CSCI 3230

Fundamentals of Artificial Intelligence

Chapter 9, Chapter 10, Sect 2, 3

INFERENCE IN FIRST-ORDER LOGIC

Outline

- Reducing first-order inference to propositional inference
- Unification
- Generalized Modus Ponens
- Forward and backward chaining
- Logic programming
- Resolution

A brief history of reasoning

450B.C.	Stoics	Propositional logic, inference (may be)
322B.C.	Aristotle	“syllogisms” (inference rules), quantifier
1565	Cardano	probability theory (propositional logic + uncertainty)
1847	Boole	propositional logic (again)
1849	Frege freg-”ga”	First-Order Logic
1922	Wittgenstein	Proof by truth table
1930	Gödel Girl-del	$\exists$ complete algorithm for FOL
1930	Herbrand	complete algorithm for FOL (reduce to propositional)
1931	Gödel	$\neg\exists$ complete algorithm for arithmetic
1960	Davis/Putnam	“practical” algorithm for propositional logic
1965	Robinson	“practical” algorithm for FOL – resolution, unification

Universal instantiation (UI)

function

- SUBST(θ, α) denotes the result of applying the substitution (or binding list) θ to the sentence α. E.g.:

  SUBST({ x/ Sam, y/ Pam}, Likes(x, y)) = Likes(Sam, Pam)

- Every instantiation of a universally quantified sentence is entailed by it:

$$\frac{\forall v \quad \alpha}{SUBST(\{v/g\},\alpha)}$$

  for any sentence α, any variable v and ground term g (no variable) *(Lemma)*

- E.g. ∀x King(x) ∧ Greedy (x) ⇒ Evil (x) yields

  King(John) ∧ Greedy (John) ⇒ Evil (John)
  King(Richard) ∧ Greedy (Richard) ⇒ Evil (Richard)
  King(Father (John)) ∧ Greedy (Father (John)) ⇒ Evil (Father (John))
  …

UI & EI: background for FOL inference and its relationship with propositional logic

Existential instantiation (EI)

- For any sentence α, variable v, and constant symbol k

  do not overload a constant name

  that does not appear elsewhere in the knowledge base:

$$\frac{\exists v \quad \alpha}{SUBST(\{v/k\}, \alpha)}$$

- E.g. ∃x Crown(x) ∧ OnHead(x, John) yields

  Crown(C_1) ∧ OnHead(C_1, John)

  provided C_1 is a new constant symbol, called a Skolem constant

- Another example: from $\exists x\ d(x^y)/dy = x^y$, we obtain

  $d(e^y)/\mathrm{d}y = e^y$

  provided e is a new constant symbol (natural log base=2.71828)

Logical Equivalence (UI & EI)

- UI can be applied several times to add new sentences;
  the new KB is logically equivalent to the old.

- EI can be applied once to replace the existential sentence;
  the new KB is *not* equivalent to the old,
  but is satisfiable iff the old KB was satisfiable
  at least one case is true

Reduction to propositional inference

- Suppose the KB contains just the following:

  $\forall x\ King(x) \land greedy(x) \Rightarrow Evil(x)$
  $King(John)$
  $Greedy(John)$
  $Brother(Richard, John)$

- Instantiating the universal sentence in *all possible* ways:

  $King(John) \land Greedy(John) \Rightarrow Evil(John)$
  $King(Richard) \land Greedy(Richard) \Rightarrow Evil(Richard)$
  $King(John)$
  $Greedy(John)$
  $Brother(Richard, John)$

- The new KB is propositionalized: proposition symbols are
  King(John), Greedy(John), Evil(John), King(Richard) etc.

