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
<table>
<thead>
<tr>
<th>Year</th>
<th>Name</th>
<th>Contribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>450BC</td>
<td>Stoics</td>
<td>propositional logic, inference (maybe)</td>
</tr>
<tr>
<td>322BC</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>∃ 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>¬∃ 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>
Reminder: Substitution

Given a sentence $S$ and a substitution $\sigma$,
$S\sigma$ denotes the result of plugging $\sigma$ into $S$; e.g.,
$S = \text{Smarter}(x, y)$
$\sigma = \{x/\text{Hillary}, y/\text{Bill}\}$
$S\sigma = \text{Smarter}(\text{Hillary}, \text{Bill})$
Universal instantiation (UI)

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

\[ \forall v \ \alpha \]

\[ \text{SUBST}(\{v/g\}, \alpha) \]

for any variable \( v \) and ground term \( g \) (a term without variables)

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:

$$\exists v \; \alpha \quad \text{Subst}\left(\{v/k\}, \alpha\right)$$

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.

E.g., 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.
Once we have rules for inferring non-quantified sentences, it becomes possible to reduce FOL to PL:

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

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

Assume KB contains:

$$\forall x \ King(x) Evil(x)$$
$$King(Father(Father(Father(John))))$$

In order to prove $Evil(Father(Father(Father(John))))$, the above method requires instantiation with depth-3 ground terms $Father(Father(Father(John)))$. 
Another example:

Assume KB contains:

\[
\forall x \text{ King}(x) \Rightarrow \text{King}(\text{Father}(x)) \\
A = \text{Father}(B) \\
B = \text{Father}(C) \\
\text{King}(C)
\]

In order to prove \text{King}(A), the above method requires instantiation with depth-2 ground terms \text{Father}(\text{Father}(C)).

You can notice that this could go on indefinitely.
Reduction contd.

**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 $n = 1$ 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:
- you can prove it if a sentence is entailed
- no algorithm exists that rejects every non-entailed sentence.
Problems with propositionalization

Propositionalization seems to generate lots of irrelevant sentences. Assume we have the query \( \text{Evil}(x) \) and the following KB:

\[
\begin{align*}
\forall x \ & \text{King}(x) \land \text{Greedy}(x) \Rightarrow \text{Evil}(x) \\
\text{King}(\text{John}) \\
\forall y \ & \text{Greedy}(y) \\
\text{Brother}(\text{Richard, John})
\end{align*}
\]

it seems obvious that \( \text{Evil}(\text{John}) \), but propositionalization produces lots of facts such as \( \text{Greedy}(\text{Richard}) \) 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!
Unification

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

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

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

Unification finds substitutions that makes different logical expressions look identical.
# Unification

Assume we have the following KB; what is the unification of each pair (row) of expression:

<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></td>
</tr>
<tr>
<td>Knows(John, x)</td>
<td>Knows(y, OJ)</td>
<td></td>
</tr>
<tr>
<td>Knows(John, x)</td>
<td>Knows(y, Mother(y))</td>
<td></td>
</tr>
<tr>
<td>Knows(John, x)</td>
<td>Knows(x, OJ)</td>
<td></td>
</tr>
</tbody>
</table>
**Unification**

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

<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>(\diamond {x/Jane})</td>
</tr>
<tr>
<td>Knows(John, (x))</td>
<td>Knows((y), OJ)</td>
<td></td>
</tr>
<tr>
<td>Knows(John, (x))</td>
<td>Knows((y), Mother((y)))</td>
<td></td>
</tr>
<tr>
<td>Knows(John, (x))</td>
<td>Knows((x), OJ)</td>
<td></td>
</tr>
</tbody>
</table>
**Unification**

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

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

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

<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>( \diamond {x/\text{Jane}} )</td>
</tr>
<tr>
<td>( \text{Knows}(\text{John}, x) )</td>
<td>( \text{Knows}(y, \text{OJ}) )</td>
<td>( \diamond {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>( \diamond {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

