

FIRST-ORDER LOGIC: PROOFS

Course “Computational Logic”



Wolfgang Schreiner

Research Institute for Symbolic Computation (RISC)

Wolfgang.Schreiner@risc.jku.at



First-Order Logic Proofs

Our core goal is to show the validity of first-order formulas.

- **Problem:** how to show $\models F$?
 - Does $M \models F$ hold for every structure M (i.e., is every structure M a model of F)?
 - But there are infinitely many structures with different domains and interpretations!

Can we reduce first-order reasoning to reasoning in some “canonical structures”?

Herbrand Structures

A **Herbrand structure** $H := (D_H, I_H)$ for a formula (language) with symbols $C, \mathcal{F}, \mathcal{P}$ consists of the Herbrand universe D_H and some Herbrand interpretation I_H .

- The **Herbrand universe** D_H is the set of all terms t formed according to the following grammar:

$$t ::= c \mid f(t_1, \dots, t_n)$$

- Every constant $c \in C$ (if $C = \{ \}$, we extend C by a constant c).
- Every n -ary function symbol $f \in \mathcal{F}$.
- D_H is the set of ground terms (no variables) that includes all constants and is closed under the application of all function symbols (thus D_H is generally infinite).
- I_H is a **Herbrand interpretation** if the following holds:

$$I(c) := c (\in D_H) \quad I(f)(t_1, \dots, t_n) := f(t_1, \dots, t_n) (\in D_H) \quad I(p)(t_1, \dots, t_n) \subseteq D_H^n$$

- I_H interprets constant c as itself, n -ary function symbol f as a term constructor, and n -ary predicate p as an arbitrary n -ary relation over D_H .

A Herbrand structure is a (generalization of a) “term algebra”.

Herbrand Structures as Models of Formulas

- **Theorem:** Let F be a quantifier-free formula. Then there exists a structure M with $M \models F$ if and only if there exists a Herbrand structure H with $H \models F$.
 - **Proof sketch:** Since the implication from right to left clearly holds, only the implication from left to right has to be shown. For this, we assume $M \models F$ for arbitrary structure $M = (D, I)$ and show $H \models F$ for the Herbrand structure $H = (D_H, I_H)$ over F with

$$I_H(p)(t_1, \dots, t_n) :\Leftrightarrow M \models p(t_1, \dots, t_n)$$

We take arbitrary valuation v_H over D_H and show $\llbracket F \rrbracket_{v_H}^H = \text{true}$. Let x_1, \dots, x_n be the free variables of F and consider the closed formula instance

$F' := F[v_H(x_1)/x_1, \dots, v_H(x_n)/x_n]$. From $M \models F$, we can show $M \models F'$. Furthermore, we can show $\llbracket F \rrbracket_{v_H}^H = \llbracket F' \rrbracket_{v'}^M$ for arbitrary valuation v' over D . From $M \models F'$, we have $\llbracket F' \rrbracket_{v'}^M = \text{true}$ and thus also $\llbracket F \rrbracket_{v_H}^H = \text{true}$.

Herbrand structures are “canonical structures” for reasoning in first-order logic; all proof calculi use these structures in some way or another.

The Sequent Calculus

An extension of the propositional sequent calculus by two additional rules.

$$\frac{\Gamma, A[t/x], (\forall x. A), \Delta \vdash \Lambda}{\Gamma, (\forall x. A), \Delta \vdash \Lambda} \quad (\forall\text{-L})$$

$$\frac{\Gamma \vdash \Delta, A[y/x], \Lambda}{\Gamma \vdash \Delta, (\forall x. A), \Lambda} \quad (\forall\text{-R})$$

$$\frac{\Gamma, A[y/x], \Delta \vdash \Lambda}{\Gamma, (\exists x. A), \Delta \vdash \Lambda} \quad (\exists\text{-L})$$

$$\frac{\Gamma \vdash \Delta, A[t/x], (\exists x. A), \Lambda}{\Gamma \vdash \Delta, (\exists x. A), \Lambda} \quad (\exists\text{-R})$$

- **Substitution $F[t/x]$:**
 - Substitution of term t for every free occurrence of variable x in formula F .
- **Eigenvariable (Skolem constant) y**
 - y must not occur in the conclusion of the rule.
- **Witness term t**
 - Term t may contain arbitrary variables, constants, and function symbols; however, every variable in t different from x must not be not bound by any quantifier in A .

