Artificial Intelligence

Blai Bonet

Universidad Simón Bolívar, Caracas, Venezuela

Inference in first-order logic

Goals for the lecture

- Language, syntax and semantics for first-order logic
- Inference problem for first-order logic
- CNF fragment, clausal form and conversion to clausal form
- First-order resolution
- Unification
- Resolution strategies

First-order logic

First-order logic (FOL) is more expressive than propositional logic, but inference is much more difficult

More expressive means that certain properties for objects cannot be expressed in propositional logic and thus, we can't do inference about such things

We address the representation and inference problem for FOL

Syntax of first-order logic: Language

Language (set of symbols) is more complex than before

Language $\mathcal{L}$ made of

- symbols $a, b, c, \ldots$ denoting **constants** (objects, persons, etc)
- symbols $x, y, z, \ldots$ denoting **variables** (instantiated to objects)
- symbols $f, g, h, \ldots$ denoting **functions** among objects (each function symbol has **fixed arity**)
- symbols $P, Q, R, \ldots$ denoting **relations** among objects (each relation symbol has **fixed arity**)
- logical connectors: $\neg, \vee, \wedge, \rightarrow, \leftrightarrow$
- quantifier symbols: $\forall$, $\exists$
- parentization symbols: (,), [,], ...

Syntax for first-order logic: Terms

A term denotes an object in the world

Terms over $\mathcal{L}$ constructed inductively:

- each constant symbol a is a term
- each variable symbol x is a term
- if f is a function symbol of arity k and $t_1, t_2, \ldots, t_k$ are terms, $f(t_1, t_2, \ldots, t_k)$ is a term

Example: for constant symbol a and function f of arity 1, the following are terms: a, $f(a)$, $f(f(a))$, $f(f(f(a)))$, ...

Syntax for first-order logic: Atoms and literals

Atoms and literals are the simplest formulas that can be constructed

If R is a relational symbol of arity k and $t_1, t_2, \ldots, t_k$ are terms, $R(t_1, t_2, \ldots, t_k)$ is an **atom** and also a **positive literal**

If $R(t_1, t_2, \ldots, t_k)$ is an atom, $\neg R(t_1, t_2, \ldots, t_k)$ is a **negative literal**

Example: for unary function f, constant a and binary relation R, the following are atoms: $R(a, a)$, $R(f(f(a)), a)$, $R(f(a), f(f(f(a))))$, ...

Syntax for first-order logic: Formulas

Formulas are statements that receive a truth valuation under a model, either true or false

Formulas over $\mathcal{L}$ built inductively from terms:

- if $\varphi = R(t_1, t_2, \ldots, t_k)$ is an atom, φ is a formula
- if φ is a formula, $\neg\varphi$ is a formula
- if φ and ψ are formulas, $\varphi \wedge \psi$, $\varphi \vee \psi$, $\varphi \rightarrow \psi$ and $\varphi \leftrightarrow \psi$ are formulas
- if φ is a formula and x is a variable, $\forall x \varphi$ and $\exists x \varphi$ are formulas

Example: the following are formulas $\forall x \exists y (R(x) \rightarrow S(y))$ and $\exists y \forall x (R(x) \rightarrow S(y))$

Free variables and sentences

We define notions of when a variable appears free in a formula

Let x be variable. We consider term and formulas:

- x doesn't appear free in term $t = a$ for constant a
- x appears free in term $t = y$ iff $y = x$
- x appears free in term $t = f(t_1, \ldots, t_k)$ iff x appears free in some t_i
- x appears free in formula $\varphi = R(t_1, \ldots, t_k)$ iff x appears free in some t_i
- x appears free in formula $\neg\varphi$ iff x appears free in φ
- x appears free in formula $\varphi \wedge \psi$ iff x appears free in φ or free in ψ
- similar for formulas $\varphi \vee \psi$, $\varphi \rightarrow \psi$ and $\varphi \leftrightarrow \psi$
- x appears free in formula $\forall y \varphi$ iff $y \neq x$ and x appears free in φ
- similar for formula $\exists x \varphi$

If no variable appears free in formula φ, we say that φ is a **sentence**

Semantics of first-order logic: Models

A sentence talks about "objects/individuals" in a "universe"

Models thus need to express all **conceivable universes** and **relations** between their constituent members

For language $\mathcal{L}$ containing constants $\{a_i\}_i$, function symbols $\{f_i^{r_i}\}_i$ and relational symbols $\{R_i^{s_i}\}_i$, a model is a **structure**

$$\mathcal{A} \;=\; (U^{\mathcal{A}}, \{a_i^{\mathcal{A}}\}_i, \{f_i^{\mathcal{A}}\}_i, \{R_i^{\mathcal{A}}\}_i)$$

where

- $U^{\mathcal{A}}$ is **universe** of $\mathcal{A}$ (of arbitrary size)
- each $a_i^{\mathcal{A}}$, member of $U^{\mathcal{A}}$, is **interpretation** of constant a_i
- each $f_i^{\mathcal{A}}$, r_i-ary function mapping r_i-tuples over $U^{\mathcal{A}}$ to $U^{\mathcal{A}}$, is interpretation of functional symbol f_i
- each $R_i^{\mathcal{A}}$, s_i-ary relation over $U^{\mathcal{A}}$, is interpretation of relational symbol R_i

Semantics of first-order logic: Assignments

Structures $\mathcal{A}$ offer interpretations for all elements in language except variables

For variables, we interpret them with assignments ν that map variable symbols to elements in universe $U^{\mathcal{A}}$ of $\mathcal{A}$

