Logical Reasoning

Goal: to have a computer automatically perform deduction or prove theorems

First, we need a language in which to communicate to the machine.

axioms

theorems

hypotheses

rules

Languages

Propositional Calculus

(or propositional logic)

1st Order Predicate Calculus

⋮

Propositional Logic

Propositions: Statements that are either true or false.

P: LISP runs on IBM PCs.

Q: IBM PCs are computers

R: Prolog runs on IBM PCs.

Propositional Logic Symbols or Connectives

$\wedge$ and

$\vee$ or

$\neg$ not

$\rightarrow$ implications

$P \wedge Q$

$P \wedge R \rightarrow Q$

$\neg R \wedge P$

Predicate Calculus

Some formulas with meanings that express a set of facts

1) man (Marcus)

2) Pompeian (Marcus)

3) born (Marcus, 40) [40 A.D.]

4) $\forall x$: man (x) $\rightarrow$ mortal (x)

5) $\forall x$: Pompeian (x) $\rightarrow$ died (x, 79)

6) erupted (volcano, 79)

7) $\forall x: \forall t_1: \forall t_2$: mortal(x) $\wedge$ born(x,t_1) $\wedge$ gt(t_2-t_1, 150) $\rightarrow$ dead(x,t_2)

8) now = 1994

9) $\forall x: \forall t$: [alive(x,t) $\rightarrow \neg$dead(x,t)] $\wedge$ [$\neg$dead(x,t) $\rightarrow$ alive(x,t)]

10) $\forall x: \forall t_1: \forall t_2$: died(x,$t_1$) $\wedge$ gt(t_2, t_1) $\rightarrow$ dead(x,t_2)

To prove: dead(Marcus, now)

One way

Pompeian(Marcus)

$\forall x$: Pompeian(x) $\rightarrow$ died(x, 79)

died(Marcus, 79)

gt(now, 79)

died(Marcus, 79) $\wedge$ gt(now, 79)

$\forall x$: $\forall t1$: $\forall t2$: died(x,t1) $\wedge$ gt(t2, t1) $\rightarrow$ dead(x,t2)

dead(Marcus, now)

This is a direct proof.

$x \rightarrow y$

$\neg y \rightarrow \neg x$

Proof by Contradiction

$\neg dead(Marcus, now)$

$\forall x: \forall t1: \forall t2: died(x,t1) \land gt(t2, t1) \rightarrow dead(x,t2)$

$\forall t_1: \neg[died(Marcus, t_1) \land gt(now, t_1)]$

$\forall t_1: \neg died(Marcus, t_1) \lor \neg gt(now, t_1)$

died(Marcus, 79)*

$\neg gt(now, 79)$ gt(now, 79)

$\otimes$ contradiction

*assume we already proved this separately

Resolution Theorem Provers for Predicate Calculus

Given: F: a database of axioms (set of formulas)

S: a conjecture (a formula)

Prove: F: logically implies S

Technique

- Construct $\neg S$: negated conjecture.
- Show $F' = F \cup \{\neg S\}$ is not satisfiable
(leads only to contradiction)

Since we are assuming F is satisfiable, we can conclude
$\neg\{\neg S\}$ or S

Part I — Preprocessing to express in homogeneous form

1) Eliminate →'s

Replace $A \to B$ by $\vee(\neg A, B)$

Running Example

$\forall x\, \forall y\, ((A(x) \to \neg C(x,y)) \to \neg \forall x\, \exists z\, \wedge(P(x,z), R(z))$

$\forall x\, \forall y\, (\vee(\neg A(x), \neg C(x,y)) \to \neg \forall x\, \exists z\, \wedge(P(x,z), R(z))$

$\forall x\, \forall y\, \vee(\neg \vee(\neg A(x), \neg C(x,y)), \neg \forall x\, \exists z\, \wedge(P(x,z), R(z))$

2) Reduce the scope of each ¬ to apply to at most one predicate, by applying rules.

