), P(z, f(z), f(u))\}$
   - $D_0 = \{a, z\}$
   - $v_0 = z$ and $t_0 = a$
   - therefore $\sigma_1 = [z/a]$ and $W_1 = \{P(a, x, f(g(y))), P(a, f(a), f(u))\}$

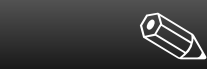
Unification algorithm: example



Is there a most general unifier of $\{P(a, x, f(g(y))), P(z, f(z), f(u))\}$?

1. $\sigma_0 = \epsilon$ and $W_0 = \{P(a, x, f(g(y))), P(z, f(z), f(u))\}$
   - $D_0 = \{a, z\}$
   - $v_0 = z$ and $t_0 = a$
   - therefore $\sigma_1 = [z/a]$ and $W_1 = \{P(a, x, f(g(y))), P(a, f(a), f(u))\}$
2. $\sigma_1 = [z/a]$ and $W_1 = \{P(a, x, f(g(y))), P(a, f(a), f(u))\}$

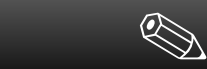
Unification algorithm: example



Is there a most general unifier of $\{P(a, x, f(g(y))), P(z, f(z), f(u))\}$?

1. $\sigma_0 = \epsilon$ and $W_0 = \{P(a, x, f(g(y))), P(z, f(z), f(u))\}$
   - $D_0 = \{a, z\}$
   - $v_0 = z$ and $t_0 = a$
   - therefore $\sigma_1 = [z/a]$ and $W_1 = \{P(a, x, f(g(y))), P(a, f(a), f(u))\}$
2. $\sigma_1 = [z/a]$ and $W_1 = \{P(a, x, f(g(y))), P(a, f(a), f(u))\}$
   - $D_1 = \{x, f(a)\}$

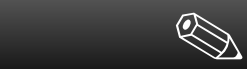
Unification algorithm: example



Is there a most general unifier of $\{P(a, x, f(g(y))), P(z, f(z), f(u))\}$?

1. $\sigma_0 = \epsilon$ and $W_0 = \{P(a, x, f(g(y))), P(z, f(z), f(u))\}$
   - $D_0 = \{a, z\}$
   - $v_0 = z$ and $t_0 = a$
   - therefore $\sigma_1 = [z/a]$ and $W_1 = \{P(a, x, f(g(y))), P(a, f(a), f(u))\}$
2. $\sigma_1 = [z/a]$ and $W_1 = \{P(a, x, f(g(y))), P(a, f(a), f(u))\}$
   - $D_1 = \{x, f(a)\}$
   - $v_1 = x$ and $t_1 = f(a)$

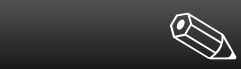
Unification algorithm: example



Is there a most general unifier of $\{P(a, x, f(g(y))), P(z, f(z), f(u))\}$?

1. $\sigma_0 = \epsilon$ and $W_0 = \{P(a, x, f(g(y))), P(z, f(z), f(u))\}$
   - $D_0 = \{a, z\}$
   - $v_0 = z$ and $t_0 = a$
   - therefore $\sigma_1 = [z/a]$ and $W_1 = \{P(a, x, f(g(y))), P(a, f(a), f(u))\}$
2. $\sigma_1 = [z/a]$ and $W_1 = \{P(a, x, f(g(y))), P(a, f(a), f(u))\}$
   - $D_1 = \{x, f(a)\}$
   - $v_1 = x$ and $t_1 = f(a)$
   - therefore $\sigma_2 = [z/a, x/f(a)]$ and $W_2 = \{P(a, f(a), f(g(y))), P(a, f(a), f(u))\}$

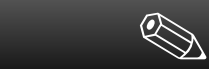
Unification algorithm: example



Is there a most general unifier of $\{P(a, x, f(g(y))), P(z, f(z), f(u))\}$?

1. $\sigma_0 = \epsilon$ and $W_0 = \{P(a, x, f(g(y))), P(z, f(z), f(u))\}$
   - $D_0 = \{a, z\}$
   - $v_0 = z$ and $t_0 = a$
   - therefore $\sigma_1 = [z/a]$ and $W_1 = \{P(a, x, f(g(y))), P(a, f(a), f(u))\}$
2. $\sigma_1 = [z/a]$ and $W_1 = \{P(a, x, f(g(y))), P(a, f(a), f(u))\}$
   - $D_1 = \{x, f(a)\}$
   - $v_1 = x$ and $t_1 = f(a)$
   - therefore $\sigma_2 = [z/a, x/f(a)]$ and $W_2 = \{P(a, f(a), f(g(y))), P(a, f(a), f(u))\}$
3. $\sigma_2 = [z/a, x/f(a)]$ and $W_2 = \{P(a, f(a), f(g(y))), P(a, f(a), f(u))\}$

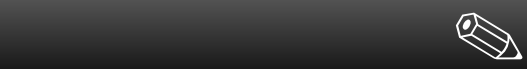
Unification algorithm: example



Is there a most general unifier of $\{P(a, x, f(g(y))), P(z, f(z), f(u))\}$?

1. $\sigma_0 = \epsilon$ and $W_0 = \{P(a, x, f(g(y))), P(z, f(z), f(u))\}$
   - $D_0 = \{a, z\}$
   - $v_0 = z$ and $t_0 = a$
   - therefore $\sigma_1 = [z/a]$ and $W_1 = \{P(a, x, f(g(y))), P(a, f(a), f(u))\}$
2. $\sigma_1 = [z/a]$ and $W_1 = \{P(a, x, f(g(y))), P(a, f(a), f(u))\}$
   - $D_1 = \{x, f(a)\}$
   - $v_1 = x$ and $t_1 = f(a)$
   - therefore $\sigma_2 = [z/a, x/f(a)]$ and $W_2 = \{P(a, f(a), f(g(y))), P(a, f(a), f(u))\}$
3. $\sigma_2 = [z/a, x/f(a)]$ and $W_2 = \{P(a, f(a), f(g(y))), P(a, f(a), f(u))\}$
   - $D_2 = \{g(y), u\}$

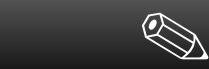
Unification algorithm: example



Is there a most general unifier of $\{P(a, x, f(g(y))), P(z, f(z), f(u))\}$?

1. $\sigma_0 = \epsilon$ and $W_0 = \{P(a, x, f(g(y))), P(z, f(z), f(u))\}$
   - $D_0 = \{a, z\}$
   - $v_0 = z$ and $t_0 = a$
   - therefore $\sigma_1 = [z/a]$ and $W_1 = \{P(a, x, f(g(y))), P(a, f(a), f(u))\}$
2. $\sigma_1 = [z/a]$ and $W_1 = \{P(a, x, f(g(y))), P(a, f(a), f(u))\}$
   - $D_1 = \{x, f(a)\}$
   - $v_1 = x$ and $t_1 = f(a)$
   - therefore $\sigma_2 = [z/a, x/f(a)]$ and $W_2 = \{P(a, f(a), f(g(y))), P(a, f(a), f(u))\}$
3. $\sigma_2 = [z/a, x/f(a)]$ and $W_2 = \{P(a, f(a), f(g(y))), P(a, f(a), f(u))\}$
   - $D_2 = \{g(y), u\}$
   - $v_2 = u$ and $t_2 = g(y)$

Unification algorithm: example



Is there a most general unifier of $\{P(a, x, f(g(y))), P(z, f(z), f(u))\}$?

1. $\sigma_0 = \epsilon$ and $W_0 = \{P(a, x, f(g(y))), P(z, f(z), f(u))\}$
   - $D_0 = \{a, z\}$
   - $v_0 = z$ and $t_0 = a$
   - therefore $\sigma_1 = [z/a]$ and $W_1 = \{P(a, x, f(g(y))), P(a, f(a), f(u))\}$
2. $\sigma_1 = [z/a]$ and $W_1 = \{P(a, x, f(g(y))), P(a, f(a), f(u))\}$
   - $D_1 = \{x, f(a)\}$
   - $v_1 = x$ and $t_1 = f(a)$
   - therefore $\sigma_2 = [z/a, x/f(a)]$ and $W_2 = \{P(a, f(a), f(g(y))), P(a, f(a), f(u))\}$
3. $\sigma_2 = [z/a, x/f(a)]$ and $W_2 = \{P(a, f(a), f(g(y))), P(a, f(a), f(u))\}$
   - $D_2 = \{g(y), u\}$
   - $v_2 = u$ and $t_2 = g(y)$
   - therefore $\sigma_3 = [z/a, x/f(a), u/g(y)]$ and $W_3 = \{P(a, f(a), f(g(y)))\}$

Factor and binary resolvent

We can now redefine the two "rules" of PL Resolution using the mgu notion.

Definition (factor)

If two literals **with the same sign** of a clause C have a most general unifier σ, then $\sigma(C)$ is called **factor of** C.

Definition (binary resolvent)

Let C_1 and C_2 be two clauses without **any common variable**. Let l_1 and l_2 be two literals of C_1 and of C_2.
If l_1 and $\neg l_2$ have a most general unifier σ, then the clause $(\sigma(C_1) - \sigma(l_1)) \vee (\sigma(C_2) - \sigma(l_2))$ is called **binary resolvent** of C_1 and of C_2.

If $\sigma(C_1) - \sigma(l_1) = \sigma(C_2) - \sigma(\neg l_2) = \phi$, the binary resolvent is noted $\square$.

Factor and binary resolvent: some examples!

Exercise (factor)

Find a factor for the following clauses:

- $R(x, a) \vee Q(y) \vee R(f(b), a)$
- $P(f(x), x) \vee Q(x) \vee P(y, a)$

Exercise (binary resolvent)

Find a binary resolvent for the following clauses:

- $P(x, a) \vee Q(y)$ and $\neg P(z, z)$
- $\neg P(x, f(x)) \vee Q(g(x))$ and $P(a, y) \vee R(y)$

Is it possible to find a binary resolvent of $P(a, x)$ and $\neg P(x, b)$?

Resolvent: combining factor and binary resolvent

We can now combine the two notions like in the PL case:

Definition (resolvent)

A **resolvent** of clauses C_1 and C_2 is a factor of one of the following binary resolvents:

- a binary resolvent of C_1 and C_2
- a binary resolvent of a factor of C_1 and of C_2
- a binary resolvent of C_1 and a factor of C_2
- a binary resolvent of a factor of C_1 and a factor of C_2

Resolution formal system $\mathcal{R}$

Definition (resolution rule R)

$$\frac{C_1 \qquad C_2}{R(C_1, C_2)}\ (R)$$

where $R(C_1, C_2)$ is a resolvent of C_1 and C_2.

Like in the PL case, there is no axiom. So, we expect to have completeness for refutation only.

Some (good) properties for Resolution

Lemma

For all clauses C_1 and C_2, $\{C_1, C_2\} \models R(C_1, C_2)$

And of course:

Theorem (completeness of resolution principle)

A set of clauses S is unsatisfiable iff there is a deduction of $\square$ in $\mathcal{R}$ (also called refutation) from S.

Socrates: back in high school...

Exercise

Using a FOL language and the Resolution formal system, prove that the following argument is correct:

- all humans are mortal
- Socrates is human
- therefore Socrates is mortal

Variable renaming is important!