Example Proof

$$\frac{\frac{\frac{\frac{\frac{\frac{}{p(\bar{x}, \bar{y}), \forall y. p(\bar{x}, y) \vdash p(\bar{x}, \bar{y}), \exists x. p(x, \bar{y})}{(AX)}}{p(\bar{x}, \bar{y}), \forall y. p(\bar{x}, y) \vdash \exists x. p(x, \bar{y})}{(\exists-R)}}{\forall y. p(\bar{x}, y) \vdash \exists x. p(x, \bar{y})}{(\forall-L)}}{\exists x. \forall y. p(x, y) \vdash \exists x. p(x, \bar{y})}{(\exists-L)}}{\exists x. \forall y. p(x, y) \vdash \forall y. \exists x. p(x, y)}{(\forall-R)}}{\vdash (\exists x. \forall y. p(x, y)) \Rightarrow (\forall y. \exists x. p(x, y))}{(\Rightarrow-R)}$$

A simple proof that applies all quantifier rules.

A Proof with More Branches

$$\begin{array}{c}
 \frac{}{p(a) \vdash p(a)} \text{ (AX)} \quad \frac{\frac{}{p(a), r(a) \vdash r(a)} \text{ (AX)}}{p(a), r(a) \vdash \exists x. r(x)} \text{ (\exists-R)}}{p(a), p(a) \Rightarrow r(a) \vdash \exists x. r(x)} \text{ (\Rightarrow-L)} \\
 \frac{}{q(b) \vdash q(b)} \text{ (AX)} \quad \frac{\frac{}{q(b), r(f(b)) \vdash r(f(b))} \text{ (AX)}}{q(b), r(f(b)) \vdash \exists x. r(x)} \text{ (\exists-R)}}{q(b), q(b) \Rightarrow r(f(b)) \vdash \exists x. r(x)} \text{ (\Rightarrow-L)} \\
 \frac{p(a), p(a) \Rightarrow r(a) \vdash \exists x. r(x)}{p(a), (\forall x. p(x) \Rightarrow r(x)) \vdash \exists x. r(x)} \text{ (\forall-L, DROP)} \quad \frac{q(b), q(b) \Rightarrow r(f(b)) \vdash \exists x. r(x)}{q(b), (\forall x. q(x) \Rightarrow r(f(x))) \vdash \exists x. r(x)} \text{ (\forall-L, DROP)} \\
 \frac{p(a) \vee q(b), (\forall x. p(x) \Rightarrow r(x)), (\forall x. q(x) \Rightarrow r(f(x))) \vdash \exists x. r(x)}{(p(a) \vee q(b)), (\forall x. p(x) \Rightarrow r(x)) \wedge (\forall x. q(x) \Rightarrow r(f(x))) \vdash \exists x. r(x)} \text{ (\wedge-L)} \\
 \frac{(p(a) \vee q(b)), (\forall x. p(x) \Rightarrow r(x)) \wedge (\forall x. q(x) \Rightarrow r(f(x))) \vdash \exists x. r(x)}{(p(a) \vee q(b)) \wedge (\forall x. p(x) \Rightarrow r(x)) \wedge (\forall x. q(x) \Rightarrow r(f(x))) \vdash \exists x. r(x)} \text{ (\wedge-L)} \\
 \frac{(p(a) \vee q(b)) \wedge (\forall x. p(x) \Rightarrow r(x)) \wedge (\forall x. q(x) \Rightarrow r(f(x))) \vdash \exists x. r(x)}{\vdash ((p(a) \vee q(b)) \wedge (\forall x. p(x) \Rightarrow r(x)) \wedge (\forall x. q(x) \Rightarrow r(f(x)))) \Rightarrow \exists x. r(x)} \text{ (\Rightarrow-R)}
 \end{array}$$

A proof by “case distinction”.

Sequent Calculus Trainer

Sequent Calculus Trainer

File Edit Help

Propositional Logic First-Order Logic

Sequent input

Antecedent: enter formulas comma separated

Succedent: $(P(a) \mid Q(b)) \& (\text{forall } x. P(x) \rightarrow R(x)) \& (\text{forall } x. Q(x) \rightarrow R(f(x))) \rightarrow (\text{exists } x. R(x))$

scan sequent

$\emptyset \Rightarrow (P(a) \vee Q(b)) \wedge (\forall x. P(x) \rightarrow R(x)) \wedge (\forall x. Q(x) \rightarrow R(f(x))) \rightarrow (\exists x. R(x))$