1. Demorgan's Laws

   $\neg \vee(x_1, \ldots, x_n) \Rightarrow \wedge(\neg x_1, \ldots, \neg x_n)$

   $\neg \wedge(x_1, \ldots, x_n) \Rightarrow \vee(\neg x_1, \ldots, \neg x_n)$

2. $\neg(\neg x) \Rightarrow x$
3. $\neg(\forall x\, A) \Rightarrow \exists x(\neg A)$
4. $\neg(\exists x\, A) \Rightarrow \forall x(\neg A)$

$$\forall x\ \forall y \vee(\neg\vee(\neg A(x), \neg C(x,y)), \neg\forall x\ \exists z \wedge(P(x,z), R(z)))$$

$$\forall x\ \forall y \vee(\wedge(A(x), C(x,y)), \neg\forall x\ \exists z \wedge(P(x,z), R(z)))$$

$$\forall x\ \forall y \vee(\wedge(A(x), C(x,y)), \exists x\ \neg\exists z \wedge(P(x,z), R(z)))$$

$$\forall x\ \forall y \vee(\wedge(A(x), C(x,y)), \exists x\ \forall z\ \neg\wedge(P(x,z), R(z)))$$

$$\forall x\ \forall y \vee(\wedge(A(x), C(x,y)), \exists x\ \forall z \vee(\underline{\neg P(x,z)}, \underline{\neg R(z)}))$$

3) Standardize Variables

Rename variables so that each quantifier binds a unique variable

Ex.

$$\underline{\forall x}\ \forall y \vee(\wedge(A(x), C(x,y)), \underline{\exists x}\ \forall z \vee(\neg P(x,z), \neg R(z)))$$

(this x is in the scope of the other one, rename it)

$$\forall x\ \forall y \vee(\wedge(A(x), C(x,y)), \underline{\exists u}\ \forall z \vee(\neg P(\underline{u},z), \neg R(z)))$$

4. Eliminate existential qualifiers by introducing *Skolem functions.*

Ex. $\forall x \forall y \, \exists z \, P(x, y, z)$

Want to eliminate $\exists z$.

Variable z depends on x and y, since $\exists z$ is within the scope of $\forall x \forall y$, so we can consider z a function of x and y.

Choose an arbitrary unused function name f and replace z by f(x, y) eliminating the $\exists$.

$$\forall x \forall y \, P(x, y, f(x, y))$$

Interpretation:

f(x, y) specifies for any x, y a value of z that exists and satisfies P(x, y, z)

$\underline{\forall x \, \forall y} \vee(\wedge(A(x), C(x,y)), \underline{\exists u} \, \forall z \vee(\neg P(u,z), \neg R(z))$

$\forall x \, \forall y \vee(\wedge(A(x), C(x,y)), \forall z \vee(\neg P(g(x, y),z), \neg R(z))$

Note: now we can move the $\forall z$ forward.

$\forall x \, \forall y \, \forall z \vee(\wedge(A(x), C(x,y)), \vee(\neg P(g(x, y),z), \neg R(z)))$

5. Rewrite the result in Conjunctive Normal Form.

Conjunctive Normal Form

$\wedge(x_1, \ldots, x_n)$ where the x_i are:

- atomic formulas
- negated atomic formulas
- disjunctions

Do this by repeatedly applying the rule:

$$\vee(x_1, \wedge(x_2, \ldots, x_n) =$$
$$\wedge(\vee(x_1, x_2), \ldots, \vee(x_1, x_n)$$

Example: $\forall x \forall y \forall z \vee(\wedge(A(x), C(x,y)), \vee(\neg P(g(x, y),z), \neg R(z)))$

To see the transformation, think of this as

$$A\ C \vee \neg P \vee \neg R$$

$$= (A\ C \vee \neg P) \vee \neg R$$

$$= (A \vee \neg P)(C \vee \neg P) \vee \neg R$$

$$= (A \vee \neg P \vee \neg R)(C \vee \neg P \vee \neg R)$$

$$\forall x\ \forall y\ \forall z \wedge(\ \vee(A(x), \neg P(g(x, y),z), \neg R(z)),$$
$$\vee(C(x,y)), \neg P(g(x, y),z), \neg R(z)))$$

6. Since all variables are now universally quantified, eliminate $\forall$ as understood.

$\wedge(\ \vee(A(x), \neg P(g(x, y),z), \neg R(z)), \vee(C(x,y)), \neg P(g(x, y),z), \neg R(z)))$

The input formula(s) are now expressed in a kind of normal form call the *clause form equivalent* of the original expression

Def (clause form equivalent)

- a *literal* is an atom or the negation of an atom.
- a *clause* is a *disjunction*of literals
- a *formula* is a *conjunction* of clauses

We can think of the clause form equivalent as a *set* of clauses, and each clause as a *set* of literals.

$$\{\text{Clause 1. } \overbrace{\{A(x), \neg P(g(x,y),z), \neg R(z)\}}^{\text{Implicit disjunction}},$$

$$\text{Clause 2. } \{C(x, y), \neg P(g(x, y), z), \neg R(z)\}\}$$

The formula is the set consisting of Clause 1 and Clause 2, with implicit conjunction.