Variable renaming between the different clauses is important, as you may do mistakes when not renaming.

For instance, consider the two wffs $\forall x \exists y\ P(x, y)$ and $\forall x \exists y\ Q(x, y)$.

Without renaming, you can deduce $\forall x \exists y\ P(x, y) \wedge Q(x, y)$ from those two wffs, which is clearly not the case...

How to use Resolution?

What to prove	**How to prove it**
Σ is unsatisfiable	$CL(\Sigma) \vdash_{\mathcal{R}} \square$
$\Sigma \models \varphi$	$CL(\Sigma \cup \{\neg\varphi\}) \vdash_{\mathcal{R}} \square$
$\models \varphi$	$CL(\neg\varphi) \vdash_{\mathcal{R}} \square$

You can also use Resolution to solve "fill-in-the-blank" questions like "who are the x such that $P(x, a)$ holds?".
In this case, consider the formula $P(x, a) \rightarrow \mathit{Goal}(x)$, use it with Resolution and stop when producing a clause with only *Goal* literals.

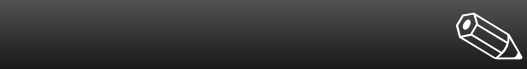
Students or not?

Exercise

Prove using the Resolution formal system that the following argument is correct:

- every student has a student card
- PhD students are students
- the only persons who benefit from ONERA work council are ONERA employees
- PhD students at ONERA benefit from ONERA work council
- therefore PhD students at ONERA are ONERA employees who have a student card

To ski or not to ski



Exercise

Represent the following facts and question in first-order logic:

> *Tony, Mike, and John belong to the Alpine Club. Every member of the Alpine Club is either a skier or a mountain climber (or both). No mountain climber likes rain, and all skiers like snow. Mike dislikes whatever Tony likes and likes whatever Tony dislikes. Tony likes rain and snow. Is there a member of the Alpine Club who is a mountain climber but not a skier?*

Using Resolution, find the answer.

6 - Program analysis with first-order logic

Disclaimer

All the material presented in this part is taken from Chang and Lee 1973.

Chang, Chin-Liang and Richard Char-Tung Lee (1973).
Symbolic logic and mechanical theorem proving.
Academic Press.

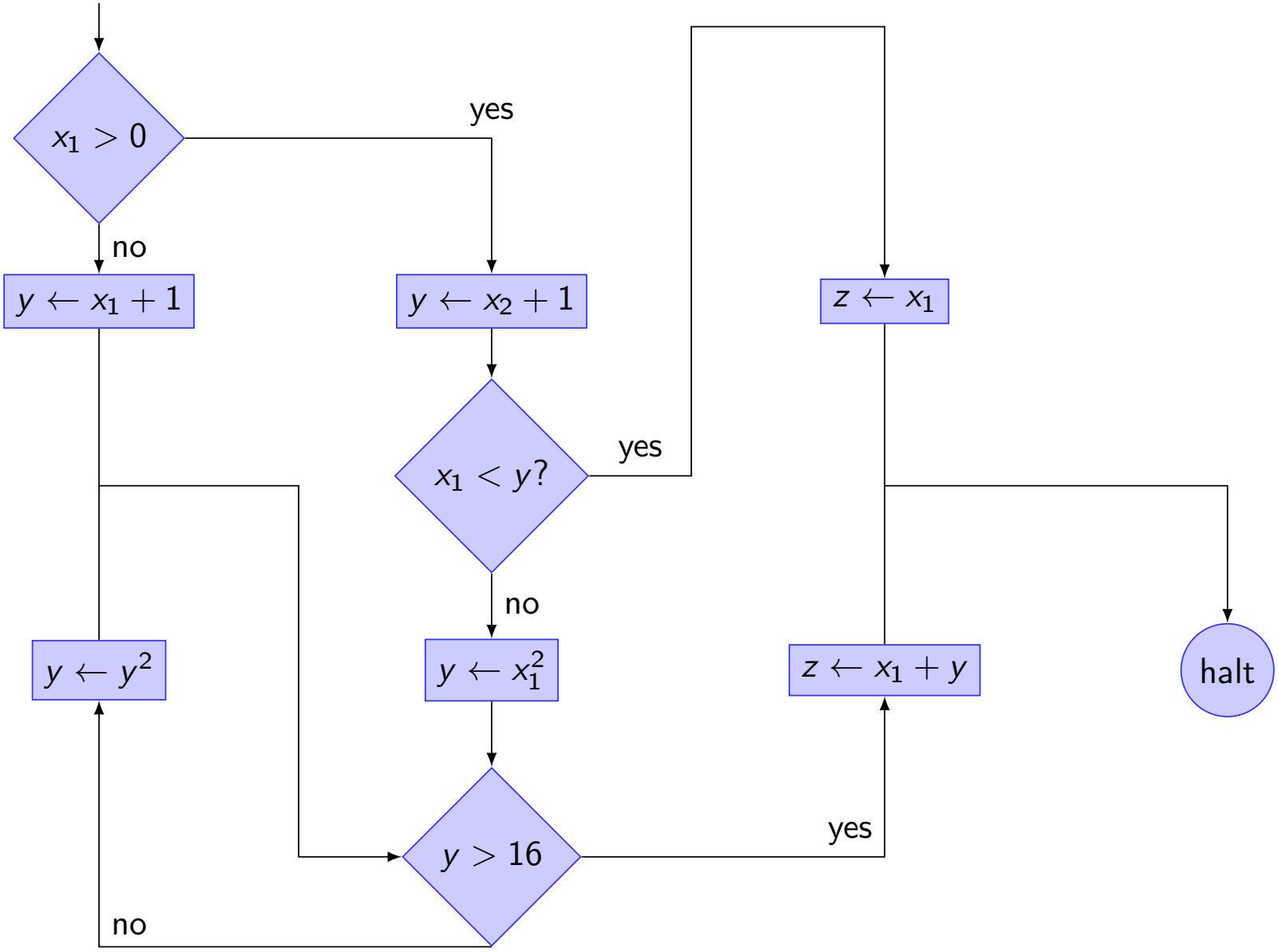
Introduction

We considering a program $\mathcal{P}$, some questions arise:

- **the halting problem**: given an input i, does $\mathcal{P}$ halt?
- **the answer problem**: given an input i and considering that $\mathcal{P}$ halts, what is the answer returned by $\mathcal{P}$?
- **the correctness problem**: given an input i, does the answer given by $\mathcal{P}$ respect its specifications?
- **the equivalence problem**: given two programs, do they give the same results with the same inputs?
- **the specialization problem**: given a set $\mathcal{I}$ of acceptable inputs for $\mathcal{P}$, when considering $\mathcal{I}^* \subseteq \mathcal{I}$, how can $\mathcal{P}$ be simplified to $\mathcal{P}^*$ such that $\mathcal{P}^*$ is faster than $\mathcal{P}$ on $\mathcal{I}^*$?

Outline of part 6 - Program analysis with FOL

“Classical” representation of a program



Directed graph

We want to model a program by **first-order formulas**.

In order to find such a representation, we will model a program by a **directed graph**.

Definition (directed graph)

A directed graph is composed of a non-empty set V, a set A such that elements of A does not appear in V and an application $D : A \to V \times V$.

- V: vertices
- A: arcs
- **path** in a directed graph: $a_1, a_2, \ldots, a_n$ where each a_i has v_{i-1} as tail vertice and v_i as head vertice.

(A) formal definition of a program

Definition (program)

A program is composed of a **vector of input variables** $\vec{x} = \{x_1, \ldots, x_n\}$, a **vector of program variables** $\vec{y} = \{y_1, \ldots, y_m\}$, a **vector of output variables** $\vec{z} = \{z_1, \ldots, z_l\}$ and a finite directed graph such that:

1. there is only one vertice that is never head of an arc: S
2. there is only one vertice that is never tail of an arc: H
3. every vertice $v \in V$ is on a path from S to H
4. every arc $a \in A$ is associated to a quantifier-free formula $P_a(\vec{x}, \vec{y})$ called **test formula for** a
5. for each arc $a \in A$:
   - if a does not have H as tail, an assignation $\vec{y} \leftarrow f_a(\vec{x}, \vec{y})$ is associated to a
   - if a has H as tail, an assignation $\vec{z} \leftarrow f_a(\vec{x}, \vec{y})$ is associated to a
6. for each vertice $v \in V$ different from H, if $a_1, \ldots, a_n$ are the outgoing arcs from v, then for all value of $\vec{x}$ and $\vec{y}$, one and only one of the $P_{a_i}(\vec{x}, \vec{y})$ is true (**determinism**)

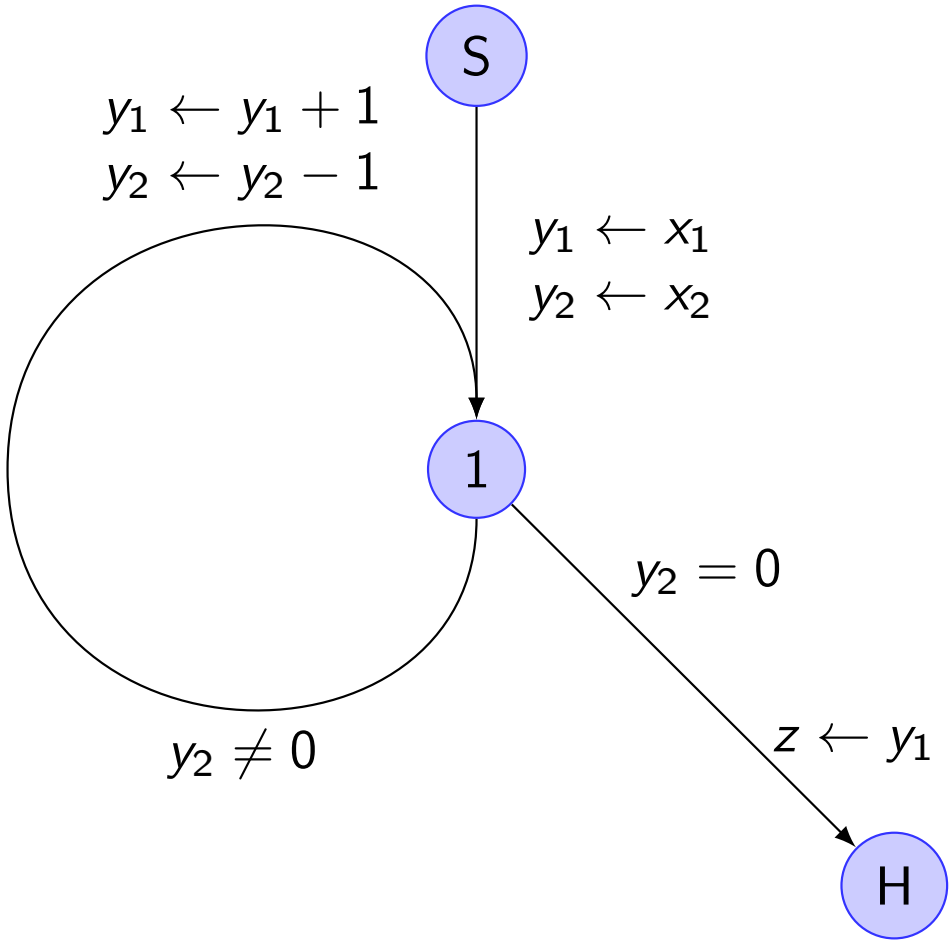
Prog. definition: example

Function sum(x_1, x_2)

Input: two natural numbers x_1 et x_2
Output: the result of the addition of x_1 and x_2

```
1 y_1 ← x_1 ;
2 y_2 ← x_2 ;
3 while y_2 ≠ 0 do
4 |   y_1 ← y_1 + 1 ;
5 |   y_2 ← y_2 − 1 ;
6 end
7 z ← y_1 ;
8 halt ;
```

Directed graph: example



Translation into FOL formulas

Definition (access predicate)

Let $v \in V$. The **access predicate** of v is a predicate Q_v s.t. $Q_v(\vec{x}, \vec{y})$ (or $Q_H(\vec{x}, \vec{z})$ if $v = H$) is defined as the condition for going from S to v with inputs vector $\vec{x}$ and variables vector **updated** to $\vec{y}$ (or $\vec{z}$).