Rule set

- Ax
- Reflexivity - R
- $\text{ff} - L$
- $\text{tt} - R$
- $\neg - L$
- $\neg - R$
- $\vee - L$
- $\vee - R$
- $\wedge - L$
- $\wedge - R$
- $\rightarrow - L$
- $\rightarrow - R$
- $\leftrightarrow - L$
- $\leftrightarrow - R$
- $\exists - L$
- $\exists - R$
- $\forall - L$
- $\forall - R$
- Substitution - L
- Substitution - R
- Contraction - L
- Contraction - R
- Reflexivity - L
- Weakening
- Delete subtree

Zoom

Sequent Calculus Trainer

Sequent Calculus Trainer
— x

File Edit Help

Propositional Logic
First-Order Logic

Proof completed

Congratulations!

You have successfully completed the proof.

OK

$$\frac{\frac{\frac{P(a), \forall x . Q(x) \rightarrow R(f(x)) \Rightarrow \exists x . R(x), P(a)}{P(a), P(a) \rightarrow R(a), \forall x . Q(x) \rightarrow R(f(x)) \Rightarrow \exists x . R(x)}^{(A_0)} \quad \frac{\frac{P(a), R(a) \Rightarrow \exists x . R(x)}{P(a), R(a) \Rightarrow \exists x . R(x)}^{(B_0)}}{P(a), R(a), \forall x . Q(x) \rightarrow R(f(x)) \Rightarrow \exists x . R(x)}^{(B_1)}}{P(a), \forall x . P(x) \rightarrow R(x), \forall x . Q(x) \rightarrow R(f(x)) \Rightarrow \exists x . R(x)}^{(A_1)} \quad \frac{\frac{Q(b) \Rightarrow \exists x . R(x), Q(b)}{Q(b), Q(b) \rightarrow R(f(b)) \Rightarrow \exists x . R(x)}^{(A_2)} \quad \frac{\frac{Q(b), R(f(b)) \Rightarrow R(f(b))}{Q(b), R(f(b)) \Rightarrow \exists x . R(x)}^{(B_2)}}{Q(b), \forall x . P(x) \rightarrow R(x), \forall x . Q(x) \rightarrow R(f(x)) \Rightarrow \exists x . R(x)}^{(B_3)}}{P(a) \vee Q(b), \forall x . P(x) \rightarrow R(x), \forall x . Q(x) \rightarrow R(f(x)) \Rightarrow \exists x . R(x)}^{(A_2)} \quad \frac{\frac{P(a) \vee Q(b) \wedge (\forall x . P(x) \rightarrow R(x)), \forall x . Q(x) \rightarrow R(f(x)) \Rightarrow \exists x . R(x)}{(P(a) \vee Q(b)) \wedge (\forall x . P(x) \rightarrow R(x)) \wedge (\forall x . Q(x) \rightarrow R(f(x))) \Rightarrow \exists x . R(x)}^{(A_3)} \quad \frac{\frac{P(a) \vee Q(b) \wedge (\forall x . P(x) \rightarrow R(x)) \wedge (\forall x . Q(x) \rightarrow R(f(x))) \Rightarrow \exists x . R(x)}{\emptyset \Rightarrow (P(a) \vee Q(b)) \wedge (\forall x . P(x) \rightarrow R(x)) \wedge (\forall x . Q(x) \rightarrow R(f(x))) \Rightarrow \exists x . R(x)}^{(A_4)}}{\emptyset \Rightarrow (P(a) \vee Q(b)) \wedge (\forall x . P(x) \rightarrow R(x)) \wedge (\forall x . Q(x) \rightarrow R(f(x))) \Rightarrow \exists x . R(x)}^{(A_4)}$$

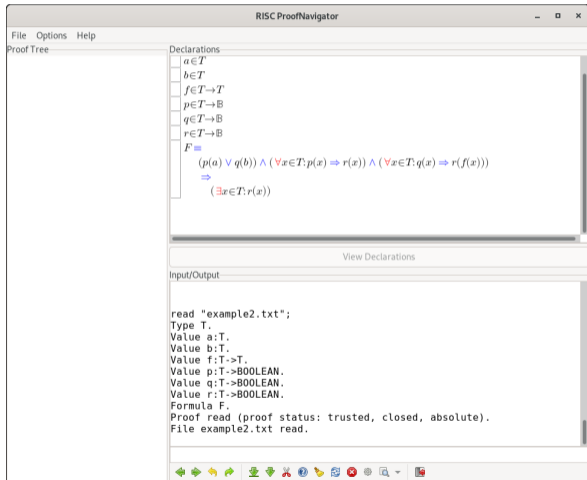
Zoom

Rule set

Ax	Reflexivity - R
ff - L	tt - R
¬ - L	¬ - R
∨ - L	∨ - R
∧ - L	∧ - R
→ - L	→ - R
↔ - L	↔ - R
∃ - L	∃ - R
∀ - L	∀ - R
Substitution - L	Substitution - R
Contraction - L	Contraction - R
Reflexivity - L	Weakening
Delete subtree	

