Automated Reasoning in First-Order Logic

Peter Baumgartner

http://users.cecs.anu.edu.au/~baumgart/

5/8/2013

Automated Reasoning in First-Order Logic

… First-Order Logic

Can express (mathematical) structures, e.g. groups

$$\forall x\ 1 \cdot x = x \qquad \forall x\ x \cdot 1 = x \qquad \text{(N)}$$
$$\forall x\ x^{-1} \cdot x = 1 \qquad \forall x\ x \cdot x^{-1} = 1 \qquad \text{(I)}$$
$$\forall x, y, z\ (x \cdot y) \cdot z = x \cdot (y \cdot z) \qquad \text{(A)}$$

… Reasoning …

- Object level: It follows $\forall x\ (x \cdot x) = 1 \rightarrow \forall x, y\ x \cdot y = y \cdot x$
- Meta-level: the word problem for groups is decidable

Automated …

Computer program to provide the above conclusions *automatically*

Application: Compiler Validation

Problem: prove equivalence of source and target program

```
1:  y := 1                   1: y := 1
2:  if z = x*x*x             2: R1 := x*x
3:    then y := x*x + y      3: R2 := R1*x
4:  endif                    4: jmpNE(z,R2,6)
                             5: y := R1+1
```

To prove: (indexes refer to values at line numbers; index 0 = initial values)

From $y_1 = 1 \ \wedge\ z_0 = x_0 * x_0 * x_0 \ \wedge\ y_3 = x_0 * x_0 + y_1$

and $y'_1 = 1 \ \wedge\ R1_2 = x'_0 * x'_0 \ \wedge\ R2_3 = R1_2 * x'_0 \ \wedge\ z'_0 = R2_3$
$\wedge\ y'_5 = R1_2 + 1 \ \wedge\ x_0 = x'_0 \ \wedge\ y_0 = y'_0 \ \wedge\ z_0 = z'_0$

it follows $y_3 = y'_5$

Issues

- Previous slides gave motivation: *logical analysis of systems*

  System can be "anything that makes sense" and can be described using logic (group theory, computer programs, ...)

- First-order logic is expressive but not too expressive, i.e., admits *complete* reasoning procedures

- So, reasoning with it can be automated on computer. BUT
  - How to do it in the first place: suitable calculi?
  - How to do it efficiently: search space control?
  - How to do it optimally: reasoning support for specific theories like equality and arithmetic?

- The lecture will touch on some of these issues and explain basic approaches to their solution

More on "Reasoning"

Example

A_1: Socrates is a human

A_2: All humans are mortal

Translation into first-order logic:

A_1: human(socrates)

A_2: $\forall X$ (human$(X) \rightarrow$ mortal(X))

Which of the following statements hold true?

1. $\{A_1,\ A_2\} \models$ mortal(socrates)
2. $\{A_1,\ A_2\} \models$ mortal(apollo)
3. $\{A_1,\ A_2\} \not\models$ mortal(socrates)
4. $\{A_1,\ A_2\} \not\models$ mortal(apollo)
5. $\{A_1,\ A_2\} \models \neg$mortal(socrates)
6. $\{A_1,\ A_2\} \models \neg$mortal(apollo)

Non-trivial issues: what do these statements mean *exactly*? How to design a theorem prover that can correctly answer all/some such questions?

Contents

- Propositional logic: syntax, semantics, some important results, automated reasoning ("Resolution") – all in view of reusability for first-order logic.
- First-order logic: syntax, semantics, automated reasoning ("Resolution")

Propositional Logic

Propositional logic (PL) is concerned with statements about truth values of propositions on account of their *form.*

Definition 1 (Syntax of Propositional Logic)**.** Given

- a denumerable set of *atomic formulas* P_i (also: "propositional variables", "atoms"), where $i = 1, 2, 3 \ldots$, and
- the *connectives* $\wedge$, $\vee$ and $\neg$, and
- the symbols (and).

The *propositional formulas (PF)* are defined inductively as follows:

1. $P_i \in PF$, where $i = 1, 2, 3 \ldots$.
2. If $F \in PF$ and $G \in PF$, then $(F \wedge G) \in PF$, $(F \vee G) \in PF$ and $\neg F \in PF$. $\square$

In the following just "formula" instead of "propositional formula".
A *subformula of a formula* F is a substring of F that is again a formula.

Abbreviations and Conventions

We use the following abbreviations, where $F_i \in PF$:

Abbreviation	*Expansion*
A, B, C, ...	P_1, P_2, P_3, ...
$(F_1 \rightarrow F_2)$	$(\neg F_1 \vee F_2)$
$(F_2 \leftarrow F_1)$	$(\neg F_1 \vee F_2)$
$(F_1 \leftrightarrow F_2)$	$((F_1 \wedge F_2) \vee (\neg F_1 \wedge \neg F_2))$
$\bigvee_{i=1}^{n} F_i$	$(\cdots((F_1 \vee F_2) \vee F_3) \vee \cdots \vee F_n)$
$\bigwedge_{i=1}^{n} F_i$	$(\cdots((F_1 \wedge F_2) \wedge F_3) \wedge \cdots \wedge F_n)$

The symbols $\rightarrow$,$\leftarrow$ and $\leftrightarrow$ are also called *connectives.*
We use the following precedences (in increasing binding power):

$$\leftrightarrow \quad \begin{matrix}\rightarrow \\ \leftarrow\end{matrix} \quad \wedge \quad \vee \quad \neg$$

A formula of the form $(F \wedge G)$ is called a *conjunction*, $(F \vee G)$ a *disjunction*, and $\neg F$ a *negation.* Parenthesis can be left away if the formula can be reconstructed modulo associativity of $\wedge$ and $\vee$.

Semantics of Propositional Logic

The set of *truth values* is $\{\mathbf{T}, \mathbf{F}\}$.

Definition 2 (Assignment)**.** An *assignment* for a set D of atomic formulas is a function $\mathcal{A}_D$ that maps each $A \in D$ to a truth value, i.e. $\mathcal{A}_D(A) \in \{\mathbf{T}, \mathbf{F}\}$ for every $A \in D$. □

Definition 3 (Suitable Assignment)**.** Let F be a formula. An assignment $\mathcal{A}$ is called *suitable for F* iff $\mathcal{A}$ is defined for all atomic subformulas in F. □

Definition 4 (Extensionality principle)**.** Let H be a formula and $\mathcal{A}$ a suitable assignment for H. The *extension of $\mathcal{A}$ to H* is the function $\mathcal{B}$ that assigns a truth value to H, recursively defined according to the form of H, as follows:

1. $\mathcal{B}(H) = \mathcal{A}(H)$ if H is an atom
2. $\mathcal{B}(F \wedge G) = \begin{cases} \mathbf{T} & \text{if } \mathcal{B}(F) = \mathbf{T} \text{ and } \mathcal{B}(G) = \mathbf{T} \\ \mathbf{F} & \text{otherwise} \end{cases}$
3. $\mathcal{B}(F \vee G) = \begin{cases} \mathbf{T} & \text{if } \mathcal{B}(F) = \mathbf{T} \text{ or } \mathcal{B}(G) = \mathbf{T} \\ \mathbf{F} & \text{otherwise} \end{cases}$
4. $\mathcal{B}(\neg F) = \begin{cases} \mathbf{T} & \text{if } \mathcal{B}(F) = \mathbf{F} \\ \mathbf{F} & \text{otherwise} \end{cases}$ □

Notation: Instead of $\mathcal{A}_D$ and $\mathcal{B}$ just $\mathcal{A}$. That is, $\mathcal{A}$ is identified with its extension to formulas.

Inductive definitions (like Definition 1) enable *inductive proofs*:

Remark 5 (Induction on the structure of formulas)**.** *To prove that a property P holds for every formula F it suffices to show the following:*

Induction start: *P holds for every atomic formula A.*

Induction step: *Assume P holds for arbitrary formulas F and G (*induction hypothesis*).*

Show that P holds for $\neg F$, $F \wedge G$ and $F \vee G$ as well.

Example application:

Lemma 6. *Let $\mathcal{A}$ and $\mathcal{A}'$ be suitable assignments for a formula H such that $\mathcal{A}(A) = \mathcal{A}'(A)$ for all atomic subformulas of H.*

Then, $\mathcal{A}(H) = \mathcal{A}'(H)$. □

Some Important Definitions

We say that an assignment $\mathcal{A}$ is *suitable* for a set M of formulas iff $\mathcal{A}$ is suitable for every $F \in M$.

The following notions are all defined to be equivalent:

- $\mathcal{A}$ is suitable for F and $\mathcal{A}(F) = \mathbf{T}$.
- $\mathcal{A} \models F$.
- $\mathcal{A}$ is a *model* of F.
- F is valid under $\mathcal{A}$.

Note that these definitions apply only to *suitable* assignments.

The notation $\mathcal{A} \not\models F$ means "not $\mathcal{A} \models F$".

For example, if $D = \{B\}$ and, say, $\mathcal{A}_D(B) = \mathbf{T}$ then $\mathcal{A}_D \not\models A \vee \neg A$ just because $\mathcal{A}_D$ is not suitable for $A \vee \neg A$.

