Inference in first-order logic

Chapter 9
Outline

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

Chapter 9   2
<table>
<thead>
<tr>
<th>Year</th>
<th>Person</th>
<th>Contribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>450 B.C.</td>
<td>Stoics</td>
<td>propositional logic, inference (maybe)</td>
</tr>
<tr>
<td>322 B.C.</td>
<td>Aristotle</td>
<td>“syllogisms” (inference rules), quantifiers</td>
</tr>
<tr>
<td>1565</td>
<td>Cardano</td>
<td>probability theory (propositional logic + uncertainty)</td>
</tr>
<tr>
<td>1847</td>
<td>Boole</td>
<td>propositional logic (again)</td>
</tr>
<tr>
<td>1879</td>
<td>Frege</td>
<td>first-order logic</td>
</tr>
<tr>
<td>1922</td>
<td>Wittgenstein</td>
<td>proof by truth tables</td>
</tr>
<tr>
<td>1930</td>
<td>Gödel</td>
<td>( \exists ) complete algorithm for FOL</td>
</tr>
<tr>
<td>1930</td>
<td>Herbrand</td>
<td>complete algorithm for FOL (reduce to propositional)</td>
</tr>
<tr>
<td>1931</td>
<td>Gödel</td>
<td>( \neg \exists ) complete algorithm for arithmetic</td>
</tr>
<tr>
<td>1960</td>
<td>Davis/Putnam</td>
<td>“practical” algorithm for propositional logic</td>
</tr>
<tr>
<td>1965</td>
<td>Robinson</td>
<td>“practical” algorithm for FOL—resolution</td>
</tr>
</tbody>
</table>
Universal instantiation (UI)

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

$$\forall v \; \alpha \quad \text{SUBST}\{\{v/g\}, \alpha\}$$

for any variable $v$ and ground term $g$

E.g., $\forall x \; King(x) \land Greedy(x) \Rightarrow Evil(x)$ yields

- $King(John) \land Greedy(John) \Rightarrow Evil(John)$
- $King(Richard) \land Greedy(Richard) \Rightarrow Evil(Richard)$
- $King(Father(John)) \land Greedy(Father(John)) \Rightarrow Evil(Father(John))$
**Existential instantiation (EI)**

For any sentence $\alpha$, variable $v$, and constant symbol $k$ that does not appear elsewhere in the knowledge base:

$$\exists v \alpha \quad \text{SUBST}\{\{v/k\}, \alpha\}$$

E.g., $\exists x \ Crown(x) \land OnHead(x, John)$ yields

$$Crown(C_1) \land OnHead(C_1, John)$$

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

$$d(e^y)/dy = e^y$$

provided $e$ is a new constant symbol
Existential instantiation contd.

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.

Can we apply EI twice, with two different Skolem constants??
Existential instantiation contd.

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.

Can we apply EI twice, with two different Skolem constants??

Yes! $\text{Crown}(C_2) \land \text{OnHead}(C_2, \text{John})$ is OK too because $C_1$ and $C_2$ could be the same crown.
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, we have

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

The new KB is propositionalized: proposition symbols are

\[ King(John), Greedy(John), Evil(John), King(Richard) \text{ etc.} \]
Reduction contd.

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 $\alpha$ is entailed by an FOL KB, it is entailed by a finite subset of the propositional KB

Idea: For $n = 0$ to $\infty$ do
create a propositional KB by instantiating with depth-$n$ terms
see if $\alpha$ is entailed by this KB

Problem: works if $\alpha$ is entailed, loops if $\alpha$ is not entailed

Theorem: Turing (1936), Church (1936), entailment in FOL is semidecidable
Problems with propositionalization

Propositionalization seems to generate lots of irrelevant sentences. E.g., from

\[ \forall x \ King(x) \land Greedy(x) \Rightarrow Evil(x) \]
\[ King(John) \]
\[ \forall y \ Greedy(y) \]
\[ Brother(Richard, John) \]

it seems obvious that \textit{Evil(John)}, but propositionalization produces lots of facts such as \textit{Greedy(Richard)} that are irrelevant