Let ν be assignment, x be variable symbol and u be element in $U^{\mathcal{A}}$. The assignment $\nu[x/u]$ is the assignment given by

$$\nu[x/u](y) \;=\; \begin{cases} u & \text{if } y = x \\ \nu(y) & \text{if } y \neq x \end{cases}$$

Semantics of first-order logic: Terms

Terms are interpreted as elements in universe of structure

If t is term, $\mathcal{A}$ is structure, and ν is assignment, the interpretation of t in $\mathcal{A}$, denoted by $\nu^{\mathcal{A}}(t)$, is defined as

- if $t = a_i$, then $\nu^{\mathcal{A}}(t) = a_i^{\mathcal{A}}$
- if $t = x$, then $\nu^{\mathcal{A}}(t) = \nu(x)$
- if $t = f_i(t_1, \ldots, t_k)$, then $\nu^{\mathcal{A}}(t) = f_i^{\mathcal{A}}(\nu^{\mathcal{A}}(t_1), \ldots, \nu^{\mathcal{A}}(t_n))$

Semantics of first-order logic: Formulas

Validity of formulas is defined using structures and assignments

Let $\mathcal{A}$ be structure and ν be assignment:

- $(\mathcal{A},\nu) \vDash R_i(t_1,\ldots,t_k)$ iff $(\nu^{\mathcal{A}}(t_1),\ldots,\nu^{\mathcal{A}}(t_k)) \in R_i^{\mathcal{A}}$
- $(\mathcal{A},\nu) \vDash \neg\varphi$ iff $(\mathcal{A},\nu) \nvDash \varphi$
- $(\mathcal{A},\nu) \vDash \varphi \wedge \psi$ iff $(\mathcal{A},\nu) \vDash \varphi$ and $(\mathcal{A},\nu) \vDash \psi$
- $(\mathcal{A},\nu) \vDash \varphi \vee \psi$ iff $(\mathcal{A},\nu) \vDash \varphi$ or $(\mathcal{A},\nu) \vDash \psi$
- $(\mathcal{A},\nu) \vDash \varphi \rightarrow \psi$ iff $(\mathcal{A},\nu) \nvDash \varphi$ or $(\mathcal{A},\nu) \vDash \psi$
- $(\mathcal{A},\nu) \vDash \varphi \leftrightarrow \psi$ iff $(\mathcal{A},\nu) \vDash \varphi \rightarrow \psi$ and $(\mathcal{A},\nu) \vDash \psi \rightarrow \varphi$
- $(\mathcal{A},\nu) \vDash \forall x\varphi$ iff $(\mathcal{A},\nu[x/u]) \vDash \varphi$ for each $u \in U^{\mathcal{A}}$
- $(\mathcal{A},\nu) \vDash \exists x\varphi$ iff $(\mathcal{A},\nu[x/u]) \vDash \varphi$ for some $u \in U^{\mathcal{A}}$

Semantics of first-order logic: Entailment

Structure $\mathcal{A}$ is a model for formula φ, written as $\mathcal{A} \vDash \varphi$, iff $(\mathcal{A},\nu) \vDash \varphi$ for **every assignment** ν

If φ is a sentence, φ has no free variables and thus $(\mathcal{A},\nu) \vDash \varphi$ is **independent** of the assignment ν

Thus, for sentences φ, $\mathcal{A} \vDash \varphi$ iff $(\mathcal{A},\nu) \vDash \varphi$ for **some assignment** ν

Logical consequence

Given two sentences φ and ψ over $\mathcal{L}$, we say that ψ is a **logical consequence** of φ (or that φ **entails** ψ, or ψ is entailed by φ) iff every model for φ is also a model for ψ

In symbols,

$$\mathcal{A} \vDash \varphi \quad \Longrightarrow \quad \mathcal{A} \vDash \psi$$

for every model $\mathcal{A}$ over $\mathcal{L}$

If φ entails ψ, we write $\varphi \vDash \psi$

Basic inference problem

Given two sentences φ and ψ over $\mathcal{L}$, the **basic inference problem** is to determine whether φ entails ψ (i.e. check whether $\varphi \vDash \psi$)

Fundamental result: $\varphi \vDash \psi$ iff $\varphi \wedge \neg\psi$ is **unsatisfiable**

Proof: $(\Rightarrow)$ $\varphi \vDash \psi$ iff for every model $\mathcal{A}$ that makes φ true, then $\mathcal{A}$ makes ψ true as well. Therefore, there is no model $\mathcal{A}$ such that $\mathcal{A} \vDash \varphi \wedge \neg\psi$ (i.e. $\varphi \wedge \neg\psi$ is unsatisfiable)

$(\Leftarrow)$ $\varphi \wedge \neg\psi$ is unsatisfiable iff there is no model $\mathcal{A}$ that makes $\varphi \wedge \neg\psi$ true. Therefore, for every model $\mathcal{A}$ over $\mathcal{L}$, if $\mathcal{A}$ makes φ true, then it must make ψ true as well

General satisfiability problem for FOL

The satisfiability problem for FOL is:

Given a sentence φ over a language $\mathcal{L}$, determine whether φ is satisfiable

The satisfiability problem is not even **decidable**: there is no algorithm to check whether a given sentence is satisfiable

The satisfiability problem is **partially decidable**

CNF Fragment

As with propositional logic, we define the CNF fragment for FOL
A FOL sentence φ is in CNF iff $\varphi = \forall x_1 \forall x_2 \cdots \forall x_n \psi$ where:

– ψ is quantifier free

– the variables in ψ are $\{x_1, x_2, \ldots, x_n\}$

– ψ is a conjunction of disjunctions of literals (i.e. a conjunction of clauses)