Satisfiability and Validity

A formula F is called

- *satisfiable* if F has at least one model
- *unsatisfiable* if F has no model
- *valid* (*tautological*, *tautology*) iff every suitable assignment is a model of F. Notation: $\models F$ for "F is tautology". $\not\models F$ for "F is not tautology".

Let M be a set of formulas. M is called *satisfiable* iff there is an assignment A such that for all $F \in M$ it holds $\mathcal{A} \models F$. If this is the case we write $\mathcal{A} \models M$.

Similarly: validity, unsatisfiability.

Proposition 7 ("$\approx$ Proof by contradiction")**.** *A formula F is a tautology iff $\neg F$ is unsatisfiable.*

Definition 8 (Logical Consequence)**.** Let M be a set of formulas and G a formula.
G is a *logical consequence* of M, written as $M \models G$, iff

for every suitable assignment $\mathcal{A}$ for M and G: if $\mathcal{A} \models M$ then $\mathcal{A} \models G$.

For a formula F define $F \models G$ as $\{F\} \models G$. □

Proposition 9. *1. The following are equivalent:*

(a) G is a logical consequence of F.

(b) $(F \to G)$ is a tautology.

   (c) $(F \land \neg G)$ *is unsatisfiable.*

2. *The following are equivalent:*

   (a) G *is a logical consequence of* M.

   (b) $M \cup \{\neg G\}$ *is unsatisfiable.*

□

Equivalence and Normal Forms

Most theorem provers assume that the input formulas have been transformed into a normal form, one that facilities the design of the core inference rules.

The most important normal form is "clause normal form", or "conjunctive normal form", introduced in the following. Clause normal form is obtained by rewriting as long as possible a formula into an equivalent one based on certain logical equivalences.

Definition 10 (Logical Equivalence)**.** Two formulas F and G are *equivalent*, written as $F \equiv G$, iff

for all suitable assignments for F and G it holds $\mathcal{A}(F) = \mathcal{A}(G)$. □

Proposition 11 (Substitution Theorem)**.** *Assume* $F \equiv G$. *If* H *is a formula with at least one occurrence of* F *as a subformula then* $H \equiv H'$, *where* H' *is obtained from* H *by replacing some occurrence of* F *in* H *by* G.

Proof. (Sketch) By induction on the formula structure. For the induction start, if $H = F$ then $H' = G$, and $H \equiv H'$ follows from $F \equiv G$. The proof of the induction step is similar to the proof of Lemma 6. □

The relevance of Proposition 11 is given by the following equivalences, which justifies to replace subformulas by equivalent ones.

Proposition 12. *All of the following hold:*

$$
\begin{aligned}
(F \land F) &\equiv F && \text{(Idempotency)}\\
(F \lor F) &\equiv F \\
(F \land G) &\equiv (G \land F) && \text{(Commutativity)}\\
(F \lor G) &\equiv (G \lor F) \\
((F \land G) \land H) &\equiv (F \land (G \land H)) && \text{(Associativity)}\\
((F \lor G) \lor H) &\equiv (F \lor (G \lor H)) \\
(F \land (F \lor G)) &\equiv F && \text{(Absorption)}\\
(F \lor (F \land G)) &\equiv F \\
(F \land (G \lor H)) &\equiv ((F \land G) \lor (F \land H)) && \text{(Distributivity)}\\
(F \lor (G \land H)) &\equiv ((F \lor G) \land (F \lor H))
\end{aligned}
$$

(Proposition 12 continued)

$$\begin{aligned}
\neg\neg F &\equiv F && \text{(Double Negation)}\\
\neg(F \wedge G) &\equiv (\neg F \vee \neg G) && \text{(deMorgan)}\\
\neg(F \vee G) &\equiv (\neg F \wedge \neg G) &&\\
(F \vee G) &\equiv F \text{ , if } F \text{ is a tautology} && \text{(Tautologies)}\\
(F \wedge G) &\equiv G \text{ , if } F \text{ is a tautology} &&\\
(F \vee G) &\equiv G \text{ , if } F \text{ is unsatisfiable} && \text{(Unsatisfiability)}\\
(F \wedge G) &\equiv F \text{ , if } F \text{ is unsatisfiable} &&
\end{aligned}$$

□

Proof. (Sketch) Use truth tables. □

Example

Proof of the equivalence

$$((A \vee (B \vee C)) \wedge (C \vee \neg A)) \equiv ((B \wedge \neg A) \vee C)$$

$$\begin{aligned}
& ((A \vee (B \vee C)) \wedge (C \vee \neg A)) &&\\
\equiv\ & (((A \vee B) \vee C) \wedge (C \vee \neg A)) && \text{(Associativity and ST)}\\
\equiv\ & ((C \vee (A \vee B)) \wedge (C \vee \neg A)) && \text{(Commutativity and ST)}\\
\equiv\ & (((C \vee (A \vee B)) \wedge C) \vee ((C \vee (A \vee B)) \wedge \neg A)) && \text{(Distributivity and ST)}\\
\equiv\ & (C \vee ((C \vee (A \vee B)) \wedge \neg A)) && \text{(Absorption and ST)}\\
\equiv\ & (C \vee ((C \wedge \neg A) \vee ((A \vee B) \wedge \neg A))) && \text{(Distributivity and ST)}\\
\equiv\ & (C \vee ((C \wedge \neg A) \vee ((A \wedge \neg A) \vee (B \wedge \neg A)))) && \text{(Distributivity and ST)}\\
\equiv\ & (C \vee ((C \wedge \neg A) \vee (B \wedge \neg A))) && \text{(Unsatisfiability and ST)}\\
\equiv\ & ((C \vee (C \wedge \neg A)) \vee (B \wedge \neg A)) && \text{(Associativity and ST)}\\
\equiv\ & (C \vee (B \wedge \neg A)) && \text{(Absorption and ST)}\\
\equiv\ & ((B \wedge \neg A) \vee C) && \text{(Commutativity and ST)}
\end{aligned}$$

Application

Proposition 13. *For every formula F there is an equivalent formula that contains the connectives $\vee$ and $\neg$ only.* □

Proof. Starting with F, repeat as long as possible removing conjunctive subformulas by using the equivalence $(F \wedge G) \equiv \neg(\neg F \vee \neg G)$. □

Conjunctive and Disjunctive Normal Form

Definition 14 (Literal, Normal Forms)**.** A *literal* is an atom or the negation of an atom. In the first case the literal is *positive*, in the second case it is *negative*.

Literals are usually denoted by the letters K and L in the following.

A formula F is in *conjunctive normal form (CNF)* iff it is a conjunction of disjunction of literals:

$$F = (\bigwedge_{i=1}^{n} (\bigvee_{j=1}^{m_i} L_{i,j}))$$

A formula F is in *disjunctive normal form (DNF)* iff it is a disjunction of conjunction of literals:

$$F = (\bigvee_{i=1}^{n} (\bigwedge_{j=1}^{m_i} L_{i,j}))$$

□

Theorem 15. *For every formula there is an equivalent one in CNF and an equivalent one in DNF.*

Proof. Sketch, for the CNF part: starting with the given formula, using Proposition 12, apply the following equivalences from left to right, in the given order, as long as possible:

$$\neg\neg G \equiv G \tag{1}$$
$$\neg(G \wedge H) \equiv (\neg G \vee \neg H) \tag{2}$$
$$\neg(G \vee H) \equiv (\neg G \wedge \neg H) \tag{3}$$
$$(F \vee (G \wedge H)) \equiv ((F \vee G) \wedge (F \vee H)) \tag{4}$$
$$((G \wedge H) \vee F) \equiv ((F \vee G) \wedge (F \vee H)) \tag{5}$$

It remains to show:

1. The above procedure always terminates.
2. The resulting formula is in CNF.

□

Clause Logic

Let $F = (\bigwedge_{i=1}^{n} (\bigvee_{j=1}^{m_i} L_{i,j}))$ be a formula in CNF. The formula F can also be written as

$$\begin{array}{lll} F = & & (\neg A_{1,1} \vee \cdots \vee \neg A_{1,k_1} \vee A_{1,k_1+1} \vee \cdots \vee A_{1,m_1}) \\ & \wedge & (\qquad \cdots \qquad) \\ & \vdots & \qquad \vdots \\ & \wedge & (\neg A_{n,1} \vee \cdots \vee \neg A_{n,k_n} \vee A_{n,k_n+1} \vee \cdots \vee A_{n,m_n}) \ , \end{array}$$

or, equivalently, as

$$\begin{array}{lll} F = & & (A_{1,1} \wedge \cdots \wedge A_{1,k_1} \rightarrow A_{1,k_1+1} \vee \cdots \vee A_{1,m_1}) \\ & \wedge & (\qquad\qquad \cdots \qquad\qquad) \\ & \vdots & \qquad\qquad \vdots \\ & \wedge & (A_{n,1} \wedge \cdots \wedge A_{n,k_n} \rightarrow A_{n,k_n+1} \vee \cdots \vee A_{n,m_n}) \end{array}$$

Relevancy: Many problems naturally come as if-then rules, i.e., following the second pattern.

