FIRST-ORDER LOGIC: PROOFS

Course “Computational Logic”

Wolfgang Schreiner
Research Institute for Symbolic Computation (RISC)
Wolfgang.Schreiner@risc.jku.at

JOHANNES KEPLER
UNIVERSITY LINZ

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 $\mathcal{C}, \mathcal{F}, \mathcal{P}$ 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:

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

  - Every constant $c \in \mathcal{C}$ (if $\mathcal{C} = \{\ \}$, we extend $\mathcal{C}$ 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:

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

  - 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

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

    We take arbitrary valuation v_H over D_H and show $[\![F]\!]^H_{v_H} = \text{true}$. Let $x_1, \ldots, x_n$ be the free variables of F and consider the closed formula instance $F' := F[v_H(x_1)/x_1, \ldots, v_H(x_n)/x_n]$. From $M \models F$, we can show $M \models F'$. Furthermore, we can show $[\![F]\!]^H_{v_H} = [\![F']\!]^M_{v'}$ for arbitrary valuation v' over D. From $M \models F'$, we have $[\![F']\!]^M_{v'} = \text{true}$ and thus also $[\![F]\!]^H_{v_H} = \text{true}$.

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.

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

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

- 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

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

A simple proof that applies all quantifier rules.

Another Proof

- We may apply some additional “convenience” rules:

$$\frac{\Gamma, \Delta \vdash \Lambda}{\Gamma, A, \Delta \vdash \Lambda}\ \text{(DROP)} \qquad\qquad \frac{\Gamma \vdash \Delta, \Lambda}{\Gamma \vdash \Delta, A, \Lambda}\ \text{(DROP)}$$

  - Reduce size of sequent; soundness can be easily derived.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{p(\overline{x}) \vdash p(\overline{x})}\ \text{(AX)} \qquad \dfrac{\dfrac{\dfrac{}{p(\overline{x}), q(\overline{x}, \overline{y}) \vdash q(\overline{x}, \overline{y})}\ \text{(AX)}}{p(\overline{x}), q(\overline{x}, \overline{y}) \vdash \exists x, y.\ q(x, y)}\ (\exists\text{-R,DROP})}{p(\overline{x}), \exists y.\ q(\overline{x}, y) \vdash \exists x, y.\ q(x, y)}\ (\exists\text{-L})}{p(\overline{x}), p(\overline{x}) \Rightarrow \exists y.\ q(\overline{x}, y) \vdash \exists x, y.\ q(x, y)}\ (\Rightarrow\text{-L})}{p(\overline{x}), \forall x.\ p(x) \Rightarrow \exists y.\ q(x, y) \vdash \exists x, y.\ q(x, y)}\ (\forall\text{-L,DROP})}{\exists x.\ p(x), \forall x.\ p(x) \Rightarrow \exists y.\ q(x, y) \vdash \exists x, y.\ q(x, y)}\ (\exists\text{-L})}{(\exists x.\ p(x)) \wedge (\forall x.\ p(x) \Rightarrow \exists y.\ q(x, y)) \vdash \exists x, y.\ q(x, y)}\ (\wedge\text{-L})}{\vdash ((\exists x.\ p(x)) \wedge (\forall x.\ p(x) \Rightarrow \exists y.\ q(x, y))) \Rightarrow \exists x, y.\ q(x, y)}\ (\Rightarrow\text{-R})$$

We may drop formulas that have served their purpose.