Reduction to propositional inference

Claim: a ground sentence* is entailed by new KB iff entailed by original KB

Claim: every FOL KB can be propositionalized so as to preserve entailment

Idea: propositionalize KB and query, apply resolution, return result

Problem: with function symbols, there are infinitely many ground terms,

e.g. *Father(Father(Father(John)))*

Theorem: Herbrand (1930). If a sentence α is entailed by an FOL KB, it is entailed by a *finite* subset of the propositional KB

Idea: For n = 0 to ∞ do
create a propositional KB by instantiating with depth-n terms
see if α is entailed by this KB

Problem: works if α is entailed, loops if α is not entailed

Theorem: Turing (1936), Church (1936), entailment in FOL is semidecidable (algorithm can say yes to every entailed sentence, cannot say no to every non-entailed sentence) ??

Problems with propositionalization

- Propositionalization generates lots of irrelevant sentences.

- E.g. From

  $\forall x\ King(x) \wedge greedy(x) \Rightarrow Evil(x)$

  $King(John)$

  $\forall y\ Greedy(y)$

  $Brother(Richard, John)$

  it seems obvious that Evil(John), but propositionalization produces lots of irrelevant facts such as Greedy(Richard).

- With p k–ary predicates and n constants, there are $p \cdot n^k$ instantiations! k: max # of arguments (var) in a predicate.

??So.??

$King(x) \wedge greedy(x) \Rightarrow Evil(x)$

$King(John)$
$\forall y\ Greedy(y)$

Unification

We can get the inference immediately if we can find a substitution θ such that King(x) and Greedy(x) match King(John) and Greedy(y)

θ = {x/John, y/John} works

UNIFY(p, q) = θ where SUBST(θ, p) = SUBST(θ, q)

p	q	θ
Knows(John, x)	Knows(John, Jane)	{x/ Jane}
Knows(John, x)	Knows(y, OJ)	{x/ OJ, y/ John}
Knows(John, x)	Knows(y, Mother(y))	{y/ John, x/ Mother(John)} recursive
Knows(John, x)	Knows(x, OJ)	fail

Standardizing apart eliminates overlap of variables, e.g. Knows(z_{17}, OJ)

Generalized Modus Ponens (GMP)

For atomic sentences p_i, p_i', and q, where there is a substitution θ such that $SUBST(\theta, p_i') = SUBST(\theta, p_i)$, for all i:

$$\frac{p_1', p_2', \ldots, p_n', (p_1 \wedge p_2 \wedge \ldots p_n \Rightarrow q)}{SUBST(\theta, q)}$$

p_1' is $King(John)$ — p_1 is $King(x)$

p_2' is $Greedy(y)$ — p_2 is $Greedy(x)$

q is $Evil(x)$

θ is $\{x/John, y/John\}$

$SUBST(\theta, q)$ is $Evil(John)$

$\forall x\ King(x) \wedge greedy(x) \Rightarrow Evil(x)$

$King(John)$

$\forall y\ Greedy(y)$

$Brother(Richard, John)$

(p.9)

facts & rules of atomic sentence

GMP used with KB of definite clauses (exactly one positive literal)

All variables assumed universally quantified. (cf. Horn clauses)

Soundness of GMP

Need to show that

$p_1', \ldots, p_n', (p_1 \wedge \ldots \wedge p_n \Rightarrow q) \vDash q\theta$

provided that $p_i'\theta = p_i\theta$ for all i

Lemma: For any definite clause p, we have $p \vDash p\theta$ by UI

1. $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \vDash (p_1 \wedge \ldots \wedge p_n \Rightarrow q)\theta = (p_1\theta \wedge \ldots \wedge p_n\theta \Rightarrow q\theta)$
2. $p_1', \ldots, p_n' \vDash (p_1' \wedge \ldots \wedge p_n')\,\theta \vDash p_1'\theta \wedge \ldots \wedge p_n'\theta$
3. From 1 and 2, $q\theta$ follows by ordinary Modus Ponens

$q\theta : SUBST(\theta, q)$ substituted conclusion

Example knowledge base

The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American. (陸軍上校)

Prove that Col. West is a criminal

Example knowledge base (definite clauses)