With \( p \) \( k \)-ary predicates and \( n \) constants, there are \( p \cdot n^k \) instantiations

With function symbols, it gets much much worse!
We can get the inference immediately if we can find a substitution \( \theta \) such that \( \text{King}(x) \) and \( \text{Greedy}(x) \) match \( \text{King}(\text{John}) \) and \( \text{Greedy}(y) \)

\[ \theta = \{x/\text{John}, y/\text{John}\} \text{ works} \]

**Unify** \((\alpha, \beta) = \theta \) if \( \alpha \theta = \beta \theta \)

<table>
<thead>
<tr>
<th>( p )</th>
<th>( q )</th>
<th>( \theta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{Knows}(\text{John}, x) )</td>
<td>( \text{Knows}(\text{John}, \text{Jane}) )</td>
<td></td>
</tr>
<tr>
<td>( \text{Knows}(\text{John}, x) )</td>
<td>( \text{Knows}(y, \text{OJ}) )</td>
<td></td>
</tr>
<tr>
<td>( \text{Knows}(\text{John}, x) )</td>
<td>( \text{Knows}(y, \text{Mother}(y)) )</td>
<td></td>
</tr>
<tr>
<td>( \text{Knows}(\text{John}, x) )</td>
<td>( \text{Knows}(x, \text{OJ}) )</td>
<td></td>
</tr>
</tbody>
</table>
Unification

We can get the inference immediately if we can find a substitution $\theta$ such that $\textit{King}(x)$ and $\textit{Greedy}(x)$ match $\textit{King}(\text{John})$ and $\textit{Greedy}(y)$

$\theta = \{x/\text{John}, y/\text{John}\}$ works

$\text{UNIFY}(\alpha, \beta) = \theta$ if $\alpha\theta = \beta\theta$

<table>
<thead>
<tr>
<th>$p$</th>
<th>$q$</th>
<th>$\theta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{Knows}(\text{John}, x)$</td>
<td>$\text{Knows}(\text{John}, \text{Jane})$</td>
<td>${x/\text{Jane}}$</td>
</tr>
<tr>
<td>$\text{Knows}(\text{John}, x)$</td>
<td>$\text{Knows}(y, \text{OJ})$</td>
<td></td>
</tr>
<tr>
<td>$\text{Knows}(\text{John}, x)$</td>
<td>$\text{Knows}(y, \text{Mother}(y))$</td>
<td></td>
</tr>
<tr>
<td>$\text{Knows}(\text{John}, x)$</td>
<td>$\text{Knows}(x, \text{OJ})$</td>
<td></td>
</tr>
</tbody>
</table>
Unification

We can get the inference immediately if we can find a substitution $\theta$ such that $\text{King}(x)$ and $\text{Greedy}(x)$ match $\text{King}(\text{John})$ and $\text{Greedy}(y)$

$\theta = \{x/\text{John}, y/\text{John}\}$ works

$\text{UNIFY}(\alpha, \beta) = \theta$ if $\alpha\theta = \beta\theta$

<table>
<thead>
<tr>
<th>$p$</th>
<th>$q$</th>
<th>$\theta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{Knows}(\text{John}, x)$</td>
<td>$\text{Knows}(\text{John}, \text{Jane})$</td>
<td>${x/\text{Jane}}$</td>
</tr>
<tr>
<td>$\text{Knows}(\text{John}, x)$</td>
<td>$\text{Knows}(y, \text{OJ})$</td>
<td>${x/\text{OJ}, y/\text{John}}$</td>
</tr>
<tr>
<td>$\text{Knows}(\text{John}, x)$</td>
<td>$\text{Knows}(y, \text{Mother}(y))$</td>
<td></td>
</tr>
<tr>
<td>$\text{Knows}(\text{John}, x)$</td>
<td>$\text{Knows}(x, \text{OJ})$</td>
<td></td>
</tr>
</tbody>
</table>
Unification

