FIRST-ORDER LOGIC: THE RESOLUTION METHOD

Course "Computational Logic"

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

JⱯU JOHANNES KEPLER
UNIVERSITY LINZ

Clause Form

Resolution can show the unsatisfiability of first-order formulas in *clause form*.

- Clause Form: a conjunction $C_1 \wedge \ldots \wedge C_n$ of clauses.
    - Clause: a closed formula $\forall x_1, \ldots, x_n. \ L_1 \vee \ldots \vee L_m$ with literals $L_1, \ldots, L_m$.
    - Literal: an atomic formula $p(t_1, \ldots, t_n)$ or its negation $\neg p(t_1, \ldots, t_n)$.
- Convention: quantifiers are dropped.
    - Every clause is implicitly universally quantified over its free variables.
    - Thus a first-order clause form can be represented as a set of sets of literals.
- Theorem: for every first-order formula F there exists a clause form that is satisfiable if and only if F is satisfiable.
    - Proof sketch: Convert F into Skolem normal form $\forall x_1, \ldots, x_n. \ G$ and convert matrix G into conjunctive normal form $G_1 \wedge \ldots \wedge G_m$. The resulting formula $\forall x_1, \ldots, x_n. \ G_1 \wedge \ldots \wedge G_m$ is logically equivalent to $(\forall x_1, \ldots, x_n. \ G_1) \wedge \ldots \wedge (\forall x_1, \ldots, x_n. \ G_m)$.
    - Note: If $F = F_1 \wedge \ldots \wedge F_n$ with closed $F_1, \ldots, F_n$, we may convert each F_i into clause form.

To show that F is valid, it suffices to show that the *clause form* of $\neg F$ is unsatisfiable.

Example

Assume our goal is to show the validity of formula $\forall y.\ \exists z.\ (p(z, y) \Leftrightarrow \exists x.\ (p(z, x) \wedge p(x, z)))$.

- **Negation:** (connective $\neg$ "pushed down" to literals)

$$\exists y.\ \forall z.\ (p(z, y) \Leftrightarrow \forall x.\ \neg p(z, x) \vee \neg p(x, z))$$

- **Eliminate $\Leftrightarrow$:** $(A \Leftrightarrow B \equiv (A \Rightarrow B) \wedge (B \Rightarrow A) \equiv (\neg A \vee B) \wedge (A \vee \neg B))$

$$\exists y.\ \forall z.\ (\neg p(z, y) \vee \forall x.\ \neg p(z, x) \vee \neg p(x, z)) \wedge (p(z, y) \vee \exists x.\ p(z, x) \wedge p(x, z))$$

- **Skolemization:** (constant c for y, function f for x)

$$\forall z.\ (\neg p(z, c) \vee \forall x.\ \neg p(z, x) \vee \neg p(x, z)) \wedge (p(z, c) \vee (p(z, f(z)) \wedge p(f(z), z)))$$

- **Prenex Form:**

$$\forall z.\ \forall x.\ (\neg p(z, c) \vee \neg p(z, x) \vee \neg p(x, z)) \wedge (p(z, c) \vee (p(z, f(z)) \wedge p(f(z), z)))$$

- **Conjunctive Normal Form:**

$$\forall z.\ \forall x.\ (\neg p(z, c) \vee \neg p(z, x) \vee \neg p(x, z)) \wedge (p(z, c) \vee p(z, f(z))) \wedge (p(z, c) \vee p(f(z), z))$$

- **Clause Form:**

$$(\forall z.\ \forall x.\ \neg p(z, c) \vee \neg p(z, x) \vee \neg p(x, z)) \wedge (\forall z.\ p(z, c) \vee p(z, f(z))) \wedge (\forall z.\ p(z, c) \vee p(f(z), z))$$

Set of set of literals $\{\{\neg p(z, c), \neg p(z, x), \neg p(x, z)\},\ \{p(z, c), p(z, f(z))\},\ \{p(z, c), p(f(z), z)\}\}$.

Ground Resolution

(Davis Putnam, 1960)

- Our goal is to show the unsatisfiability of the clause form
$$\{\{\neg p(z,c), \neg p(z,x), \neg p(x,z)\}, \ \{p(z,c), p(z,f(z))\}, \ \{p(z,c), p(f(z),z)\}\}.$$

  - Herbrand's Theorem: it suffices to show the unsatisfiability of a set of ground instances.

- We show the unsatisfiability of
$$\{\{\neg p(c,c)\}, \ \{\{\neg p(f(c),c), \neg p(c,f(c))\}, \ \{p(c,c), p(c,f(c))\}, \ \{p(c,c), p(f(c),c)\}\}$$

  - Two instances of clause 1, one instance of clause 2, and one instance of clause 3.