The Resolution method (see below) uses a set representation of formulas in CNF, as follows:

$$\begin{array}{lll} M_F = & \{ & \{\neg A_{1,1}, \ldots, \neg A_{1,k_1}, A_{1,k_1+1}, \ldots, A_{1,m_1}\}, \\ & & \vdots \\ & & \underbrace{\{\neg A_{n,1}, \ldots, \neg A_{n,k_n}, A_{n,k_n+1}, \ldots, A_{n,m_n}\}}_{\text{Clause}} \end{array}$$

Definition 16 (Clause, Clause Set)**.** A *clause* is a set of literals. The empty clause is written as $\Box$. A *clause set* is a set of clauses. $\Box$

Any clause set corresponds in an obvious way to a formula in CNF and to a (different) formula in DNF, and vice versa. We sometimes also write clauses as disjunctions, e.g., $A \vee \neg B \vee C$ instead of $\{A, \neg B, C\}$.

More Notation

A clause

$$\{\neg A_1 \vee \cdots \vee \neg A_k \vee A_{k+1} \vee \cdots \vee A_m\}$$

is also written as

$$A_1, \ldots, A_k \rightarrow A_{k+1}, \ldots, A_m$$

The atoms $A_1, \ldots, A_k$ are also called the *body (of the clause)* and the atoms $A_{k+1}, \ldots, A_m$ are also called the *head (of the clause).*

Special cases:

$$\begin{array}{rll} m = 0: & A_1, \ldots, A_k \rightarrow & (\textit{Negative clause}) \\ k = 0: & \rightarrow A_1, \ldots, A_m & (\textit{Positive clause}) \end{array}$$

Semantic Trees

(Robinson 1968, Kowalski and Hayes 1969.)

Semantic trees are a convenient device to represent assignments (for possibly infinitely many) atoms.

Applications

- To prove the completeness of the propositional Resolution calculus.
- Characterizes a specific, refined Resolution calculus.
- To prove the compactness theorem of propositional logic. Application: completeness proof of first-order logic Resolution.

Definition 17 (Tree)**.** A *tree*

- is an acyclic, connected, directed graph, where
- every node has at most one incoming edge.

A *rooted tree* has a dedicated node, called *root* that has no incoming edge.

A tree is *finite* iff it has finitely many vertices (and edges) only.

In a *finitely branching tree* every node has only finitely many edges.

A *binary* tree every node has at most two outgoing edges. It is *complete* iff every node has either no or two outgoing edges.

A *path P in a rooted tree* is a possibly infinite sequence of nodes $P = (N_0, N_1, \ldots)$, where N_0 is the root, and N_i is a direct successor of N_{i-1}, for all $i = 1, \ldots, n$.

A *path to a node N* is a finite path of the form $(N_0, N_1, \ldots, N_n)$ such that $N = N_n$; the value n is the *length* of the path. The node N_{n-1} is called the *immediate predecessor of N* Every node $N_0, N_1, \ldots, N_{n-1}$ is called a *predecessor of N*.

A *(node-)labelled tree* is a tree together with a labelling function λ that maps each node to an element in a given set. □

Let L be a literal. The *complement of L* is the literal

$$\overline{L} := \begin{cases} \neg A & \text{if } L \text{ is the atom } A \\ A & \text{if } L \text{ is the negated atom } \neg A. \end{cases}$$

Definition 18 (Semantic Tree)**.** A *semantic tree $\mathcal{B}$ (for a set of atoms D)* is a labelled, complete, rooted, binary tree such that

1. the root is labelled by the Symbol $\top$,
2. for every inner node N, one successor of N is labeled with the literal A, and the other successor is labeled with the literal $\neg A$, for some $A \in D$, and
3. for every node N, there is no literal L such that $L \in \mathcal{I}(N)$ and $\overline{L} \in \mathcal{I}(N)$, where

$$\mathcal{I}(N) = \{\lambda(N_i) \mid N_0, N_1, \ldots, (N_n = N) \text{ is a path to } N \text{ and } 1 \leq i \leq n\} \ . \quad \square$$

Remark 19 (Semantics of $\top$)**.** *Convention: the symbol* $\top$ *is identified with* $\mathbf{T}$.

Definition 20 (Atom Set)**.** For a clause set M let the *atom set (of M)* be the set of atoms occurring in clauses in M. A *semantic tree for M* is a semantic tree for the atom set M. $\square$

Definition 21 (Complete Semantic Tree)**.** A semantic tree for D is *complete* iff for every leaf N it holds that

$A \in \mathcal{I}(N)$ or $\neg A \in \mathcal{I}(N)$, for all $A \in D$. $\square$

Interpretation Induced by a Semantic Tree

1. Every node N in a semantic tree for D induces an assignment $\mathcal{A}_N$ for some $D' \subseteq D$ as follows:
$$\mathcal{A}_N(A) = \begin{cases} \mathbf{T} & \text{if } A \in \mathcal{I}(N) \\ \mathbf{F} & \text{if } \neg A \in \mathcal{I}(N) \end{cases}$$
2. If the atom set of M is finite, in every complete semantic tree and each of its leafs N the assignment $\mathcal{A}_N$ is suitable for M.
3. If the atom set of M is infinite, every complete semantic tree for M is infinite (even more: does not have any leafs).
4. A complete semantic tree can be seen as an enumeration of all possible assignments for M (it holds $\mathcal{A}_N \neq \mathcal{A}_{N'}$ whenever $N \neq N'$).

If a clause set M is unsatisfiable then every assignment $\mathcal{A}$ falsifies some clause in M (by definition), i.e., $\mathcal{A} \not\models \mathcal{C}$ for some $\mathcal{C} \in M$. This motivates the following definition:

Definition 22 (Failure Node)**.** A node N in a semantic tree for M is a *failure node*, if

1. there is a clause $\mathcal{C} \in M$ such that $\mathcal{A}_N$ is suitable for $\mathcal{C}$ and $\mathcal{A}_N \not\models \mathcal{C}$, and
2. for every predecessor N' of N it holds:

   there is no clause $\mathcal{C} \in M$ such that $\mathcal{A}_{N'}$ is suitable for $\mathcal{C}$ and $\mathcal{A}_{N'} \not\models \mathcal{C}$. $\square$

Definition 23 (Open, Closed)**.** A path P in a semantic tree for M is *closed* iff P contains failure node, otherwise it is P *open*.

A semantic tree $\mathcal{B}$ for M is *closed* iff every path is closed, otherwise $\mathcal{B}$ is *open*. $\square$

Every closed semantic tree can be turned into a finite closed one by removing all subtrees below all failure nodes.

The construction of (closed or open) finite semantic trees is the core of the propositional DPLL procedure.

Compactness

Lemma 24. *A clause set M is unsatisfiable iff there is a closed semantic tree for M.*

Proof. See whiteboard. $\square$

Theorem 25 (Compactness)**.** *A clause set M is unsatisfiable iff some finite subset of M is unsatisfiable.*

Proof. The if-direction is trivial. For the only-if direction, Lemma 24 gives us a finite unsatisfiable subset of M as identified by the finitely many failure nodes in the semantic tree. $\square$

Theorem 25 gives a hint how to reduce proof search in first-order logic to propositional logic, see below for details.

The Resolution Calculus

The Resolution calculus (short: Resolution) (Robinson, 1965) is a calculus for first-order logic. Refined versions are the most widely used calculi that are implemented in contemporary automated theorem provers for first-order logic.

We discuss the version for propositional logic first.

Calculus

A *calculus* consists of, roughly,

- a decidable set of formulas, called *axioms*,
- a collection of transformation rules between formulas, called *inference rules*,
- a notion of *derivation* that prescribes how inference rules, axioms and another given formula (*hypothesis*) are to be combined, and
- a notion of *proof* that singles out certain derivations.

  "Proofs" then give rise to soundness and completeness theorems.

In the case of Resolution:

- Axioms: None
- Inference rules: the *Resolution inference rule* combines two clauses into a new clause. (The first-order version requires an additional "factoring" rule.)
- Derivation: a sequence of clauses, starting with the hypothesis clauses, called a *Resolution derivation.*
- Proof: a Resolution derivation that contains the empty clause $\square$, also called a *(Resolution) refutation*

Refutations thus have the following form:

$$\underbrace{\mathcal{C}_1, \ldots, \mathcal{C}_k}_{\substack{\text{Hypothesis}\\\text{clauses}}}, \underbrace{\mathcal{C}_{k+1}, \ldots, (\mathcal{C}_n = \Box)}_{\substack{\text{Derived}\\\text{clauses}}}$$

Common problem statement:

Given: (i) $\mathcal{T} = \{Ax_1, \ldots, Ax_n\}$ a finite set of formulas, and
(ii) A formula F.

Question: does $\mathcal{T} \models F$ hold ? (is F a consequence of $\mathcal{T}$?)

How to show that with Resolution:

	$\mathcal{T} \models F$	(1)
iff	$\mathcal{T} \cup \{\neg F\}$ is unsatisfiable (Proposition 9-2)	(2)
iff	the clausal form of $Ax_1 \wedge \cdots \wedge Ax_n \wedge \neg F$ is unsatisfiable (Theorem 15)	(3)
iff	there is a Resolution refutation of $(Ax_1, \ldots, Ax_n, \neg F)$	(4)

The transition from (3) to (4) is given by the *completeness theorem*, and the transition from (4) to (3) is given by the *soundness theorem* of Resolution.

