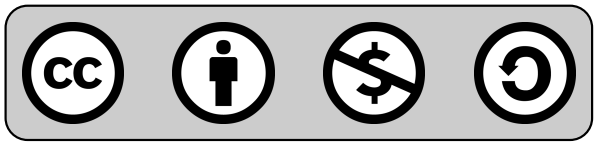
N7PD Declarative Programming

Logic Programming – Resolution

Christophe Garion

ISAE-SUPAERO/DISC

License CC BY-NC-SA 3.0



Lectures on Declarative Programming

1. **logic programming** (C. Garion, ISAE-SUPAERO)
   2 lectures + 1 evaluated lab
2. **constraint programming** (N. Barnier, ENAC)
   1 lecture + 1 lab + 1 evaluated lab
3. **SAT** (C. Garion, ISAE-SUPAERO)
   1 lecture + 1 lab
4. **SMT** (K. Delmas, ONERA)
   1 lecture + 1 lab + 2 evaluated labs

Team: the ones above +

- G. Dupont (ENSEEIHT), you should know him ☺

Introduction: what is declarative programming?

Informal definition from Wikipedia

Declarative programming: express the **logic** of computation without describing its control flow.

Broadly speaking: **what** is the problem and **not how** to compute a solution.

You already know some examples of declarative programming!

- SQL
- regular expressions
- ...

Here: logic programming

We are interested in this first part of the lecture in logic programming.

In logic programming, programs are written as sentences in **logical form**.

What can we do with such a program?

➥ find a correct conclusion from the program to solve a problem

Example:

Program

$$\begin{aligned} \forall x\ add(zero, x) &= x \\ \forall x \forall y\ add(succ(x), y) &= succ(add(x, y)) \end{aligned}$$

Question: $\exists w\ add(succ(succ(zero)), succ(succ(zero))) = w$?

Logic programming, Prolog, Resolution

We will use the Prolog logic programming language.

Prolog use **first-order logic** as a formal language to write programs and a **formal system** called Resolution to answer queries on programs.

Therefore we will

- first study Resolution and show how it can be used to solve problems
- introduce Prolog and its mechanisms

Notice that

- Prolog will also be used during the lecture on constraint programming
- Resolution will also be used during the lecture on SAT

Learning outcomes

At the end of this session, you should

- have understood syntax of propositional logic and first-order logic
- be able to compute normal forms (CNF for propositional logic and Skolem standard form for first-order logic)
- be able to use the Resolution formal system to verify that an argument is correct
- have understood on an example that Resolution can be used for declarative programming

Outline

Outline

Syntax of propositional logic

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

Let *Var* be a set of propositional variables. The syntax of well-formed formulas of $\mathcal{L}_{PL}$ is given by the following EBNF:

$$\varphi ::= \text{'}A\text{'} \mid \text{'}\top\text{'} \mid \text{'}\bot\text{'} \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \rightarrow \varphi \mid \varphi \leftrightarrow \varphi$$

where $A \in \mathit{Var}$.

The semantics of propositional logic is defined classically (cf. MOD first year lecture).

The validity of formula φ, denoted by $\models \varphi$, and the logical consequence of φ from Σ, denoted by $\Sigma \models \varphi$, are defined as usual.

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 $\varphi \equiv \top$.

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 $L_1 \vee \ldots \vee L_n$ 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:

$$\varphi \rightarrow \psi \equiv \neg\varphi \vee \psi$$

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 algorithm (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 algorithm (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 algorithm (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: do return φ;
3 |   case φ is φ1 ∧ φ2: do return remove-imp(φ1) ∧ remove-imp(φ2);
4 |   case φ is φ1 ∨ φ2: do return remove-imp(φ1) ∨ remove-imp(φ2);
5 |   case φ is φ1 → φ2: do return ¬ remove-imp(φ1) ∨ remove-imp(φ2);
6 end
```

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: do return φ;
3 |   case φ is ¬¬φ1: do return φ1;
4 |   case φ is φ1 ∧ φ2: do return NNF(φ1) ∧ NNF(φ2);
5 |   case φ is φ1 ∨ φ2: do return NNF(φ1) ∨ NNF(φ2);
6 |   case φ is ¬(φ1 ∧ φ2): do return NNF(¬φ1) ∨ NNF(¬φ2);
7 |   case φ is ¬(φ1 ∨ φ2): do return NNF(¬φ1) ∧ NNF(¬φ2);
8 end
```