- For this, we may apply the resolution method of propositional logic:
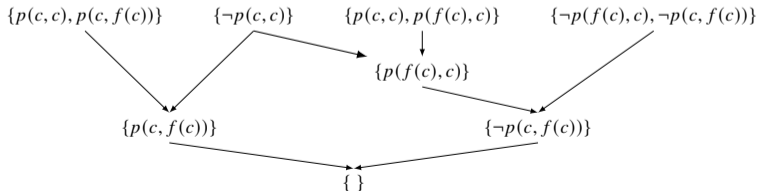
Rather than "guessing" appropriate instances, we better apply *unification*.   3/33

First-Order Resolution

John Alan Robinson, 1965: a calculus with judgement $F \vdash$ ("F is unsatisfiable").

$$\frac{\{\,\} \in F}{F \vdash} \text{ (AX)} \qquad \frac{C \in F \quad \sigma \text{ is a bijective renaming of the variables in } C \quad F \cup \{C\sigma\} \vdash}{F \vdash} \text{ (REN)}$$

$$\frac{\begin{array}{c} C \cup \{p(t_1, \ldots, t_n)\} \in F \quad D \cup \{\neg p(u_1, \ldots, u_n)\} \in F \quad \sigma \text{ is mgu of } \{(t_1, u_1), \ldots, (t_n, u_n)\} \\ C \cup \{p(t_1, \ldots, t_n)\} \text{ and } D \cup \{\neg p(u_1, \ldots, u_n)\} \text{ have no common variables} \quad F \cup \{C\sigma \cup D\sigma\} \vdash \end{array}}{F \vdash} \text{ (RES)}$$

$$\frac{\begin{array}{c} C \cup \{p(t_1, \ldots, t_n), p(u_1, \ldots, u_n)\} \in F \quad \sigma \text{ is mgu of } \{(t_1, u_1), \ldots, (t_n, u_n)\} \\ F \cup \{C\sigma \cup \{p(t_1\sigma, \ldots, t_n\sigma)\}\} \vdash \end{array}}{F \vdash} \text{ (FACT)}$$

- Axiom (AX): a formula with an empty clause is unsatisfiable.
- Resolution (RES): if two clauses contain literals that become complimentary when applying most general unifier σ, we may combine the clauses after dropping these literals and applying σ.
- To make (RES) applicable, it may be necessary to apply two auxiliary rules:
  - Renaming (REN): rename the variables in a clause to become distinct from those in another clause.
  - Factoring (FACT): if a clause contains two literals that become identical when applying most general unifier σ, we may drop one of the literals and apply σ to the resulting clause.
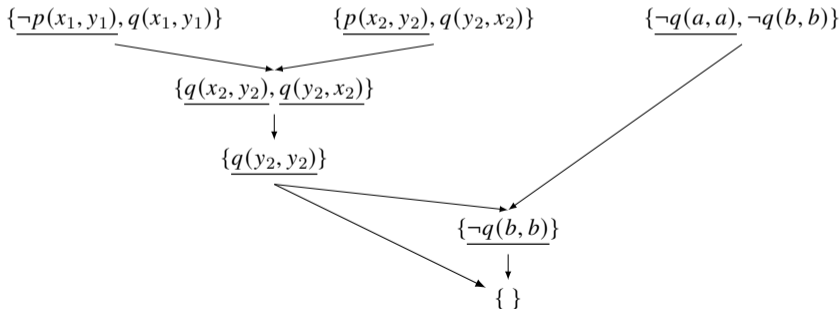
A Simple Example

We show the unsatisfiability of the formula

$$(\forall x, y.\ \neg p(x, y) \lor q(x, y)) \land (\forall x, y.\ p(x, y) \lor q(y, x)) \land (\neg q(a, a) \lor \neg q(b, b))$$

which is represented by the set of clauses (with no common variables)

$$\{\{\neg p(x_1, y_1), q(x_1, y_1)\},\ \{p(x_2, y_2), q(y_2, x_2)\},\ \{\neg q(a, a), \neg q(b, b)\}\}$$
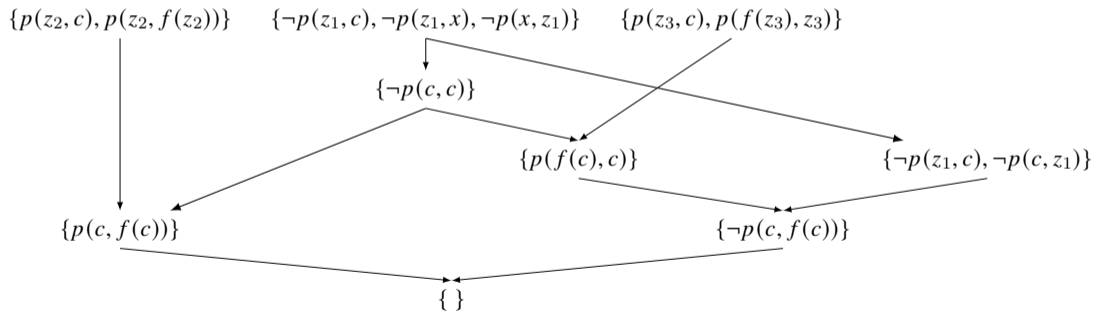
by the following refutation proof:

Three resolution steps and one factoring step.