We can get the inference immediately if we can find a substitution \( \theta \) such that \( \text{King}(x) \) and \( \text{Greedy}(x) \) match \( \text{King}(\text{John}) \) and \( \text{Greedy}(y) \)

\( \theta = \{x/\text{John}, y/\text{John}\} \) works

\( \text{UNIFY}(\alpha, \beta) = \theta \) if \( \alpha\theta = \beta\theta \)

<table>
<thead>
<tr>
<th>( p )</th>
<th>( q )</th>
<th>( \theta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{Knows}(\text{John}, x) )</td>
<td>( \text{Knows}(\text{John}, \text{Jane}) )</td>
<td>{x/\text{Jane}}</td>
</tr>
<tr>
<td>( \text{Knows}(\text{John}, x) )</td>
<td>( \text{Knows}(y, \text{OJ}) )</td>
<td>{x/\text{OJ}, y/\text{John}}</td>
</tr>
<tr>
<td>( \text{Knows}(\text{John}, x) )</td>
<td>( \text{Knows}(y, \text{Mother}(y)) )</td>
<td>{y/\text{John}, x/\text{Mother}(\text{John})}</td>
</tr>
<tr>
<td>( \text{Knows}(\text{John}, x) )</td>
<td>( \text{Knows}(x, \text{OJ}) )</td>
<td></td>
</tr>
</tbody>
</table>
Unification

We can get the inference immediately if we can find a substitution \( \theta \) such that \( \text{King}(x) \) and \( \text{Greedy}(x) \) match \( \text{King}(\text{John}) \) and \( \text{Greedy}(y) \)

\[ \theta = \{x/\text{John}, y/\text{John}\} \text{ works} \]

\( \text{Unify}(\alpha, \beta) = \theta \) if \( \alpha\theta = \beta\theta \)

<table>
<thead>
<tr>
<th>( p )</th>
<th>( q )</th>
<th>( \theta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Knows(John, x)</td>
<td>Knows(John, Jane)</td>
<td>{x/Jane}</td>
</tr>
<tr>
<td>Knows(John, x)</td>
<td>Knows(y, OJ)</td>
<td>{x/OJ, y/John}</td>
</tr>
<tr>
<td>Knows(John, x)</td>
<td>Knows(y, Mother(y))</td>
<td>{y/John, x/Mother(John)}</td>
</tr>
<tr>
<td>Knows(John, x)</td>
<td>Knows(x, OJ)</td>
<td>fail</td>
</tr>
</tbody>
</table>

Standardizing apart eliminates overlap of variables, e.g., \( \text{Knows}(z_{17}, OJ) \)
Generalized Modus Ponens (GMP)

\[
\begin{array}{c}
p_1', \ p_2', \ \ldots, \ p_n', \ (p_1 \land p_2 \land \ldots \land p_n \Rightarrow q) \\
q\theta
\end{array}
\]

where \( p'_i\theta = p_i\theta \) for all \( i \)

\( p_1' \) is \( \text{King}(John) \) \hspace{1cm} p_1 \) is \( \text{King}(x) \)
\( p_2' \) is \( \text{Greedy}(y) \) \hspace{1cm} p_2 \) is \( \text{Greedy}(x) \)
\( \theta \) is \( \{x/John, y/John\} \) \hspace{1cm} q \) is \( \text{Evil}(x) \)
\( q\theta \) is \( \text{Evil}(John) \)

GMP used with KB of definite clauses (exactly one positive literal)
All variables assumed universally quantified
Soundness of GMP

Need to show that

\[ p_1', \ldots, p_n', (p_1 \land \ldots \land p_n \Rightarrow q) \models q\theta \]

provided that \( p_i' \theta = p_i \theta \) for all \( i \)

Lemma: For any definite clause \( p \), we have \( p \models p \theta \) by UI

1. \( (p_1 \land \ldots \land p_n \Rightarrow q) \models (p_1 \land \ldots \land p_n \Rightarrow q)\theta = (p_1\theta \land \ldots \land p_n\theta \Rightarrow q\theta) \)

2. \( p_1', \ldots, p_n' \models p_1' \land \ldots \land p_n' \models p_1'\theta \land \ldots \land p_n'\theta \)

3. From 1 and 2, \( q\theta \) follows by ordinary Modus Ponens
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.

Prove that Col. West is a criminal
Example knowledge base contd.