Example: for language $\mathcal{L} = \{a, f^1, P^2\}$, the sentence

$$\forall x \forall y \forall z \forall w[(\neg P(y,a) \vee P(y,x)) \wedge \neg P(x,z) \wedge (P(w,a) \vee P(w,f(w)))]$$

is in CNF

Clausal form

Like propositional logic, there is a clausal form for FOL in which a CNF sentence is represented by a collection of subsets of literals

Let $\varphi = \forall x_1 \cdots \forall x_n [C_1 \wedge C_2 \wedge \cdots \wedge C_m]$ be a CNF sentence. We have $\varphi \equiv \psi_1 \wedge \psi_2 \wedge \cdots \wedge \psi_m$ where

$$\begin{aligned} \psi_1 &= \forall x_1 \cdots \forall x_n C_1 \\ \psi_2 &= \forall x_1 \cdots \forall x_n C_2 \\ &\cdots \\ \psi_m &= \forall x_1 \cdots \forall x_n C_m \end{aligned}$$

Now **rename variables** so that no variable is shared between ψ_i and ψ_j

Clausal form is obtained by representing each clause ψ as the set of its literals, and φ as the collection of subsets for its clauses

Example of clausal form

Consider the CNF sentence

$$\forall x \forall y \forall z \forall w[(\neg P(y,a) \vee P(y,x)) \wedge \neg P(x,z) \wedge (P(w,a) \vee P(w,f(w)))]$$

Its clausal form is

$$\{\ \{\neg P(y_1,a), P(y_1,x_1)\},\ \{\neg P(x_2,z_2)\},\ \{P(w_3,a), P(w_3,f(w_3))\}\ \}$$

Conversion to clausal form

Every sentence φ can be transformed into a sentence ψ in clausal form (i.e. CNF) while **preserving satisfiability**:

$$\varphi \text{ is satisfiable} \iff \psi \text{ is satisfiable}$$

In general, φ is not **logically equivalent** to ψ, yet this doesn't matter as we are interested in checking only whether φ is satisfiable or not

Algorithm for converting sentences into clausal form

1. Remove logical connectors $\rightarrow$ and $\leftrightarrow$ by expressions involving $\vee$, $\wedge$ and $\neg$
2. Use De Morgan's rules to move negations down to literal level
3. Rename quantified variables uniquely (for each quantifier, replace quantified variable by a new unused variable)
4. Remove existential quantification by **skolemization**
5. Distribute $\vee$ over $\wedge$ until obtaining a conjunction of disjunctions
6. [Sentence is now in CNF] Separate into individual quantified clauses
7. Obtain clausal form

Skolemization: Intuition for simple case

Consider sentence $\varphi = \exists x \psi(x)$ where notation $\psi(x)$ means the free variables in ψ are in $\{x\}$

Let $\mathcal{A}$ be structure and ν be such that $(\mathcal{A}, \nu) \vDash \varphi$. By definition, there is $u^* \in U^{\mathcal{A}}$ such that $(\mathcal{A}, \nu[x/u^*]) \vDash \psi$

Therefore, we can "create" constant a and new model $\mathcal{A}'$ equal to $\mathcal{A}$ but **extended** with constant a mapped to object u^*. For new model, $(\mathcal{A}', \nu) \vDash \psi[x/a]$

That is, sentence $\exists x \psi$ is satisfiable iff sentence $\psi[x/a]$ is satisfiable where a is **new constant symbol** (a is called **skolem constant**)

Skolemization: Intuition for general case

Consider sentence $\varphi = \forall z \exists x \psi(z, x)$ where notation $\psi(z, x)$ means the free variables in ψ are in $\{z, x\}$

Let $\mathcal{A}$ be structure and ν be such that $(\mathcal{A}, \nu) \vDash \varphi$. By definition, for every $u \in U^{\mathcal{A}}$ there is $u^* \in U^{\mathcal{A}}$ such that $(\mathcal{A}, \nu[z/u, x/u^*]) \vDash \psi$

The object u^* **depends** on the object u and thus we cannot create constant for it. We need a **function** for u^*

Therefore, we "create" a new function f^1 and a new model $\mathcal{A}'$ equal to $\mathcal{A}$ but extended with function f such that $(\mathcal{A}', \nu) \vDash \forall z \psi[x/f(z)]$

That is, sentence $\forall z \exists x \psi$ is satisfiable iff sentence $\forall z \psi[x/f(z)]$ is satisfiable where a is **new function symbol** (f is called **skolem function**)

Skolemization

Let φ by sentence containing the formula $\exists x \psi(x, z_1, \ldots, z_n)$ where notation $\psi(x, z_1, \ldots, z_n)$ means that the free variables in ψ are among $\{x, z_1, \ldots, z_n\}$

Assume also that variables in $\{x, z_1, \ldots, z_n\}$ are all **universally quantifed** in scope of ψ

Sentence φ is satisfiable iff sentence φ' is satisfiable where φ' is equal to φ but where formula $\exists x \psi$ is replaced by $\psi[x/f(z_1, \ldots, z_n)]$ with f being a **new skolem function** of arity n

If $n = 0$ (i.e. formula ψ has no universal quantified free variable), the skolem function is a skolem constant

Example of conversion (steps 1, 2 and 3)

Consider sentence $\exists y \forall z[P(z, y) \leftrightarrow \neg\exists x(P(z, x) \land P(x, z))]$

1. Remove logical connectors $\rightarrow$ and $\leftrightarrow$:

$$\exists y \forall z\big[\,[P(z, y) \rightarrow \neg\exists x(P(z, x) \land P(x, z))] \land [P(z, y) \leftarrow \neg\exists x(P(z, x) \land P(x, z))]\,\big]$$

$$\exists y \forall z\big[\,[\neg P(z, y) \lor \neg\exists x(P(z, x) \land P(x, z))] \land [P(z, y) \lor \exists x(P(z, x) \land P(x, z))]\,\big]$$