$Q_S(\vec{x}, \vec{y})$ is supposed to be always "true".

Q_H is called the **halting predicate**.

Definition (arc formula)

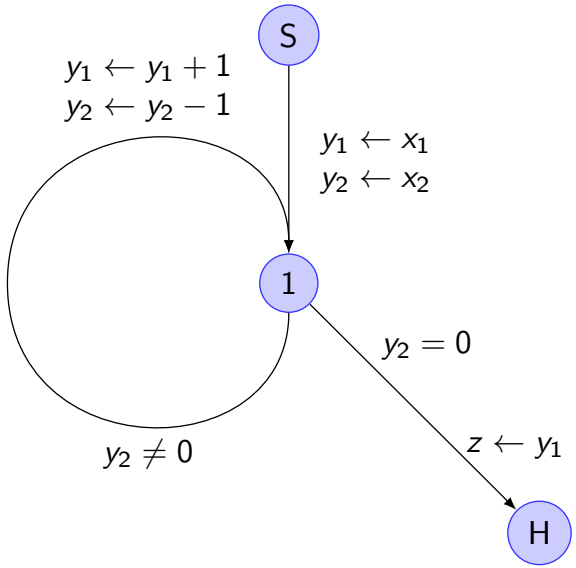
Let $a = (v_i, v_j)$ be an arc of a program, $P_a(\vec{x}, \vec{y})$ be the test formula associated to a, $f_a(\vec{x}, \vec{y})$ be the assignation associated to a, then the formula of a is a formula W_a such that

$$W_a \equiv Q_i(\vec{x}, \vec{y}) \wedge P_a(\vec{x}, \vec{y}) \rightarrow Q_j(\vec{x}, f_a(\vec{x}, \vec{y}))$$

Logical description: example

sum(x_1,x_2)

$$
\begin{aligned}
Q_{S\to 1} \equiv & Q_1(x_1, x_2, x_1, x_2) \\
Q_{1\to 1} \equiv & Q_1(x_1, x_2, y_1, y_2) \wedge y_2 \neq 0 \to \\
& Q_1(x_1, x_2, y_1 + 1, y_2 - 1) \\
Q_{1\to H} \equiv & Q_1(x_1, x_2, y_1, y_2) \wedge y_2 = 0 \to \\
& Q_H(x_1, x_2, y_1)
\end{aligned}
$$



Formula describing a program

Definition (formula describing a program)

Let $a_1, \ldots, a_n$ be the arcs of a program $\mathcal{P}$. Then $\forall \vec{y}\ W_{a_1} \wedge \ldots \wedge W_{a_n}$ is called the **formula describing** $\mathcal{P}$.

Remark: inputs are considered as constants.

Theorem

Let $\mathcal{P}$ be a program and $A_{\mathcal{P}}$ be the set of clauses representing the formula describing $\mathcal{P}$. Then $A_{\mathcal{P}}$ is satisfiable.

Outline of part 6 - Program analysis with FOL

Program analysis

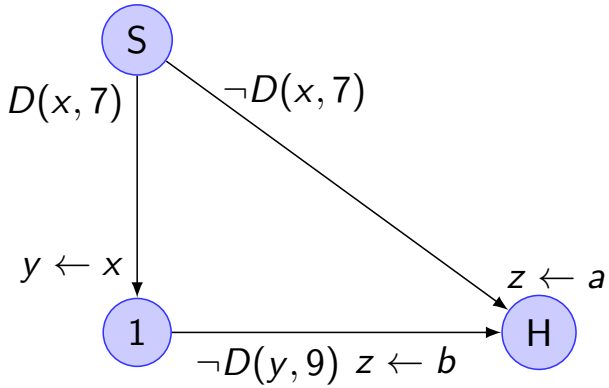
First idea: use Resolution to generate conclusions on a program $\mathcal{P}$ from $A_{\mathcal{P}}$.

➥ "redefine" the program specifications

$?(x)$

$$
\begin{array}{ll}
Q_{S\to 1} & D(x,7) \to Q_1(x,x) \\
Q_{S\to H} & \neg D(x,7) \to Q_H(x,a) \\
Q_{1\to H} & Q_1(x,y) \wedge \neg D(y,9) \to Q_H(x,b)
\end{array}
$$

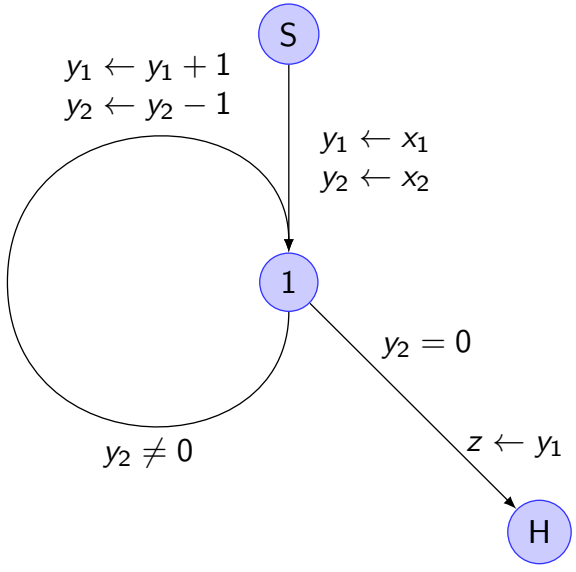
$$
\begin{array}{ll}
C_1 & \neg D(x,7) \vee Q_1(x,x) \\
C_2 & D(x,7) \vee Q_H(x,a) \\
C_3 & \neg Q_1(x,y) \vee D(y,9) \vee Q_H(x,b)
\end{array}
$$



Program analysis: loops

sum(x_1,x_2)

$$
\begin{array}{ll}
C_1 & Q_1(x_1, x_2, x_1, x_2) \\
C_2 & \neg Q_1(x_1, x_2, y_1, y_2) \vee y_2 = 0 \vee \\
 & Q_1(x_1, x_2, y_1 + 1, y_2 - 1) \\
C_3 & \neg Q_1(x_1, x_2, y_1, y_2) \vee y_2 \neq 0 \vee \\
 & Q_H(x_1, x_2, y_1)
\end{array}
$$



induction schema (not obvious. . .)

$$
\begin{aligned}
&\exists y_2\ [(y_2 > 0 \wedge Q_1(x_1, x_2, x_1, y_2)) \wedge \\
&\quad \forall y_3\ (y_3 > 0 \wedge Q_1(x_1, x_2, x_1, y_3) \rightarrow Q_1(x_1, x_2, x_1 + 1, y_3 - 1))] \\
&\qquad\qquad \rightarrow Q_1(x_1, x_2, x_1 + x_2, 0)
\end{aligned}
$$

Program analysis: loop

$$
\begin{array}{ll}
C_1 & Q_1(x_1, x_2, 0, x_2) \\
C_2 & \neg Q_1(x_1, x_2, y_1, y_2) \vee y_2 = 0 \vee Q_1(x_1, x_2, y_1 + x_1, y_2 - 1) \\
C_3 & \neg Q_1(x_1, x_2, y_1, y_2) \vee y_2 \neq 0 \vee Q_H(x_1, x_2, y_1) \\
C_4 & y_2 \not> 0 \vee \neg Q_1(x_1, x_2, x_1, y_2) \vee f(y_2) > 0 \vee Q_1(x_1, x_2, x_1 + x_2, 0) \\
C_5 & y_2 \not> 0 \vee \neg Q_1(x_1, x_2, x_1, y_2) \vee Q_1(x_1, x_2, x_1, f(y_2)) \vee \\
 & Q_1(x_1, x - 2, x_1 + x_2, 0) \\
C_6 & y_2 \not> 0 \vee \neg Q_1(x_1, x_2, x_1, y_2) \vee \neg Q_1(x_1, x_2, x_1 + 1, f(y_2) - 1) \vee \\
 & Q_1(x_1, x_2, x_1 + x_2, 0) \\
C_7 & x_2 > 0 \\
C_8 & 0 = 0 \\
C_9 & u \not> 0 \vee u \neq 0
\end{array}
$$

$Q_H(x_1, x_2, x_1 + x_2)$ can be produced (but it is very long...).

Outline of part 6 - Program analysis with FOL

Halting and response

- **halting**: given $\mathcal{P}$ and a specification of its inputs, does $\mathcal{P}$ halt?
- **answer**: given $\mathcal{P}$ and a specification of its inputs, what is the answer provided by $\mathcal{P}$?

Those two problems are linked...

We need the following informations:

- $A_{\mathcal{P}}$: formulas describing $\mathcal{P}$
- A_S: formulas describing test predicates and assignation functions (for instance, axioms for equality etc.)
- A_I: formulas characterizing inputs

Theorem

If $A_i \wedge A_S$ is consistent, then $A_i \wedge A_s \wedge A_{\mathcal{P}}$ is satisfiable.

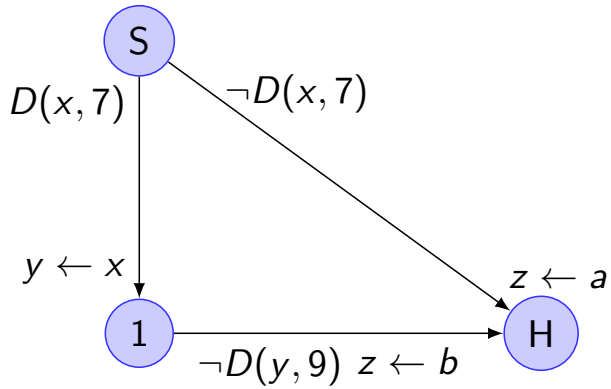
Halting problem

Theorem

Let $\mathcal{P}$ be a program whose description is $A_{\mathcal{P}} \wedge A_I \wedge A_S$. Then $\mathcal{P}$ halts iff $A_{\mathcal{P}} \wedge A_I \wedge A_S \models \exists \vec{z}\ Q_H(\vec{x}, \vec{z})$.

For instance:

$$Q_{S\to 1} \wedge Q_{S\to H} \wedge Q_{1\to H} \models Q_H(x, a) \vee Q_H(x, b)$$



Halting and anwser

Definition (halting clause)

A clause containing only the literal Q_H is called halting clause.