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

\[ American(x) \land Weapon(y) \land Sells(x, y, z) \land Hostile(z) \Rightarrow Criminal(x) \]

Nono ... has some missiles
Example knowledge base contd.

\[\text{American}(x) \land \text{Weapon}(y) \land \text{Sells}(x, y, z) \land \text{Hostile}(z) \Rightarrow \text{Criminal}(x)\]

Nono . . . has some missiles, i.e., \( \exists x \ \text{Owns}(\text{Nono}, x) \land \text{Missile}(x) \):

\[\text{Owns}(\text{Nono}, M_1) \text{ and } \text{Missile}(M_1)\]

. . . all of its missiles were sold to it by Colonel West
Example knowledge base contd.

... it is a crime for an American to sell weapons to hostile nations:
\[
\text{American}(x) \land \text{Weapon}(y) \land \text{Sells}(x, y, z) \land \text{Hostile}(z) \Rightarrow \text{Criminal}(x)
\]

Nono ... has some missiles, i.e., \( \exists x \ \text{Owns}(\text{Nono}, x) \land \text{Missile}(x) \):
\[
\text{Owns}(\text{Nono}, M_1) \ \text{and} \ \text{Missile}(M_1)
\]

... all of its missiles were sold to it by Colonel West
\[
\forall x \ \text{Missile}(x) \land \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono})
\]

Missiles are weapons:
Example knowledge base contd.

... it is a crime for an American to sell weapons to hostile nations:
\[ \text{American}(x) \land \text{Weapon}(y) \land \text{Sells}(x, y, z) \land \text{Hostile}(z) \Rightarrow \text{Criminal}(x) \]

Nono ... has some missiles, i.e., \( \exists x \ \text{Owns}(\text{Nono}, x) \land \text{Missile}(x) \):
\[ \text{Owns}(\text{Nono}, M_1) \text{ and } \text{Missile}(M_1) \]

... all of its missiles were sold to it by Colonel West
\[ \forall x \ \text{Missile}(x) \land \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono}) \]

Missiles are weapons:
\[ \text{Missile}(x) \Rightarrow \text{Weapon}(x) \]

An enemy of America counts as "hostile":
Example knowledge base contd.

... it is a crime for an American to sell weapons to hostile nations:
\[ \text{American}(x) \land \text{Weapon}(y) \land \text{Sells}(x, y, z) \land \text{Hostile}(z) \Rightarrow \text{Criminal}(x) \]

Nono ... has some missiles, i.e., \( \exists x \ \text{Owns}(\text{Nono}, x) \land \text{Missile}(x) \):
\[ \text{Owns}(\text{Nono}, M_1) \text{ and } \text{Missile}(M_1) \]

... all of its missiles were sold to it by Colonel West
\[ \forall x \ \text{Missile}(x) \land \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono}) \]

Missiles are weapons:
\[ \text{Missile}(x) \Rightarrow \text{Weapon}(x) \]

An enemy of America counts as “hostile”:
\[ \text{Enemy}(x, \text{America}) \Rightarrow \text{Hostile}(x) \]

West, who is American ...
\[ \text{American}(\text{West}) \]

The country Nono, an enemy of America ...
\[ \text{Enemy}(\text{Nono}, \text{America}) \]
Forward chaining algorithm

function FOL-FC-Ask\((KB, \alpha)\) returns a substitution or false

repeat until new is empty

\(\text{new} \leftarrow \{\}\) 

for each sentence \(r\) in \(KB\) do

\((p_1 \land \ldots \land p_n \Rightarrow q) \leftarrow \text{Standardize-Apart}(r)\)

for each \(\theta\) such that \((p_1 \land \ldots \land p_n)\theta = (p'_1 \land \ldots \land p'_n)\theta\)

for some \(p'_1, \ldots, p'_n\) in \(KB\)