2. Move negations down to literal level:

$$\exists y \forall z\big[\,[\neg P(z, y) \lor \forall x(\neg P(z, x) \lor \neg P(x, z))] \land [P(z, y) \lor \exists x(P(z, x) \land P(x, z))]\,\big]$$

3. Rename quantified variables uniquely:

$$\exists y \forall z\big[\,[\neg P(z, y) \lor \forall x_1(\neg P(z, x_1) \lor \neg P(x_1, z))] \land [P(z, y) \lor \exists x_2(P(z, x_2) \land P(x_2, z))]\,\big]$$

Example of conversion (steps 4 and 5)

Sentence after step 3:

$$\exists y \forall z\big[\,[\neg P(z, y) \lor \forall x_1(\neg P(z, x_1) \lor \neg P(x_1, z))] \land [P(z, y) \lor \exists x_2(P(z, x_2) \land P(x_2, z))]\,\big]$$

4. Remove existential quantification (skolemization):

$$\forall z\big[\,[\neg P(z, a) \lor \forall x_1(\neg P(z, x_1) \lor \neg P(x_1, z))] \land [P(z, a) \lor \exists x_2(P(z, x_2) \land P(x_2, z))]\,\big]$$

$$\forall z\big[\,[\neg P(z, a) \lor \forall x_1(\neg P(z, x_1) \lor \neg P(x_1, z))] \land [P(z, a) \lor (P(z, f(z)) \land P(f(z), z))]\,\big]$$

5. Distribute $\lor$ over $\land$:

$$\begin{aligned}\forall z\big[\,&[\neg P(z, a) \lor \forall x_1(\neg P(z, x_1) \lor \neg P(x_1, z))] \land \\ &[P(z, a) \lor P(z, f(z))] \land \\ &[P(z, a) \lor P(f(z), z)]\,\big]\end{aligned}$$

Example of conversion (steps 6 and 7)

Sentence after step 5:

$$\begin{aligned}\forall z\big[\,&[\neg P(z, a) \lor \forall x_1(\neg P(z, x_1) \lor \neg P(x_1, z))] \land \\ &[P(z, a) \lor P(z, f(z))] \land \\ &[P(z, a) \lor P(f(z), z)]\,\big]\end{aligned}$$

6. Separate into individual quantified clauses sharing no variables:

$$\begin{aligned}&\forall z_1 \forall x_1\big[\neg P(z_1, a) \lor \neg P(z_1, x_1) \lor \neg P(x_1, z_1)\big] \land \\ &\forall z_2\big[P(z_2, a) \lor P(z_2, f(z_2))\big] \land \\ &\forall z_3\big[P(z_3, a) \lor P(f(z_3), z_3)\big]\end{aligned}$$

7. Obtain clausal form:

$$\begin{aligned}\big\{\,&\{\neg P(z_1, a), \neg P(z_1, x_1), \neg P(x_1, z_1)\}, \\ &\{P(z_2, a), P(z_2, f(z_2))\}, \\ &\{P(z_3, a), P(f(z_3), z_3)\}\,\big\}\end{aligned}$$

Tseiting transformation

Distributing $\vee$ over $\wedge$ may take **exponential time/space** if done naively

For example, distributing $(\ell_1 \wedge \cdots \wedge \ell_n) \vee (\ell'_1 \wedge \cdots \wedge \ell'_m)$ generates conjunction with $n \times m$ disjunctions of size 2

Explosion can be avoided by introducing new relations. Denote

$$\begin{aligned} \varphi_1(\bar{x}) &\equiv \ell_1 \wedge \cdots \wedge \ell_n \\ \varphi_2(\bar{x}) &\equiv \ell'_1 \wedge \cdots \wedge \ell'_m \end{aligned}$$

Original formula is equivalent to $R_1(\bar{x}) \vee R_2(\bar{x})$ where R_1 and R_2 are **new relation symbols** denoting φ_1 and φ_2 respectively

For R_1 theory extended with (similarly for R_2)

$$\begin{aligned} R_1(\bar{x}) &\rightarrow \ell_1 \wedge \cdots \wedge \ell_n \\ \ell_1 \wedge \cdots \wedge \ell_n &\rightarrow R_1(\bar{x}) \end{aligned}$$

First-order resolution: Example

Consider $\varphi = \exists y \forall z[P(z,y) \leftrightarrow \neg\exists x(P(z,x) \wedge P(x,z))]$ and its clausal form:

$$\{\underbrace{\{\neg P(z_1,a), \neg P(z_1,x_1), \neg P(x_1,z_1)\}}_{C_1}, \underbrace{\{P(z_2,a), P(z_2,f(z_2))\}}_{C_2}, \underbrace{\{P(z_3,a), P(f(z_3),z_3)\}}_{C_3}\}$$

Make following **substitutions** for the (implicitly universally quantifed) variables:

$$\begin{aligned} \theta_1 &= [x_1/a,\ z_1/a] && \text{in clause } C_1 \\ \theta_2 &= [z_2/a] && \text{in clause } C_2 \\ \theta_3 &= [z_3/a] && \text{in clause } C_3 \\ \theta_4 &= [z_1/f(a),\ x_1/a] && \text{in clause } C_1 \end{aligned}$$

We obtain set G of **grounded clauses** (i.e. without variables)

$$\{\{\neg P(a,a)\}, \{P(a,a), P(a,f(a))\}, \{P(a,a), P(f(a),a)\}, \{\neg P(f(a),a), \neg P(a,f(a))\}\}$$