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

<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>( \diamond ) {x/Jane}</td>
</tr>
<tr>
<td>Knows(John, x)</td>
<td>Knows(y, OJ)</td>
<td>( \diamond ) {x/OJ, y/John}</td>
</tr>
<tr>
<td>Knows(John, x)</td>
<td>Knows(y, Mother(y))</td>
<td>( \diamond ) {y/John, x/Mother(John)}</td>
</tr>
<tr>
<td>Knows(John, x)</td>
<td>Knows(x, OJ)</td>
<td>( \diamond ) fail</td>
</tr>
</tbody>
</table>
Unification-Standardizing Apart

UNIFY(\textit{Knows}(\textit{John}, x), \textit{Knows}(x, \textit{Elizabeth})) = fail

Is there a problem? Why?
Unification—Standardizing Apart

$\text{UNIFY}(\text{Knows}(\text{John}, x), \text{Knows}(x, \text{OJ})) = \text{fail}$

Remember: $\text{Knows}(\text{John}, x)$ means "John knows everyone" (universally quantified) and since "Everyone knows OJ" as well, so the unification should NOT fail.

The problem is due to both predicates using the same variable, which is solved by "standardizing apart" (renaming variables) before unification.

Standardizing apart eliminates overlap of variables, e.g., $\text{Knows}(z_{17}, \text{OJ})$
Unification-Most General Unifier

UNIFY(Knows(John, x), Knows(y, z))?

\{y/John, x/z\} or \{y/John, x/John, z/John\}?
Unification-Most General Unifier

\[ \text{UNIFY}(\text{Knows}(John, x), \text{Knows}(y, z))? \]

\[ \{y/John, x/z\} \text{ (gives } \text{Knows}(John, z)\text{) or } \{y/John, x/John, z/John\} \text{ (gives } \text{Knows}(John, John)\text{).} \]

◊ When there are more than one possible unifier, pick the one that leaves the most choice for the variables.

◊ There is always a single Most General Unifier for every pair of expressions.
Proof with Basic inference rules

The next few slides show how a proof can be made with basic inference rules.

Then we present Generalized Modus Ponens (GMP) that applies a common pattern of 3 steps in one.
Example proof using basic inference rules

Using the unification and inference rules, we can prove some statements:

Assuming that the KB contains the following:

- Buffy is a buffalo
- Pat is a pig
- Buffaloes outrun pigs

Prove that Bob outruns Pat

1. \( \text{Buffalo}(Bob) \)
2. \( \text{Pig}(Pat) \)
3. \( \forall x, y \ \text{Buffalo}(x) \land \text{Pig}(y) \Rightarrow \text{Faster}(x, y) \)
Example proof

Prove Bob outruns Pat

Bob is a buffalo
Pat is a pig
Buffaloes outrun pigs

1. $Buffalo(Bob)$
2. $Pig(Pat)$
3. $\forall x, y \ Buffalo(x) \land Pig(y) \Rightarrow Faster(x, y)$

And-Intro. 1 & 2

4. $Buffalo(Bob) \land Pig(Pat)$
Example proof

Prove Bob outruns Pat

Bob is a buffalo
Pat is a pig
Buffaloes outrun pigs

1. Buffalo(Bob)
2. Pig(Pat)
3. \( \forall x, y \ Buffalo(x) \land Pig(y) \Rightarrow Faster(x, y) \)

And-Intro. 1 & 2
Univ.El. 3, \( \{x/Bob, y/Pat\} \)

4. Buffalo(Bob) \land Pig(Pat)
5. Buffalo(Bob) \land Pig(Pat) \Rightarrow Faster(Bob, Pat)
## Example proof

Prove Bob outruns Pat

<table>
<thead>
<tr>
<th>Bob is a buffalo</th>
<th>1. $Buffalo(Bob)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pat is a pig</td>
<td>2. $Pig(Pat)$</td>
</tr>
<tr>
<td>Buffaloes outrun pigs</td>
<td>3. $\forall x, y \ Buffalo(x) \land Pig(y) \Rightarrow Faster(x, y)$</td>
</tr>
</tbody>
</table>

| And-Intro. 1 & 2       | 4. $Buffalo(Bob) \land Pig(Pat)$        |
| Univ.El. 3, $\{x/Bob, y/Pat\}$ | 5. $Buffalo(Bob) \land Pig(Pat) \Rightarrow Faster(Bob, Pat)$ |
| Mod.Pon. 4 & 5        | 6. $Faster(Bob, Pat)$                    |
Search with primitive inference rules