A Proof with More Branches

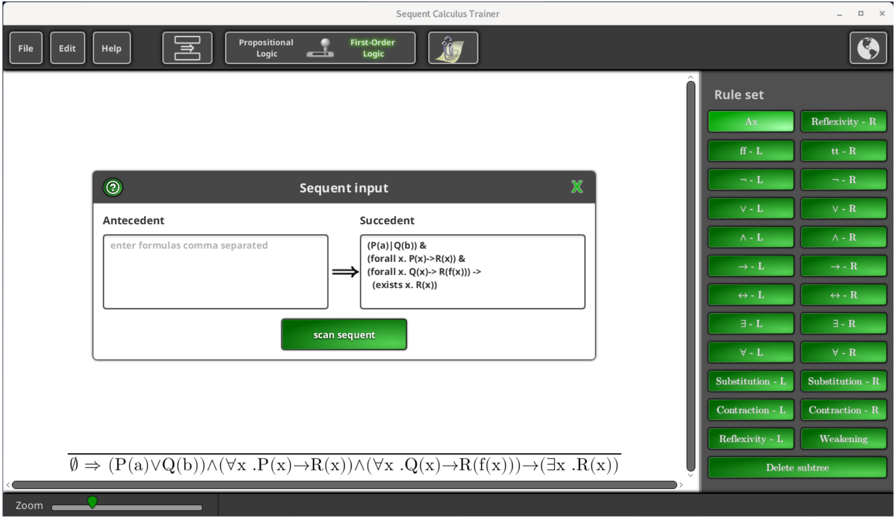
$$
\dfrac{
\dfrac{\dfrac{}{p(a) \vdash p(a)}\,(\text{AX}) \qquad \dfrac{\dfrac{}{p(a), r(a) \vdash r(a)}\,(\text{AX})}{p(a), r(a) \vdash \exists x.\, r(x)}\,(\exists\text{-R})}{p(a), p(a) \Rightarrow r(a) \vdash \exists x.\, r(x)}\,(\Rightarrow\text{-L})
}{p(a), (\forall x.\, p(x) \Rightarrow r(x)) \vdash \exists x.\, r(x)}\,(\forall\text{-L,DROP})
$$

$$
\dfrac{
\dfrac{\dfrac{}{q(b), \vdash q(b)}\,(\text{AX}) \qquad \dfrac{\dfrac{}{q(b), r(f(b)) \vdash r(f(b))}\,(\text{AX})}{q(b), r(f(b)) \vdash \exists x.\, r(x)}\,(\exists\text{-R})}{q(b), q(b) \Rightarrow r(f(b)) \vdash \exists x.\, r(x)}\,(\Rightarrow\text{-L})
}{q(b), (\forall x.\, q(x) \Rightarrow r(f(x))) \vdash \exists x.\, r(x)}\,(\forall\text{-L,DROP})
$$

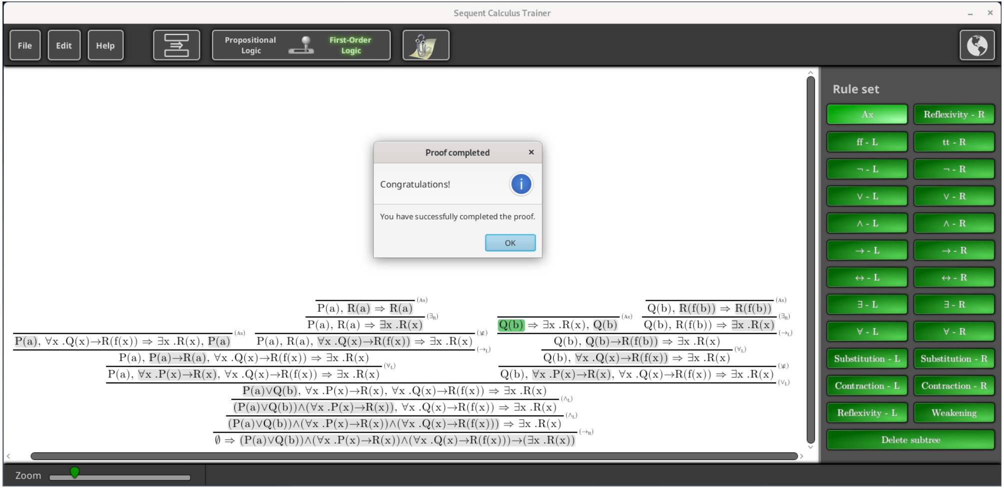
$$
\dfrac{\dfrac{\dfrac{\dfrac{p(a), (\forall x.\, p(x) \Rightarrow r(x)) \vdash \exists x.\, r(x) \qquad q(b), (\forall x.\, q(x) \Rightarrow r(f(x))) \vdash \exists x.\, r(x)}{p(a) \vee q(b), (\forall x.\, p(x) \Rightarrow r(x)), (\forall x.\, q(x) \Rightarrow r(f(x))) \vdash \exists x.\, r(x)}\,(\vee\text{-L,DROP})}{(p(a) \vee q(b)), (\forall x.\, p(x) \Rightarrow r(x)) \wedge (\forall x.\, q(x) \Rightarrow r(f(x))) \vdash \exists x.\, r(x)}\,(\wedge\text{-L})}{(p(a) \vee q(b)) \wedge (\forall x.\, p(x) \Rightarrow r(x)) \wedge (\forall x.\, q(x) \Rightarrow r(f(x))) \vdash \exists x.\, r(x)}\,(\wedge\text{-L})}{\vdash \big((p(a) \vee q(b)) \wedge (\forall x.\, p(x) \Rightarrow r(x)) \wedge (\forall x.\, q(x) \Rightarrow r(f(x)))\big) \Rightarrow \exists x.\, r(x)}\,(\Rightarrow\text{-R})
$$