Definition 26 (Resolution Inference Rule)**.** Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be clauses. A clause $\mathcal{C}$ is called a *(binary) resolvent* of $\mathcal{C}_1$ and $\mathcal{C}_2$ iff

1. there is a literal L with $L \in \mathcal{C}_1$ and $\overline{L} \in \mathcal{C}_2$, and
2. $\mathcal{C} = (\mathcal{C}_1 \setminus \{L\}) \cup (\mathcal{C}_2 \setminus \{\overline{L}\})$ □

Resolution inference rule schematically:

$$\frac{\mathcal{C}_1 \qquad \mathcal{C}_2}{(\mathcal{C}_1 \setminus \{L\}) \cup (\mathcal{C}_2 \setminus \{\overline{L}\})}$$

An *inference* is an instance of an inference rule. The upper clauses are called the *premises* of the inference rule or inference, and the lower clause the *conclusion.*

Soundness of Resolution

The following lemma is essential in proving the soundness of the Resolution calculus.

Lemma 27. *Let M be a clause set and $\mathcal{C}$ a resolvent of of $\mathcal{C}_1 \in M$ and $\mathcal{C}_2 \in M$. Then $M \equiv M \cup \{\mathcal{C}\}$.*

Theorem 28 (Soundness)**.** *The resolution calculus is refutationally sound. That is, if there is a Resolution refutation starting with the hypothesis clauses $\mathcal{C}_1, \ldots, \mathcal{C}_k$, then $\{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$ is unsatisfiable.*

Proof. By contradiction. Assume a refutation $\mathcal{C}_1, \ldots, \mathcal{C}_k, \mathcal{C}_{k+1}, \ldots, (\mathcal{C}_n = \Box)$ and that $M := \{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$ is satisfiable. Let I be a model of M. By applying Lemma 27 $n - (k+1)$ times conclude $I \models \Box$. However, no interpretation can satisfy the empty clause, a plain contradiction. $\Box$

Finding Resolution Refutations

It is unpractical to enumerate derivations until a refutation comes up. It is better to close the given set of hypothesis M under (all possible) Resolution inferences and extract a refutation afterwards if $\Box$ is among the derived clauses More precisely:

Definition 29 (Resolution Closure)**.** Let M be a clause set. Define

1. $\text{Res}(M) = M \cup \{\mathcal{C} \mid \mathcal{C} \text{ is a resolvent of two clauses in } M\}$

2. $$\begin{aligned} \text{Res}^0(M) &= M \\ \text{Res}^{n+1}(M) &= \text{Res}(\text{Res}^n(M)) \quad \text{,for all } n \geq 0. \end{aligned}$$

3. $\text{Res}^\star(M) = \bigcup_{n \geq 0} \text{Res}^n(M)$ $\Box$

Theorem 30 (Completeness, Closure Version)**.** *The resolution calculus is refutationally complete. That is, if $M = \{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$ is unsatisfiable then $\Box \in \text{Res}^\star(M)$.*

Proof. Use semantic trees, see whiteboard. $\Box$

Propositional Logic – Final Remarks

The above Resolution calculus is very basic and can be improved considerably. There are two important classes of improvements:

Inference rule restrictions

That is, forbid certain Resolution inferences (that are not needed to get a complete calculus).

Example: the completeness proof based on semantic trees justifies certain "ordering restrictions".

Clause Deletion

E.g., *subsumption deletion*, the most important deletion rule: remove a clause $\mathcal{C}$ from M if there is a clause $\mathcal{C}' \in M$ such that $\mathcal{C}' \subset \mathcal{C}$.

That subsumption deletion preserves completeness can also be justified by the semantic tree completeness proof.

First-Order Logic

First-order logic (FOL), or predicate logic, is an extension of propositional logic by language elements for formulating that certain relations hold between all or some objects of a domain.

Plan of Attack

1. Syntax and Semantics
2. Normal forms
3. Herbrand theory
4. First-Order Resolution

First-Order Logic (FOL)

"The function f is continuous", expressed in FOL:

$$\forall \epsilon (0 < \epsilon \implies \forall a \exists \delta (0 < \delta \wedge \forall x (|x - a| < \delta \implies |f(x) - f(a)| < \epsilon)))$$

Underlying Language

Variables	ϵ, a, δ, x
Function symbols	0, $\lvert \_ \rvert$, $\_ - \_$, $f(\_)$
Predicate symbols	$\_ < \_$, $\_ = \_$
Boolean connectives	$\wedge$, $\vee$, $\implies$, $\neg$
Quantifiers	$\forall$, $\exists$

First-Order Logic

"The function f is continuous", expressed in FOL:

$$\forall \epsilon (0 < \epsilon \implies \forall a \exists \delta (0 < \delta \wedge \forall x (|x - a| < \delta \implies |f(x) - f(a)| < \epsilon)))$$

Meaning of language elements (informally)

A structure is a pair $\mathcal{A} = (U_{\mathcal{A}}, I_{\mathcal{A}})$ where $U_{\mathcal{A}}$ is a denumerable set ("universe") and $I_{\mathcal{A}}$ provides the meaning of function and predicate symbols:

Variables	$\mapsto$ elements from $U_{\mathcal{A}}$ ("valuation")
Function symbols	$\mapsto$ (total) function $U_{\mathcal{A}}^n \mapsto U_{\mathcal{A}}$
Predicate symbols	$\mapsto$ relation $\subseteq U_{\mathcal{A}}^n$
Boolean connectives	$\mapsto$ the usual boolean functions
Quantifiers	$\mapsto$ "for all ... holds", "there is a ..., such that"

Syntax

We need denumerable sets of variables, predicate symbols and function symbols. Let $i = 1, 2, 3, \ldots$ and $k = 0, 1, 2, \ldots$ (i numbers these symbols, k is the arity):

- A *variable* has the form x_i.
- A *predicate symbol* has the form P_i^k
- A *function symbol* has the form f_i^k A 0-ary function symbol is also called a *constant*.

Notational conventions:

u, v, w, x, y, z	for variables
a, b, c	for constants
f, g, h	for function symbols
P, Q, R	for predicate symbols

Terms

The set of *terms* is defined inductively as follows:

1. Every variable is a term.
2. If f is a k-ary function symbol and $t_1, \ldots, t_k$ are terms then $f(t_1, \ldots, t_k)$ is a term (a *function term*).

Notation: c instead of $c()$, where c is a constant.

We take the liberty to write function terms in infix notation for better readability. For example, $f(x) - f(a)$ instead of $-(f(x), f(a))$.

Formulas

The set FOF of first-order logic formulas is defined inductively as follows:

1. If P is a k-ary predicate symbol and $t_1, \ldots, t_k$ are terms then $P(t_1, \ldots, t_k) \in \text{FOF}$ (called *atomic formula* or *atom*).
2. If $F \in \text{FOF}$ and $G \in \text{FOF}$ then $(F \wedge G) \in \text{FOF}$ and $(F \vee G) \in \text{FOF}$.
3. If $F \in \text{FOF}$ then $\neg F \in \text{FOF}$.
4. If $F \in \text{FOF}$ and x is a variable then
   - $\forall x\ F \in \text{FOF}$ (*universally quantified formula*), and
   - $\exists x\ F \in \text{FOF}$ (*existentially quantified formula*).

   The symbols $\forall$ and $\exists$ are called *universal quantifier* and *existential quantifier*, respectively.

Conventions

The following notions carry over from propositional logic in the expected way:

- The connectives $\to$, $\leftarrow$, $\leftrightarrow$.
- The precedences of the connectives. The quantifiers $\forall$ and $\exists$ have highest precedence.
- The definition of subformula.

Free and Bound Variables

Definition 31 (Free and Bound Variables, Sentence)**.** An occurrence of a variable x in a formula F is called *bound* if that occurrence is within a subformula of F of the form $\exists x\ G$ or $\forall x\ G$. Otherwise that occurrence is called *free.*

The formula G is called the *scope* of $\exists x$ or $\forall x$

A formula without occurrences of free variables is called *closed*, or a *sentence.* $\square$

Example

$$\forall y \quad \overbrace{(\forall \overbrace{x \quad P(x)}^{\textit{scope}} \quad \to \quad Q(x,y))}^{\textit{scope}}$$

The occurrence of y is bound, as is the first occurrence of x. The second occurrence of x is a free occurrence.

Semantics

Definition 32 (Structure)**.** A *structure* is a pair $\mathcal{A} = (U_\mathcal{A}, I_\mathcal{A})$, where

1. $U_\mathcal{A}$ is a non-empty set, called *domain* or *universe*), and
2. $I_\mathcal{A}$ is a function (*interpretation function*) that maps
   - every k-ary predicate symbol P in its domain to a k-ary relation over $U_\mathcal{A}$, that is, $I_\mathcal{A}(P) \subseteq U_\mathcal{A}^k$,
   - every k-ary function symbol in its domain f to a k-ary function over $U_\mathcal{A}$ to $U_\mathcal{A}$ that is, $I_\mathcal{A}(f) : U_\mathcal{A}^k \mapsto U_\mathcal{A}$, and
   - every variable x in its domain to an element from $U_\mathcal{A}$, that is, $I_\mathcal{A}(x) \in U_\mathcal{A}$.