Refutation: Example

Set G of grounded clauses is **propositional theory** where every literal becomes a propositional symbol

Grounded clauses:

$$\{\underbrace{\{\neg P(a,a)\}}_{G_1}, \underbrace{\{P(a,a), P(a,f(a))\}}_{G_2}, \underbrace{\{P(a,a), P(f(a),a)\}}_{G_3}, \underbrace{\{\neg P(f(a),a), \neg P(a,f(a))\}}_{G_4}\}$$

G contains a **refutation:**

$$\begin{aligned} G_1 : G_2 &\vdash G_5 = \{P(a,f(a)\} \\ G_1 : G_3 &\vdash G_6 = \{P(f(a),a)\} \\ G_4 : G_5 &\vdash G_7 = \{\neg P(f(a),a)\} \\ G_6 : G_7 &\vdash G_8 = \{\} \end{aligned}$$

Therefore, φ is inconsistent!

First-order refutation

Let Δ be a theory (sentence) in **clausal form**

An **instantiation** Γ of Δ is theory in clausal form such that each clause C in Γ is equal to $C'[\theta]$ where $C' \in \Delta$ and θ is some substitution

If Γ has no variables, Γ is **grounded instantiation**

Fundamental result:

Δ is inconsistent iff there is **finite** and **gounded instantiation** Γ that contains a refutation

If Γ is finite and grounded instantiation containing refutation, the refutation can be found with **propositional resolution**

Partial decidability

If Δ is unsatisfiable, there is finite and grounded instantiation that proves it

Yet, there is **infinite number** of such instantiations

An algorithm that **systematically enumerates** finite and grounded instantiations, and uses resolution to find refutation is the **best** we can do in **worst case**

If we run algorithm on Δ and it doesn't finish after 10 hours, we can't conclude anything:

– It is possible that Δ is consistent and there is no refutation

– It is possible that Δ is inconsistent but algorithm has not yet generated an instantiation containing refutation

If Δ is consistent, the algorithm **never terminates** because there is no Γ containing refutation

Partial decidability

Behavior will be the same with any correct algorithm

Behavior doesn't depend on algorithm. It is a **property** of the SAT problem for FOL

We say that SAT problem for FOL is **partially decidable**

On other hand, SAT problem (for propositional logic) is decidable but **intractable**

First-order resolution

Essentially the same as resolution for propositional logic

Consider two first-order clauses containing a complemented literal ℓ:

$$\begin{aligned} C_1 &= \forall\bar{x}(\ell \lor \ell_1 \lor \cdots \lor \ell_n) \\ C_2 &= \forall\bar{y}(\neg\ell \lor \ell'_1 \lor \cdots \lor \ell'_m) \end{aligned}$$

First-order resolution allows us to derive a new clause:

$$C_1 : C_2 \;\vdash\; \forall\bar{x}\forall\bar{y}(\ell_1 \lor \cdots \lor \ell_n \lor \ell'_1 \lor \cdots \lor \ell'_m)$$

First-order resolution is a **correct** rule of inference: if Δ contains C_1 and C_2, then $\Delta \vDash C$ where $C_1 : C_2 \vdash C$

Unifier

Problem is that for clausal form Δ:

– two different clauses in Δ share no variables (due to renaming)

– even if variables are renamed, there may be no two clauses in Δ on which to make resolution

Unifier for clauses C_1 and C_2 is substitution θ such that $C'_1 = C_1[\theta]$ and $C'_2 = C_2[\theta]$ contain a complemented literal and thus available for resolution

Example: $\theta = [z_1/a, z_2/a]$ is a unifier for clauses

$$\begin{aligned} C_1 &= \{\neg P(z_1, a), \neg P(z_1, x_1), \neg P(x_1, z_1)\} \\ C_2 &= \{P(z_2, a), P(z_2, f(z_2))\} \end{aligned}$$

Unification: Finding unifiers

Given literals ℓ and ℓ', we want to find unifier for pair (ℓ, ℓ'):

– substitution θ such that $\ell[\theta] = \ell'[\theta]$

– θ should be as general as possible; i.e. make least commitments

Unification algorithm finds such unifier for pair (ℓ, ℓ'). It uses subroutine for finding unifier for pairs (t, t') made of terms

Disagreements

When finding a unifier for either pair (ℓ, ℓ') or pair (t, t'), we can encounter different types of **disagreements**

Fatal disagreements that cannot be solved:

F1. $\ell = R(\bar{t})$ and $\ell' = P(\bar{t}')$ for relations $R \neq P$

F2. $t = a$ and $t' = b$ for constants $a \neq b$

F3. $t = a$ and $t' = f(\bar{t})$

F4. $t = f(\bar{t})$ and $t' = g(\bar{t}')$ for functions $f \neq g$

F5. $t = x$, $t' \neq x$, and variable x appears in t'; e.g. $t = x$ and $t' = g(x)$

Repairable disagreements:

R1. $t = x$, $t' \neq x$, x doesn't appear in t', and there is no substitution for x: make substitution x/t'

Unification algorithm: Literals

```
Unify($\ell, \ell'$) :
1. If $\ell = R(\bar{t})$ and $\ell' = P(\bar{t}')$ with $R \neq P$, return FAIL        (disagreement F1)
2. Let $\ell = R(t_1, t_2, \ldots, t_n)$ and $\ell' = R(t'_1, t'_2, \ldots, t'_n)$
3. Let $\theta = []$ be the empty substitution
4. For $i = 1$ to $n$:
5.     $\theta :=$ Unify($t_i[\theta], t'_i[\theta], \theta$)
6.     If $\theta ==$ FAIL, return FAIL
7. Return $\theta$
```