Original Example

We show the unsatisfiability of

$$\{\{\neg p(z_1,c), \neg p(z_1,x), \neg p(x,z_1)\}, \{p(z_2,c), p(z_2,f(z_2))\}, \{p(z_3,c), p(f(z_3),z_3)\}\}.$$

by the following refutation proof:

$\{p(z_2,c), p(z_2,f(z_2))\}$ $\{\neg p(z_1,c), \neg p(z_1,x), \neg p(x,z_1)\}$ $\{p(z_3,c), p(f(z_3),z_3)\}$

$\{\neg p(c,c)\}$

$\{p(f(c),c)\}$ $\{\neg p(z_1,c), \neg p(c,z_1)\}$

$\{p(c,f(c))\}$ $\{\neg p(c,f(c))\}$

$\{\,\}$

The Importance of Factoring

Consider clause form $\{\{p(x,x), p(c,x)\}, \{\neg p(y,y), \neg p(c,y)\}\}$.

- Without Factoring:

$$\{\underline{p(x,x)}, p(c,x)\}, \{\underline{\neg p(y,y)}, \neg p(c,y)\} \rightarrow \{p(c,y), \neg p(c,y)\} \equiv \top$$

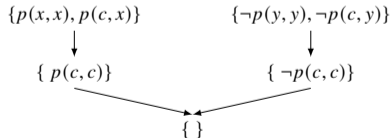$$\{\underline{p(x,x)}, p(c,x)\}, \{\neg p(y,y), \underline{\neg p(c,y)}\} \rightarrow \{p(c,c), \neg p(c,c)\} \equiv \top$$

$$\{p(x,x), \underline{p(c,x)}\}, \{\underline{\neg p(y,y)}, \neg p(c,y)\} \rightarrow \{p(c,c), \neg p(c,c)\} \equiv \top$$

$$\{p(x,x), \underline{p(c,x)}\}, \{\neg p(y,y), \underline{\neg p(c,y)}\} \rightarrow \{p(y,y), \neg p(y,y)\} \equiv \top$$

  - By using only resolution, just trivial consequences can be derived.
  - Thus no progress towards proof of unsatisfiability can be made.

- With Factoring:

$$
\begin{array}{ccc}
\{p(x,x), p(c,x)\} & & \{\neg p(y,y), \neg p(c,y)\} \\
\downarrow & & \downarrow \\
\{\, p(c,c)\} & & \{\, \neg p(c,c)\} \\
& \searrow \quad \swarrow & \\
& \{\,\} &
\end{array}
$$

  - By using also factoring, unsatisfiability can be easily shown.

Factoring is indeed essential for the completeness of the calculus.

First-Order Resolution

Actually, a single rule may subsume the work of renaming, factoring, and resolution.

$$\frac{\begin{array}{c} C \in F \qquad D \in F \qquad C' \subseteq C \qquad D' \subseteq D \\ \text{all literals in } C' \text{ are unnegated, all literals in } D' \text{ are negated (or the other way round)} \\ \sigma_1 \text{ and } \sigma_2 \text{ are bijective renamings of the variables in } C \text{ and } D \\ \text{such that } C\sigma_1 \text{ and } D\sigma_2 \text{ have no common variables} \\ \sigma \text{ is mgu of all pairs of literals in } C'\sigma_1 \cup \overline{D'}\sigma_2 \\ C'' = (C \backslash C')\sigma_1 \qquad D'' = (D \backslash D')\sigma_2 \qquad F \cup \{C''\sigma \cup D''\sigma\} \vdash \end{array}}{F \vdash} \text{ (RES')}$$

- Generalized Resolution (RES'):
  - Renames clauses to have disjoint sets of variables.
  - Resolves a *set* of positive literals with a *set* of negative literals.
  - Factors the literals within each set.

The calculus only requires the two rules (AX) and (RES').
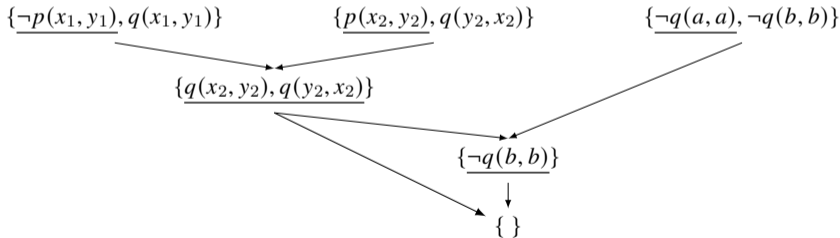
A Simple Example (Revisited)

We show the unsatisfiability of the formula

$$(\forall x, y. \; \neg p(x, y) \vee q(x, y)) \wedge (\forall x, y. \; p(x, y) \vee q(y, x)) \wedge (\neg q(a, a) \vee \neg q(b, b))$$

which is represented by the set of clauses (with no common variables)

$$\{\{\neg p(x_1, y_1), q(x_1, y_1)\}, \; \{p(x_2, y_2), q(y_2, x_2)\}, \; \{\neg q(a, a), \neg q(b, b)\}\}$$