Lemma (program halting)

Let $\mathcal{P}$ a program and A_T the formula obtained by removing all literals containing Q_H from $A_\mathcal{P}$. Then $\mathcal{P}$ halts iff $A_T \wedge A_S \wedge A_I$ is unsatisfiable.

Theorem (program answer)

Let $\mathcal{P}$ be a program. Then $\mathcal{P}$ halts iff there is a deduction from an halting clause from $A_\mathcal{P} \wedge A_S \wedge A_I$.

Program halting: example

C_1 $Q_1(x_1, x_2, 0, x_2)$

C_2 $\neg Q_1(x_1, x_2, y_1, y_2) \vee y_2 = 0 \vee Q_1(x_1, x_2, y_1 + x_1, y_2 - 1)$

C_3 $\neg Q_1(x_1, x_2, y_1, y_2) \vee y_2 \neq 0$

C_4 $y_2 \not> 0 \vee \neg Q_1(x_1, x_2, x_1, y_2) \vee f(y_2) > 0 \vee Q_1(x_1, x_2, x_1 + x_2, 0)$

C_5 $y_2 \not> 0 \vee \neg Q_1(x_1, x_2, x_1, y_2) \vee Q_1(x_1, x_2, x_1, f(y_2)) \vee$
$Q_1(x_1, x - 2, x_1 + x_2, 0)$

C_6 $y_2 \not> 0 \vee \neg Q_1(x_1, x_2, x_1, y_2) \vee \neg Q_1(x_1, x_2, x_1 + 1, f(y_2) - 1) \vee$
$Q_1(x_1, x_2, x_1 + x_2, 0)$

C_7 $x_2 > 0$

C_8 $0 = 0$

C_9 $u \not> 0 \vee u \neq 0$

Outline of part 6 - Program analysis with FOL

Correctness and equivalence: definition

Definition (correctness)

Let $\mathcal{P}$ be a program and $R(\vec{x}, \vec{z})$ a specification of $\mathcal{P}$ expressed using a relation. $\mathcal{P}$ is correct with respect to $R(\vec{x}, \vec{z})$ iff $A_{\mathcal{P}} \wedge A_S \wedge A_i \models \exists \vec{z}\ Q_H(\vec{x}, \vec{z}) \wedge R(\vec{x}, \vec{z})$.

Definition (equivalence)

Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two programs. $\mathcal{P}_1$ and $\mathcal{P}_2$ are equivalent iff $A_{\mathcal{P}_1} \wedge A_{\mathcal{P}_2} \wedge A_S \wedge A_i \models \exists \vec{z}\ Q^1_H(\vec{x}, \vec{z}) \wedge Q^2_H(\vec{x}, \vec{z})$.

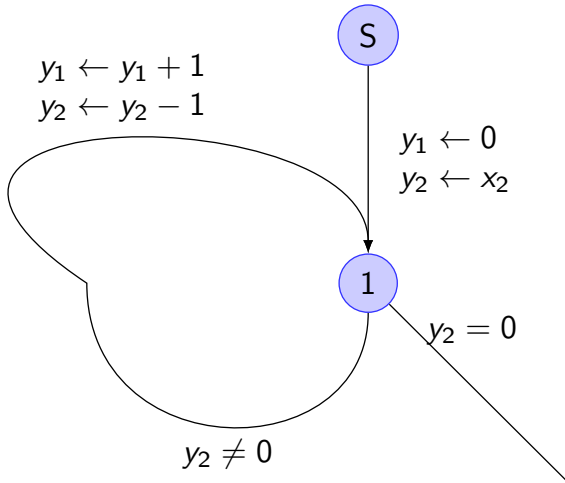
Correctness: example

Constraints:

- $x_1 > 0$
- $x_2 > 0$

"Axioms":

- $\neg(x_1 > 0) \vee \neg(x_2 > 0) \vee (x_1 + x_2 > x_1)$



Specification

$z > x_1$

$\exists z\ Q_H(x_1, x_2, z) \wedge (z > x_1)$ can be easily proved.

Outline of part 6 - Program analysis with FOL

Specialization procedure

Definition (specialization)

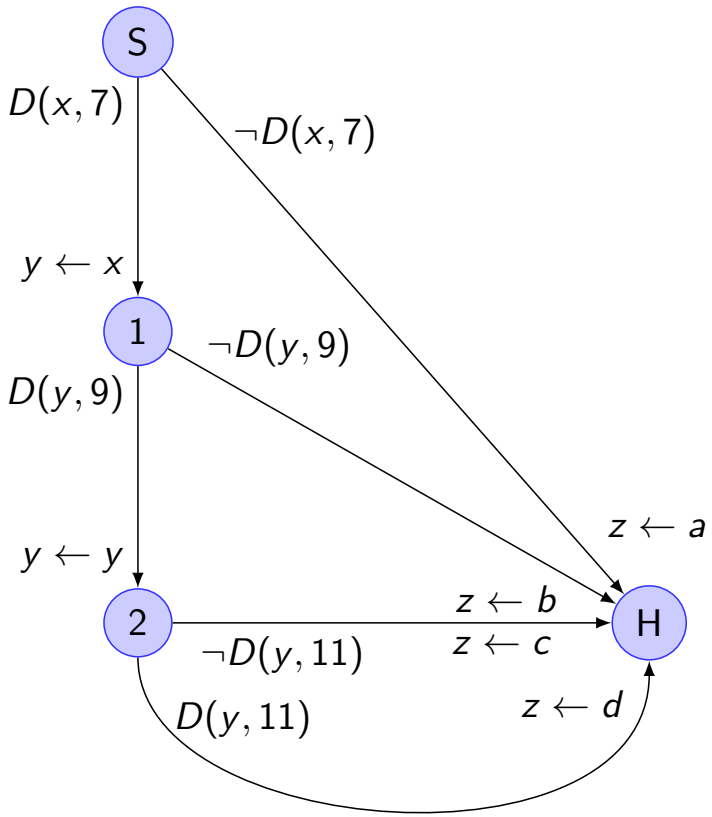
Given a set $\mathcal{I}$ of acceptable input for $\mathcal{P}$, if we consider $\mathcal{I}^* \subseteq \mathcal{I}$, how can $\mathcal{P}$ be simplified into $\mathcal{P}^*$ such that $\mathcal{P}^*$ is faster than $\mathcal{P}$ on $\mathcal{I}^*$?

Function specialization($\mathcal{P}$, $\mathcal{I}^*$)

Input: a program $\mathcal{P}$ and a set of inputs $\mathcal{I}^*$
Output: a program $\mathcal{P}^*$ specialization of $\mathcal{P}$ for $\mathcal{I}^*$

1 $S \leftarrow A_{\mathcal{P}} \wedge A_S \wedge A_{\mathcal{I}^*}$;
2 deduce an halting clause C_H from S using a deduction of D ;
3 $\mathcal{P}^* = \mathcal{P} - \{a_k = (v_i, v_j) \ : \ Q_{i \to j}$ is not used in$D\}$;
4 **return** $\mathcal{P}^*$;

Specialization: example



Initially: $\mathcal{I} = \mathbb{N}$

Now: $\mathcal{I}^* = \{x \in \mathbb{N} \ : \ x < 10\}$

Specialization: example

$A_{\mathcal{P}}$:

C_1	$\neg D(x,7) \vee Q_1(x,x)$
C_2	$D(x,7) \vee Q_H(x,a)$
C_3	$\neg Q_1(x,y) \vee \neg D(y,9) \vee Q_2(x,y)$
C_4	$\neg Q_1(x,y) \vee D(y,9) \vee Q_H(x,b)$
C_5	$\neg Q_2(x,y) \vee D(y,11) \vee Q_H(x,c)$
C_6	$\neg Q_2(x,y) \vee \neg D(y,11) \vee Q_H(x,d)$

$A_{\mathcal{S}}$:

C_7	$\neg(x < 10) \vee \neg D(x,11)$

$A_{\mathcal{I}^*}$:

C_8	$x < 10$

$Q_H(x,a) \vee Q_H(x,b) \vee Q_H(x,c)$ can be deduced without using C_6.

7 - Formal number theory

Introduction

Hilbert's program: "mathematize" mathematics!

Particularly, we are interested in mathematics **consistency**: from a "good" formalization of math., we cannot deduce φ and $\neg\varphi$.

Let's start with **arithmetics**!

Two Gödel incompleteness theorems:

1. there a "sentence" in arithmetics that says that it is not provable, so there are **undecidable** statements in arithmetics
2. consistency of arithmetics can not be expressed into arithmetics

Two fundamental ideas:

- **fixed point** notion
- **coding** of arithmetics (and proofs of arithmetics) in arithmetics

Outline of part 7 - Formal number theory

Definition (first-order theory)

Let $\mathcal{L}_{FOL}$ be a first-order language. A **first-order theory** is a formal theory K whose symbols and wffs are the symbols and wffs of $\mathcal{L}_{FOL}$ and whose axioms and inference rules are defined as follows.

Definition (logical axioms)

Let φ, ψ and γ be wffs of $\mathcal{L}_{FOL}$.

A1 $\varphi \to (\psi \to \varphi)$

A2 $(\varphi \to (\psi \to \gamma)) \to ((\varphi \to \psi) \to (\varphi \to \gamma))$

A3 $(\neg\psi \to \neg\varphi) \to ((\neg\psi \to \varphi) \to \psi)$

A4 $\forall x_i\ \varphi(x_i) \to \varphi(t)$ if $\varphi(x_i)$ is a wff of $\mathcal{L}_{FOL}$ and t a free term for x_i in $\varphi(x_i)$

A5 $(\forall x_i\ (\varphi \to \psi)) \to (\varphi \to \forall x_i\ \psi)$ if φ does not have free occurrences of x_i

Non-logical axioms

Non-logical axioms or **proper axioms** of each theory are defined in the theory.

N.B.

They are not necessary tautologies.

N.B.

A first-order theory without non-logical axioms is called **predicate calculus**.

Inference rules

Definition (Modus Ponens rule (MP))

From A and $A \to B$ infer B:

$$\frac{A \qquad A \to B}{B}\ (MP)$$

Definition (generalization rule ($\forall$))

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

Model of a first-order theory

Definition (model of a theory)

Let K be a first-order theory expressed in the language $\mathcal{L}_{FOL}$. A **model** of K is an interpretation I of $\mathcal{L}_{FOL}$ in which all axioms of K are valid formulas.

Properties of first-order theories

Theorem (soundness)

Every theorem of predicate calculus is valid.

Theorem (consistency)

Predicate calculus is consistent.

Theorem (completeness (Gödel, 1930))

Every valid formula of predicate calculus is a theorem.

First-order theory with equality

Definition (first-order theory with equality)

Let K be a first-order theory with a binary predicate A. We will note $t = s$ as an abbreviation of $A(t, s)$ and $t \neq s$ as an abbreviation of $\neg A(t, s)$.
K is a **first-order theory with equality** if the following wff are proper axioms (schematas) of K:

A6 $\forall x\ x = x$

A7 $\forall x \forall y\ x = y \rightarrow (\varphi \rightarrow \varphi[x|y])$ for every wff φ such that x is freely substitutable by y in φ

The second axiom is called **Leibniz's law**.