RISC ProofNavigator

ProofNavigator &



```
% example2.txt
newcontext "example2";
```

```
T:TYPE;
a:T;
b:T;
f:T->T;
p:(T)->BOOLEAN;
q:(T)->BOOLEAN;
r:(T)->BOOLEAN;
```

```
F: FORMULA
  (p(a) OR q(b)) AND
  (FORALL(x:T): p(x) => r(x)) AND
  (FORALL(x:T): q(x) => r(f(x))) =>
  (EXISTS(x:T): r(x));
```

RISC ProofNavigator

The screenshot displays the RISC ProofNavigator application window. The interface is divided into several sections:

- File Options Help**: The top menu bar.
- Proof Tree**: A tree view on the left showing the proof structure. The current node is `[6dd]: auto`, which is a child of `[xyc]: instantiate b in 1xo`.
- Proof State**: The main area on the right showing the current state of the proof. It includes:
 - Formula [F] proof state [6dd] (autosimp,CVC3,first order,boolean): auto**
 - Constants (with types):** $a, b, f, p, q, r.$
 - guv** $\forall x \in T: p(x) \Rightarrow r(x)$
 - 1xo** $\forall x \in T: q(x) \Rightarrow r(f(x))$
 - ymf** $q(b)$
 - 3br** $r(f(b))$
 - oy5** $\exists x \in T: r(x)$
 - Parent: [xyc] Children: [xas]**
- View Declarations**: A button below the proof state.
- Input/Output**: A text area at the bottom showing the execution log:

```
Proof state [141] is closed by decision procedure.
Proof state [xas] is closed by decision procedure.
Proof replay successful.
Use 'proof F' to see proof.
proof F:
proof status: trusted, closed, absolute
[gca]: scatter
  [yxh]: split pqg
    [wcy]: auto
      [141]: proved (CVC3)
        [xyc]: instantiate b in 1xo
          [6dd]: auto
            [xas]: proved (CVC3)
```
- Bottom Bar**: A toolbar with various navigation and utility icons.

Soundness of the Sequent Calculus

Theorem: Every derivable sequent is valid.

Proof Sketch: It suffices to show that, if the conclusion of a rule is not valid, also some premise is not valid.

$$\frac{\Gamma, A[t/x], (\forall x. A), \Delta \vdash \Lambda}{\Gamma, (\forall x. A), \Delta \vdash \Lambda} \quad (\forall\text{-L})$$

$$\frac{\Gamma \vdash \Delta, A[y/x], \Lambda}{\Gamma \vdash \Delta, (\forall x. A), \Lambda} \quad (\forall\text{-R})$$