... it is a crime for an American to sell weapons to hostile nations:

American(x) ∧ Weapon(y) ∧ Sells(x, y, z) ∧ Hostile(z) ⇒ Criminal(x)

Nono... has some missiles, i.e., ∃x Owns(Nono, x) ∧ Missile(x):

Owns(Nono, M1) and Missile(M1)

... all of its missiles were sold to it by Colonel West

Missile(x) ∧ Owns(Nono, x) ⇒ Sells(West, x, Nono)

Missiles are weapons: (common sense)

Missile(x) ⇒ Weapon(x)

An enemy of America counts as "hostile": (common sense)

Enemy(x, America) ⇒ Hostile(x)

West, who is American…

American(West)

The country Nono, an enemy of America…

Enemy(Nono, America)   4 rules & 4 facts

Forward chaining algorithm

```
function FOL-FC-Ask(KB, α) returns a substitution or false
   inputs: KB, a knowledge base, a set of first-order definite clauses
           α, the query, an atomic sentence
   local variables: new, the new sentences inferred on each iteration  //new facts inferred

   repeat until new is empty
      new ← {}
      for each sentence r in KB do
         (p1 ∧ ... ∧ pn ⇒ q) ← Standardize-Apart(r)
         for each θ such that (p1 ∧ ... ∧ pn)θ = (p1' ∧ ... ∧ pn')θ  //matched
              for some p1', ... pn' in KB //facts
            q' ← SUBST(θ, q) //a new fact (fire)
            if q' is not a renaming of a sentence already in KB or new then do
               add q' to new
               φ ← Unify(q', α)  //query to be matched with new fact
               if φ is not fail then return φ
      add new to KB
   return false
```

Forward chaining proof

American(x) ∧ Weapon(y) ∧ Sells(x, y, z) ∧ Hostile(z) ⇒ Criminal(x)

Missile(x) ∧ Owns(Nono, x) ⇒ Sells(West, x, Nono) y

Missile(x) ⇒ Weapon(x) y

Enemy(x, America) ⇒ Hostile(x) z

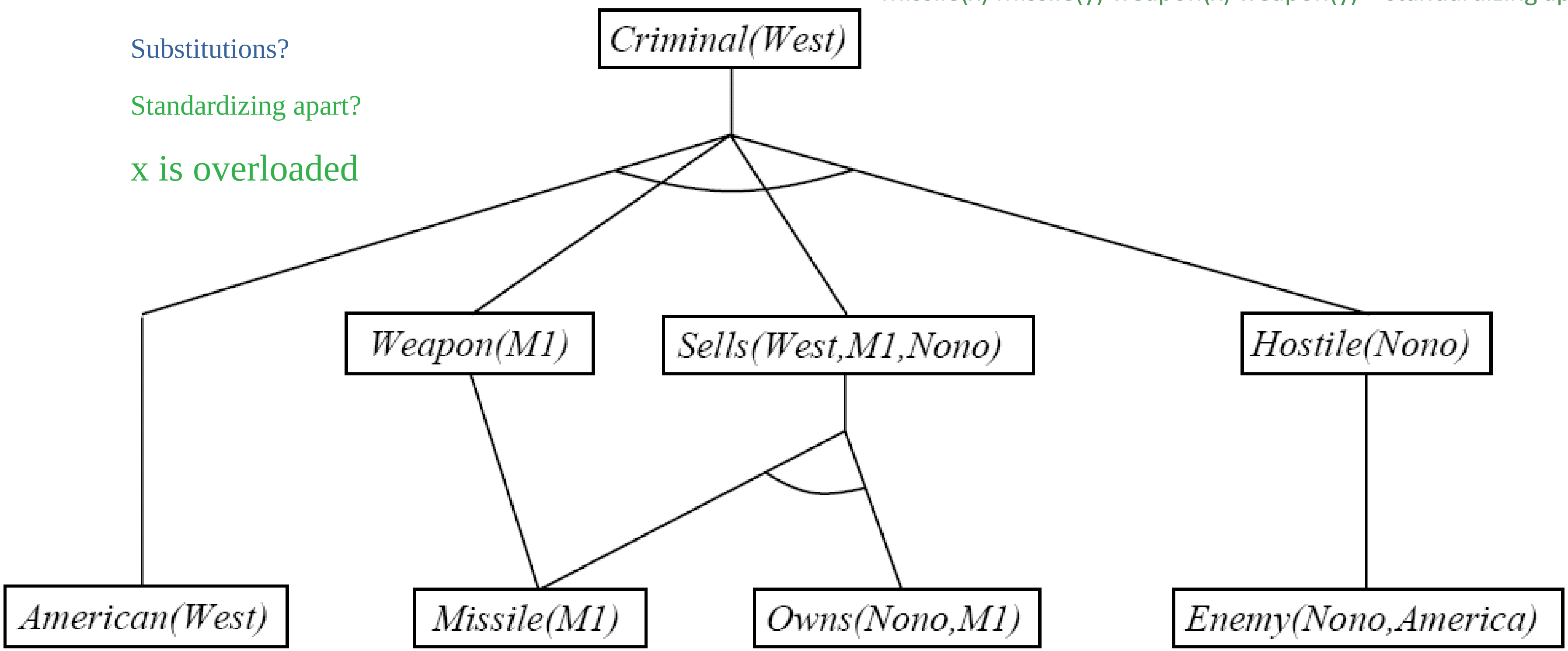
American(West) Missile(M1)
Owns(Nono, M1) Enemy(Nono, America)