\(q' \leftarrow \text{Subst}(\theta, q)\)

if \(q'\) is not a renaming of a sentence already in \(KB\) or \(new\) then do

add \(q'\) to \(new\)

\(\phi \leftarrow \text{UNIFY}(q', \alpha)\)

if \(\phi\) is not fail then return \(\phi\)

add \(new\) to \(KB\)

return false
| American(West) | Missile(M1) | Owns(Nono,M1) | Enemy(Nono,America) |
Forward chaining proof

American(West) -> Missile(M1) -> Owns(Nono,M1) -> Sells(West,M1,Nono) -> Hostile(Nono)
Forward chaining proof

Criminal(West)

Weapon(M1)  Sells(West,M1,Nono)  Hostile(Nono)

American(West)  Missile(M1)  Owns(Nono,M1)  Enemy(Nono,America)
Properties of forward chaining

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

**Datalog** = first-order definite clauses + no functions (e.g., crime KB)
FC terminates for Datalog in poly iterations: at most \( p \cdot n^k \) literals

With functions, may not terminate if \( \alpha \) is not entailed

This is unavoidable: entailment with definite clauses is semidecidable
Efficiency of forward chaining

Simple observation: no need to match a rule on iteration $k$ if a premise wasn’t added on iteration $k - 1$

$\Rightarrow$ match each rule whose premise contains a newly added literal

Matching itself can be expensive

**Database indexing** allows $O(1)$ retrieval of known facts

E.g., query $\text{Missile}(x)$ retrieves $\text{Missile}(M_1)$

Matching conjunctive premises against known facts is NP-hard

Forward chaining is widely used in **deductive databases**
\[
\text{Diff}(\text{wa}, \text{nt}) \land \text{Diff}(\text{wa}, \text{sa}) \land \\
\text{Diff}(\text{nt}, \text{q}) \land \text{Diff}(\text{nt}, \text{sa}) \land \\
\text{Diff}(\text{q}, \text{nsw}) \land \text{Diff}(\text{q}, \text{sa}) \land \\
\text{Diff}(\text{nsw}, \text{v}) \land \text{Diff}(\text{nsw}, \text{sa}) \land \\
\text{Diff}(\text{v}, \text{sa}) \Rightarrow \text{Colorable}()
\]

\text{Diff}(\text{Red}, \text{Blue}) \quad \text{Diff}(\text{Red}, \text{Green}) \\
\text{Diff}(\text{Green}, \text{Red}) \quad \text{Diff}(\text{Green}, \text{Blue}) \\
\text{Diff}(\text{Blue}, \text{Red}) \quad \text{Diff}(\text{Blue}, \text{Green})

\text{Colorable()} \text{ is inferred iff the CSP has a solution}

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

function FOL-BC-Ask(\(KB, goals, \theta\)) returns a set of substitutions

inputs: \(KB\), a knowledge base

\(goals\), a list of conjuncts forming a query (\(\theta\) already applied)

\(\theta\), the current substitution, initially the empty substitution \{

local variables: \(answers\), a set of substitutions, initially empty

if \(goals\) is empty then return \{\(\theta\)\}

\(q' \leftarrow \text{Subst}(\theta, \text{First}(goals))\)

for each sentence \(r\) in \(KB\)

where \(\text{Standardize-Apart}(r) = (p_1 \land \ldots \land p_n \Rightarrow q)\)

and \(\theta' \leftarrow \text{Unify}(q, q')\) succeeds

\(\text{new_goals} \leftarrow [p_1, \ldots, p_n] \mid \text{Rest}(goals)\)

\(\text{answers} \leftarrow \text{FOL-BC-Ask}(KB, \text{new_goals}, \text{Compose}(\theta', \theta)) \cup \text{answers}\)

return \(\text{answers}\)
Backward chaining example

Criminal(West)
Backward chaining example

\[
\text{Criminal(West)} \quad \{x/West\}
\]

\[
\text{American}(x) \quad \text{Weapon}(y) \quad \text{Sells}(x,y,z) \quad \text{Hostile}(z)
\]
Backward chaining example

\begin{align*}
\text{Criminal}(&\text{West}) \\
\text{American}(&\text{West}) \quad \text{Weapon}(&\text{y}) \quad \text{Sells}(&\text{x, y, z}) \\
\text{Hostile}(&\text{z}) \\
\end{align*}