- Rule ($\forall\text{-L}$):** Since the conclusion is not valid, we have some structure M and valuation v with $\llbracket \Gamma \rrbracket_v^M = \text{true}$, $\llbracket \forall x. A \rrbracket_v^M = \text{true}$, $\llbracket \Delta \rrbracket_v^M = \text{true}$, and $\llbracket \Lambda \rrbracket_v^M = \text{false}$. Let $d := \llbracket t \rrbracket_v^M$. From above, to show that the premise is not valid, it suffices to show $\llbracket A[t/x] \rrbracket_v^M = \text{true}$. From the side condition on t , we can show $\llbracket A[t/x] \rrbracket_v^M = \llbracket A \rrbracket_{v[x \mapsto d]}^M$. From $\llbracket \forall x. A \rrbracket_v^M = \text{true}$, we know $\llbracket A \rrbracket_{v[x \mapsto d]}^M = \text{true}$ and are done.
- Rule ($\forall\text{-R}$):** Since the conclusion is not valid, we have some structure M and valuation v with $\llbracket \Gamma \rrbracket_v^M = \text{true}$, $\llbracket \Delta \rrbracket_v^M = \text{false}$, $\llbracket \forall x. A \rrbracket_v^M = \text{false}$, and $\llbracket \Lambda \rrbracket_v^M = \text{false}$. From $\llbracket \forall x. A \rrbracket_v^M = \text{false}$, there is some $d \in D$ such that $\llbracket A \rrbracket_{v[x \mapsto d]}^M = \text{false}$. Let $v' := v[y \mapsto d]$. Since y does not occur in the conclusion, we have $\llbracket \Gamma \rrbracket_{v'}^M = \text{true}$, $\llbracket \Delta \rrbracket_{v'}^M = \text{false}$, and $\llbracket \Lambda \rrbracket_{v'}^M = \text{false}$. Thus, to show that the premise is not valid, it suffices to show $\llbracket A[y/x] \rrbracket_{v'}^M = \text{false}$, i.e., $\llbracket A[y/x] \rrbracket_{v[y \mapsto d]}^M = \text{false}$. Since y does not occur in A , we can show $\llbracket A[y/x] \rrbracket_{v[y \mapsto d]}^M = \llbracket A \rrbracket_{v[x \mapsto d]}^M = \text{false}$ and are done.
- Rules ($\exists\text{-L}$) and ($\exists\text{-R}$):** analogously.

Proof Tree Construction: Data

To construct a proof tree for sequent $\Gamma \vdash \Delta$, we use the following data:

- $y = [y_0, y_1, \dots]$: an infinite sequence of variables that do *not* occur in $\Gamma \vdash \Delta$.
 - These variables can be used as eigenvariables in rules (\forall -R) and (\exists -L).
- $a = [a_0, a_1, \dots]$: an infinite sequence of term sequences:
 - The terms in these sequences are available as witnesses in rules (\forall -L) and (\exists -R).
 - If some function symbols occur in $\Gamma \vdash \Delta$, all sequences a_0, a_1, \dots are infinite.
 - $[t_0] \circ a_0 = [t_0, \dots]$: an enumeration of all terms constructed from the free variables, constants, and function symbols in $\Gamma \vdash \Delta$.
 - If $\Gamma \vdash \Delta$ does not contain any free variable or constant, we use $t_0 := y_0$.
 - $a_{i \geq 1}$: an enumeration $[y_i, \dots]$ of all terms that contain y_i and are constructed from y_1, \dots, y_i and the free variables, constants, and function symbols in $\Gamma \vdash \Delta$.

During the proof tree construction, the value of program variable n indicates that y_1, \dots, y_n have been used as eigenvariables in rules (\forall -R) or (\exists -L); the sequences a_0, a_1, \dots, a_n contain all terms in which these variables may occur.

Proof Tree Construction: Algorithm

```
procedure SEARCH( $\Gamma \vdash \Delta$ )
  INITIALIZE( $y, a, t_0$ )
   $T, ts, n \leftarrow \langle \Gamma \vdash \Delta \rangle, [t_0], 0$ 
  while  $T$  has some open leaf node do
    for every open leaf node  $N$  in  $T$  do
      EXPAND( $N, T, ts, y, n$ )
    end for
    for  $i$  from 0 do  $n$  do
      if  $\neg$ empty( $a_i$ ) then
         $ts, a_i \leftarrow ts \circ [\text{head}(a_i)], \text{tail}(a_i)$ 
      end if
    end for
  end while
  if  $T$  is complete then
    WRITE("T proves  $\Gamma \vdash \Delta$ ")
  else
    WRITE("T refutes  $\Gamma \vdash \Delta$ ")
  end if
end procedure
```

```
procedure EXPAND( $N, T, ts, y, \uparrow n$ )
  Let  $S$  be the subtree of  $T$  with root  $N$ 
  Apply the propositional rules until the formulas
  in all leaf nodes of  $S$  are atomic or quantified
  for every leaf formula in  $S$  to which ( $\forall$ -L) or ( $\exists$ -R) applies do
    repeatedly apply the rule for every  $t \in ts$ 
  end for
  for every leaf formula in  $S$  to which ( $\forall$ -R) or ( $\exists$ -L) applies do
     $n \leftarrow n + 1$ 
    apply the rule for  $x \leftarrow y_n$ 
  end for
end procedure
```

A leaf node is **open** if it does not match any axiom and there is a non-atomic node formula whose outermost symbol is

- either a connective
- or a quantifier to which (\forall -L) or (\exists -R) has not yet been applied for every term in ts .
 - This has to be recorded in EXPAND.