Missile(x) Missile(y) weapon(x) weapon(y) – standardizing apart;

Substitutions?

Standardizing apart?

x is overloaded



Θ={x/West, y/M1, z/Nono}

Properties of forward chaining

Sound and complete for first-order definite clauses
(proof similar to propositional proof)

(recursive)

- Datalog KB = first-order definite clauses + no functions (e.g. crime KB, p14)
- FC terminates for Datalog in poly iterations: at most $p \bullet n^k$ literals
  (p k-ary predicates and n constants) k arguments
- Datalog: a query & rule language for deductive databases (syntactically a subset of Prolog) – Database-logic

- May not terminate in general if α is not entailed
- This is unavoidable: entailment with definite clauses is semidecidable (p.8)

Efficiency of forward chaining

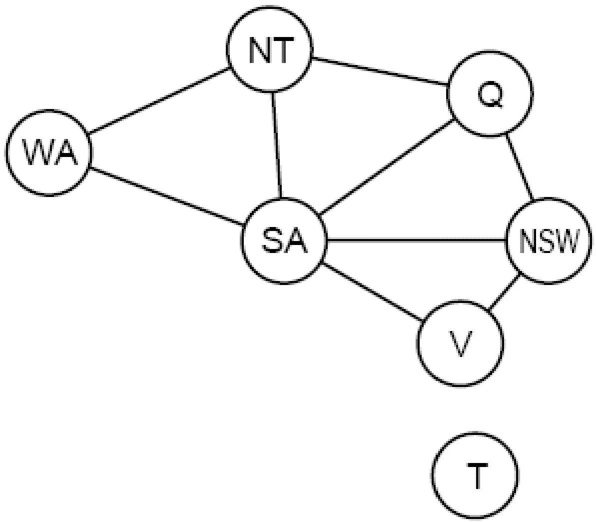
- Simple observation: no need to match a rule on iteration k

  if a premise wasn't added on iteration k-1

  ⇒ match each rule whose premise contains a newly added literal

  unification
- Matching itself can be expensive
- Database indexing allows O(1) retrieval of known facts
  e.g., query Missile(x) retrieves Missile(M)
- Matching conjunctive premises against known facts is NP-hard
- Forward chaining is widely used in deductive databases