Unification algorithm: Terms

```
Unify($t, t', \theta$) :
 1. If $t = a$ and $t' = a$, return $\theta$
 2. If $t = a$ and $t' = x$, or $t = x$ and $t' = a$, return $\theta[x/a]$      (extend $\theta$ with $x/a$)
 3. If $t = a$ or $t' = a$, return FAIL                            (disagreement F2 or F3)

 4. If $t = x$ and $t' = x$, return $\theta$
 5. If $t = x$ and $x$ appears in $t'$, return FAIL                     (disagreement F5)
 6. If $t' = x$ and $x$ appears in $t$, return FAIL                     (disagreement F5)
 7. If $t = x$, return $\theta[x/t']$                 (disagreement R1, extend $\theta$ with $x/t'$)
 8. If $t' = x$, return $\theta[x/t]$                  (disagreement R1, extend $\theta$ with $x/t$)

 9. If $t = f(\cdots)$ and $t' = g(\cdots)$ with $f \neq g$, return FAIL      (disagreement F4)
10. Let $t = f(t_1, t_2, \ldots, t_n)$ and $t' = f(t'_1, t'_2, \ldots, t'_n)$
11. For $i = 1$ to $n$:
12.     $\theta :=$ Unify($t_i[\theta], t'_i[\theta], \theta$)
13.     If $\theta ==$ FAIL, return FAIL
14. Return $\theta$
```

Example of unification

Let's try to find unifier for $t = f(x, f(x, y))$ and $t' = f(g(y), f(g(a), z))$

```
Unify($f(x, f(x, y)), f(g(y), f(g(a), z)), []$)
    Unify($x, g(y), []$)
        ↳ $\theta = [x/g(y)]$
    Unify($f(g(y), y), f(g(a), z), [x/g(y)]$)
        Unify($g(y), g(a), [x/g(y)]$)
            Unify($y, a, [x/g(y)]$)
                ↳ $\theta = [x/g(y), y/a]$
            ↳ $\theta = [x/g(y), y/a]$
        Unify($a, z, [x/g(y), y/a]$)
            ↳ $\theta = [x/g(y), y/a, z/a]$
        ↳ $\theta = [x/g(y), y/a, z/a]$
    ↳ $\theta = [x/g(y), y/a, z/a]$
```

Let's verify: $t[\theta] = f(g(a), f(g(a), a))$ and $t'[\theta] = f(g(a), f(g(a), a))$

Resolution algorithm

Not necessary to generated a grounded instantiation Γ and then find a refutation in Γ

Resolution algorithm performs concurrently:

– Instantiate Δ using unifiers θ

– Look for refutation among already instantiated clauses

Resolution algorithm

```
FindRefutation($\Delta$):
1. While $\Delta$ does not contain the empty clause $\emptyset$
2.     Choose two clauses $C$ and $C'$ in $\Delta$
3.     Choose positive literal $\ell$ in $C$
4.     Choose negative literal $\ell'$ in $C'$
5.     Let $\theta$ := Unify($\ell, \neg\ell'$)
6.     If $\theta$ != FAIL
7.         $R$ := Resolvent($C[\theta], C'[\theta], \ell[\theta]$)
8.         $\Delta := \Delta \cup \{$ RenameVariables($R$) $\}$
```

FindRefutation is guaranteed to terminate if Δ is unsatisfiable

Complete example

Consider the following situation, described in natural language:

1. Jack owns a dog
2. Every dog owner is an animal lover
3. No animal lover kills an animal
4. Either Jack or Curiosity killed the cat, named Tuna

Did Curiosity kill the cat?

Use first-order resolution to answer the question

Complete example

Express the situation and question in first-order logic

First define first-order language $\mathcal{L}$

– **Constant symbols:** Jack, Curiosity, Tuna

– **Function symbols:** none

– **Relational symbols:** Cat^1, Dog^1, Animal^1, AnimalLover^1, Owns^2, Kills^2

Complete example

1. Jack owns a dog:
   – $\exists x[\text{Dog}(x) \land \text{Owns}(\text{Jack}, x)]$

2. Every dog owner is an animal lover
   – $\forall x[\exists y[\text{Dog}(y) \land \text{Owns}(x, y)] \rightarrow \text{AnimalLover}(x)]$

3. No animal lover kills an animal
   – $\forall x[\text{AnimalLover}(x) \rightarrow \forall y[\text{Animal}(y) \rightarrow \neg\text{Kills}(x, y)]]$

4. Either Jack or Curiosity killed the cat, named Tuna
   – $\text{Kills}(\text{Jack}, \text{Tuna}) \lor \text{Kills}(\text{Curiosity}, \text{Tuna})$
   – $\text{Cat}(\text{Tuna})$
   – $\forall x[\text{Cat}(x) \rightarrow \text{Animal}(x)]$ (implicit!)