Notation: We write $P^\mathcal{A}$ instead of $I_\mathcal{A}(P)$, $f^\mathcal{A}$ instead of $I_\mathcal{A}(f)$, and $x^\mathcal{A}$ instead of $I_\mathcal{A}(x)$.

Notice that $I_\mathcal{A}$ is allowed to be a partial function. This gives rise to the following definition, in analogy to Definition 3 for PL:

Definition 33 (Suitable Structure)**.** Let $\mathcal{A}$ be a structure and F a formula. We say that $\mathcal{A}$ is *suitable for* F iff $I_{\mathcal{A}}$ is defined on every predicate symbol, function symbol and every variable that occurs free in F, and analogously for sets of formulas. $\square$

Remarks

- We have fixed a priori *one* set of predicate symbols and *one* set of function symbols. Hence we deal with "the" first-order logic. It is also customary to parametrize the logic wrt a *signature*, i.e., sets of predicate symbols and function symbols.
- It is also customary to separate $I_{\mathcal{A}}$ into two components, corresponding to the interpretations functions for predicate symbols and function symbols on the one hand, and the interpretation function of free variables on the other hand, called a *valuation.*

None of the above makes an essential difference.

Example

Let $F = \forall x\ P(x, f(x)) \wedge Q(g(a, z))$.

A suitable structure $\mathcal{A}$ for F is:

$$\begin{array}{rcl} U_{\mathcal{A}} & = & \{0, 1, 2, \ldots\} \\ P^{\mathcal{A}} & = & \{(m, n) \mid m, n \in U_{\mathcal{A}} \text{ and } m < n\} \\ Q^{\mathcal{A}} & = & \{n \in U_{\mathcal{A}} \mid n \text{ is a prime number}\} \\ f^{\mathcal{A}} & = & \text{the successor function on } U_{\mathcal{A}}\text{, i.e., } f^{\mathcal{A}}(n) = n + 1 \\ g^{\mathcal{A}} & = & \text{the addition function on } U_{\mathcal{A}}\text{, i.e. } g^{\mathcal{A}}(m, n) = m + n \\ a^{\mathcal{A}} & = & 2 \\ z^{\mathcal{A}} & = & 3 \end{array}$$

Example of a different universe ("Herbrand universe"):

$$U_{\mathcal{A}} \quad = \quad \{a, f(a), g(a, a), f(g(a, a)), g(f(a), a), \ldots\}$$

Evaluation of Terms and Formulas

Definition 34 (Evaluation of Terms)**.** Let t be a term and $\mathcal{A}$ a suitable structure for t. The *value of t in $\mathcal{A}$, $\mathcal{A}(t)$,* is defined recursively as follows:

1. If t is a variable x then $\mathcal{A}(x) = x^{\mathcal{A}}$.
2. If t has the form $f(t_1, \ldots, t_k)$, where f is a k-ary function symbol and $t_1, \ldots, t_k$ are terms, then

$$\mathcal{A}(f(t_1, \ldots, t_k)) = f^{\mathcal{A}}(\mathcal{A}(t_1), \ldots, \mathcal{A}(t_k)) \quad \square$$

In order to define the evaluation of quantified formulas we need an "update" operation on variable valuations:

Definition 35 (Update)**.** For any structure $\mathcal{A}$, $\mathcal{A}_{[x/d]}$ is the structure that is the same as except for the value of x in $\mathcal{A}$, which is d. More formally,

$$\mathcal{A}_{[x/d]}(y) = \begin{cases} d & \text{if } y = x \\ \mathcal{A}(x) & \text{otherwise} \end{cases} \qquad \square$$

Definition 36 (Evaluation of Formulas)**.** Let H be a formula and $\mathcal{A}$ a suitable structure for H. The *value of H in $\mathcal{A}$, $\mathcal{A}(F)$,* is defined recursively as follows:

1. If H is of the form $P(t_1, \ldots, t_k)$, where P is a k-ary predicate symbol and $t_1, \ldots, t_k$ are terms,

$$\mathcal{A}(P(t_1, \ldots, t_k)) = \begin{cases} \mathbf{T} & \text{if } (\mathcal{A}(t_1), \ldots, \mathcal{A}(t_k)) \in P^{\mathcal{A}} \\ \mathbf{F} & \text{otherwise} \end{cases}$$

2. If H is of the form $F \wedge G$, $F \vee G$ or $\neg F$ then $\mathcal{A}(H)$ is defined analogously as for PL, see Def. 4.

3. If H is of the form $\forall x\ G$ then

$$\mathcal{A}(\forall x\ G) = \begin{cases} \mathbf{T} & \text{if for all } d \in U_{\mathcal{A}} \text{ it holds } A_{[x/d]}(G) = \mathbf{T} \\ \mathbf{F} & \text{otherwise} \end{cases}$$

4. If H is of the form $\exists x\ G$ then

$$\mathcal{A}(\exists x\ G) = \begin{cases} \mathbf{T} & \text{if there is a } d \in U_{\mathcal{A}} \text{ with } A_{[x/d]}(G) = \mathbf{T} \\ \mathbf{F} & \text{otherwise} \end{cases} \qquad \square$$

Some Important Definitions

Let F be a formula and $\mathcal{A}$ a structure.
The following notions are all defined to be equivalent:

- $\mathcal{A}$ is suitable for F and $\mathcal{A}(F) = \mathbf{T}$.
- $\mathcal{A} \models F$.
- $\mathcal{A}$ is a *model* of F.
- F is valid under $\mathcal{A}$.

The notation $\mathcal{A} \not\models F$ means "not $\mathcal{A} \models F$".

Notice these definitions have the same formulations as their counterparts for PL. The only difference is that $\mathcal{A}$ is now a structure instead of an assignment.

The notions of *(un)satisfiability* and *validity* are adapted in the same way from PL to FOF.

Definition 37 (Logical Consequence)**.** Let M be a set of formulas and G a formula. G is a *logical consequence* of M, written as $M \models G$, iff

for every suitable structure $\mathcal{A}$ for M and G: if $\mathcal{A} \models M$ then $\mathcal{A} \models G$.

For a formula F define $F \models G$ as $\{F\} \models G$. □

Notice that free variables in M and in G with the same name are evaluated to the same value by $\mathcal{A}$.

Proposition 38. *1. The following are equivalent:*

(a) G is a logical consequence of F.

(b) $(F \to G)$ is a tautology.

(c) $(F \wedge \neg G)$ is unsatisfiable.

2. The following are equivalent:

(a) G is a logical consequence of M.

(b) $M \cup \{\neg G\}$ is unsatisfiable. □

Proof. As for Proposition 9. □

Equivalence and Normal Forms

- All equivalences in Proposition 12 are valid in FOF, too.
- The Substitution Theorem (Theorem 11) holds analogously.
- In addition, all of the following equivalences hold:

1. $\neg\forall x\ F \equiv \exists x\ \neg F$
   $\neg\exists x\ F \equiv \forall x\ \neg F$

2. If x does not occur free in G: [1ex]
   $(\forall x\ F \wedge G) \equiv \forall x\ (F \wedge G)$
   $(\forall x\ F \vee G) \equiv \forall x\ (F \vee G)$
   $(\exists x\ F \wedge G) \equiv \exists x\ (F \wedge G)$
   $(\exists x\ F \vee G) \equiv \exists x\ (F \vee G)$

3. $(\forall x\ F \wedge \forall x\ G) \equiv \forall x\ (F \wedge G)$
   $(\exists x\ F \vee \exists x\ G) \equiv \exists x\ (F \vee G)$

4. $\forall x\ \forall y\ F \equiv \forall y\ \forall x\ F$
   $\exists x\ \exists y\ F \equiv \exists y\ \exists x\ F$

Definition 39 (Prenex Normal Form)**.** A formula F is in *prenex normal form* iff it is of the form

$$F = Q_1 x_1 \ \cdots \ Q_n x_n \ G$$

where $n \geq 0$, $Q_1, \dots, Q_n \in \{\forall, \exists\}$, and G, the *matrix of* F, contains no quantifiers. □

To apply the resolution calculus, the given formula needs first to be converted into a specific prenex normal form, more precisely it needs to be of the form

$$F' = \forall x_1 \ \cdots \ \forall x_n \ G' \ \text{, where } G' \text{ is in CNF.}$$

Ideas/Problems for doing that:

1. *Idea:* The equivalences 1–3 above can be used to push quantifiers outwards. *Problem 1:* The equivalences 2 are applicable only under certain circumstances
2. *Problem 2:* How to remove the existential quantifiers?
3. *Idea:* Finally, given $\forall x_1 \ \cdots \ \forall x_n \ G$, the CNF G' is obtained by purely propositional means from G (cf. Theorem 15).