Hard matching example



$$Diff(wa, nt) \wedge Diff(wa, sa) \wedge$$
$$Diff(nt, q) \wedge Diff(nt, sa) \wedge$$
$$Diff(q, nsw) \wedge Diff(q, sa) \wedge$$
$$Diff(nsw, v) \wedge Diff(nsw, sa) \wedge$$
$$Diff(v, sa) \Rightarrow Colorable()$$

$$Diff(Red, Blue) \quad Diff(Red, Green)$$
$$Diff(Green, Red) \quad Diff(Green, Blue)$$
$$Diff(Blue, Red) \quad Diff(Blue, Green)$$

Colorable() is inferred iff the CSP has a solution (?any)

CSPs include 3SAT as a special case, hence matching is NP-hard

3-satisfiability problems

e.g. 3-coloring

Backward chaining algorithm

function FOL-BC-Ask (*KB*, *goals*, θ) **returns** a set of substitutions

inputs: *KB*, a knowledge base

goals, list of conjuncts forming a query //goal stack

θ, the current substitution, initially the empty substitution {}

local variable: *ans*, a set of substitutions, initially empty

if *goals* is empty **then return** {θ}

$q' \leftarrow$ SUBST(θ, First(*goals*)) //pop from goal stack

for each *r* in *KB* where Standard-Apart(*r*) = $(p_1 \land \ldots \land p_n \Rightarrow q)$

and $\theta' \leftarrow$ Unify(q, q') succeeds //(sub)-goal matches then-part or fact

ans ← FOL-BC-Ask(*KB*, $[p_1, \ldots, p_n |$ Rest(*goals*)], Compose(θ', θ)) ∪ *ans*