The above can be implemented as a search process where:

◊ Operators are inference rules
◊ States are sets of sentences
◊ Goal test checks state to see if it contains query sentence

Problem: branching factor huge, esp. for UE

Idea: AI, UE, MP is a common inference pattern. Find a substitution that makes the rule premise match some known facts

⇒ a single, more powerful inference rule: Generalized Modus Ponens
Generalized Modus Ponens (GMP)

Idea (Gen. Mod. Ponens): Unify rule premises with known facts, apply unifier to conclusion

E.g., KB contains:
\[ \text{Knows}(\text{John}, \text{Jane}) \]
\[ \text{Knows}(\text{John}, x) \implies \text{Likes}(\text{John}, x) \]

We can infer: \[ \text{Likes}(\text{John}, \text{Jane}) \] after the unification with substitution \[ \{x/Jane\} \]
Generalized Modus Ponens (GMP)

**GMP does this in one step:** And-Introduction, Universal Elimination, Modus Ponens

\[
P_1', \ P_2', \ldots, \ P_n', \ (P_1 \land P_2 \land \ldots \land P_n \Rightarrow Q) \quad q\sigma
\]

where \( p_i' \sigma = p_i\sigma \) for all \( i \)

E.g.

\[
P_1' = \text{Faster(Bob,Pat)}
\]
\[
P_2' = \text{Faster(Pat,Steve)}
\]

\[P_1 \land P_2 \Rightarrow Q = \text{Faster}(x, y) \land \text{Faster}(y, z) \Rightarrow \text{Faster}(x, z) \]

\[\sigma = \{x/Bob, y/Pat, z/Steve\}\]

\[q\sigma = \text{Faster}(Bob, Steve)\]
GMP is sensible because:

◊ It takes big steps

◊ It takes sensible steps: it uses substitutions that are guaranteed to help
  \(\{x/\text{Bob}, y/\text{Pat}, z/\text{Steve}\}\)
Generalized Modus Ponens (GMP)

With the knowledgebase containing:

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

Applying GMP, we can entail \( Evil(John) \):

\[
\frac{p_1', p_2', \ldots, p_n', (p_1 \land p_2 \land \ldots \land p_n \Rightarrow q)}{q\theta}
\]

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

\( p_1' \) is \( King(John) \)
\( p_1 \) is \( King(x) \)
\( p_2' \) is \( Greedy(y) \)
\( p_2 \) is \( Greedy(x) \)
\( \theta \) is \( \{x/John, y/John\} \)
\( q \) is \( Evil(x) \)
\( q\theta \) is \( Evil(John) \)
Inference mechanism with one inference rule (GMP) \( \Rightarrow \) all the sentences in the KB should be in a form to match the premise of Modus Ponens.

Convert the KB into Canonical Form:

◊ Apply Existential Elimination

\[ \exists x \, \text{Missile}(x) \text{ is converted to } \text{Missile}(M1) \]

◊ Remove Universal Quantifiers (all variables are assumed universally quantified)

\[ \forall x \, \text{Missile}(x) \Rightarrow \text{Weapon}(x) \text{ is written as } \text{Missile}(x) \Rightarrow \text{Weapon}(x) \]

◊ Put sentences in Horn form
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.