Example: partial orders

Let us consider a FO language with a binary predicate $<$, without function nor constant symbol.

Non-logical axioms for partial orders

A6 $\forall x \neg(x < x)$

A7 $\forall x \forall y \forall z \; x < y \wedge y < z \rightarrow x < z$

Example: groups

Let us consider a FO language **with equality** and with a function symbol $+$ and a constant symbol 0.

Non-logical axioms for groups

A8 $\forall x \forall y \forall z \ (x + (y + z) = (x + y) + z)$

A9 $\forall x \ 0 + x = x$

A10 $\forall x \exists y \ x + y = 0$

Theorem

Group theory is decidable.

This is not the case for instance for free groups.

Outline of part 7 - Formal number theory

Objective

Find a **complete** and **consistent** first-order theory representing arithmetics.

We will see:

- a **formalization** of arithmetics due to Peano
- a **logical theory** representing Peano's arithmetics
- that we can **express** all **interesting functions** on natural numbers in this theory
- that **such a theory does not exist** (Gödel's theorems)

Outline of part 7 - Formal number theory

Peano postulates

Peano postulates for arithmetics

P1 0 is a natural number

P2 if x is a natural number, there is another natural number called **successor** of x and denoted by x'

P3 $0 \neq x'$ for all natural number x

P4 if $x' = y'$ then $x = y$

P5 if Q is a property on natural numbers and if a) 0 verifies Q and b) if when natural number x verifies Q then x' verifies Q, then all natural numbers verify Q

A theory for arithmetics: S

Let us consider a FO language L_A with a single predicate (noted $=$), a single constant (noted 0) and three functions (noted $'$, $+$ and .). Let S be a FO theory on L_A.

Non-logical axioms of S

S1 $x_1 = x_2 \rightarrow (x_1 = x_3 \rightarrow x_2 = x_3)$

S2 $x_1 = x_2 \rightarrow x_1' = x_2'$

S3 $0 \neq x_1'$

S4 $x_1' = x_2' \rightarrow x_1 = x_2$

S5 $x_1 + 0 = x_1$

S6 $x_1 + x_2' = (x_1 + x_2)'$

S7 $x_1.0 = 0$

S8 $x_1.x_2' = (x_1.x_2) + x_1$

S9 $\varphi(0) \rightarrow (\forall x\ (\varphi(x) \rightarrow \varphi(x')) \rightarrow \forall x\ \varphi(x))$ for all wff φ

Properties of S

Theorem

S is a FO theory with equality.

Easy to prove...

Definition (standard interpretation)

The **standard interpretation** (or **standard model**) of S is an "intuitive" interpretation of S such that:

- S domain is $\mathbb{N}$
- the natural number 0 is the interpretation of the symbol 0
- adding of 1 is the interpretation of $'$
- addition and multiplication on natural numbers are the interpretations of $+$ and .
- the interpretation of $=$ is identity on $\mathbb{N}$

Properties of S

Some theorems can be exhibited in S:

- "generalization" of axioms to any terms
  $t = r \to (t = s \to r = s)$
- distributivity of multiplication
  $t.(r + s) = (t.r) + (r.s)$
- associativity of multiplication
  $(t.r).s = t.(r.s)$
- cancellation law for addition
  $t + s = r + s \to t = r$

Numerals

The terms 0, $0'$, $0''$ etc. are called **numerals** and are denoted by $\overline{0}$, $\overline{1}$, $\overline{2}$ etc.

More precisely, $\overline{0}$ is 0 and for all natural number n, $\overline{n+1} = \overline{n}'$.

We can easily prove "classical" theorems, like for instance $t + \overline{1} = t'$ and the following properties for m and n if m and n are natural numbers:

- if $m \neq n$ then $\vdash_S \overline{m} \neq \overline{n}$
- $\vdash_S \overline{m+n} = \overline{m} + \overline{n}$
- $\vdash_S \overline{m.n} = \overline{m}.\overline{n}$

Outline of part 7 - Formal number theory

Definition of number-theoretic functions

Definition (number-theoretic function)

A number-theoretic function is a **total** function from $\mathbb{N}^p$ to $\mathbb{N}$.

Definition (function expressibility)

Let K be a theory with equality on L_A. Let f be a number-theoretic function with arity n. f is **representable** in K iff there is a wff $\varphi(x_1, \ldots, x_n, y)$ ($x_1, \ldots, x_n, y$ being free variables in φ) such that for all natural numbers $k_1, \ldots, k_n, m$:

1. if $f(k_1, \ldots, k_n) = m$ then $\vdash_K \varphi(\overline{k_1}, \ldots, \overline{k_n}, \overline{m})$
2. $\vdash_K \exists y\ \varphi(\overline{k_1}, \ldots, \overline{k_n}, y)$

f is **strongly representable** in K if condition 2 is replaced by:

2. $\vdash_K \exists y\ \varphi(x_1, \ldots, x_n, y)$

Number-theoretic functions expressibility

Theorem (weak/strong repr. equivalence)

A function f is strongly representable in K iff f is representable in K.

This proposition allows to work only with strongly representable functions, i.e. functions which "work" with all terms!

Primitive recursive and recursive functions

Definition (initial functions)

- the **zero** function, $Z(x) = 0$ for all x
- the **successor** function, $N(x) = x + 1$ for all x
- the **projection** function, $U_i^n(x_1, \ldots, x_n) = x_i$ for all $x_1, \ldots, x_n$

Definition (building rules)

- substitution:

$$f(x_1, \ldots, x_n) = g(h_1(x_1, \ldots, x_n), \ldots, h_n(x_1, \ldots, x_n))$$

- recursion:

$$\begin{aligned} f(x_1, \ldots, x_n, 0) &= g(x_1, \ldots, x_n) \\ f(x_1, \ldots, x_n, y+1) &= h(x_1, \ldots, x_n, y, f(x_1, \ldots, x_n, y)) \end{aligned}$$

- restrictive μ-operator: $\mu y(g(x_1, \ldots, x_n, y) = 0)$ is the smallest y such that $g(x_1, \ldots, x_n, y) = 0$.

Primitive recursive and recursive functions

Definition (recursive functions)

A function is **primitive recursive** iff it can be obtained by using in a **finite number of steps** initial functions, substitution and recursion.
A function is **recursive** iff it can be obtained by using in a **finite number of steps** initial functions, substitution, recursion and restrictive μ-operator.

Initial functions are (of course) primitive recursive.

Some recursive functions: $+$, ., power, predecessor, absolute value, factorial, minimum, maximum, remaining of euclidian division, sum, product etc.

Some examples of recursive functions

predecessor

$$\begin{cases} \delta(0) &= 0 \\ \delta(y+1) &= y \end{cases}$$

function $\dot{-}$

$$\begin{cases} x \dot{-} 0 &= x \\ x \dot{-} (y+1) &= \delta(x \dot{-} y) \end{cases}$$

absolute value

$$|x - y| = (x \dot{-} y) + (y \dot{-} x)$$

$= 0$?

$$sg(x) = x \dot{-} \delta(x)$$

remaining of division

$$\begin{cases} rm(x,0) &= 0 \\ rm(x,y+1) &= N(rm(x,y)) * sg(|x - N(rm(x,y$$

Gödel β function

Definition (Gödel β function)

$\beta(x_1, x_2, x_3) = rm(1 + (x_3 + 1).x_2, x_1)$

Some properties:

- β is primitive recursive
- $\beta(x_1, x_2, x_3)$ is representable by the following FOL wff:

$$\exists w\ (x_1 = (1 + (x_3 + 1).x_2).w + y \wedge y < 1 + (x_3 + 1).x_2)$$

Theorem

For all sequence of natural numbers $k_0, \ldots, k_n$ there is b and c such that $\beta(b, c, i) = k_i$ for $0 \leq i \leq n$.

The previous propositions are used to prove the following result:

Theorem (representation of recursive functions)

Every recursive function is representable in S.

Outline of part 7 - Formal number theory

Gödel numbers: symbols

Definition (Gödel numbers of symbols)

Let K be a FOL theory. Every symbol of K is associated to a natural number in the following way:

- $g(() = 3$, $g()) = 5$, $g(,) = 7$, $g(\neg) = 9$, $g(\rightarrow) = 11$, $g(\forall) = 13$
- $g(x_k) = 13 + 8k$ for $k \geq 1$
- $g(a_k) = 7 + 8k$ for $k \geq 1$
- $g(x_k) = 13 + 8k$ for $k \geq 1$
- $g(f_k^n) = 1 + 8(2^n 3^k)$ for $k, n \geq 1$
- $g(A_k^n) = 3 + 8(2^n 3^k)$ for $k, n \geq 1$

Different symbols have different Gödel numbers.

Gödel numbers: expressions and sequences

Definition (Gödel number of an expression)

Let $u_0 u_1 \dots u_r$ be an expression, then:

$$g(u_0 u_1 \dots u_r) = 2^{g(u_0)} 3^{g(u_1)} \dots p_r^{g(u_k)}$$

where p_i is the i-th prime number.

Gödel numbers of expressions are different from the symbols ones.

Definition (Gödel number of a sequence)

Let $e_0, e_1, \dots, e_r$ be a finite sequence of expression of K, then:

$$g(e_0, e_1, \dots, e_r) = 2^{g(e_0)} 3^{g(e_1)} \dots p_r^{g(e_k)}$$

The Gödel number of sequences are all different and different of Gödel numbers of expressions and symbols.

Primitive recursive vocabulary

Definition (primitive recursive vocabulary)

A theory K have a primitive recursive vocabulary (a recursive vocabulary) if the following properties are primitive recursive (or recursive):

- $IC(x)$: x is the Gödel number of a constant symbol of K
- $FL(x)$: x is the Gödel number of a function symbol of K
- $PL(x)$: x is the Gödel number of a predicate symbol of K

A theory K which have a **finite number** of constant symbols, function symbols and predicate symbols have a primitive recursive vocabulary...

...paricularly S!

Recursive functions and relations

Let us consider a theory K with a recursive vocabulary. Then a lot of interesting recursive functions are representable in this theory:

- $EVbl(x)$: x is the Gödel number of an expression which is a variable and variants on functions, terms, atomic formulas, formulas, substitution, axiom instances etc.
- $MP(x, y, z)$: the expression with Gödel number z is a consequence of application of Modus Ponens on the expressions with Gödel numbers x and y.
- $Sub(y, u, v)$ the Gödel number of the result of the substitution of all free occurrences of the variable which has for Gödel number v has for Gödel number u.
- $Num(y)$: the Gödel number of the expression $\overline{y}$.
- $Nu(x)$: x is the Gödel number of a numeral.
- $D(u)$: the Gödel number of $\varphi(\overline{u})$ if u is the Gödel number of the wff $\varphi(x_1)$. D is called the **diagonal function**.
  Let us remark that $D(u) = Sub(u, Num(u), 21)$.

Recursive set of axioms

Definition (recursive set of axioms)

A theory K has a (primitive) recursive set of axioms if the following function is (primitive) recursive:
$PrAx(y)$: y is the Gödel number of a non-logical axiom of K.

Theorem

S has a primitive recursive set of axioms.