//Depth first recursive call push

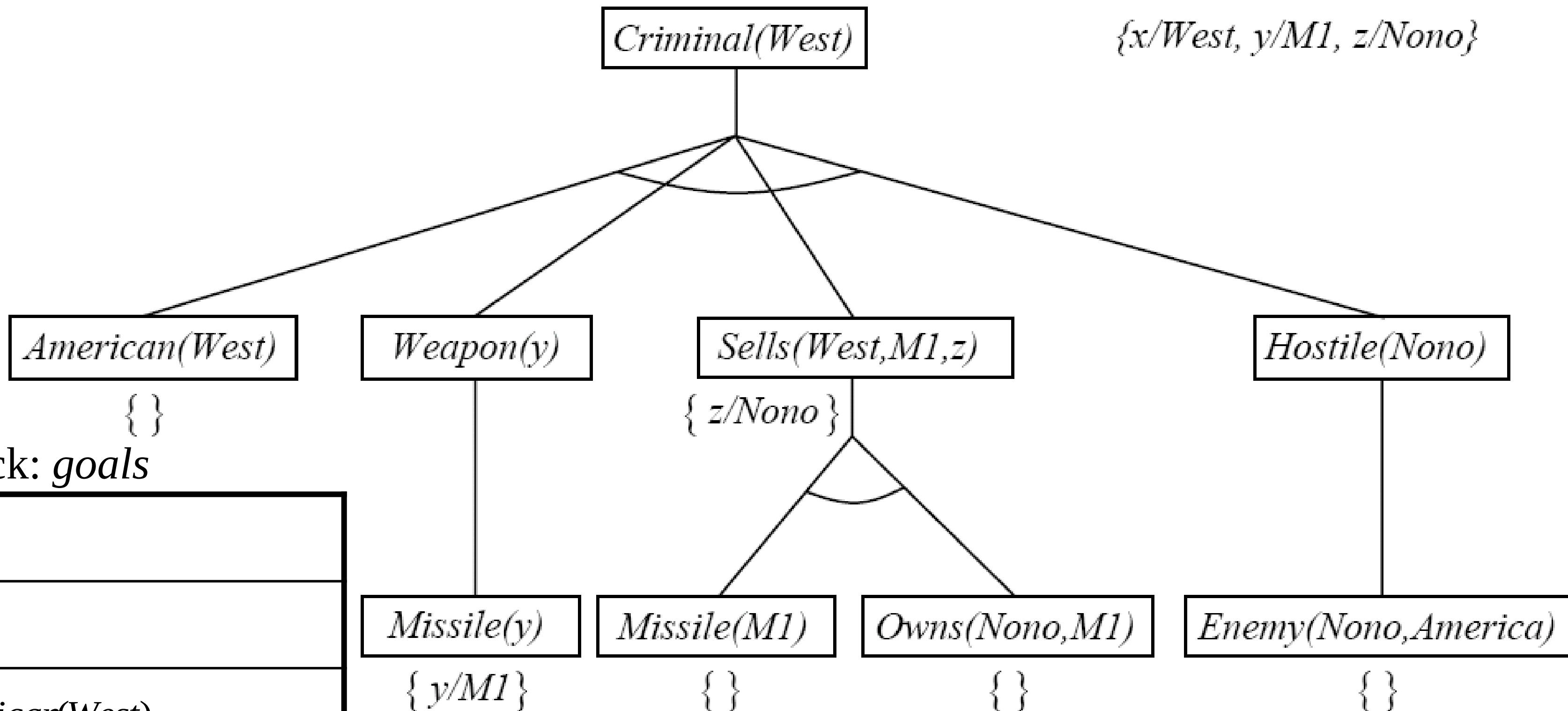
return *ans*

Composition of substitutions, apply each substitution in turn:

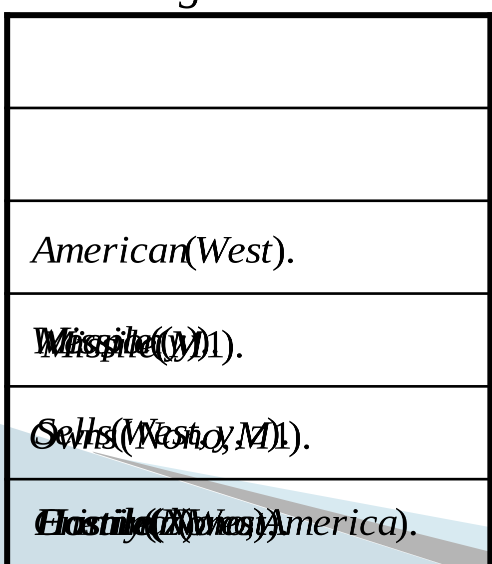
$$\text{SUBST}(\text{Compose}(\theta_1, \theta_2), p) = \text{SUBST}(\theta_2, \text{SUBST}(\theta_1, p))$$

r: sentences in KB including facts and rules; q is the head of a rule or simply a fact;

Backward chaining example



Stack: *goals*



American(x) ∧ Weapon(y) ∧ Sells(x, y, z) ∧ Hostile(z) ⇒ Criminal(x)

Missile(x) ∧ Owns(Nono, x) ⇒ Sells(West, x, Nono)

Missile(x) ⇒ Weapon(x)

Enemy(x, America) ⇒ Hostile(x)

American(West) | Missile(M1) | Owns(Nono, M1) | Enemy(Nono, America)

Properties of backward chaining

- Depth-first recursive proof search: space is linear in size of proof

- Incomplete due to infinite loops
$\Rightarrow$fix by checking current goal against every goal on stack

- Inefficient due to repeated sub-goals (both success and failure)
$\Rightarrow$fix using caching of previous results (extra space!)

- Widely used (without improvements!) for logic programming

Logic programming

Soundbite: computation as inference on logical KBs

	Logic programming	Ordinary programming
1	Identify problem	Identify problem
2	Assemble information	Assemble information
3	Tea break	Figure out solution (method)
4	Encode information in KB	Program solution
5	Encode problem instance as <u>facts</u>	Encode problem instance as <u>data</u>
6	Ask queries	Apply program to data
7	Find false facts and rules	Debug procedural errors

Should be easier to debug *Capital(New York, US)* than $x := x + 2$!