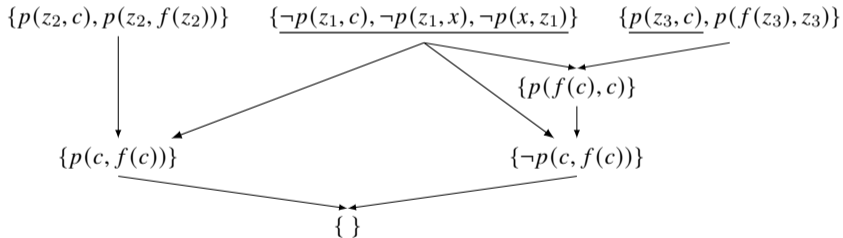by the following refutation proof:

Three (generalized) resolution steps.

Original Example (Revisited)

We show the unsatisfiability of

$$\{\{\neg p(z_1,c), \neg p(z_1,x), \neg p(x,z_1)\}, \{p(z_2,c), p(z_2,f(z_2))\}, \{p(z_3,c), p(f(z_3),z_3)\}\}.$$

by the following refutation proof:

Four (generalized) resolution steps.

Soundness and Completeness of First-Order Resolution

- Soundness: if $F \vdash$ can be derived, F is unsatisfiable.
  - Proof sketch: The soundness of each rule can be shown according to the semantics of first-order logic (compare also with the proof sketch of the soundness of resolution in propositional logic).

- Completeness: if F is unsatisfiable, $F \vdash$ can be derived.
  - Proof sketch: Assume that F is unsatisfiable. By the corollary of Herbrand's theorem and the compactness theorem, some finite set S of ground instances of clauses of F is unsatisfiable. Thus, by the completeness of propositional resolution, there exists a propositional refutation proof of S. Now, one can show that for each application of rule (RES) in propositional resolution to derive a new propositional clause C' there exists an application of rule (RES') in first-order resolution to derive a first-order clause C such that C' is a ground instance of C. Since a propositional refutation proof can only be completed by application of rule (AX), propositional resolution has derived the empty clause C'. Thus the first-order refutation proof has derived a clause C such that the empty clause C' is an instance of C which implies that also C is the empty clause. Therefore also the first-order derivation proof can be completed by application of rule (AX).

This notion of completeness is often called "refutation completeness".

First-Order Resolution in OCaml

```
let rec mgu l env = ... ;;
let unifiable p q = ... ;;
let rename pfx cls = ... ;;


let resolvents cl1 cl2 p acc = (* General resolution rule, incorporating factoring. *)
  let ps2 = filter (unifiable(negate p)) cl2 in
  if ps2 = [] then acc else
  let ps1 = filter (fun q -> q <> p & unifiable p q) cl1 in
  let pairs = allpairs (fun s1 s2 -> s1,s2)
                       (map (fun pl -> p::pl) (allsubsets ps1))
                       (allnonemptysubsets ps2) in
  itlist (fun (s1,s2) sof ->
            try image (subst (mgu (s1 @ map negate s2) undefined))
                      (union (subtract cl1 s1) (subtract cl2 s2)) :: sof
            with Failure _ -> sof) pairs acc;;


let resolve_clauses cls1 cls2 =
  let cls1' = rename "x" cls1 and cls2' = rename "y" cls2 in
  itlist (resolvents cls1' cls2') cls1' [];;
```

First-Order Resolution in OCaml

```
let rec resloop0 (used,unused) = (* Basic "Argonne" loop, the "given clause algorithm" *)
  match unused with
    [] -> failwith "No proof found"
  | cl::ros ->
      print_string(string_of_int(length used) ^ " used; "^
                   string_of_int(length unused) ^ " unused.");
      print_newline();
      let used' = insert cl used in
      let news = itlist(@) (mapfilter (resolve_clauses cl) used') [] in
      if mem [] news then true else resloop0 (used',ros@news);;

let pure_resolution0 fm = resloop0([],simpcnf(specialize(pnf fm)));;

let resolution0 fm =
  let fm1 = askolemize(Not(generalize fm)) in
  map (pure_resolution0 ** list_conj) (simpdnf fm1);;
```

Main loop picks clause `cl` from list `unused`, generates all resolvents of `cl` with clauses from `used`, and appends the results to `unused`; thus every pair of clauses is tried <u>once</u>.

First-Order Resolution in OCaml

```
# let davis_putnam_example = resolution0
 <<exists x. exists y. forall z.
         (F(x,y) ==> (F(y,z) /\ F(z,z))) /\
         ((F(x,y) /\ G(x,y)) ==> (G(x,z) /\ G(z,z)))>>;;
0 used; 3 unused.
1 used; 2 unused.
2 used; 3 unused.
...
80 used; 468 unused.
81 used; 473 unused.
82 used; 478 unused.
83 used; 483 unused.
84 used; 488 unused.
val davis_putnam_example : bool list = [true]
```

The number of clauses explodes, because a lot of them are actually redundant.