Example: Application of Equivalences 1–3
(Notice the "unofficial" connective $\rightarrow$)

$$\forall \epsilon (0 < \epsilon \rightarrow \underline{\forall a} \exists \delta (0 < \delta \land \forall x (|x - a| < \delta \rightarrow |f(x) - f(a)| < \epsilon)))$$

$$\Downarrow$$

$$\forall \epsilon \forall a (0 < \epsilon \rightarrow \underline{\exists \delta} (0 < \delta \land \forall x (|x - a| < \delta \rightarrow |f(x) - f(a)| < \epsilon)))$$

$$\Downarrow$$

$$\forall \epsilon \forall a \exists \delta (0 < \epsilon \rightarrow 0 < \delta \land \underline{\forall x} (|x - a| < \delta \rightarrow |f(x) - f(a)| < \epsilon))$$

$$\Downarrow$$

$$\forall \epsilon \forall a \exists \delta (0 < \epsilon \rightarrow \underline{\forall x} (0 < \delta \land |x - a| < \delta \rightarrow |f(x) - f(a)| < \epsilon))$$

$$\Downarrow$$

$$\underline{\forall \epsilon \forall a \exists \delta \forall x} (0 < \epsilon \rightarrow (0 < \delta \land (|x - a| < \delta \rightarrow |f(x) - f(a)| < \epsilon)))$$

"Problem 1" does not show up here

Problem 1: The equivalences 2 are applicable only under certain circumstances

Definition 40 (Substitution)**.** Let F be a formula, x a variable and t a term. Then, $F[x/t]$ denotes the formula that is obtained from F by replacing every free occurrence of x in F by t.

Lemma 41 (Bound Renaming)**.** *Let $F = Qx\ G$ be a formula, where $Q \in \{\exists, \forall\}$. Let y be a variable that does not occur in G. Then $F \equiv Qy\ G[x/y]$*

Proposition 42. *For every formula F there is an equivalent formula F' in prenex normal form.*

Proof. Similarly to the proof of Theorem 15, using additionally the equivalences 1-3 and Lemma 41 to enable the application of the equivalences 2. □

Problem 2: How to remove the existential quantifiers?

Definition 43 (Skolemization)**.** Let F be a formula of the form

$$F = \forall x_1\ \cdots \forall x_n\ \exists y\ G$$

The *immediate Skolemization of F* is the formula

$$F' = \forall x_1\ \cdots \forall x_n\ G[y/f(x_1, \ldots, x_n)]\ ,$$

where f is a new (wrt. F) n-ary function symbol. □

The *Skolemization of F* (or *Skolem normal form of F*) is the formula F^{Sk} that is obtained from F by repeated immediate Skolemization, as long as possible.

Obviously, F^{Sk} does not contain $\exists$-quantifiers, if F is in prenex normal form.

Example

$$\forall\epsilon\forall a\underline{\exists\delta}\forall x(0 < \epsilon \to 0 < \underline{\delta} \wedge (|x - a| < \underline{\delta} \to |f(x) - f(a)| < \epsilon))$$

$$\Downarrow \text{ (Skolemization)}$$

$$\forall\epsilon\forall a\forall x\underline{(0 < \epsilon \to (0 < d(\epsilon, a) \wedge (|x - a| < d(\epsilon, a) \to |f(x) - f(a)| < \epsilon)))}$$

$$\Downarrow \text{ (matrix in CNF)}$$

$$\forall\epsilon\forall a\forall x((0 < \epsilon \to 0 < d(\epsilon, a)) \wedge (0 < \epsilon \wedge |x - a| < d(\epsilon, a) \to |f(x) - f(a)| < \epsilon))$$

Theorem 44. *A formula F is satisfiable iff the Skolem normal form F^{Sk} is satisfiable.*

Notice that Theorem 44 is about equisatisfiability, not logical equivalence.

Summary: Transformation Steps

Input: a predicate logic formula F, possibly containing free variables.

Output: an equisatisfiable formula in Skolem normal form and matrix in CNF.

1. Let $y_1, \ldots, y_n$ be all variables that occur free in F. Let $F_1 = \exists y_1 \cdots \exists y_n\ F$ (F_1 is equisatisfiable with F).
2. Let F_2 be the prenex normal form of $F1$ (see Proposition 42).
3. Let F_3 be the Skolem normal form of F_2 (F_3 is equisatisfiable with F_2, see Theorem 44).
4. Let F_4 be obtained from F_3 by replacing the matrix of F_3 by an equivalent CNF (see Theorem 15).

Then, F_4 is the desired output formula.

Herbrand Theory

"Problem"

In a structure $\mathcal{A} = (U_\mathcal{A}, I_\mathcal{A})$ the universe $U_\mathcal{A}$ can be an *arbitrary* set, and the interpretation function $I_\mathcal{A}$ can be arbitrary, too.

How could a calculus deal with that? "Search" all possible $U_\mathcal{A}$ and $I_\mathcal{A}$?

"Solution"

Work with *Herbrand structures.*

A Herbrand structure has the following properties:

- It fixes a priori *a single* domain $U_\mathcal{A}$, the *Herbrand universe.*
- The interpretation function $I_\mathcal{A}$ for function symbols is fixed, too.
- Only the interpretation function $I_\mathcal{A}$ for predicate symbols can vary in Herbrand structures.

Herbrand Universe

Definition 45 (Herbrand Universe)**.** Let F be a sentence in Skolem normal form. The *Herbrand universe for F, $D(F)$,* is defined inductively as follows:

1. Every constant symbol occurring in F is in $D(F)$.

   If no constant occurs in F then a fresh constant c is in $D(F)$.
2. For every n-ary function symbol occurring in F and terms $t_1, \ldots, t_n$ in $D(F)$ the term $f(t_1, \ldots, t_n)$ is in $D(F)$. □

Example
Assume a constant 0, a unary function symbol +1 written postfix and a binary predicate symbol >. Let $\mathcal{A}$ the structure over the natural numbers $U_{\mathcal{A}} = \{\mathbf{0}, \mathbf{1}, \mathbf{2}, \ldots\}$ with the usual interpretation functions.

Let $F = \forall x \quad x +1 > 0$.

Then, $D(F) = \{0,\ 0 +1,\ 0 +1 +1, \ldots\}$

Herbrand Structure

Definition 46 (Herbrand Structure)**.** Let F be a sentence in Skolem normal form. A suitable structure $\mathcal{A} = (U_{\mathcal{A}}, I_{\mathcal{A}})$ is called a *Herbrand structure (for F)* if all of the following holds:

1. $U_{\mathcal{A}} = D(F)$
2. for every n-ary function symbol f occurring in F and $t_1, \ldots, t_n \in D(F)$ it holds $f^{\mathcal{A}}(t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$ □

Remarks

- Variable-free terms are mapped to "themselves": syntax and semantics coincide.

  Example: $\mathcal{A}(0 +1) = 0 +1$.

- *Notation:* the interpretation function $I_{\mathcal{A}}$ for predicate symbols can be specified indirectly, by a set of variable-free atoms, those that are true in $I_{\mathcal{A}}$, and $\mathcal{A}$ is identified with that set, e.g.,

$$\mathcal{A} = \{0 +1 > 0,\ 0 +1 +1 > 0,\ \ldots\}$$

  stands for $>^{\mathcal{A}} = \{(0 +1, 0),\ (0 +1 +1, 0), \ldots\}$

Herbrand Expansion
Because syntax and semantics of variable-free terms coincide, universal quantification of x is equivalent to expanding x with all variable-free terms. More precisely:

Definition 47 (Herbrand Expansion)**.** Let

$$F = \forall y_1 \ \cdots \forall y_n\ G$$

be a sentence in Skolem normal form. Define $E(F)$, the *Herbrand expansion of* F as

$$E(F) = \{G[y_1/t_1] \cdots [y_n/t_n] \mid t_1, \ldots, t_n \in D(F)\}$$

It follows immediately:

Lemma 48. *Let F be a sentence in Skolem normal form and $\mathcal{A}$ a Herbrand structure for F. Then $\mathcal{A} \models F$ iff $\mathcal{A} \models E(F)$.*

The relevance of Herbrand structures is given by the following theorem:

Theorem 49. *Let F be a sentence in Skolem normal form. Then F is satisfiable iff F has a Herbrand model.*

The relevance of Theorem 49 is given by the following chain, which provides a (naive) semi-decision procedure for first-order logic by reduction to propositional logic ("Gilmore procedure").

	$M \models F$, for a finite set M of sentences and a sentence F
iff	$M \cup \{\neg F\}$ is unsatisfiable (Proposition 38.2.a)
iff	$G = \bigwedge_{H \in M} H \wedge \neg F$ is unsatisfiable
iff	the Skolem normal form G^{Sk} of G is unsatisfiable (Theorem 44)
iff	G^{Sk} has no Herbrand model (Theorem 49)
iff	the Herbrand expansion $E(G^{Sk})$ is unsatisfiable (Lemma 48)
iff	some finite subset $N \subseteq E(G^{Sk})$ is unsatisfiable (Compactness, Theorem 25)
iff	N is unsatisfiable (N can be identified with a set of propositional logic clauses)

Corollary: every satisfiable formula has a model with denumerable domain. As a consequence, real number arithmetic cannot be axiomatized in FOL.

For the proof of Theorem 49 we need the following lemma (recall Definition 40 "Substitution"), which can be proven by structural induction.