Prolog systems

definite clauses

- Basis: backward chaining with Horn clauses + Control
  Widely used in Europe, Japan (basis of 5th Generation project 80's)
  Compilation techniques ⇒ 60 million LIPS
- Program = set of clauses = head :- $\text{literal}, \ldots \text{literal}_n$.
  criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z)
- Depth-first, left-to-right backward chaining.
- Built-in predicates for arithmetic etc., e.g., X is Y*Z+3
  Does not solve: equation "5 is X+Y" fails
- Closed-world assumption ("negation as failure")

  database is complete | if a fact is not in the database, then it is not true

  e.g. given alive(X) :- not dead (X).
  alive(Joe) succeeds if dead(Joe) fails

Prolog examples

- Appending two lists to produce a third:

  append([ ], Y, Y).

  append([X| L], Y, [X| Z]) :- append(L,Y,Z).

  (| : adjoins)

- query: append(A, B, [1,2]) ?
  (What 2 lists can be appended to give [1,2]?

  Answers: A = [ ]   B = [1, 2]
  A = [1]   B = [2]
  A = [1,2]   B = [ ]

Resolution: brief summary

- Full first-order version:

$$\frac{l_1 \vee \ldots \vee l_k, \quad m_1 \vee \ldots \vee m_n}{(l_1 \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n)\theta}$$

where Unify$(l_i, \neg m_j) = \theta$

- For example,

$$\frac{\begin{array}{l}\neg Rich(x) \vee Unhappy(x) \\ Rich(Ken)\end{array}}{Unhappy(Ken)}$$

with $\theta = \{x/Ken\}$

- Apply resolution steps to $CNF(KB \wedge \neg\alpha)$; complete for FOL

CNF: conjunctive normal form

Conversion to CNF (Conjunctive Normal Form)

Everyone who loves all animal is loved by someone:

$\forall x[\forall y\ Animal(y) \Rightarrow Loves(x, y)] \Rightarrow [\exists y\ Loves(y, x)]$

1. Eliminate biconditional and implications $(\alpha \Rightarrow \beta) \equiv (\neg \alpha \vee \beta)$

   $\forall x[\neg\forall y\ \neg Animal(y) \vee Loves(x, y)] \vee [\exists y\ Loves(y, x)]$

2. Move $\neg$ inwards: $\underline{\neg \forall x, p} \equiv \underline{\exists x\ \neg p}$; $\underline{\neg\exists x, p \equiv \forall x\ \neg p}$:

   $\forall x\ [\exists y\ \neg(\neg Animal(y) \vee Loves(x, y))] \vee [\exists y\ Loves(y, x)]$

   $\forall x\ [\exists y\ \neg\neg Animal(y) \wedge \neg Loves(x, y)] \vee [\exists y\ Loves(y, x)]$

   $\forall x\ [\exists y\ Animal(y) \wedge \neg Loves(x, y)] \vee [\exists y\ Loves(y, x)]$

3. Standardize variables: each quantifier should use a difference one

   $\forall x\ [\exists y\ Animal(y) \wedge \neg Loves(x, y)] \vee [\exists \underline{z}\ Loves(\underline{z}, x)]$

Conversion to CNF

4. Skolemize: a more general form of existential instantiation. Each existential variable is replaced by a Skolem function of the enclosing universally quantified variables:
   $\forall x \, [Animal(F(x)) \land \neg Loves(x, F(x))] \lor Loves(G(x), x)$
5. Drop universal quantifiers:
   $[Animal(F(x)) \land \neg Loves(x, F(x))] \lor Loves(G(x), x)$
6. Distribute ∧ over ∨
   $[Animal(F(x)) \lor Loves(G(x), x)]$
   $\land \, [\neg Loves(x, F(x)) \lor Loves(G(x), x)]$

Resolution proof: definite clauses