5. Did Curiosity kill the cat?
   – $\neg\text{Kills}(\text{Curiosity}, \text{Tuna})$

Entailment test

We want to check whether $\Delta \vDash \varphi$

This is equivalent to checking satisfiability of $\Delta \land \neg\varphi$

In example, Δ is all the knowledge we know about the situation and φ is the question asked

Complete example

In example, $\neg\varphi = \neg\text{Kills}(\text{Curiosity}, \text{Tuna})$ while Δ is

$$\begin{aligned}\Delta \;=\; & \exists x[\text{Dog}(x) \land \text{Owns}(\text{Jack}, x)] \;\land \\ & \forall x[\exists y[\text{Dog}(y) \land \text{Owns}(x, y)] \rightarrow \text{AnimalLover}(x)] \;\land \\ & \forall x[\text{AnimalLover}(x) \rightarrow \forall y[\text{Animal}(y) \rightarrow \neg\text{Kills}(x, y)]] \;\land \\ & [\text{Kills}(\text{Jack}, \text{Tuna}) \lor \text{Kills}(\text{Curiosity}, \text{Tuna})] \;\land \\ & \text{Cat}(\text{Tuna}) \;\land \\ & \forall x[\text{Cat}(x) \rightarrow \text{Animal}(x)]\end{aligned}$$

Convert Δ into clausal form. Step 1, remove $\rightarrow$ and $\leftrightarrow$:

$$\begin{aligned}\Delta \;=\; & \exists x[\text{Dog}(x) \land \text{Owns}(\text{Jack}, x)] \;\land \\ & \forall x[\neg\exists y[\text{Dog}(y) \land \text{Owns}(x, y)] \lor \text{AnimalLover}(x)] \;\land \\ & \forall x[\neg\text{AnimalLover}(x) \lor \forall y[\neg\text{Animal}(y) \lor \neg\text{Kills}(x, y)]] \;\land \\ & [\text{Kills}(\text{Jack}, \text{Tuna}) \lor \text{Kills}(\text{Curiosity}, \text{Tuna})] \;\land \\ & \text{Cat}(\text{Tuna}) \;\land \\ & \forall x[\neg\text{Cat}(x) \lor \text{Animal}(x)]\end{aligned}$$

Complete example

Step 2, move $\neg$ down to literal level:

$$\begin{aligned}\Delta \;=\; & \exists x[\mathsf{Dog}(x) \land \mathsf{Owns}(\mathsf{Jack}, x)] \land \\ & \forall x[\forall y[\neg\mathsf{Dog}(y) \lor \neg\mathsf{Owns}(x,y)] \lor \mathsf{AnimalLover}(x)] \land \\ & \forall x[\neg\mathsf{AnimalLover}(x) \lor \forall y[\neg\mathsf{Animal}(y) \lor \neg\mathsf{Kills}(x,y)]] \land \\ & [\mathsf{Kills}(\mathsf{Jack},\mathsf{Tuna}) \lor \mathsf{Kills}(\mathsf{Curiosity},\mathsf{Tuna})] \land \\ & \mathsf{Cat}(\mathsf{Tuna}) \land \\ & \forall x[\neg\mathsf{Cat}(x) \lor \mathsf{Animal}(x)]\end{aligned}$$

Step 3, rename quantified variables uniquely:

$$\begin{aligned}\Delta \;=\; & \exists x_1[\mathsf{Dog}(x_1) \land \mathsf{Owns}(\mathsf{Jack}, x_1)] \land \\ & \forall x_2[\forall y_2[\neg\mathsf{Dog}(y_2) \lor \neg\mathsf{Owns}(x_2,y_2)] \lor \mathsf{AnimalLover}(x_2)] \land \\ & \forall x_3[\neg\mathsf{AnimalLover}(x_3) \lor \forall y_3[\neg\mathsf{Animal}(y_3) \lor \neg\mathsf{Kills}(x_3,y_3)]] \land \\ & [\mathsf{Kills}(\mathsf{Jack},\mathsf{Tuna}) \lor \mathsf{Kills}(\mathsf{Curiosity},\mathsf{Tuna})] \land \\ & \mathsf{Cat}(\mathsf{Tuna}) \land \\ & \forall x_4[\neg\mathsf{Cat}(x_4) \lor \mathsf{Animal}(x_4)]\end{aligned}$$

Complete example

Step 4, remove existential quantification (skolemization):

$$\begin{aligned}\Delta \;=\; & \mathsf{Dog}(a) \land \\ & \mathsf{Owns}(\mathsf{Jack}, a)] \land \\ & \forall x_2[\forall y_2[\neg\mathsf{Dog}(y_2) \lor \neg\mathsf{Owns}(x_2,y_2)] \lor \mathsf{AnimalLover}(x_2)] \land \\ & \forall x_3[\neg\mathsf{AnimalLover}(x_3) \lor \forall y_3[\neg\mathsf{Animal}(y_3) \lor \neg\mathsf{Kills}(x_3,y_3)]] \land \\ & [\mathsf{Kills}(\mathsf{Jack},\mathsf{Tuna}) \lor \mathsf{Kills}(\mathsf{Curiosity},\mathsf{Tuna})] \land \\ & \mathsf{Cat}(\mathsf{Tuna}) \land \\ & \forall x_4[\neg\mathsf{Cat}(x_4) \lor \mathsf{Animal}(x_4)]\end{aligned}$$

Step 5, distribute $\lor$ over $\land$ until getting CNF: nothing to do!

Complete example

Step 6, obtain clauses in prenex form:

$$\begin{aligned}\Delta \;=\; & \mathsf{Dog}(a) \land \\ & \mathsf{Owns}(\mathsf{Jack}, a)] \land \\ & \forall x_2\forall y_2[\neg\mathsf{Dog}(y_2) \lor \neg\mathsf{Owns}(x_2,y_2) \lor \mathsf{AnimalLover}(x_2)] \land \\ & \forall x_3\forall y_3[\neg\mathsf{AnimalLover}(x_3) \lor \neg\mathsf{Animal}(y_3) \lor \neg\mathsf{Kills}(x_3,y_3)] \land \\ & [\mathsf{Kills}(\mathsf{Jack},\mathsf{Tuna}) \lor \mathsf{Kills}(\mathsf{Curiosity},\mathsf{Tuna})] \land \\ & \mathsf{Cat}(\mathsf{Tuna}) \land \\ & \forall x_4[\neg\mathsf{Cat}(x_4) \lor \mathsf{Animal}(x_4)]\end{aligned}$$