Correctness Properties of the Algorithm

By the soundness of the calculus, if SEARCH terminates with a complete proof tree, $\Gamma \vdash \Delta$ is valid.

- **Theorem:** if $\Gamma \vdash \Delta$ is valid, SEARCH terminates with a complete proof tree.
 - **Proof Sketch:** we assume that $\Gamma \vdash \Delta$ is valid but SEARCH does not terminate with a complete proof tree; from this, we derive a contradiction. There are two cases:

First, SEARCH may terminate with an incomplete tree T , i.e., there is a leaf node $\Gamma_k \vdash \Delta_k$ at some depth k that does not match any axiom. But, from the loop condition, no leaf node of T is open. Thus, $\Gamma_k \vdash \Delta_k$ only contains atoms and quantified formulas to which (\forall -L) and (\exists -R) have been applied for every term in ts . Consider every node $\Gamma_i \vdash \Delta_i$ along the path $\Gamma \vdash \Delta \rightarrow \dots \rightarrow \Gamma_k \vdash \Delta_k$ from the root $\Gamma \vdash \Delta$ to the leaf $\Gamma_k \vdash \Delta_k$. Let $S := \bigcup \{ \Gamma_i \cup \neg \Delta_i \mid 0 \leq i \leq k \}$ where $\neg \Delta := \{ \neg A \mid A \in \Delta \}$. Now it is possible to prove that every formula in S is satisfied by the Herbrand structure $H_S = (D_S, I_S)$ where (considering all free variables as constants) $D_S := \bigcup \{ a_i \mid 0 \leq i \leq n \} \cup ts$ (for the final values of ts, a, n) and $I_S(p)(t_1, \dots, t_n) := \Leftrightarrow p(t_1, \dots, t_n) \in \bigcup \{ \Gamma_i \mid 0 \leq i \leq k \}$. Since $\Gamma_0 = \Gamma$ and $\Delta_0 = \Delta$, this structure H_S refutes $\Gamma \vdash \Delta$, which contradicts the assumption that $\Gamma \vdash \Delta$ is valid.

Second, SEARCH may not terminate. Then its execution describes the construction of an infinite tree T (even if only a finite part of T is ever computed). Since T is infinite but finitely branching, by König's lemma it contains some infinite path $\Gamma \vdash \Delta \rightarrow \dots$. Analogously to the first case, we can construct from this path a satisfiable set S and structure H_S that refutes $\Gamma \vdash \Delta$ (to show this, it is essential that for every universal formula in some Γ_i respectively existential formula in some Δ_i , every instance of that formula appears in the branch in some $\Gamma_{j \geq i}$ respectively $\Delta_{j \geq i}$).

Fundamental Properties of First-Order Logic

- **Completeness:** every valid first-order formula is provable.
 - Kurt Gödel, 1929 (for another proof calculus of first-order logic).
 - A corollary of the previous theorem: given a valid formula F , procedure SEARCH finds a complete proof tree for the sequent $\vdash F$.
 - However, if F is invalid, SEARCH may run forever.
- **Undecidability:** there cannot exist any procedure that, when given an arbitrary first-order formula F , always halts and correctly states whether F is valid.
 - Alonzo Church/Alan Turing, 1936/1937.
 - The halting problem for computing machines is undecidable.
 - The halting problem can be reduced to the decision problem of first-order logic.

The power and the limit of reasoning in first-order logic.

The Problem of the Sequent Calculus

Procedure SEARCH looks a bit difficult to implement.

- Complex traversal of proof tree to make sure that all quantified formulas in all leafs to which the rules $(\forall\text{-L})$ and $(\exists\text{-R})$ are applicable are indeed instantiated by all possible terms.

Is there no “easier” way to achieve the same result?

Herbrand's Theorem

Actually, the **Gödel-Herbrand-Skolem** theorem (≈ 1930).