Theorem

Let K be a theory with a recursive vocabulary and a recursive axioms set. Then the following relations are recursive:

- *$Ax(y)$: y is the Gödel number of an axiom of K.*
- *$Prf(y)$: y is the Gödel number of a proof in K.*
- *$Pf(x, y)$: y is the Gödel number of a proof in K of the wff with Gödel number x.*

Functions representativity

Theorem

Let K be a theory with equality with a constant 0 and a function f_1^1 and such that it has a recursive axioms set. Let us suppose that for all natural numbers r and s, if $\vdash_K \overline{r} = \overline{s}$ then $r = s$, then every function that is representable K is recursive.

Outline of part 7 - Formal number theory

Fixed point theorems

Let φ be an expression of a theory and q its Gödel number. We will note $\overline{q}$ by $n(\varphi)$ (it is the "name" of φ in L_A).

Theorem (diagonalization lemma)

*Let us suppose that the diagonalization function is representable in a theory K with equality. Then for all wff $\psi(x_1)$ in which x_1 is the only free variable, there is a **closed** formula φ such that:*

$$\vdash_K \varphi \leftrightarrow \psi(n(\varphi))$$

Theorem (fixed point theorem)

*Let us suppose that all recursive functions are representable in a theory with equality K. Then for all formula $\psi(x_1)$ in which x_1 is the only free variable, there is a **closed** wff φ such that:*

$$\vdash_K \varphi \leftrightarrow \psi(n(\varphi))$$

ω-consistency

Definition

Let K be a theory with constant symbol 0 and function symbol f_1^1. K is ω-consistent if for all wff $\varphi(x_1)$ with x_1 as only free variable, if $\vdash_K \neg\varphi(\overline{n})$ for all natural number n, then it is **false** that $\vdash_K \exists x\ \varphi(x)$.

Theorem

If K is ω-consistent, then K is consistent.

Easy: consider $\psi(x) \equiv \varphi(x) \wedge \neg\varphi(x)$.

First Gödel incompleteness theorem

Theorem (first incompleteness theorem)

Let K be a theory with equality in L_A satisfying:

1. *K has a recursive axioms set*
2. *$\vdash_K 0 \neq \overline{1}$*
3. *every recursive function is representable in K*

Let us consider the relation $Pf(x, y)$ meaning that x is the Gödel number of a proof in K of a formula having for Gödel number y, then there is a closed formula $\mathcal{G}$ such that:

$$\vdash_K \mathcal{G} \leftrightarrow \forall x \neg Pf(x, n(\mathcal{G}))$$

In this case, if K is ω-consistent, then it is false that $\vdash_K \mathcal{G}$ and it is false that $\vdash_K \neg\mathcal{G}$.
$\mathcal{G}$ is ***undecidable****.*

Second Gödel incompleteness theorem

Theorem (second Gödel incompleteness theorem)

Let K be an extension of S having a recursive axioms set. Let $\mathcal{C}_K$ be the following closed wff:

$$\forall x_1 \forall x_2 \forall x_3 \forall x_4 \ \neg(\mathcal{PF}(x_1, x_3) \wedge \mathcal{PF}(x_2, x_4) \wedge \mathcal{N}(x_3, x_4))$$

$\vdash_K \mathcal{C}_K \rightarrow \mathcal{G}$ *therefore if K is consistent, then $\mathcal{C}_K$ is not provable in K.*

8 - Logic programming

Introduction

There are several programming paradigms:

- procedural (C, Pascal...)
- object-oriented (SmallTalk, Objective C, C++, Java, Python, Ruby, ...)
- functional (Caml, LISP, Haskell)
- **logic** (Prolog)
- etc.

A Prolog program defines **relations**, not a algorithm.

Prolog program: an example

Example (sum of two natural numbers)

`add(z,X,X).`	for all X, $0 + X = X$
`add(s(Y),X,s(V)) :- add(Y,X,V).`	for all X, Y, V, if $X + Y = V$ then $s(Y) + X = s(V)$

We can ask Prolog some questions: `add(s(s(z)),s(s(s(z))),W)`?

- ➥ meaning: is there a `W` such that `add(s(s(z)),s(s(s(z))),W)` holds, i.e. `W = 2 + 3`?

In this case, Prolog answers `W = s(s(s(s(s(z)))))`.

Prolog basic principles

Instructions in a Prolog program can be viewed as the **premises** of an argument.

A request can be viewed as the **conclusion** of an argument from the previous premises.

The fact that the conclusion can be deduced from the premises is proved by using **Resolution**. **Variables** are used and instanciated.

Basic Prolog syntax

- representing data: using **terms**
- identifier beginning by a **lowercase** letter: function or predicate symbol
- identifier beginning by a **uppercase** letter: variable symbol

Outline of part 8 - Logic programming

Program clause

Definition (Prolog program)

A Prolog program is a **sequence** of clauses.

Syntax (Program clause)

$$A \ \texttt{:-} \ B_1, \ldots, B_n \ \texttt{.}$$

Intuition: for all possible values for variables, if $B_1, \ldots, B_n$ are all true, then A is true.

If $n = 0$, then the clause is a **fact** and is simply denoted by A**.**

Program clause

Definition (Prolog program)

A Prolog program is a **sequence** of clauses.

Syntax (Program clause)

$$A \ \texttt{:-} \ B_1, \ldots, B_n \ \texttt{.}$$

Intuition: for all possible values for variables, if $B_1, \ldots, B_n$ are all true, then A is true.

If $n = 0$, then the clause is a **fact** and is simply denoted by A**.**

Program clause

Definition (Prolog program)

A Prolog program is a **sequence** of clauses.

Syntax (Program clause)

$$\underbrace{A}_{\text{head}} \;\texttt{:-}\; \underbrace{B_1, \ldots, B_n}_{\text{body}} \texttt{.}$$

Intuition: for all possible values for variables, if $B_1, \ldots, B_n$ are all true, then A is true.

If $n = 0$, then the clause is a **fact** and is simply denoted by A**.**

Program clause

Definition (Prolog program)

A Prolog program is a **sequence** of clauses.

Syntax (Program clause)

$$\underbrace{A}_{\text{head}} \;\texttt{:-}\; \underbrace{B_1, \ldots, B_n}_{\text{body}} \texttt{.}$$

Intuition: for all possible values for variables, if $B_1, \ldots, B_n$ are all true, then A is true.

If $n = 0$, then the clause is a **fact** and is simply denoted by A**.**

Program clause

Definition (Prolog program)

A Prolog program is a **sequence** of clauses.

Syntax (Program clause)

$$\underbrace{A}_{\text{head}} \;\texttt{:-}\; \underbrace{B_1, \ldots, B_n}_{\text{body}} \texttt{.}$$

Intuition: for all possible values for variables, if $B_1, \ldots, B_n$ are all true, then A is true.

If $n = 0$, then the clause is a **fact** and is simply denoted by A**.**

Program clause

Definition (Prolog program)

A Prolog program is a **sequence** of clauses.

Syntax (Program clause)

A :- $B_1, \ldots, B_n$. ----- **beware of us!**

head body

Intuition: for all possible values for variables, if $B_1, \ldots, B_n$ are all true, then A is true.

If $n = 0$, then the clause is a **fact** and is simply denoted by A.

Query clause

Syntax (Query clause)

$$:\!-B_1, \ldots, B_n\,.$$

Intuition: are there values for variables such that $B_1, \ldots, B_n$ are **all** true?

Beware

For the program clause $B(X)$:- $C(Y, X)$.

- X is **universally** quantified
- Y is **existentially** quantified

Terminology: Prolog resolvent

Definition (Most general unifier)

If $p(t_1, \ldots, t_n)$ and $p(s_1, \ldots, s_n)$ are two atoms such thath the subsitution σ is a most general unifier of those atoms, then $p(t_1, \ldots, t_n)$ and $p(s_1, \ldots, s_n)$ are unifiable by **mgu** σ.

Definition (Prolog resolvent)

Let $R = \texttt{:-}\ A_1, \ldots, A_m$ be a query clause and $C = A'_1\ \texttt{:-}\ B_1, \ldots, B_p$ be a program clause with $m > 0$ and $p \geq 0$. If A_1 and A'_1 are unifiable by σ, then the new query clause $R' = \texttt{:-}\ \sigma(B_1, \ldots, B_p, A_2, \ldots, A_m)$ is called **Prolog resolvent** of R and C.

Idea behind Prolog resolvent

The idea behind Prolog resolvent is the same as behind Resolution in First-Order Logic (cf. last slides in this part).

More intuitively, consider a program clause $A' :\!- B_1, \ldots, B_n$. It can be read as "if $B_1, \ldots, B_n$ hold, then A' holds"...
...but you can also understand the clause as "to prove A', it is sufficient to prove $B_1, \ldots, B_n$".

Thus, when you have a request involving A such that A and A' are unifiable, you can replace the A part of the request by $B_1, \ldots, B_n$ (given the substitution).

Does it end? It depends (cf. next slides), but facts can be used, e.g.:

clause	`add(X, z, X).`
request	`add(s(s(z)), z, W).`
resolvent	empty clause with substitution `X/s(s(z))`

Evaluation algorithm

There is a **non-determinist** evaluation algorithm for Prolog.

Inputs: a Prolog program P and a query clause R

Output: two possibilities

- a substitution σ for the variables appearing in R (if R does not contain variables, the output is `YES`);
- `NO`

Given a Prolog program P and a query R, an evaluation of R can have three issues:

- ending with success
- ending with `NO`
- **no ending!**

Algorithm

Algorithm 23.1: Prolog evaluator

Input: a Prolog program P and a query `R`
Output: a substitution σ for variables appearing in `R`, else `NO`

1 $R_c \leftarrow R$;
2 $mgu_c \leftarrow \emptyset$;
3 **while** $R_c =$ **:-** $G_1, \ldots, G_k \neq \emptyset$ **do**
4 | **choose** $C = G'_1$ **:-** $D_1, \ldots D_t \in P$ st G_1 et G'_1 unifiable by σ ;
5 | **if** C **does not exist then**
6 | | `break` ;
7 | **end**
8 | $R_c \leftarrow$ Prolog resolvent of G_1 and C (replace G_1) ;
9 | $mgu_c \leftarrow mgu_c \circ \sigma$;
10 **end**
11 **if** $R_c = \emptyset$ **then**
12 | compute restriction σ' of mgu_c to R variables ;
13 | **if** $\sigma' = \emptyset$ **then**
14 | | **return YES** ;
15 | **else**
16 | | **return** σ' ;
17 | **end**
18 **else**
19 | **return No** ;
20 **end**

Outline of part 8 - Logic programming

Example

Program

1. `parent(jack,mary).`
2. `parent(louise,jack).`
3. `parent(franck,john).`
4. `ancestor(X,Y) :- parent(X,Y).`
5. `ancestor(X,Y) :- ancestor(X,Z), parent(Z,Y).`

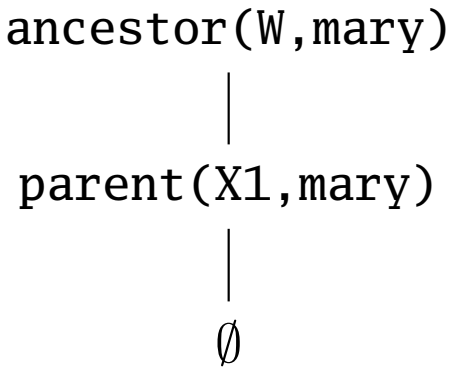
Query

```
:- ancestor(W,mary)
```

Some evaluation **examples** are presented in the next slides.