Step 7, obtain clausal form:

$$\begin{aligned}\Delta \;=\; \{ & \{\mathsf{Dog}(a)\}, \\ & \{\mathsf{Owns}(\mathsf{Jack}, a)\}, \\ & \{\neg\mathsf{Dog}(y_2), \neg\mathsf{Owns}(x_2,y_2), \mathsf{AnimalLover}(x_2)\}, \\ & \{\neg\mathsf{AnimalLover}(x_3), \neg\mathsf{Animal}(y_3), \neg\mathsf{Kills}(x_3,y_3)\}, \\ & \{\mathsf{Kills}(\mathsf{Jack},\mathsf{Tuna}), \mathsf{Kills}(\mathsf{Curiosity},\mathsf{Tuna})\}, \\ & \{\mathsf{Cat}(\mathsf{Tuna})\}, \\ & \{\neg\mathsf{Cat}(x_4), \mathsf{Animal}(x_4)\}\;\}\end{aligned}$$

Complete example

Clausal form for query is $\{\neg\mathsf{Kills}(\mathsf{Curiosity},\mathsf{Tuna})\}$

Clausal form for $\Delta' = \Delta \land \neg\varphi$ is

$$\begin{aligned}\Delta' \;=\; \{\; & C_1 = \{\mathsf{Dog}(a)\}, \\ & C_2 = \{\mathsf{Owns}(\mathsf{Jack}, a)\}, \\ & C_3 = \{\neg\mathsf{Dog}(y_2), \neg\mathsf{Owns}(x_2,y_2), \mathsf{AnimalLover}(x_2)\}, \\ & C_4 = \{\neg\mathsf{AnimalLover}(x_3), \neg\mathsf{Animal}(y_3), \neg\mathsf{Kills}(x_3,y_3)\}, \\ & C_5 = \{\mathsf{Kills}(\mathsf{Jack},\mathsf{Tuna}), \mathsf{Kills}(\mathsf{Curiosity},\mathsf{Tuna})\}, \\ & C_6 = \{\mathsf{Cat}(\mathsf{Tuna})\}, \\ & C_7 = \{\neg\mathsf{Cat}(x_4), \mathsf{Animal}(x_4)\}, \\ & C_8 = \{\neg\mathsf{Kills}(\mathsf{Curiosity},\mathsf{Tuna})\}\;\}\end{aligned}$$

Complete example

We now try to find refutation in Δ' to prove $\Delta \vDash \varphi$

- $C_8 : C_5 \ \vdash \ C_9 = \{\text{Kills(Jack, Tuna)}\}$

Kills(Jack, Tuna) and Kills(x_3, y_3) unify with $\theta_1 = [x_3/\text{Jack}, y_3/\text{Tuna}]$,

- $C_9[\theta_1] : C_4[\theta_1] \ \vdash \ C_{10} = \{\neg\text{AnimalLover(Jack)}, \neg\text{Animal(Tuna)}\}$

Animal(Tuna) and Animal(x_4) unify with $\theta_2 = [x_4/\text{Tuna}]$,

- $C_{10}[\theta_2] : C_7[\theta_2] \ \vdash \ C_{11} = \{\neg\text{AnimalLover(Jack)}, \neg\text{Cat(Tuna)}\}$
- $C_{11} : C_6 \ \vdash \ C_{12} = \{\neg\text{AnimalLover(Jack)}\}$

AnimalLover(Jack) and AnimalLover(x_2) unify with $\theta_3 = [x_2/\text{Jack}]$,

- $C_{12}[\theta_3] : C_3[\theta_3] \ \vdash \ C_{13} = \{\neg\text{Dog}(y_2), \neg\text{Owns}(\text{Jack}, y_2)\}$

Owns(Jack, y_2) and Owns(Jack, a) unify with $\theta_4 = [y_2/a]$,

- $C_{13}[\theta_4] : C_2[\theta_4] \ \vdash \ C_{14} = \{\neg\text{Dog}(a)\}$
- $C_{14} : C_1 \ \vdash \ C_{15} = \emptyset$

Refutation graph

$$\begin{tikzcd}
C_8 \arrow[dr] & & C_5 \arrow[dl] \\
& C_9 \arrow[dr] & & C_4 \arrow[dl] \\
& & C_{10} \arrow[dr] & & C_7 \arrow[dl] \\
& & & C_{11} \arrow[dr] & & C_6 \arrow[dl] \\
& & & & C_{12} \arrow[dr] & & C_3 \arrow[dl] \\
& & & & & C_{13} \arrow[dr] & & C_2 \arrow[dl] \\
& & & & & & C_{14} \arrow[dr] & & C_1 \arrow[dl] \\
& & & & & & & \emptyset
\end{tikzcd}$$

Resolution strategies

FindRefutation is not fully specified because we don't say how clauses/literals are chosen

Most important choice is for clauses; once we have two clauses, we can find all possible resolutions of both clauses

A **resolution strategy** says how to choose clasues

Linear resolution: starting from the clause(s) for $\neg\varphi$, always choose the last clause generated together with another clause

Linear resolution is **complete:** if there is a refutation, linear resolution finds it

Summary

- First-order logic is more expressive than propositional logic, but inference is partially decidable
- Satisfiability problem for FOL is solved by first-order resolution
- First-order resolution is refutation complete: if theory is unsatisfiable, there is a derivation of null clause
- Unification is necessary for interleaving instantiation/search for null clause
- Linear resolution is a complete resolution strategy