First of all you shd. be able to extract the facts and axioms to put in the knowledgebase, from this paragraph:
it is a crime for an American to sell weapons to hostile nations:
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
... 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
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., \( \exists x \ \text{Owns}(\text{Nono}, x) \land \text{Missile}(x) \):

\( \text{Owns}(\text{Nono}, M_1) \) 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:
... 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”:
... 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., \( \exists x \ \text{Owns}(\text{Nono}, x) \land \text{Missile}(x): \)

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

... 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 and Backward Chaining

GMP can be used to ways:

♦ forward chaining (new fact is added to the KB and we want to generate its consequences)

♦ backward chaining (we start with something we want to prove, find implication sentences that would conclude it, and attempt to establish their premises)
Forward chaining

When a new fact $p$ is added to the KB
for each rule such that $p$ unifies with a premise
if the other premises are known
then add the conclusion to the KB and continue chaining

Forward chaining is data-driven
  e.g., inferring properties and categories from percepts
function FOL-FC-Ask(KB, α) returns a substitution or false

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 = (p'₁ ∧ ... ∧ p'ₙ)θ 
      for some p'₁, ..., p'ₙ in KB
        q' ← SUBST(θ, q)
        if q' is not a renaming of a sentence already in KB or new then do
          add q' to new
          φ ← UNIFY(q', α)
          if φ is not fail then return φ
      add new to KB
  return false
Forward chaining proof

American(West)  Missile(M1)  Owns(Nono,M1)  Enemy(Nono,America)
Three sentences were found to be of the form of "premises implies conclusion" and unified with known facts; then resulting substitution is applied to the conclusion of each rule (e.g. 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).

◊ FC may not terminate in general if $\alpha$ is not entailed. This is unavoidable since entailment with definite clauses is semidecidable:
- can find a proof of $\alpha$ if $KB \models \alpha$
- cannot always prove that $KB \not\models \alpha$

◊ Terminates for Datalog in polynomial iterations. There can be at most $p \cdot n^k$ facts to be added, where $k$ is the maximum arity (num. arguments) of any predicate and $p$ is the number of predicates and $n$ is the number of constant symbols.

**Definite clause** = Horn clause with exactly one positive literal

**Datalog** = first-order definite clauses + **no functions** (e.g., crime KB)

◊ With function symbols, infinitely many new facts can be added.
Efficiency of forward chaining

- Matching itself can be expensive: matching conjunctive premises against known facts is NP-hard!

**Database indexing** allows $O(1)$ retrieval of known facts
- e.g., query $\text{Missile}(x)$ retrieves $\text{Missile}(M_1)$

**Incremental forward chaining:** 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

- Forward chaining is widely used in **deductive databases**
Backward chaining

When a query $q$ is asked
   if a matching fact $q'$ is known, return the unifier
   for each rule whose consequent $q'$ matches $q$
      attempt to prove each premise of the rule by backward chaining

(Some added complications in keeping track of the unifiers)

(More complications help to avoid infinite loops)

Two versions: find any solution, find all solutions

Backward chaining is the basis for logic programming, e.g., Prolog
function FOL-BC-Ask(KB, goals, θ) returns a set of substitutions

inputs: KB, a knowledge base
        goals, a list of conjuncts forming a query
        θ, the current substitution, initially the empty substitution { }

local variables: answers, a set of substitutions, initially empty

if goals is empty then return {θ}

q' ← SUBST(θ, FIRST(goals))

for each sentence r in KB
    where STANDARDIZE-Apart(r) = ( p₁ ∧ ... ∧ pₙ ⇒ q)
    and θ' ← UNIFY(q, q') succeeds
    new_goals ← [p₁, ..., pₙ | REST(goals)]
    answers ← FOL-BC-Ask(KB, new_goals, COMPOSE(θ', θ)) ∪ answers

return answers
Backward chaining example

Criminal(West)
Backward chaining example

Criminal(West)

\{x/West\}

American(x)  Weapon(y)  Sells(x,y,z)  Hostile(z)
Backward chaining example

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

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

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

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

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

- \text{American}(\text{West})
  \[
  \{ \}
  \]
- \text{Weapon}(y)
- \text{Sells}(x,y,z)
- \text{Hostile}(z)