Example 1: evaluation with success

$\sigma = \emptyset\ \{W/X1, Y1/mary, X1/jack\ \}$

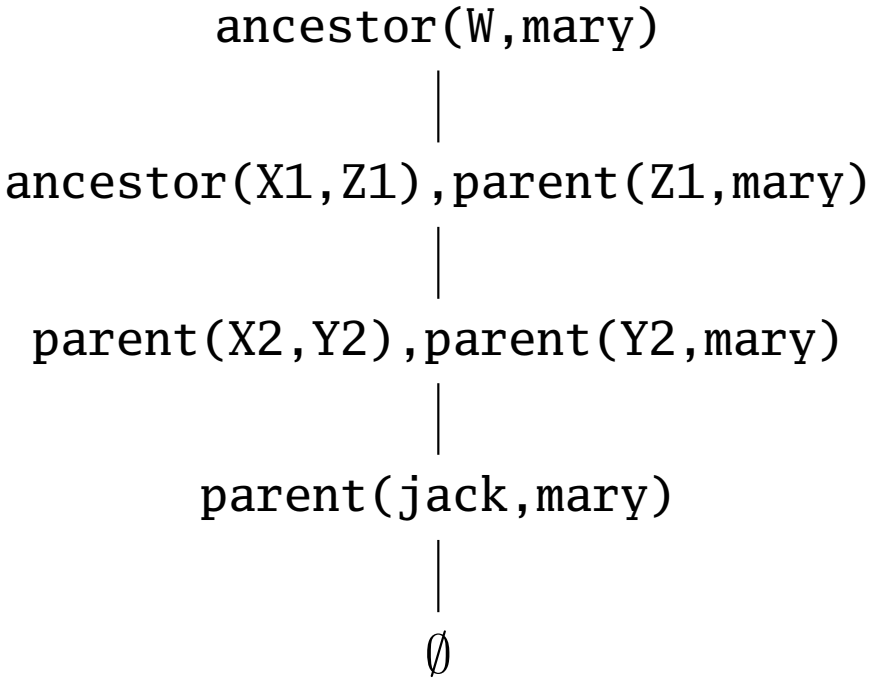



```
parent(jack,mary).
parent(louise,jack).
parent(franck,john).
ancestor(X,Y) :- parent(X,Y).
ancestor(X,Y) :- ancestor(X,Z),
                 parent(Z,Y).
```

Answer: $\{W/jack\}$

Example 2: evaluation with success

$\sigma = \emptyset\ \{W/X1, Y1/\mathit{mary}, X1/X2, Z1/Y2, X2/\mathit{louise}, Y2/\mathit{jack}\ \}$

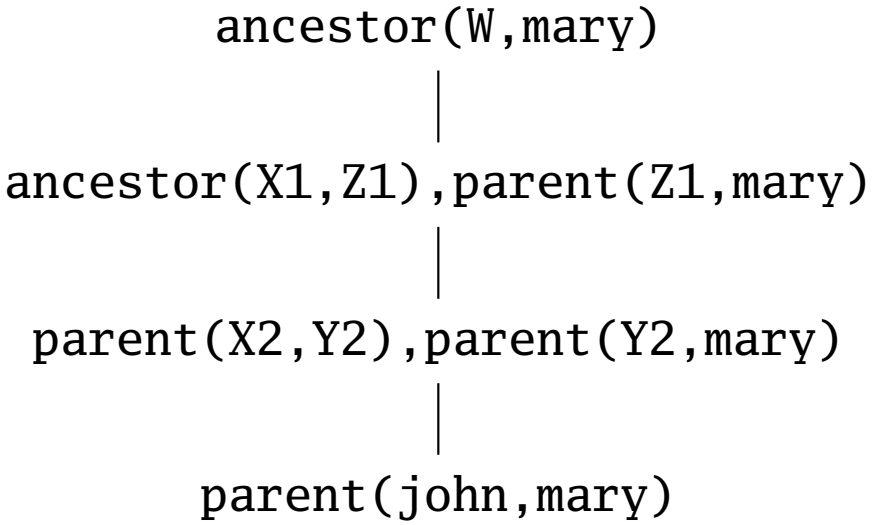



```
parent(jack,mary).
parent(louise,jack).
parent(franck,john).
ancestor(X,Y) :- parent(X,Y).
ancestor(X,Y) :- ancestor(X,Z),
                 parent(Z,Y).
```

Answer: $\{W/\mathit{louise}\}$

Example 3: evaluation with failure

$\sigma = \emptyset\ \{W/X1, Y1/\mathit{mary}, X1/X2, Z1/Y2, X2/\mathit{franck}, Y2/\mathit{john}\ \}$

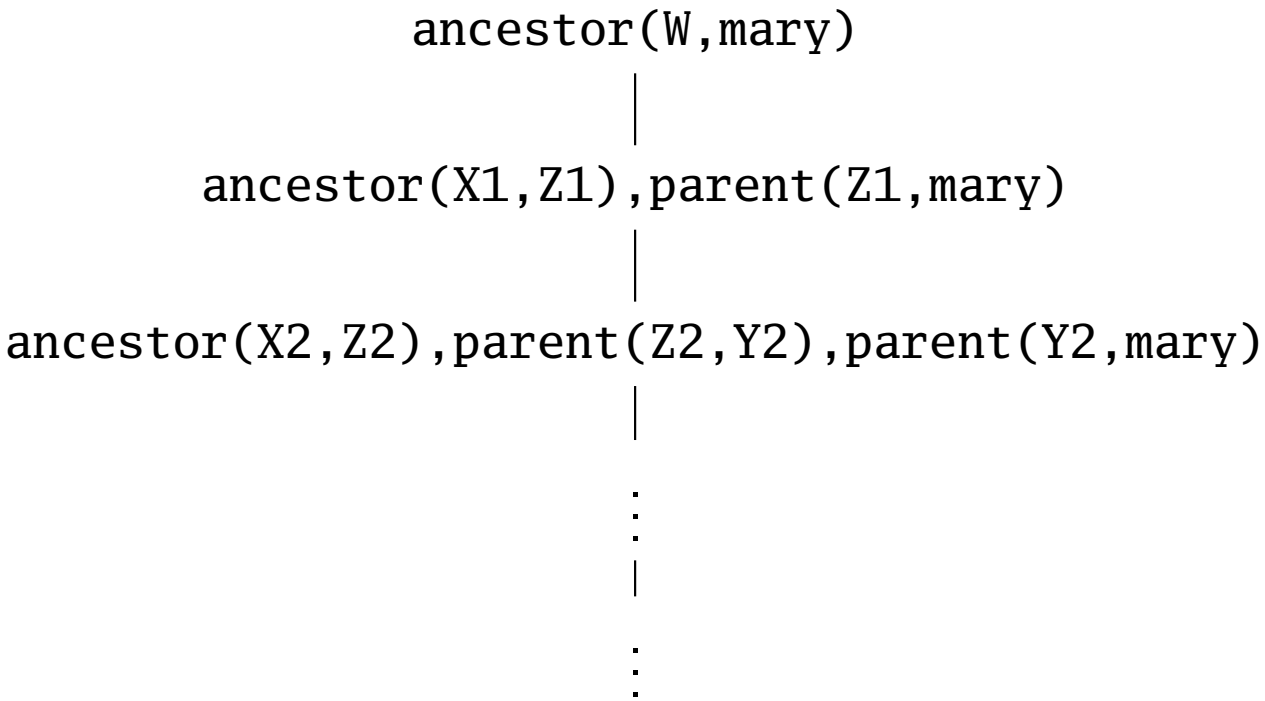



```
parent(jack,mary).
parent(louise,jack).
parent(franck,john).
ancestor(X,Y) :- parent(X,Y).
ancestor(X,Y) :- ancestor(X,Z),
                 parent(Z,Y).
```

Answer: NO

Example: evaluation with no ending

Using only clause `ancestor(X,Y) :- ancestor(X,Z),parent(Z,Y).`



No ending!

Outline of part 8 - Logic programming

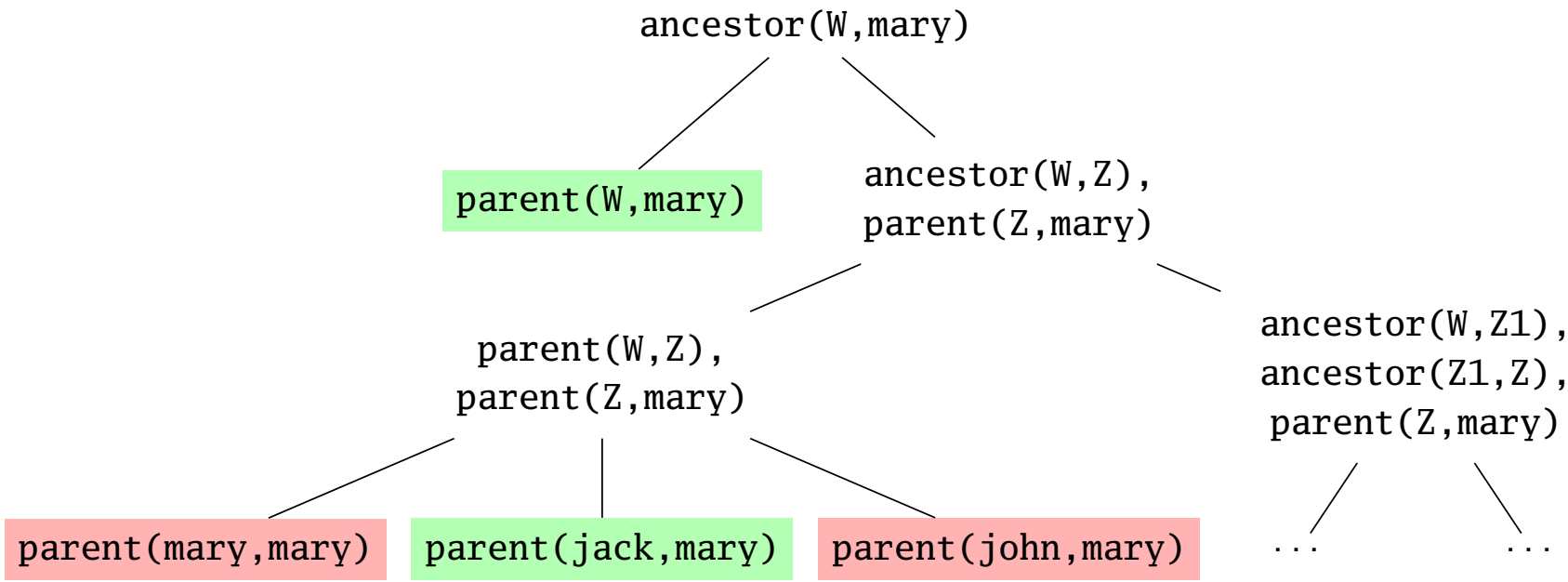
Prolog search tree

Definition (Prolog search tree)

A Prolog search tree for a program P and a query R is a tree whose nodes are query clauses and such that:

- its **root** is R;
- if a node is a non-empty query $\texttt{:-}\ A_1, \ldots, A_n$ (where $n > 0$) and $C_1, \ldots, C_k$ (where $k > 0$) are the program clauses (**appearing in this order** in P) whose heads are unifiable with A_1, then this node has k children $Res_1, \ldots, Res_k$ where for $i \in \{1, \ldots, k\}$, Res_i **is the Prolog resolvent** of A_1 with the clause C_i;
- if a node is a non-empty query $\texttt{:-}\ A_1, \ldots, A_n$ (where $n > 0$) and if there is no program clause whose head is unifiable with A_1, then this node is a **failure leaf**;
- if a node is the empty query, then this node is a **success leaf**.