Steps in Proving a Conjecture

I. Find the clause form equivalent C of

$F' = F \cup \neg S$ (F is the axiom, $\neg S$ the conjecture)

II. Try to find the new clauses that are logically implied by C.

If NIL is one of the clauses, then F′ is unsatisfiable and S is proved.

Resolution Procedure for Propositional Logic

1) Convert F to clause form.

2) Negate S, convert to clause form, and add in the clause form of F to get a set of clauses.

3) Repeat until a contradiction or no progress.

   a) select two "parent" clauses.

   b) produce their *resolvent.*

   Let $C_1 = L_1 \vee L_2 \vee \ldots \vee L_n$

   $C_2 = L_1' \vee L_2' \vee \ldots \vee L_n'$

   If C_1 has a literal L and C_2 has a literal ¬L

   Then resolvent(C_1, C_2) =

   $L_1 \vee L_2 \vee \ldots \vee L_n \vee L_1' \vee L_2' \vee \ldots \vee L_n'$

   with L and ¬L removed

   else resolvent(C_1, C_2) =

   $L_1 \vee L_2 \vee \ldots \vee L_n \vee L_1' \vee L_2' \vee \ldots \vee L_n'$

   with nothing removed

   c) if resolvent = NIL we are done; else add it to the set.

Propositional Logic Example

F: $P \vee Q,\ P \rightarrow Q,\ Q \rightarrow R$

S: R

Clause form of $F \cup \neg S$

$$\{P \vee Q,\ \neg P \vee Q,\ \neg Q \vee R,\ \neg R\}$$

① ② ③ ④

① & ② ⇒ Q ⑤

③ & ④ ⇒ $\neg Q$ ⑥

⑤ & ⑥ ⇒ NIL

done

In propositional logic, we just look for some literal L in C_1 and its negation ¬L in C_2

To find resolvents in predicate logic, we need a *matching procedure* that compares 2 literals and determines whether there is a set of *substitutions* that makes them identical. This procedure is called *unification.*

Example:

$C_1 =$ eats(Tom, x)

$C_2 =$ eats(Tom, ice cream)

Substituting "ice cream" for variable x in C_1 gives

$C_1' =$ eats(Tom, ice cream) $\equiv C_2$

The subsitituion is ice cream/x

Proof by Contradiction using Unification

Given: $\forall x\ P(x) \rightarrow R(x)$ $\quad \{\neg P(x),\ R(x)\}$

$\forall x\ R(x) \rightarrow Q(x)$ $\quad \{\neg R(z),\ Q(z)\}$

Prove: $\forall x\ P(x) \rightarrow Q(x)$

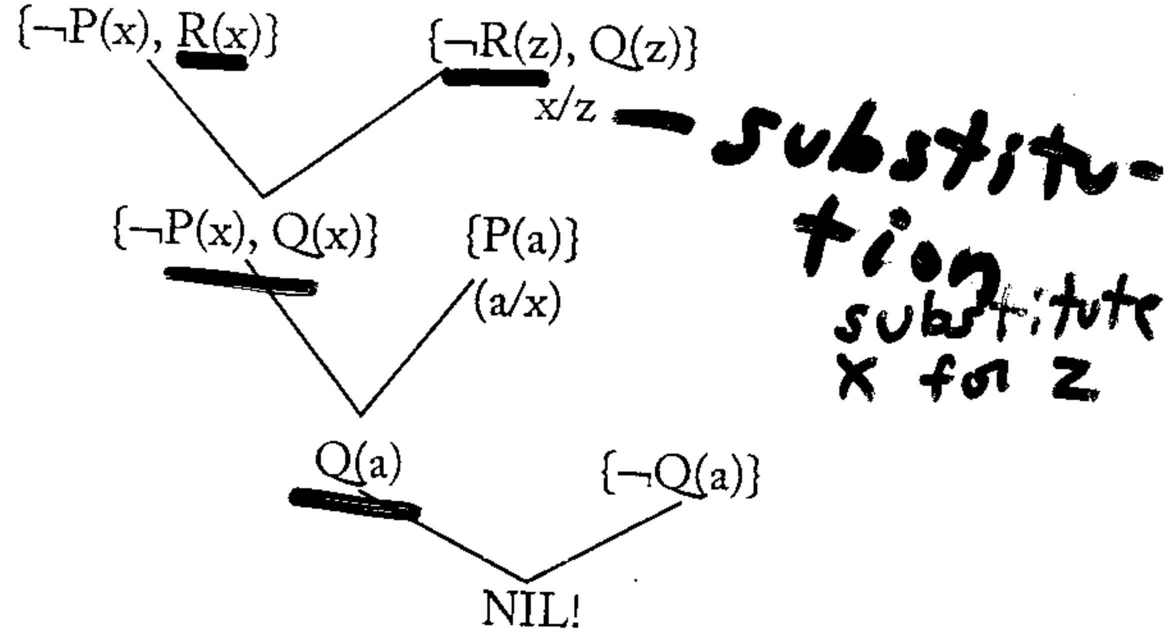
Negation: $\neg \forall x\ P(x) \rightarrow Q(x)$