- \text{Missile}(y)
  \[
  \{ y/M1 \}
  \]
Backward chaining example

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

American(West)  \{\}\n
Weapon(y)  \{\}\n
Sells(West,M1,z)  \{z/Nono\}

Missile(y)  \{y/M1\}

Missile(M1)  \{}\n
Owns(Nono,M1)

Hostile(z)
Backward chaining example

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

American(West) \{\}

Weapon(y) \{\}

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

Missile(y) \{y/M1\}

Missile(M1) \{\}

Owns(Nono, M1) \{\}

Hostile(Nono) \{\}

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

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

Incomplete due to infinite loops
  ⇒ fix by checking current goal against every goal on stack

Inefficient due to repeated subgoals (both success and failure)
  ⇒ fix using caching of previous results (extra space!)

Widely used (without improvements!) for logic programming
Completeness of Modus Ponens

Unfortunately, Modus Ponens is **incomplete**. Assume we have the following KB:

\[
P(x) \Rightarrow Q(x) \\
\neg P(x) \Rightarrow Q(x) \\
Q(x) \Rightarrow S(x) \\
R(x) \Rightarrow S(x)
\]

But we cannot prove \( S(A) \) because \( \neg P(x) \Rightarrow Q(x) \) cannot be put in Horn form (\( P_1 \land P_2 \land P_3 \Rightarrow Q \) where \( P_i \) are non-negated atoms)

What other inference mechanism can we use, which will be complete?
Resolution for FOL is complete as in Propositional Logic.

Basic propositional version:

\[
\frac{\alpha \lor \beta, \neg \beta \lor \gamma}{\alpha \lor \gamma} \quad \text{or equivalently} \quad \frac{\neg \alpha \Rightarrow \beta, \beta \Rightarrow \gamma}{\neg \alpha \Rightarrow \gamma}
\]
Resolution with CNF

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

Note that in POL, two literals are complementary if one is the negation of another one, whereas in FOL, they are complementary if one can be unified with the negation of the other one.
Resolution with CNF

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

\[
\frac{(\ell_1 \lor \cdots \lor \ell_{i-1} \lor \ell_{i+1} \lor \cdots \lor \ell_k \lor m_1 \lor \cdots \lor m_{j-1} \lor m_{j+1} \lor \cdots \lor m_n)\theta}{\text{where } \text{UNIFY}(\ell_i, \neg m_j) = \theta.}
\]

Example:

\[
\neg \text{Rich}(x) \lor \text{Unhappy}(x)
\]

\[
\frac{\text{Rich}(Ken)}{\text{Unhappy}(Ken)}
\]

with \( \theta = \{x/Ken\} \)
Conjunctive Normal Form

Resolution with disjunctions requires the CNF:

**Literal** = (possibly negated) atomic sentence, e.g., \( \neg Rich(Me) \)

**Clause** = disjunction of literals, e.g., \( \neg Rich(Me) \lor Unhappy(Me) \)

The KB is a **conjunction** of clauses.

**Any FOL KB can be converted to CNF** as follows:

1. Replace \( P \Rightarrow Q \) by \( \neg P \lor Q \)
2. Move \( \neg \) inwards, e.g., \( \neg \forall x P \) becomes \( \exists x \neg P \)
3. Standardize variables apart, e.g., \( \forall x P \lor \exists x Q \) becomes \( \forall x P \lor \exists y Q \)
4. Move quantifiers left in order, e.g., \( \forall x P \lor \exists y Q \) becomes \( \forall x \exists y P \lor Q \)
5. Eliminate \( \exists \) by Skolemization (next slide)
6. Drop universal quantifiers
7. Distribute \( \land \) over \( \lor \), e.g., \( (P \land Q) \lor R \) becomes \( (P \lor Q) \land (P \lor R) \)
Skolemization

$\exists x \text{Rich}(x)$ becomes $\text{Rich}(G1)$ where $G1$ is a new “Skolem constant”

More tricky when $\exists$ is inside $\forall$

E.g., “Everyone has a heart”

$\forall x \text{Person}(x) \Rightarrow \exists y \text{Heart}(y) \land \text{Has}(x, y)$
Skolemization