Prolog search tree for our example (simplified)



The Prolog tree on our example is **infinite on the right**.

Exploring the search tree

Depth-first exploration

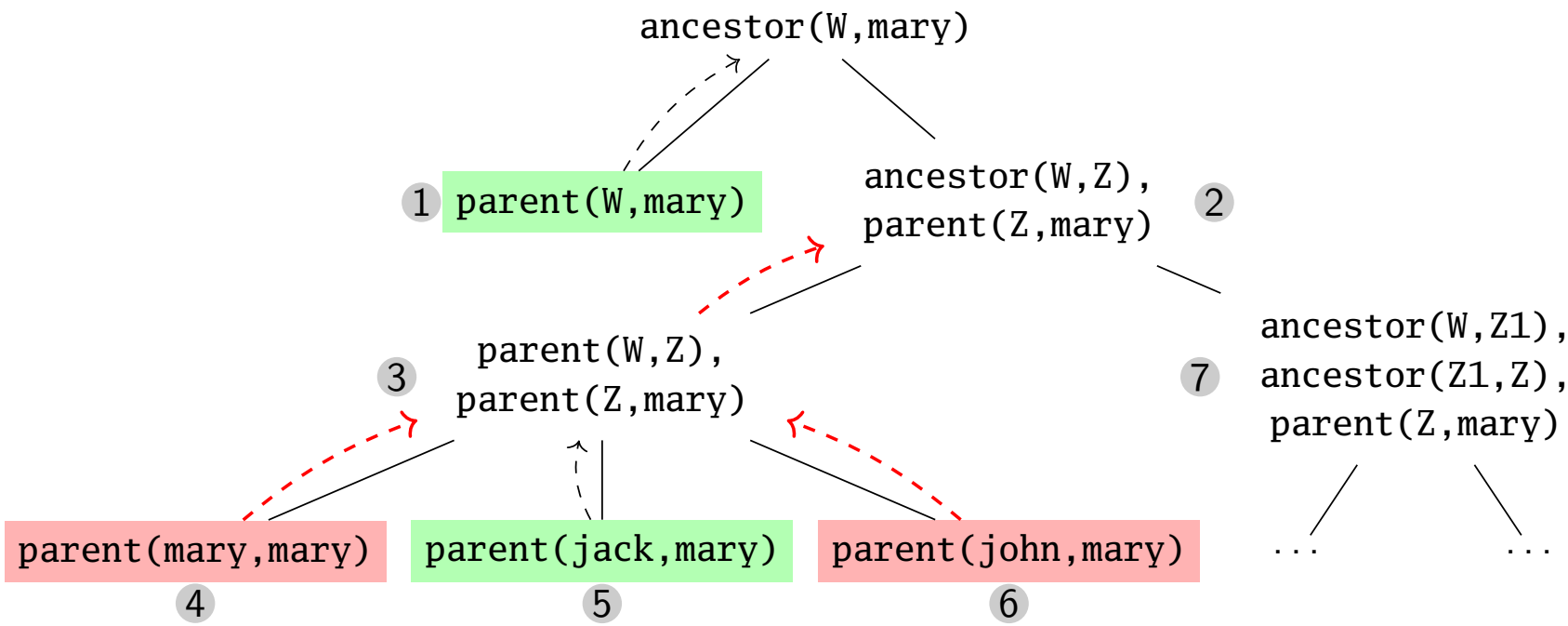
- if the current branch ends with success, the evaluation stops and give the corresponding answer;
- if the current branch ends with failure, the next branch is considered. The next clause usable for the query represented by the parent node of the current node is chosen (**backtracking**);
- if after having given the result of a success branch the user send a "continue" instruction, the next branch is considered as if the current branch had failed (**backtracking**);
- if the branch does not end, the evaluator does not stop.

Exploring the search tree

Thus...

- the answers to the query R are given into an order that depends of the writing order of clauses and atoms in P;
- infinite branch $\Rightarrow$ the interpreter does not stop;
- the answer `NO` corresponds to the case where the tree is **finite** and where every branch is a failure branch.

Prolog search tree building on our example (simplified)



- - - - - - - - - → user backtracking after success

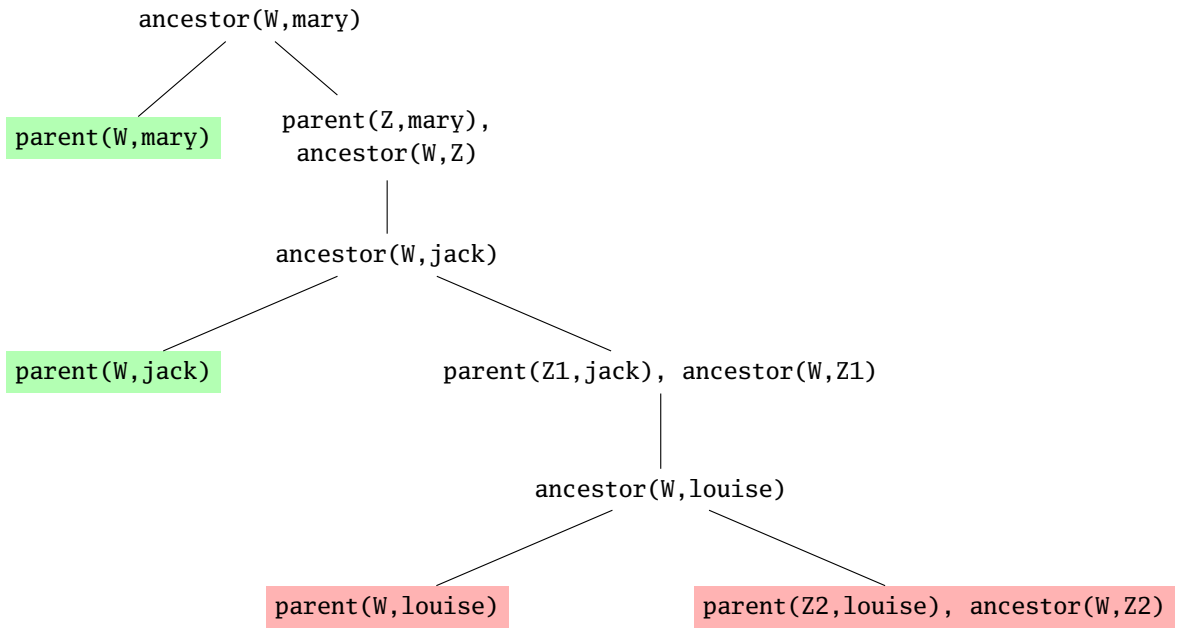
- - - - - - - - - → Prolog backtracking after failure

Atoms order is important!

Theorem

*The order of **atoms** in the clauses bodies determine the structure of the search tree.*

For instance, replace in the previous program the clause 5 by `ancestor(X,Y) :- parent(Z,Y),ancestor(X,Z).`



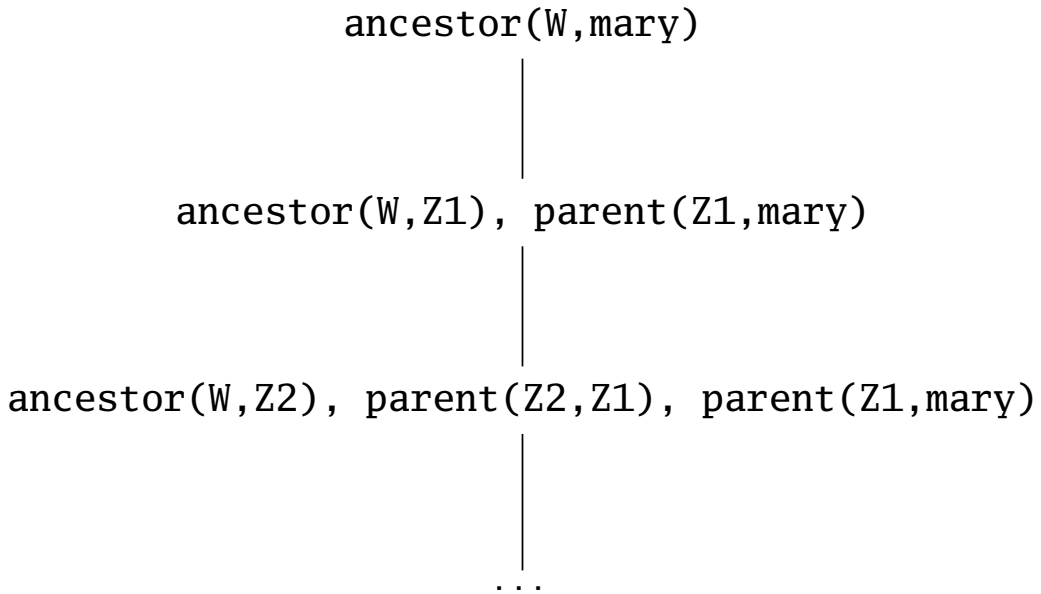
Clauses order is important!

Theorem

*The order of the **clauses** in the program is important:*

- *the answers order can change*
- *the Prolog interpretor can loop*

For instance, swap clauses 4 and 5...



Outline of part 8 - Logic programming

Prolog and Resolution?

The formula associated to $A_1 : -B_1, \ldots, B_m$ with variables $X_1, \ldots, X_n$ is $\forall x_1 \ldots \forall x_n \ ((B_1 \wedge \ldots \wedge B_m) \rightarrow A_1)$, thus $\neg B_1 \vee \ldots \vee \neg B_m \vee A_1$.

NB: there is only one positive literal in the clause.

The formula associated to the query $: -B_1, \ldots, B_m$ is $\forall x_1 \ldots \forall x_n \ (B_1 \wedge \ldots \wedge B_m) \rightarrow \bot$, thus $\neg B_1 \vee \ldots \vee \neg B_m$.

Intuition: find a refutation with Resolution!

Definition (Horn clause)

A clause is **defined** if it contains one and only one positive literal. A clause is **negative** if it does not contain positive literal.
A **Horn clause** is either a defined clause, either a negative clause.

Link between Prolog and Resolution

For instance: the first successful branch.

A successful branch **is just a refutation using Resolution** from the set of Horn clauses.

$\exists x_1 \ldots \exists x_n \ (B_1 \wedge \ldots \wedge B_m)$ is a logical consequence of the program.

A **constructive** proof is found (the variables are assigned).

Prolog uses the **Linear Resolution with Selection Function**.