$\exists x\ \neg(P(x) \rightarrow Q(x))$

$\exists x\ \neg(\neg P(x) \vee Q(x))$

$\exists x\ P(x) \wedge \neg Q(x))$

$P(a) \wedge \neg Q(a)$ $\quad \{P(a)\}$ $\quad \{\neg Q(a)\}$



Given C_1 and C_2, the computer tries to find all possible resolvents.

If one resolvent is NIL, then C_1 and C_2 cannot together be satisfied.

Ex.

$$\left.\begin{array}{l} C_1 = \{P(x)\} \\ C_2 = \{\neg P(a)\} \end{array}\right\rangle \text{ trivially unifiable } \lambda = (a,x) \qquad a/x$$

C_3 = NIL

i.e. $\forall x\, P(x)$ and $\neg P(a)$ are inconsistent

Binary Resolution Procedure Restated

0. S = axioms $\cup$ $\neg$theorem

1. Let S be a set of clauses

   $S = \{C_1, C_2, \ldots, C_n\}$. $R_1(S) = S$. $i = 1$.

2. Apply the resolution process to each pair C_i, C_j, $i \neq j$ in $R_i(S)$.

3. Place any resolvents in RES

   $R_{i+1}(S) = R_i(S) \cup$ RES

   $i = i+1$

   go to 2

If NIL is ever implied, STOP and succeed. The proof is the refutation graph leading from NIL through its ancestors, up to the original S. If we run out of time, STOP and say