A proof by “case distinction”.

Sequent Calculus Trainer

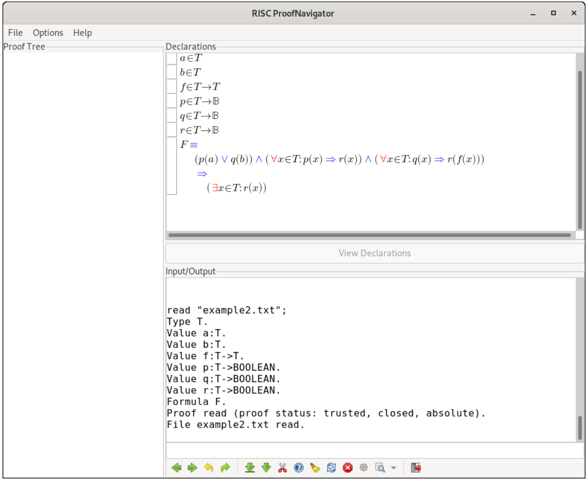


Sequent Calculus Trainer



RISC ProofNavigator

```
ProofNavigator &
```

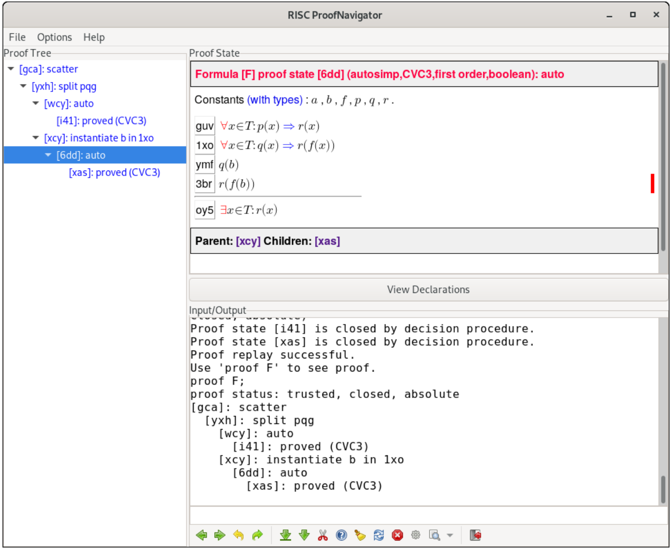



```
% example2.txt
newcontext "example2";

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



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.

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

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

Proof Tree Construction: Data

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

- $y = [y_0, y_1, \ldots]$: an infinite sequence of variables that do *not* occur in $\Gamma \vdash \Delta$.
  - These variables can be used as eigenvariables in rules (∀-R) and (∃-L).
- $a = [a_0, a_1, \ldots]$: an infinite sequence of term sequences:
  - The terms in these sequences are available as witnesses in rules (∀-L) and (∃-R).
    - If some function symbols occur in $\Gamma \vdash \Delta$, all sequences $a_0, a_1, \ldots$ are infinite.
  - $[t_0] \circ a_0 = [t_0, \ldots]$: 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 $[y_i, \ldots]$ of all terms that contain y_i and are constructed from $y_1, \ldots, y_i$ 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 $y_1, \ldots, y_n$ have been used as eigenvariables in rules (∀-R) or (∃-L); the sequences $a_0, a_1, \ldots, a_n$ contain all terms in which these variables may occur.

Proof Tree Construction: Algorithm