Removing Redundant Clauses

Assume that `resolve_clauses` generates a new clause C.

- Tautologies: If C contains $p(t_1, \ldots, t_n)$ and $\neg p(t_1, \ldots, t_n)$, we can delete C.
  - C is a tautology, i.e., logically equivalent to $\top$.
  - One can show that for every refutation that uses C there exists one that does not.
- Subsumption: C may subsume or be subsumed by an existing clause D.
  - C subsumes D if $C\sigma \subseteq D$ for some instantiation σ (thus D is a logical consequence of C).
    - Theorem: if C subsumes C', then any resolvent of C' and some clause D is subsumed either by C itself or by a resolvent of C and D.
  - Forward Deletion: if C is subsumed by D in `unused`, we can delete C.
    - Anything that would be generated from C will be generated from D.
  - Backward Replacement: if C subsumes D in `unused`, we can replace D by C.
    - Anything that would be generated from D will be generated from C.

Simple optimizations that may help to keep the clause set in check.

Removing Redundancies in OCaml

```ocaml
let subsumes_clause cls1 cls2 =
  let rec subsume env cls =
    match cls with
      [] -> env
    | l1::clt ->
        tryfind (fun l2 -> subsume (match_literals env (l1,l2)) clt) cls2
  in can (subsume undefined) cls1;;


let incorporate gcl cl unused =
  if trivial cl or
     exists (fun c -> subsumes_clause c cl) (gcl::unused)
  then unused else replace cl unused;;


let rec resloop (used,unused) =
  match unused with
    [] -> failwith "No proof found"
  | cl::ros ->
      ...
      if mem [] news then true
      else resloop(used',itlist (incorporate cl) news ros);;
```

Removing Redundancies in OCaml

```
# let davis_putnam_example = resolution
 <<exists x. exists y. forall z.
        (F(x,y) ==> (F(y,z) /\ F(z,z))) /\
        ((F(x,y) /\ G(x,y)) ==> (G(x,z) /\ G(z,z)))>>;;
0 used; 3 unused.
1 used; 2 unused.
2 used; 3 unused.
3 used; 6 unused.
4 used; 5 unused.
5 used; 4 unused.
6 used; 3 unused.
7 used; 2 unused.
val davis_putnam_example : bool list = [true]
```

Now a refutation is found very quickly.

The Resolution Method

Roughly speaking, resolution combines ideas from the DP method with unification.

- The resolution method has bottom-up and local characteristics:
  - Bottom-Up: Resolution does not consider the proof "goal" but extends the current "knowledge" by new "lemmas"; these are closed formulas that are (independently of any context) generally valid and can be later instantiated in different ways.
  - Local: When combining two clauses C and D to a resolvent

    $$C''\sigma \cup D''\sigma$$

    the substitution σ is only applied to the resolvent and does not affect the variables in the rest of the formula.

Dual to the top-down and global characteristics of the tableau method; both have their strengths and weaknesses.

Horn Clause Formulas

Generally, finding refutations by general resolution can be very costly; however, for restricted clause forms there exist more efficient search strategies.

- **Horn Clause:** a clause with at most one positive literal (Alfred Horn, 1951).
  - **Definite Clause:** a Horn clause with exactly one positive literal.
- **Concrete Syntax:** let $P_1, \ldots, P_n, P$ be unnegated literals (i.e., atomic formulas).
  - **Fact:** $\top \Rightarrow P$ (alternatively: $\Rightarrow P$)

$$\{P\}$$

  - **Rule:** $P_1 \wedge \ldots \wedge P_{n \geq 1} \Rightarrow P$

$$\{\neg P_1, \ldots, \neg P_n, P\}$$

  - **Goal (Query):** $P_1 \wedge \ldots \wedge P_{n \geq 1} \Rightarrow \bot$ (alternatively: $P_1 \wedge \ldots \wedge P_n \Rightarrow$)

$$\{\neg P_1, \ldots, \neg P_n\}$$

A Horn clause formula is a conjunction of Horn clauses (represented as a set).

Core Ideas

From now on, let F be a Horn clause formula with definite clauses only.

- We will prove the following kind of formulas (with unnegated literals $P_1, \ldots, P_m$):
  $$F \Rightarrow \exists x_1, \ldots, x_n.\ P_1 \wedge \ldots \wedge P_m$$
- For this, it suffices to show the unsatisfiability of the formula
  $$F \wedge \forall x_1, \ldots, x_n.\ \neg P_1 \vee \ldots \vee \neg P_m$$
  i.e., of the Horn clause formula
  $$F \wedge (P_1 \wedge \ldots \wedge P_m \Rightarrow \bot)$$
  which is represented by the following set:
  $$F \cup \{\{\neg P_1, \ldots, \neg P_n\}\}$$
- A successful refutation determines ground terms $t_1, \ldots, t_n$ that make this formula valid:
  $$F \Rightarrow (P_1 \wedge \ldots \wedge P_m)[t_1/x_1, \ldots t_1/x_n]$$

The core is to show the unsatisfiability of $F \cup \{G\}$ with goal $G = \{\neg P_1, \ldots, \neg P_n\}$.