Lemma 50 (Substitution Lemma)**.** *Let $\mathcal{A}$ be a suitable structure for a formula G and t a variable-free term. Then*

$$\mathcal{A}_{[x/\mathcal{A}(t)]}(G) = \mathcal{A}(G[x/t])$$

Example for the base case
It holds

$$\mathcal{A}_{[x/\mathcal{A}(0\ +1)]}(x > 0) = ((\mathcal{A}_{[x/\mathcal{A}(0\ +1)]}(x),\ \mathcal{A}_{[x/\mathcal{A}(0\ +1)]}(0)) \in >^{\mathcal{A}}) = ((\mathbf{1},\ \mathbf{0}) \in >^{\mathcal{A}})$$

and

$$\mathcal{A}(x > 0\ [x/0\ +1]) = \mathcal{A}(0\ +1 > 0) = ((\mathcal{A}(0\ +1),\ \mathcal{A}(0)) \in >^{\mathcal{A}}) = ((\mathbf{1},\ \mathbf{0}) \in >^{\mathcal{A}})$$

(We could continue $((\mathbf{1}, \mathbf{0}) \in >^{\mathcal{A}}) = (\mathbf{1}>\mathbf{0}) = \mathbf{T}$)

Proof of Theorem 49

(Sketch)

The "if"-direction is trivial.

For the "only-if" direction assume $\mathcal{B} \models F$, for some suitable structure $\mathcal{B}$. Let $\mathcal{A}$ be the Herbrand structure with each predicate symbol P defined as follows:

$$P(t_1, \ldots, t_n) \in \mathcal{A} \quad \text{iff} \quad (\mathcal{B}(t_1), \ldots, \mathcal{B}(t_n)) \in P^{\mathcal{B}}$$

(In the example, $\mathcal{A} = \{0\ {+1} > 0, 0\ {+1}\ {+1} > 0, \ldots, 0\ {+1}\ {+1} > 0\ {+1}, 0\ {+1}\ {+1}\ {+1} > 0\ {+1}, \ldots\}$)

It suffices to show $\mathcal{A} \models F$.

The claim is proven by induction over the number k of universal quantifiers in F.

Induction start ($k = 0$)
In this case F is variable-free. Use structural induction, where the base case (atom case) follows immediately from the definition of $\mathcal{A}$.

Induction step ($k > 0$)
As the induction hypothesis assume $\mathcal{A} \models F'$ whenever $\mathcal{B} \models F'$ and $\mathcal{F}'$ has strictly less universal quantifiers than $\mathcal{F}$, for every sentence $\mathcal{F}'$.

With $k > 0$ it follows F is of the form $\forall x\ G$. We have

$$\begin{array}{rll}
 & \mathcal{B} \models F & (1) \\
\text{iff} & \mathcal{B} \models \forall x\ G & (2) \\
\text{iff} & \text{for all } d \in U_{\mathcal{A}}\text{: } \mathcal{B}_{[x/d]}(G) = \mathbf{T} & (3) \\
\text{then} & \text{for all } d \in U_{\mathcal{B}} \text{ s. th. } d = \mathcal{B}(t) \text{ for some } t \in D(G)\text{: } \mathcal{B}_{[x/d]}(G) = \mathbf{T} & (4) \\
\text{iff} & \text{for all } t \in D(G)\text{: } \mathcal{B}_{[x/\mathcal{B}(t)]}(G) = \mathbf{T} & (5) \\
\text{iff} & \text{for all } t \in D(G)\text{: } \mathcal{B}(G[x/t]) = \mathbf{T} \quad \text{(by Lemma 50)} & (6) \\
\text{then} & \text{for all } t \in D(G)\text{: } \mathcal{A}(G[x/t]) = \mathbf{T} \quad \text{(by Ind. Hyp.)} & (7) \\
\text{iff} & \mathcal{A} \models \forall x\ G & (8) \\
\text{iff} & \mathcal{A} \models F \qquad \qquad \Box &
\end{array}$$

Gilmore's Algorithm

```
(1)  Input: a sentence $F$ in Skolem normal form.
(2) Let $F_1, F_2, \ldots, F_n, \ldots$ an enumeration of $E(F)$.
(3) $n$ := 0;
(4) Repeat
(5)       $n$ := $n + 1$
(6) until $(F_1 \wedge F_2 \wedge \cdots \wedge F_n)$ is unsatisfiable
(7)
(8)  Output: "unsatisfiable"
```

Gilmore's algorithm is partially correct, i.e., if it terminates then F is unsatisfiable.

Gilmore's algorithm is very inefficient, and algorithms based on the same of reduction to propositional logic were the state of the art until the invention of the Resolution calculus for first-order logic (in 1965), introduced next.

The First-Order Resolution Calculus

Motivation

The satisfiability test in Gilmore's algorithm can be done – in principle – by propositional Resolution, say. However, that would not be optimal.

Example: Let $F = \forall x\ (P(x) \land \neg P(f(x)))$ be in Skolem normal form. Then:

$$\begin{aligned}
D(F) &= \{a, f(a), f(f(a)),\ \ldots\} \\
E(F) &= \{P(a) \land \neg P(f(a)),\ P(f(a)) \land \neg P(f(f(a))), \ldots\}
\end{aligned}$$

The clause representation of the first two elements of $E(F)$ is

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

Of those clauses, only two are needed to derive the empty clause $\Box$, but there is no guarantee that such "relevant" clauses are enumerated early.

The restriction to those elements of $E(D)$ that were needed to derive the empty clause can be expressed as follows:

$$\begin{tikzcd}
\{P(x)\} \arrow[dd, "{[x/f(a)]}"] & \{\neg P(f(x))\} \arrow[dd, "{[x/a]}"] & \text{First-order clauses} \\
 & & \text{Substitutions} \\
\{P(f(a))\} & \{\neg P(f(a))\} & \text{Variable-free clauses} \\
\Box & & \text{Resolution inference}
\end{tikzcd}$$

In the following: a first-order logic Resolution calculus that works directly on the first-order logic clauses, without having to "guess" the substitutions.

Clause Form

Let $F = \forall x_1 \cdots \forall x_n\ \mathcal{C}_1 \land \cdots \land \mathcal{C}_n$ in Skolem normal form, where each $\mathcal{C}_i$ is a disjunction of literals.

The *clause form of* F is the clause set $\{\mathcal{C}'_1, \ldots, \mathcal{C}'_n\}$ where $\mathcal{C}'_i$ is the set representation of $\mathcal{C}_i$.

Because $\forall$ distributes over $\land$, the universal quantifiers can be left away, and each clause is implicitly universally quantified over all its variables. Obviously, the clause form of F is equivalent to F.

Substitutions

A *substitution* σ is a finite set of pairs x/t, called *bindings*, with x a variable and t a term, such that all bindings are pairwise different wrt. their variables.

Notation:

$$\sigma = [x_1/t_1, \ldots, x_n/t_n] \ .$$

Because $x_i \neq x_j$ for $i \neq j$, σ can be seen as a function on terms and formulas that replaces free variables by terms, as specified by the bindings.

We write $F\sigma$ for the formula that is obtained by simultaneously replacing every free occurrence of x_i by t_i. (This generalizes Definition 40.) The formula $F\sigma$ is called an *instance of F (via σ)*.

Similarly for $t\sigma$ where t is a term, and also for sets of terms, formulas, and sets of formulas (in particular clauses).

Ground Substitution

Let X be a term, an atom, a literal or a set of these. A substitution σ is called a *ground substitution for X* if $X\sigma$ is variable-free.

Every such $X\sigma$ is called a *ground instance (of X).*

With the notions above the results so far can be reformulated as follows:

Proposition 51 (Propositional Resolution Correctness)**.** *A clause set M is unsatisfiable iff there is a refutation of M^{Gr} by propositional resolution, where M^{Gr} is some finite set of ground instances of clauses from M.*

Proposition 51 will be instrumental for proving the completeness of first-order Resolution.

Unification

Question: given two terms s and t, possible containing variables. What are the terms that match both s and t, i.e., the common instances? *Unification* will find the answer.

Definition 52 (Unifier)**.** Given two terms s and t. A substitution σ is a *unifier (for s and t)* iff $s\sigma = t\sigma$.

A unifier σ is called a *most general unifier (mgu)* iff for every unifier σ' (of the same terms) there is a substitution δ such that $\sigma\delta = \sigma'$ □

Definition 52 also applies to atoms. The expression $\sigma\delta$ denotes functional composition of σ and δ (first apply σ, then δ).

Example

$s = car(red, y, z)$ $\quad t = car(u, v, ferrari)$

Then $\sigma' = [u/red,\ y/fast,\ v/fast,\ z/ferrari]$ is a unifier for s and t,

and $\sigma = [u/red,\ y/v,\ z/ferrari]$ is a mgu for s and t.

With $\delta = [v/fast]$ obtain $\sigma\delta = \sigma'$.