- **Theorem:** Let F be a quantifier-free first-order formula. Then F is first-order satisfiable if the set of all its ground instances $\{F_1, F_2, \dots\}$ is propositionally satisfiable.
 - F is **first-order satisfiable**: there exists some structure M such that $M \models F$.
 - F' is a **ground instance** of F if F' is identical to F except that every variable has been replaced by a term in which only constants and function symbols appear.
 - F is **propositionally satisfiable**: F is satisfied by some valuation ν , considering every atom as a propositional variable. A set $\{F_1, F_2, \dots\}$ is propositionally satisfiable if there exists some valuation ν that satisfies every formula F_i in the set.
- **Example:** formula $p(x) \wedge \neg q(x, y)$.
 - Ground instances: $\{p(c) \wedge \neg q(c, c), p(c) \wedge \neg q(c, f(c)), p(f(c)) \wedge \neg q(f(c), c), \dots\}$
 - Valuation: $[p(c) \mapsto \text{true}, q(c, c) \mapsto \text{false}, q(c, f(c)) \mapsto \text{false}, p(f(c)) \mapsto \text{true}, q(f(c), c) \mapsto \text{false}, \dots]$

The previously stated theorem about Herbrand structures as models is actually a consequence of Herbrand's theorem.

Corollaries of Herbrand's Theorem

- **Theorem:** Quantifier-free F is first-order **satisfiable** if every conjunction $F_1 \wedge \dots \wedge F_n$ of a finite subset of its instances is propositionally satisfiable.
 - **Proof sketch:** a corollary of the “compactness theorem” of propositional logic: a set of propositional formulas is satisfiable, if each finite subset is satisfiable.
- **Theorem:** Quantifier-free F is first-order **unsatisfiable** if some conjunction $F_1 \wedge \dots \wedge F_n$ of a finite subset of its instances is propositionally unsatisfiable.
 - **Proof sketch:** the contraposition of the previous theorem.
- **Theorem:** Formula $\forall x_1, \dots, x_n. F$ in **Skolem normal form** is **unsatisfiable** if some conjunction $F_1 \wedge \dots \wedge F_n$ of a finite number of instances of its matrix F is propositionally unsatisfiable.
 - **Proof sketch:** by induction on n , using the previous theorem as the induction base.

The basis of various “Herbrand procedures” for first-order proving.

The Gilmore Algorithm

Paul C. Gilmore, 1960.

```
procedure GILMORE( $G$ )  
   $F \leftarrow$  SKOLEMNORMALFORMMATRIX( $\neg G$ )  
   $F_s \leftarrow \top$   
   $i \leftarrow 1$   
  loop  
     $F_s \leftarrow F_s \wedge F(i)$      $\triangleright$  Add instance  $i$  of  $F$   
    if  $F_s$  is propositionally unsatisfiable then  
      WRITE("G is first-order valid")  
      return  
    end if  
     $i \leftarrow i + 1$   
  end loop  
end procedure
```

A systematic enumeration of all instances of the matrix.

The Gilmore Algorithm in OCaml

```
(* Get the constants for Herbrand base, adding nullary one if necessary. *)
let herbfuns fm =
  let cns,fns = partition (fun (_,ar) -> ar = 0) (functions fm) in
  if cns = [] then ["c",0],fns else cns,fns;;

(* Enumeration of ground terms and m-tuples, ordered by total fns. *)
let rec groundterms cntms funcs n =
  if n = 0 then cntms else
  itlist (fun (f,m) l -> map (fun args -> Fn(f,args))
        (groundtuples cntms funcs (n - 1) m) @ l)
        funcs []

and groundtuples cntms funcs n m =
  if m = 0 then if n = 0 then [[]] else [] else
  itlist (fun k l -> allpairs (fun h t -> h::t)
        (groundterms cntms funcs k)
        (groundtuples cntms funcs (n - k) (m - 1)) @ l)
        (0 -- n) [];
```

The Gilmore Algorithm in OCaml

```
let rec herbloop mfn tfn fl0 cntms funcs fvs n fl tried tuples =
  print_string(string_of_int(length tried)^" ground instances tried; "^
    string_of_int(length fl)^" items in list"); print_newline();
  match tuples with
  [] -> let newtups = groundtuples cntms funcs n (length fvs) in
    herbloop mfn tfn fl0 cntms funcs fvs (n + 1) fl tried newtups
  | tup::tups -> let fl' = mfn fl0 (subst(fpf fvs tup)) fl in
    if not(tfn fl') then tup::tried else
      herbloop mfn tfn fl0 cntms funcs fvs n fl' (tup::tried) tups;;

let gilmore_loop fl0 cntms funcs fvs n fl tried tuples =
  let mfn djs0 ifn djs = filter (non trivial) (distrib (image (image ifn) djs0) djs) in
  herbloop mfn (fun djs -> djs <> []) fl0 cntms funcs fvs n fl tried tuples;;

let gilmore fm =
  let sfm = skolemize(Not(generalize fm)) in
  let fvs = fv sfm and consts,funcs = herbfuns sfm in
  let cntms = image (fun (c,_) -> Fn(c,[])) consts in
  length(gilmore_loop (simpdnf sfm) cntms funcs fvs 0 [[]] [] []);;
```

Verify propositional unsatisfiability of a formula in DNF by finding a pair of complimentary literals in each disjunct.