Backward chaining example

\[ \text{Criminal(West)} \]

\{ x/\text{West} \}

\[ \begin{align*}
\text{American(West)} \\
\text{Weapon(y)} \\
\text{Sells(x,y,z)} \\
\text{Hostile(z)} \\
\text{Missile(y)}
\end{align*} \]
Backward chaining example

American(West)
{ }

Weapon(y)

Sells(x,y,z)

Hostile(z)

Missile(y)
{ y/M1 }

Criminal(West) {x/West, y/M1}
Backward chaining example

\[
\text{Criminal} (\text{West}) \quad \{x/\text{West}, y/M1, z/\text{Nono}\}
\]

- \text{American} (\text{West})
  - \{\}\n- \text{Weapon} (y)
- \text{Sells} (\text{West}, M1, z)
  - \{z/\text{Nono}\}
- \text{Hostile} (z)
- \text{Missile} (y)
  - \{y/M1\}
- \text{Missile} (M1)
- \text{Owns} (\text{Nono}, M1)
Backward chaining example

\[ \text{Criminal(West)} \]
\[ \{ x/West, y/M1, z/Nono \} \]

\[ \text{American(West)} \]
\[ \{ \} \]

\[ \text{Weapon(y)} \]
\[ \{ z/Nono \} \]

\[ \text{Sells(West,M1,z)} \]
\[ \{ z/Nono \} \]

\[ \text{Missile(y)} \]
\[ \{ y/M1 \} \]

\[ \text{Missile(M1)} \]
\[ \{ \} \]

\[ \text{Owns(Nono,M1)} \]
\[ \{ \} \]

\[ \text{Enemy(Nono,America)} \]
\[ \{ \} \]
Properties of backward chaining

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

Incomplete due to infinite loops
  \[ \Rightarrow \text{ fix by checking current goal against every goal on stack} \]

Inefficient due to repeated subgoals (both success and failure)
  \[ \Rightarrow \text{ fix using caching of previous results (extra space!)} \]

Widely used (without improvements!) for logic programming
Logic programming

Sound bite: computation as inference on logical KBs

Logic programming

1. Identify problem
2. Assemble information
3. Tea break
4. Encode information in KB
5. Encode problem instance as facts
6. Ask queries
7. Find false facts

Ordinary programming

Identify problem
Assemble information
Figure out solution
Program solution
Encode problem instance as data
Apply program to data
Debug procedural errors

Should be easier to debug $Capital(New\text{\textbar}York,US)$ than $x := x + 2$ !
Prolog systems

Basis: backward chaining with Horn clauses + bells & whistles
Widely used in Europe, Japan (basis of 5th Generation project)
Compilation techniques ⇒ roughly a billion LIPS on workstation

Program = set of clauses = head :- literal₁, ... literalₙ.

criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).

Efficient unification by open coding
Efficient retrieval of matching clauses by direct linking
Depth-first, left-to-right backward chaining
Built-in predicates for arithmetic etc., e.g., X is Y*Z+3
Closed-world assumption (“negation as failure”)
  e.g., given alive(X) :- not dead(X).
  Here not means not provable
So alive(joe) succeeds if dead(joe) fails
Logic (as defined so far) is **monotonic**: for any \( \alpha, \beta, \gamma \),

\[
\text{if } \alpha \models \gamma \text{ then } \alpha \land \beta \models \gamma
\]

I.e., as facts are added, set of entailed consequences grows monotonically

[Proof: \( M(\alpha \land \beta) \subseteq M(\alpha) \)]