Semantics of Horn Clause Formulas

- Theorem: $M_F \models F$ where M_F is the "minimal" Herbrand structure $M_F = (D_M, I_M)$ defined by the following predicate interpretation:

  $I_M(p)(t_1, \ldots, t_n) :\Leftrightarrow$ for every Herbrand structure H with $H \models F$ we have $I_H(p)(t_1, \ldots, t_n)$

  - "Minimal" interpretation: M_F interprets literal $p(t_1, \ldots, t_n)$ only as "true" if $F \models p(t_1, \ldots, t_n)$.
  - Proof sketch: the theorem can be shown from the fact that every Horn clause has at most one unnegated literal, i.e., is effectively an implication with unnegated assumptions (we omit the details). The theorem would not be true if we allowed a clause of form $\{p(a), q(a)\}$ with two unnegated literals: there exists an Herbrand structure where only $p(a)$ is true, and another one where only $q(a)$ is true; thus in M both $p(a)$ and $q(a)$ are false; therefore M does not satisfy the clause.

- Corollary: $M_F \models F \cup \{G_1, \ldots, G_n\}$, if $F \cup \{G_1, \ldots, G_n\}$ with goals $G_1, \ldots, G_n$ is satisfiable.
  - Proof sketch: We assume that $F \cup \{G_1, \ldots, G_n\}$ is satisfiable but not satisfied by M_F and show a contradiction. By the previous theorem M_F satisfies F. Therefore, since $F \cup \{G_1, \ldots, G_n\}$ is not satisfied by M_F and all G_i are closed formulas, we have some $G_i = \{\neg P_1, \ldots, \neg P_m\}$ such that M_F satisfies $P_1, \ldots, P_m$. Since M_F is minimal, every Herbrand model of F satisfies $P_1, \ldots, P_n$, i.e., no Herbrand model of F satisfies $F \cup \{G_1, \ldots, G_n\}$. This, by Herbrand's theorem, contradicts out assumption that $F \cup \{G_1, \ldots, G_n\}$ is satisfiable.

Every satisfiable Horn clause formula has a canonical model.

Semantics of Horn Clause Formulas

- Theorem: If $F \cup \{G\}$ is unsatisfiable for goal G, then there exists some ground instance G' of G such that $F \cup \{G'\}$ is unsatisfiable.

  - Proof sketch: From Herbrand's theorem and the compactness theorem one can show that, since F is satisfiable but $F \cup \{G\}$ is not, there exist ground instances $G_1, \ldots, G_n$ of G such that $F \cup \{G_1, \ldots, G_n\}$ is unsatisfiable. By the previous theorem, the minimal Herbrand structure M_F satisfies F, but since $F \cup \{G_1, \ldots, G_n\}$ is unsatisfiable, M_F does not satisfiy $\{G_1, \ldots, G_n\}$. Since all the G_i are ground formulas, M_F does not satisfy some $G_i = \{P_1, \ldots, P_m\}$. Thus M_F satisfies $\neg P_1, \ldots, \neg P_m$. By the definition of M_F, every Herbrand model of F satisfies $\neg P_1, \ldots, \neg P_m$. Therefore no Herbrand model of F satisfies G_i, consequently $F \cup \{G_i\}$ is unsatisfiable.

- Corollary: if $\models F \Rightarrow \exists x_1, \ldots, x_n. \ P_1 \wedge \ldots \wedge P_m$, then $\models F \Rightarrow (P_1 \wedge \ldots \wedge P_m)[t_1/x_1, \ldots t_n/x_n]$ for some ground terms $t_1, \ldots, t_n$.

  - Proof sketch: from above, using $G = \{\neg P_1, \ldots, \neg P_n\}$.

Every successful query has a "definite answer".

Proofs of Horn Clause Formulas: SLD-Resolution

SLD ≈ "Selection-Linear Resolution with Definite Clauses" (Robert Kowalski, 1974)

$$\overline{F, \{\,\} \vdash} \quad \text{(AX)}$$

$$\{\neg P_1, \ldots, \neg P_{n \geq 0}, P\} \in F \qquad \neg Q \in G$$

σ_0 is a bijective renaming of the variables in $\{P_1, \ldots, P_n, P\}$ such that
$\{P_1 \sigma_0, \ldots, P_n \sigma_0, P \sigma_0\}$ and G have no common variables

$$\frac{\sigma \text{ is mgu of } P\sigma_0 \text{ and } Q \qquad F, G \backslash \{\neg Q\} \cup \{\neg P_1 \sigma_0 \sigma, \ldots, \neg P_n \sigma_0 \sigma\} \vdash}{F, G \vdash} \quad \text{(SLD)}$$

- Judgement $F, G \vdash$
  - Horn clause formula F with only definite clauses, goal clause G.
  - Interpreted as "$F \cup \{G\}$ is unsatisfiable".
- Rule (AX): an empty goal clause is unsatisfiable.
- Rule (SLD): "matches" rule/fact $\{\neg P_1, \ldots, \neg P_{n \geq 0}, P\}$ in F to literal Q in goal G.
  - Application of rule replaces literal by (appropriately substituted) rule prerequisites.