The Gilmore Algorithm in OCaml

```
# gilmore << (P(a) \\/ Q(b)) /\ (forall x. P(x) ==> R(x)) /\ (forall x. Q(x) ==> R(f(x)))
  ==> (exists x. R(x)) >>;
0 ground instances tried; 1 items in list
1 ground instances tried; 2 items in list
2 ground instances tried; 2 items in list
2 ground instances tried; 2 items in list
3 ground instances tried; 2 items in list
- : int = 4

# skolemize << ~((P(a) \\/ Q(b)) /\ (forall x. P(x) ==> R(x)) /\ (forall x. Q(x) ==> R(f(x))))
  ==> (exists x. R(x)) >>;
<<((P(a) \\/ Q(b)) /\ (~P(x) \\/ R(x)) /\ (~Q(x) \\/ R(f(x)))) /\ ~R(x) >>
# satisfiable <<
  ((P(a) \\/ Q(b)) /\ (~P(a) \\/ R(a)) /\ (~Q(a) \\/ R(f(a))) /\ ~R(a)) /\
  ((P(a) \\/ Q(b)) /\ (~P(b) \\/ R(b)) /\ (~Q(b) \\/ R(f(b))) /\ ~R(b)) /\
  ((P(a) \\/ Q(b)) /\ (~P(f(b)) \\/ R(f(b))) /\ (~Q(f(b)) \\/ R(f(f(b)))) /\ ~R(f(b))) >> ;;
- : bool = false
```

Our example formula can be proved with 3 ground instances: $x = a, x = b, x = f(b)$.

The Gilmore Algorithm in OCaml

```
# val p45 = gilmore <<
  (forall x. P(x) /\ (forall y. G(y) /\ H(x,y) ==> J(x,y))
    ==> (forall y. G(y) /\ H(x,y) ==> R(y))) /\
  ~(exists y. L(y) /\ R(y)) /\
  (exists x. P(x) /\ (forall y. H(x,y) ==> L(y)) /\ (forall y. G(y) /\ H(x,y) ==> J(x,y)))
  ==> (exists x. P(x) /\ ~(exists y. G(y) /\ H(x,y))) >>;;
0 ground instances tried; 1 items in list
1 ground instances tried; 13 items in list
1 ground instances tried; 13 items in list
2 ground instances tried; 57 items in list
3 ground instances tried; 84 items in list
4 ground instances tried; 405 items in list
val p45 : int = 5
```

DNF representations explode, problems soon become intractable.

The Davis Putnam Algorithm in OCaml

```
let dp_mfn cjs0 ifn cjs = union (image (image ifn) cjs0) cjs;;
let dp_loop = herbloop dp_mfn dpll;;
let davisputnam fm =
  let sfm = skolemize(Not(generalize fm)) in
  let fvs = fv sfm and consts,funcs = herbfuns sfm in
  let cntms = image (fun (c,_) -> Fn(c,[])) consts in
  length(dp_loop (simpcnf sfm) cntms funcs fvs 0 [] [] []);;

# let p20 = gilmore <<(forall x y. exists z. forall w. P(x) /\ Q(y) ==> R(z) /\ U(w))
  ==> (exists x y. P(x) /\ Q(y)) ==> (exists z. R(z))>>;
...
18 ground instances tried; 15060 items in list
val p20 : int = 19
# let p20 = davisputnam <<(forall x y. exists z. forall w. P(x) /\ Q(y) ==> R(z) /\ U(w))
  ==> (exists x y. P(x) /\ Q(y)) ==> (exists z. R(z))>>;
...
18 ground instances tried; 37 items in list
val p20 : int = 19
```

Much faster propositional satisfiability testing of the smaller CNF via DPLL. 25/26

The Problem with Herbrand Procedures

However, optimizing satisfiability checking does not eliminate the core problem.

Davis, 1983: ... effectively eliminating the truth-functional satisfiability obstacle only uncovered the deeper problem of the combinatorial explosion inherent in unstructured search through the Herbrand universe ...

A more intelligent way of choosing instances is required rather than blindly trying out all possibilities.