A Unification Algorithm

A *unification problem* U is a finite set of pairs of terms, written as

$$U = \{s_1 = t_1, \ldots, s_n = t_n\} \ .$$

Input: Two terms s and t. Let $U = \{s = t\}$ initially, and apply the following transformation rules as long as possible.

$$\{x = x\} \cup N \longrightarrow N \qquad \text{(Trivial)}$$

$$\{x = t\} \cup N \longrightarrow \{x = t\} \cup N[x/t] \qquad \text{(Binding)}$$
$$\text{if } x \text{ occurs in } N \text{ and } x \text{ does not occur in } t$$

$$\{x = t\} \cup N \longrightarrow FAIL \qquad \text{(Occur check)}$$
$$\text{if } t \text{ is not a variable and } x \text{ occurs in } t$$

$$\{f(s_1, \ldots, s_m) = f(t_1, \ldots, t_m)\} \cup N \longrightarrow$$
$$\{s_1 = t_1, \ldots, s_m = t_m\} \cup N \qquad \text{(Decomposition)}$$

$$\{f(s_1, \ldots, s_m) = g(t_1, \ldots, t_m)\} \cup N \longrightarrow FAIL \quad \text{if } f \neq g \qquad \text{(Conflict)}$$

$$\{t = x\} \cup N \longrightarrow \{x = t\} \cup N \qquad \text{(Orientation)}$$
$$\text{if } t \text{ is not a variable}$$

Output: See Proposition 53

Proposition 53 (Correctness and Completeness of the Unification Algorithm)**.** *The unification algorithm above terminates for any input terms s and t, and one of the following cases applies.*

1. *Success:*

   (a) *U is of the form $U = \{x_1 = t_1, \ldots, x_n = t_n\}$,*

   (b) *$x_i \neq x_j$, for all $1 \leq i < j \leq n$, and*

   (c) *x_i does not occur in t_j, for all $1 \leq i \leq j \leq n$*

   Furthermore,

   $$\sigma = [x_1/t_1, \ldots, x_n/t_n]$$

   is a mgu of s and t

2. *Failure: $U = FAIL$.*

   In this case there is no unifier of s and t

We need one more preliminary definition for defining the Resolution inference rules.

Definition 54 (Variant, variable disjoint)**.** Two clauses $\mathcal{C}_1$ and $\mathcal{C}_2$ are called *variants* if there are substitutions ρ_1 and ρ_2 such that

$$\mathcal{C}_1\rho_1 = \mathcal{C}_2 \quad \text{and} \quad \mathcal{C}_1 = \mathcal{C}_2\rho_2 \ .$$

The substitutions ρ_1 and ρ_2 are called *renaming substitutions.*

Two clauses $\mathcal{C}_1$ and $\mathcal{C}_2$ are called *variable disjoint* iff there is no variable that occurs both in $\mathcal{C}_1$ and in $\mathcal{C}_2$. □

Intuitively, a variant of a clause is obtained by systematically replacing all variables by other variables in a one-to-one way.

Examples

- $\{p(x), q(x)\}$ and $\{p(y), q(y)\}$ are variants
- $\{p(x, y), q(x, y)\}$ and $\{p(y, x), q(y, x)\}$ are variants
- $\{p(x), q(x)\}$ and $\{p(y), q(z)\}$ are no variants
- $\{p(x), q(x)\}$ and $\{p(y), q(a)\}$ are no variants

First-Order Logic Resolution

We need two inference rules: (FO-)Resolution and Factoring

Definition 55 ((First-Order) Resolution Inference Rule)**.** Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be clauses. W.l.o.g. assume $\mathcal{C}_1$ and $\mathcal{C}_2$ are variable disjoint (otherwise take a variant of $\mathcal{C}_1$ that is variable disjoint with $\mathcal{C}_2$).

A clause $\mathcal{C}$ is called a *(binary) resolvent* of $\mathcal{C}_1$ and $\mathcal{C}_2$ if

1. there are literals $P(s_1, \ldots, s_n) \in \mathcal{C}_1$ and $\neg P(t_1, \ldots, t_n) \in \mathcal{C}_2$, and
2. there es a mgu σ of $P(s_1, \ldots, s_n)$ and $P(t_1, \ldots, t_n)$, and
3. $\mathcal{C} = (\mathcal{C}_1\sigma \setminus \{P(s_1, \ldots, s_n)\sigma\}) \cup (\mathcal{C}_2\sigma \setminus \{\neg P(t_1, \ldots, t_n)\sigma\})$ $\square$

Schematic Notation

$$\frac{\mathcal{C}_1 \qquad\qquad \mathcal{C}_2}{(\mathcal{C}_1\sigma \setminus \{P(s_1, \ldots, s_n)\sigma\}) \cup (\mathcal{C}_2\sigma \setminus \{\neg P(t_1, \ldots, t_n)\sigma\})}\ \sigma$$

Example

$$\frac{\{\underline{P(x, y)},\ P(y, x),\ P(x, a)\} \qquad \{\underline{\neg P(f(z), f(z))},\ Q(z)\}}{\{P(y(z), a), Q(z)\}}\ \sigma$$

where $\sigma = [x/f(z),\ y/f(z)]$.

Notice the two literals $P(x, y)$ and $P(y, x)$ in the left premise collapse to the same literal after applying σ to the left premise.

Definition 56 (Factoring Inference Rule)**.** Let $\mathcal{C}_1$ be a clause. A clause $\mathcal{C}$ is called a *factor* of $\mathcal{C}_1$ if

1. there are two literals $P(s_1, \ldots, s_n), P(t_1, \ldots, t_n) \in \mathcal{C}_1$, and

2. there is a mgu σ of $P(s_1, \ldots, s_n)$ and $P(t_1, \ldots, t_n)$, and

3. $\mathcal{C} = \mathcal{C}_1\sigma$ $\square$

Remark: this rule is not needed in the propositional resolution calculus; it is subsumed by the set notation of clauses

Schematic Notation

$$\frac{\mathcal{C}_1}{\mathcal{C}_1\sigma} \quad \sigma$$

Example

$$\frac{\{\underline{P(x,y)},\, P(y,x),\, \underline{P(x,a)}\}}{\{P(x,a),\, P(a,x)\}} \quad \sigma$$

where $\sigma = [y/a]$.

Resolution Closure

As for propositional logic (Def. 29):

Definition 57 (Resolution Closure)**.** Let M be a clause set. Define

1. $\mathrm{Res}(M) = M \;\cup\; \{\mathcal{C} \mid \mathcal{C} \text{ is a binary resolvent of two clauses in } M\}$
   $\qquad\qquad\;\cup\; \{\mathcal{C} \mid \mathcal{C} \text{ is a factor of a clause in } M\}$

2. $$\begin{aligned} \mathrm{Res}^0(M) &= M \\ \mathrm{Res}^{n+1}(M) &= \mathrm{Res}(\mathrm{Res}^n(M)) \text{ ,for all } n \geq 0. \end{aligned}$$

3. $\mathrm{Res}^\star(M) = \bigcup_{n\geq 0} \mathrm{Res}^n(M)$ $\square$

Main Result

Theorem 58 (Soundness and Completeness of Resolution)**.** *A clause set M is unsatisfiable iff $\square \in \mathrm{Res}^\star(M)$.*

Proof Sketch: Soundness: similarly as for propositional Resolution calculus.

Completeness: assume M is unsatisfiable. By Proposition 51 there is a finite set M^{Gr} of ground instances of clauses in M and a propositional Resolution refutation R^{Gr} of M^{Gr}.

The refutation R^{Gr} can be simulated step by step by a first-order refutation R. Illustration:

$$\begin{tikzcd}
M: \arrow[d] & \overbrace{\mathcal{C}_1^1,\ldots,\mathcal{C}_1^{m_1},}^{\text{Factoring}} \arrow[d] & \overbrace{\mathcal{C}_2^1,\ldots,\mathcal{C}_2^{m_2},}^{\text{Factoring}} \arrow[d] & \cdots\cdots & \Box \arrow[d] & (=R) \\
M^{Gr}: & \mathcal{C}_1^{Gr}, & \mathcal{C}_2^{Gr}, & \cdots\cdots & \Box & (=R^{Gr})
\end{tikzcd}$$

More precisely:

- Every clause in M^{Gr} is an instance of a clause in M (this is easy to see), and
- If $\mathcal{C}^{Gr}$ in R^{Gr} is a binary resolvent of two clauses $\mathcal{C}_1^{Gr}$ and $\mathcal{C}_2^{Gr}$ in R^{Gr} then put into R a binary resolvent $\mathcal{C}$ obtained from factors of clauses $\mathcal{C}_1$ and of factors $\mathcal{C}_2$ in R such that $\mathcal{C}^{Gr}$ is an instance of $\mathcal{C}$. Such a clause $\mathcal{C}$ exists by Lemma 59.

Hence, for every clause in the refutation R^{Gr} there is a corresponding more general clause in R. Because R^{Gr} ends with the empty clause $\Box$ so does R. $\Box$

Lifting Lemma

Lemma 59 (Lifting). *Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be variable disjoint clauses. If*

$$\begin{tikzcd}
\mathcal{C}_1 \arrow[d, "\gamma_1"] & \mathcal{C}_2 \arrow[d, "\gamma_2"] \\
\mathcal{C}_1\gamma_1 & \mathcal{C}_2\gamma_2
\end{tikzcd}$$

$$\frac{\mathcal{C}_1\gamma_1 \qquad \mathcal{C}_2\gamma_2}{\mathcal{C}'} \qquad \textit{(propositional resolvent)}$$

then there exists clauses $\mathcal{C}_1^f$ and $\mathcal{C}_2^f$ obtained from $\mathcal{C}_1$ and $\mathcal{C}_2$, respectively, by repeated factoring, and there exists a substitution δ such that

$$\frac{\mathcal{C}_1^f \qquad \mathcal{C}_2^f}{\mathcal{C}''} \qquad \textit{(first-order resolvent)}$$

$$\begin{tikzcd}
\mathcal{C}'' \arrow[d, "\delta"] \\
\mathcal{C}' = C''\delta
\end{tikzcd}$$