NO PROOF FOUND

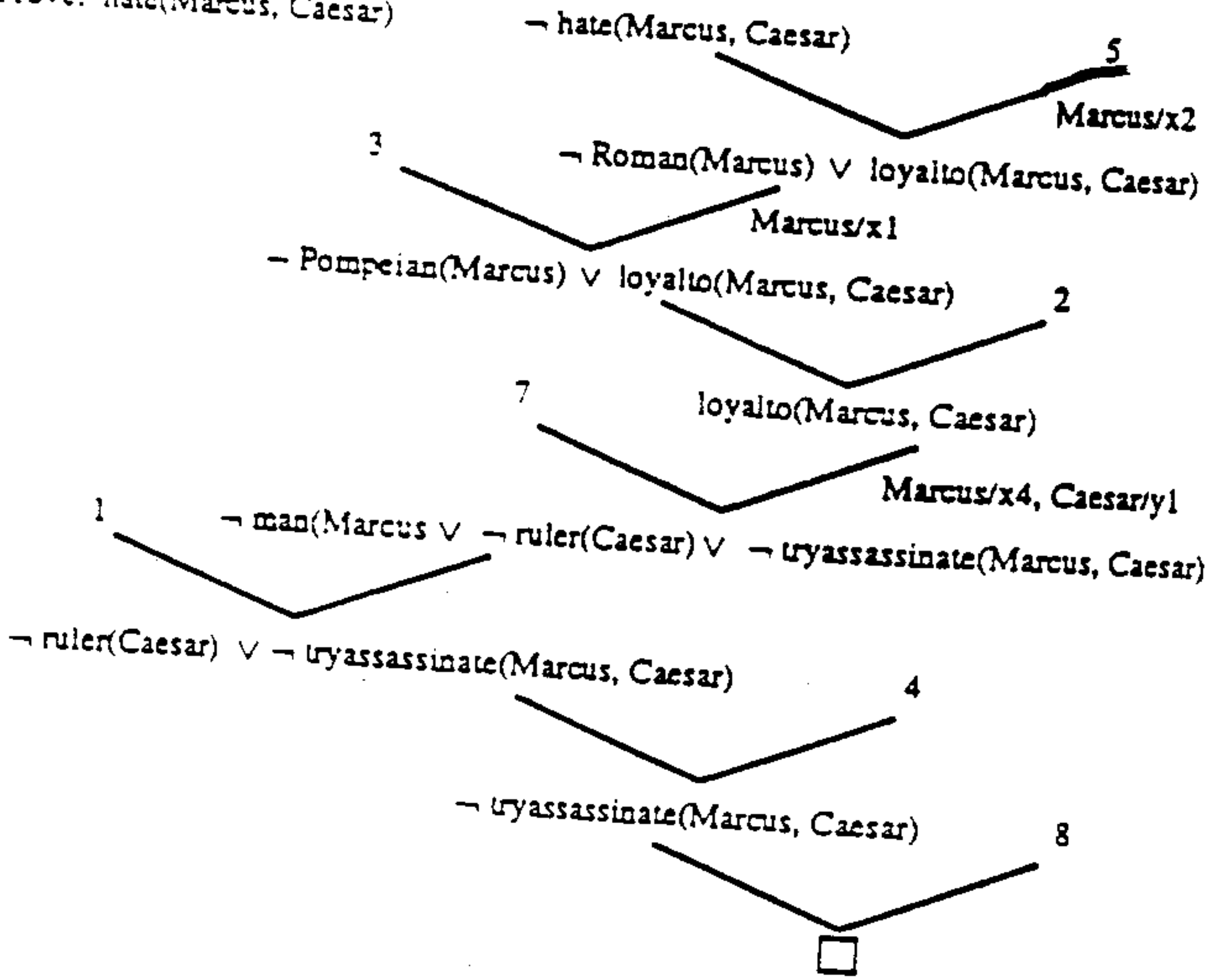
Example

Axioms in clause form:

1. man(Marcus)
2. Pompeian(Marcus)
3. ¬ Pompeian(x1) ∨ Roman(x1)
4. ruler(Caesar)
5. ¬ Roman(x2) ∨ loyalto(x2, Caesar) ∨ hate(x2, Caesar)
6. loyalto(x3, f1(x3))
7. ¬ man(x4) ∨ ¬ ruler(y1) ∨ ¬ tryassassinate(x4, y1) ∨ ¬ loyalto(x4, y1)
8. tryassassinate(Marcus, Caesar)

(a)

Prove: hate(Marcus, Caesar)



The Monkey-Bananas Problem (Simplified)

Axioms

1) $\forall x \, \forall s\{\neg ONBOX(s) \rightarrow AT(box, x, pushbox(x,s))\}$

   For each position x and state s, if the monkey isn't on the box in state s, then the box will be pushed to position x and the new state is pushbox(x,s).

2) $\forall s\{ONBOX(climbbox(s))\}$

   For all states s, the monkey will be on the box in the state achieved by applying climbbox to s.

3) $\forall s\{ONBOX(s) \wedge AT(box, c, s) \rightarrow HB(grasp(s))\}$

   For all states s, if the monkey is on the box and the box is at position c in state s, then HB is true of the state attained by applying grasp to s.

4) $\forall x \forall s\{AT(box, x, s) \rightarrow AT(box, x, climbbox(s))\}$

   The position of the box does not change when the monkey climbs on it, but the state does.

5) $\neg ONBOX(s_o)$

Conjecture Negation

$\exists s\ HB(s)$ $\forall s \neg HB(s)$ or $\neg HB(s)$

Refutation Graph

(ax. 3)

$\neg HB(t)$ $\{\neg ONBOX(s), \neg AT(box, c, s), HB(grasp(s))\}$

grasp(s)|t

$\{\neg ONBOX(s), \neg AT(box, c, s)\}$

(ax. 2)

$\{ONBOX(climbbox(s))\}$

climbbox(s)|s

$\{\neg AT(box, c, climbbox(s))\}$

(ax. 4)

$\{\neg AT(box, x, s), AT(box, x, climbbox(s))\}$

c|x

$\{\neg AT(box, c, s)\}$

(ax. 1)

$\{ONBOX(s), AT(box, x, pushbox(x, s))\}$

c|x

pushbox(c, s)|s

(ax. 5)

$ONBOX(s)$ $\neg ONBOX(s_o)$

NIL

If we change the conjecture to $\{\neg HB(s), HB(s)\}$, the result becomes

$HB(grasp(climbbox(pushbox(c,s_o))))$