A "goal-oriented" form of resolution.

Example

We infer the unsatisfiability of the following Horn clause formula:

$$p_3 \wedge p_4 \wedge (p_3 \wedge p_4 \Rightarrow p_1) \wedge (p_3 \Rightarrow p_2) \wedge (p1 \wedge p2 \Rightarrow \bot)$$

$$\frac{\frac{\frac{\frac{\frac{\frac{}{\{\{p_3\}, \{p_4\}, \{\neg p_3, \neg p_4, p_1\}, \{\neg p_3, p_2\}\}, \{\,\} \vdash}}{\{\{\underline{p_3}\}, \{p_4\}, \{\neg p_3, \neg p_4, p_1\}, \{\neg p_3, p_2\}\}, \{\neg \underline{p_3}\} \vdash}}{\{\{p_3\}, \{p_4\}, \{\neg p_3, \neg p_4, p_1\}, \{\neg p_3, \underline{p_2}\}\}, \{\neg \underline{p_2}\} \vdash}}{\{\{p_3\}, \{\underline{p_4}\}, \{\neg p_3, \neg p_4, p_1\}, \{\neg p_3, p_2\}\}, \{\neg \underline{p_4}, \neg p_2\} \vdash}}{\{\{\underline{p_3}\}, \{p_4\}, \{\neg p_3, \neg p_4, p_1\}, \{\neg p_3, p_2\}\}, \{\neg \underline{p_3}, \neg p_4, \neg p_2\} \vdash}}{\{\{p_3\}, \{p_4\}, \{\neg p_3, \neg p_4, \underline{p_1}\}, \{\neg p_3, p_2\}\}, \{\neg \underline{p_1}, \neg p_2\} \vdash}$$

A "linear" proof where only the goal formula changes.

Example

We infer the unsatisfiability of the following Horn clause formula:

$$p(x, c, x) \land \big(p(x, y, z) \Rightarrow p(x, f(y), f(z)) \big) \land \neg p(f(c), f(f(c)), z)$$

$$
\cfrac{
\cfrac{
\cfrac{
\{p(x, c, x)\}, \{\neg p(x, y, z), p(x, f(y), f(z))\} \atop \{\,\}
}{
\{p(x, c, x)\}, \{\neg p(x, y, z), p(x, f(y), f(z))\} \atop \{\neg p(f(c), c, z_2)\}
} \vdash
}{
\{p(x, c, x)\}, \{\neg p(x, y, z), p(x, f(y), f(z))\} \atop \{\neg p(f(c), f(c), z_1)\}
} \vdash
}{
\{p(x, c, x)\}, \{\neg p(x, y, z), p(x, f(y), f(z))\} \atop \{\neg p(f(c), f(f(c)), z)\}
} \vdash
$$

$\sigma_0 = \quad [\ ]$
$\sigma = \quad [x \mapsto f(c), z_2 \mapsto f(c)]$

$\sigma_0 = \quad [z \mapsto z_2]$
$\sigma = \quad [x \mapsto f(c), y \mapsto c, z_1 \mapsto f(z_2)]$

$\sigma_0 = \quad [z \mapsto z_1]$
$\sigma = \quad [x \mapsto f(c), y \mapsto f(c), z \mapsto f(z_1)]$

Composed substitution $[x \mapsto f(c), y \mapsto f(c), z \mapsto f(f(f(c)))]$.

The composition of substitutions $(\sigma_0 \circ \sigma) \circ \ldots$ performed by a sequence of applications of rule (SLD) determines terms $t_1, \ldots, t_n$ for the variables $x_1, \ldots, x_n$ in the original goal G.

Correctness of SLD-Resolution

Let F be a Horn clause formula with only definite clauses and G a goal clause.

- Soundness: if $F, G \vdash$ is derivable, then $F \cup \{G\}$ is unsatisfiable.
  - Proof sketch: Rule (AX) is clearly sound. The correctness of rule (SLD) can be established from the correctness of rule (RES').

- Completeness: if $F \cup \{G\}$ is unsatisfiable, then $F, G \vdash$ is derivable.
  - Proof sketch: First the completeness of SLD-resolution in propositional logic is proved by showing that every resolution proof of $F \cup \{G\}$ can be transformed into a proof by SLD-resolution. By Herbrand's theorem and compactness, if $F \cup \{G\}$ is unsatisfiable, there is a finite set of ground instances of $F \cup \{G\}$ that is unsatisfiable. Therefore this set has a propositional refutation by SLD-resolution. Finally, it can be shown that this refutation is an instance of SLD-resolution in first-order logic.