```
procedure SEARCH(Γ ⊢ Δ)
    INITIALIZE(y, a, t_0)
    T, ts, n ← ⟨Γ ⊢ Δ⟩, [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 ¬empty(a_i) then
                ts, a_i ← ts ∘ [head(a_i)], tail(a_i)
            end if
        end for
    end while
    if T is complete then
        WRITE("T proves Γ ⊢ Δ")
    else
        WRITE("T refutes Γ ⊢ Δ")
    end if
end procedure
```

```
procedure EXPAND(N, T, ts, y, ↕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 (∀-L) or (∃-R) applies do
        repeatedly apply the rule for every t ∈ ts
    end for
    for every leaf formula in S to which (∀-R) or (∃-L) applies do
        n ← n + 1
        apply the rule for x ← 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 (∀-L) or (∃-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 (∀-L) and (∃-R) have been applied for every term in ts. Consider every node $\Gamma_i \vdash \Delta_i$ along the path $\Gamma \vdash \Delta \to \ldots \to \Gamma_k \vdash \Delta_k$ 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 $I_S(p)(t_1, \ldots, t_n) :\Leftrightarrow p(t_1, \ldots, t_n) \in \bigcup\{\Gamma_i \mid 0 \leq i \leq k\}$. 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 $\Gamma \vdash \Delta \to \ldots$. 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 ($\forall$-L) and ($\exists$-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 $\{F_1, F_2, \ldots\}$ 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 v, considering every atom as a propositional variable. A set $\{F_1, F_2, \ldots\}$ is propositionally satisfiable if there exists some valuation v that satisfies every formula F_i in the set.
- Example: formula $p(x) \wedge \neg q(x, y)$.
  - Ground instances: $\{p(c) \wedge \neg q(c, c), p(c) \wedge \neg q(c, f(c)), p(f(c)) \wedge \neg q(f(c), c), \ldots\}$
  - Valuation: $[p(c) \mapsto \text{true}, q(c, c) \mapsto \text{false}, q(c, f(c)) \mapsto \text{false}, p(f(c)) \mapsto \text{true}, q(f(c), c) \mapsto \text{false}, \ldots]$

The previously stated theorem abound 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 $F_1 \wedge \ldots \wedge F_n$ 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 $F_1 \wedge \ldots \wedge F_n$ of a finite subset of its instances is propositionally unsatisfiable.
  - Proof sketch: the contraposition of the previous theorem.
- Theorem: Formula $\forall x_1, \ldots, x_n.\ F$ in Skolem normal form is unsatisfiable if some conjunction $F_1 \wedge \ldots \wedge F_n$ 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 ← SKOLEMNORMALFORMMATRIX(¬G)
    Fs ← ⊤
    i ← 1
    loop
        Fs ← Fs ∧ F(i)        ▹ Add instance i of F
        if Fs is propositionally unsatisfiable then
            WRITE("G is first-order valid")
            return
        end if
        i ← 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.

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 blindingly trying out all possibilities.