Negation-as-failure is **nonmonotonic**:

with no facts about joe in KB, \texttt{alive(joe)} is a consequence

with \texttt{dead(joe)} in KB, \texttt{alive(joe)} is no longer a consequence

Perhaps this behavior is natural? Humans jump to conclusions, then retract

Finding a satisfactory semantics and effective inference methods for nonmonotonic logics has proved very difficult
Prolog examples

Depth-first search from a start state X:

\[
\text{dfs}(X) :\text{- goal}(X).
\]
\[
\text{dfs}(X) :\text{- successor}(X,S),\text{dfs}(S).
\]

No need to loop over S: successor succeeds for each

Appending two lists to produce a third:

\[
\text{append}([], Y, Y).
\]
\[
\text{append}([X|L], Y, [X|Z]) :\text{- append}(L, Y, Z).
\]

query: \text{append}(A,B,[1,2]) ?
answers: A=[] B=[1,2]
A=[1,2] B=[]
Resolution: brief summary

Full first-order version:

\[
\ell_1 \lor \cdots \lor \ell_k, \quad m_1 \lor \cdots \lor m_n
\]

\[
(\ell_1 \lor \cdots \lor \ell_{i-1} \lor \ell_{i+1} \lor \cdots \lor \ell_k \lor m_1 \lor \cdots \lor m_{j-1} \lor m_{j+1} \lor \cdots \lor m_n)\theta
\]

where \( \text{UNIFY}(\ell_i, \neg m_j) = \theta \).

For example,

\[
\neg \text{Silent}(x) \lor \text{Supreme}(x)
\]

\[
\text{Silent}(\text{Harriet})
\]

\[
\text{Supreme}(\text{Harriet})
\]

with \( \theta = \{x/\text{Harriet}\} \)

Apply resolution steps to \( \text{CNF}(KB \land \neg \alpha) \); complete for FOL
Conversion to CNF

Everyone who loves all animals is loved by someone:
\[ \forall x \ [\forall y \ Animal(y) \Rightarrow Loves(x, y)] \Rightarrow [\exists y \ Loves(y, x)] \]

1. Eliminate biconditionals and implications

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

2. Move \( \neg \) inwards: \( \neg \forall x, p \equiv \exists x \ \neg p \), \( \neg \exists x, p \equiv \forall x \ \neg p \):

\[ \forall x \ [\exists y \ \neg (\neg Animal(y) \lor Loves(x, y))] \lor [\exists y \ Loves(y, x)] \]
\[ \forall x \ [\exists y \ \neg Animal(y) \land \neg Loves(x, y)] \lor [\exists y \ Loves(y, x)] \]
\[ \forall x \ [\exists y \ Animal(y) \land \neg Loves(x, y)] \lor [\exists y \ Loves(y, x)] \]
Conversion to CNF contd.

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

\[ \forall x \ [\exists y \ Animal(y) \land \neg Loves(x, y)] \lor [\exists z \ Loves(z, x)] \]

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 \( \land \) over \( \lor \):

\[ [Animal(F(x)) \lor Loves(G(x), x)] \land [\neg Loves(x, F(x)) \lor Loves(G(x), x)] \]
Resolution proof: definite clauses

\neg American(x) \lor \neg Weapon(y) \lor \neg Sells(x,y,z) \lor \neg Hostile(z) \lor Criminal(x)

\neg Criminal(West)

American(West)

\neg American(West) \lor \neg Weapon(y) \lor \neg Sells(West,y,z) \lor \neg Hostile(z)

\neg Weapon(y) \lor \neg Sells(West,y,z) \lor \neg Hostile(z)

Missile(M1)

\neg Missile(y) \lor \neg Sells(West,y,z) \lor \neg Hostile(z)

\neg Sells(West,M1,z) \lor \neg Hostile(z)

Missile(M1)

\neg Missile(M1) \lor \neg Owns(Nono,M1) \lor \neg Hostile(Nono)

\neg Owns(Nono,M1) \lor \neg Hostile(Nono)

Owns(Nono,M1)

\neg Enemy(x,America) \lor Hostile(x)

\neg Hostile(Nono)

Enemy(Nono,America)

Enemy(Nono,America)