As the tableaux method, we can implement SLD-resolution by "iterative deepening".

Implementation in OCaml

```
let renamerule k (asm,c) = ... ;;

let rec backchain rules n k env goals =
  match goals with
     [] -> env
   | g::gs ->
       if n = 0 then failwith "Too deep" else
       tryfind (fun rule ->
           let (a,c),k' = renamerule k rule in
           backchain rules (n - 1) k' (unify_literals env (c,g)) (a @ gs))
       rules;;

let hornify cls = (* replace goal G by rule "G => False" *)
  let pos,neg = partition positive cls in
  if length pos > 1 then failwith "non-Horn clause"
  else (map negate neg,if pos = [] then False else hd pos);;

let hornprove fm = (* start proof search with goal False *)
  let rules = map hornify (simpcnf(skolemize(Not(generalize fm)))) in
  deepen (fun n -> backchain rules n 0 undefined [False],n) 0;;
```

Implementation in OCaml

```
# let p32 = hornprove
 <<(forall x. P(x) /\ (G(x) \/ H(x)) ==> Q(x)) /\
    (forall x. Q(x) /\ H(x) ==> J(x)) /\
    (forall x. R(x) ==> H(x))
    ==> (forall x. P(x) /\ R(x) ==> J(x))>>;;
Searching with depth limit 0
...
Searching with depth limit 7
Searching with depth limit 8
val p32 : (string, term) func * int = (<func>, 8)
```

Limitations of Horn Clause Logic

Horn clause logic is not as expressive as general first-order logic.

- Consider formula $P_1 \wedge \ldots \wedge \neg P_i \wedge \ldots \wedge P_n \Rightarrow P$
  - There does not exist any logically equivalent Horn clause formula.
  - Horn clause logic cannot deal with negative premises.

Horn clause logic is often less interesting for proving than for *computing*.

Prolog

Programming in Logic (Alain Colmerauer and Philippe Roussel, 1972).

- An implementation of SLD resolution as a programming language.
  - A "pragmatic" implementation: efficient, but logically not complete.
    - Proof search is implemented in a "depth-first, left-to-right" fashion; this may run into "infinite recursion", even if an SLD refutation exists.
    - Omits costly occurs check in unification; thus cyclic terms may be created (typically unintentionally, as results of programming errors).
- Concrete syntax for facts, rules, goals:

  ```
  P :- P1, P2, P3 .   %% a rule: P holds, if P1 and P2 and P3 hold.
  P .                 %% a fact: P holds unconditionally.
  ?- P1, P2, P3 .     %% a goal: prove P1 and P2 and P3.
  ```

  - Builtin implementation of various data types and operations.
  - Implementation of side effects such as input and output.

A "full-fledged" (Turing-complete) programming language.

Example

```
%% file fol4.pl with predicate add(X,Y,Z) interpreted as "adding X and Y gives Z".
add(X,zero,X).                          %% adding X and zero gives X.
add(X,succ(Y),succ(Z)) :- add(X,Y,Z). %% adding X and Y+1 gives Z+1, if adding X and Y gives Z.

debian10!1> prolog fol4.pl
Welcome to SWI-Prolog (threaded, 64 bits, version 8.0.2)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit http://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- add(succ(zero),succ(succ(zero)),Z).
Z = succ(succ(succ(zero))) .

?-
```

This is (up to renaming) our previous example of an SLD resolution.

Example

```
?- add(succ(zero),Y,succ(succ(succ(zero)))).
Y = succ(succ(zero)) .

?- add(X,Y,succ(succ(succ(zero)))).
X = succ(succ(succ(zero))),
Y = zero ;
X = succ(succ(zero)),
Y = succ(zero) ;
X = succ(zero),
Y = succ(succ(zero)) ;
X = zero,
Y = succ(succ(succ(zero))) ;
false.
```

Prolog programs can be executed "inversely" and also produce multiple solutions.

Example

```
append([],Ys,Ys).
append([X|Xs],Ys,[X|Zs]) :- append(Xs,Ys,Zs).

partition([X|Xs],Y,[X|Ls],Rs) :- X =< Y, partition(Xs,Y,Ls,Rs).
partition([X|Xs],Y,Ls,[X|Rs]) :- X > Y, partition(Xs,Y,Ls,Rs).
partition([],Y,[],[]).

quicksort([X|Xs],Ys) :-
  partition(Xs,X,Left,Right),
  quicksort(Left,Ls),
  quicksort(Right,Rs),
  append(Ls,[X|Rs],Ys).
quicksort([],[]).

?- quicksort([3,1,4,1,5,9,2,7],X).
X = [1, 1, 2, 3, 4, 5, 7, 9] .
```

Prolog programs are written as "recursively defined" predicates.