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: do return φ;
3 |   case φ is φ1 ∧ φ2: do return CNF(φ1) ∧ CNF(φ2);
4 |   case φ is φ1 ∨ φ2: do return DISTR(CNF(φ1), CNF(φ2));
5 end
```

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

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

Outline

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 $\frac{e_1 \;\; e_2}{e_3}$ 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:

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

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 $q \vee \neg q \to p \vee r$:

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

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

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).

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.

Soundness of $\mathcal{R}$

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

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

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 $\Gamma \models c$.

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.

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!

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).

John's train problem

Exercise

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.*

Verify in $\mathcal{R}$ that the previous argument is correct.

Outline

Outline

Syntax of first-order logic

Definition (syntax of $\mathcal{L}_{FOL}$)

Let $\mathcal{V}$ be a set of variables, $\mathcal{F}$ be a set of function symbols and $\mathcal{P}$ be a set of predicate symbols. The syntax of well-formed formulas of $\mathcal{L}_{FOL}$ is given by the following EBNF:

$$\begin{aligned} \text{term} &::= \text{'}x\text{'} \mid \text{'}f\text{'}\ (\mathit{term}, ..., \mathit{term}) \\ \varphi &::= \text{'}P\text{'}(\text{term}) \mid \text{'}\top\text{'} \mid \text{'}\bot\text{'} \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \rightarrow \varphi \mid \\ &\qquad \varphi \leftrightarrow \varphi \mid \forall x\ \varphi \end{aligned}$$

where $x \in \mathcal{V}$, $f \in \mathcal{F}$ and $P \in \mathcal{P}$ with the adequate arities.

The semantics of first-order logic is defined classically (cf. MOD first year lecture).

The validity of formula φ, denoted by $\models \varphi$, and the logical consequence of φ from Σ, denoted by $\Sigma \models \varphi$, are defined as usual.

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.

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

- transform matrix into a formula in CNF

$$\varphi \rightsquigarrow Q_1 x_1 \ldots Q_n x_n \quad M[x_1, \ldots, x_n] \text{ where } M \text{ is a formula in CNF.}$$

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 Σ is the union of the clausal forms of the wffs appearing in Σ

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!

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 Σ be a set of wffs and $CL(\Sigma)$ the set of clausal forms obtained from Σ using the previous procedure. Σ is satisfiable iff $CL(\Sigma)$ is satisfiable.

For the Resolution formal system, think about this alternative formulation of the theorem: Σ is unsatisfiable iff $CL(\Sigma)$ unsatisfiable...

Outline

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.

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 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.

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 3: there are some variables that must be used in the Resolution rule with some ground terms.

clauses set	deduced wff	OK?
$\{P(x), \neg P(a)\}$	$\square$	✔
$\{P(x) \vee Q(b), \neg P(a) \vee R(c)\}$	$Q(b) \vee R(c)$	✔
$\{P(x) \vee Q(x), \neg P(a) \vee R(c)\}$	$Q(\mathrm{a}) \vee R(c)$	✔

Remember that **the variables are universally quantified**, so you can **substitute** the variables by some other terms.

Notice that you have to substitute all the concerned variable instances.

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 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 are substitutions that "make the formulas identical". 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 $\{P(x, f(y, z)), P(x, a), P(x, g(h(k(x))))\}$:

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 $\{P(a, x, f(g(y))), P(z, f(z), f(u))\}$?

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 $\square$.

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.

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 $P(x, a) \rightarrow Goal(x)$, use it with Resolution and stop when producing a clause with only $Goal$ literals.

Conclusion

We have shown in the last example that FOL and Resolution can be used to model and solve problems.

Questions:

- can we write a "real" programming language with this formal system?
- which strategy should we use when applying Resolution (in particular, which clauses to select)?

See you for the next lecture on Prolog!