More tricky when $\exists$ is inside $\forall$

E.g., “Everyone has a heart”
$$\forall x \text{ Person}(x) \Rightarrow \exists y \text{ Heart}(y) \land \text{Has}(x, y)$$

Incorrect:
$$\forall x \text{ Person}(x) \Rightarrow \text{Heart}(H1) \land \text{Has}(x, H1)$$

Correct:
$$\forall x \text{ Person}(x) \Rightarrow \text{Heart}(H(x)) \land \text{Has}(x, H(x))$$
where $H$ is a new symbol (“Skolem function”)

Skolem function arguments: all enclosing universally quantified variables
Conversion to CNF

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

1. Eliminate biconditionals and implications
\[ \forall x \left[ \neg \forall y \ \neg \ Animal(y) \lor Loves(x, y) \right] \lor \left[ \exists y \ Loves(y, x) \right] \]

2. Move \( \neg \) inwards:
\[ \neg \forall x, p \equiv \exists x \ \neg p, \quad \neg \exists x, p \equiv \forall x \ \neg p: \]
\[ \forall x \left[ \exists y \ \neg \neg \ Animal(y) \lor \neg Loves(x, y) \right] \lor \left[ \exists y \ Loves(y, x) \right] \]
\[ \forall x \left[ \exists y \ \neg \neg \ Animal(y) \land \neg Loves(x, y) \right] \lor \left[ \exists y \ Loves(y, x) \right] \]
\[ \forall x \left[ \exists y \ Animal(y) \land \neg Loves(x, y) \right] \lor \left[ \exists y \ Loves(y, x) \right] \]
3. Standardize variables: each quantifier should use a different one

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

4. Skolemize: a more general form of existential instantiation.

Each existential variable is replaced by a Skolem function of the enclosing universally quantified variables:

\[ \forall x \ [Animal(F(x)) \land \lnot Loves(x, F(x))] \lor Loves(G(x), x) \]

5. Drop universal quantifiers:

\[ [Animal(F(x)) \land \lnot Loves(x, F(x))] \lor Loves(G(x), x) \]

6. Distribute $\land$ over $\lor$:

\[ [Animal(F(x)) \lor Loves(G(x), x)] \land [\lnot Loves(x, F(x)) \lor Loves(G(x), x)] \]
Chaining with Resolution

Chaining with resolution is more powerful than Modus Ponens but still not complete.

Ex. How to prove $P \lor \neg P$ with an empty KB.

Solution: refutation (proof by contradiction):
To prove $P$, assume $P$ is false (add $\neg P$ to the KB) and prove a contradiction.

In other words, to prove $KB \models \alpha$, show that $KB \land \neg \alpha$ is unsatisfiable.
Refutation

To prove $\alpha$:
- negate it
- convert to CNF
- add to CNF KB
- infer contradiction

E.g., to prove $\text{Rich}(\text{Me})$, add $\neg \text{Rich}(\text{Me})$ to the CNF KB

\begin{align*}
\neg \text{PhD}(x) \lor \text{HighlyQualified}(x) \\
\text{PhD}(x) \lor \text{EarlyEarnings}(x) \\
\neg \text{HighlyQualified}(x) \lor \text{Rich}(x) \\
\neg \text{EarlyEarnings}(x) \lor \text{Rich}(x)
\end{align*}
Resolution proof

\[ \neg \text{PhD}(x) \lor \text{HQ}(x) \]

\[ \neg \text{HQ}(x) \lor \text{Rich}(x) \]

\[ \neg \text{PhD}(x) \lor \text{Rich}(x) \]

\[ \text{PhD}(x) \lor \text{ES}(x) \]

\[ \text{Rich}(x) \lor \text{ES}(x) \]

\[ \neg \text{ES}(x) \lor \text{Rich}(x) \]

\[ \text{Rich}(x) \]

\[ \neg \text{Rich}(\text{Me}) \]

\{x/\text{Me}\}
Resolution proof: definite clauses

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

\[ \neg \text{American}(x) \lor \neg \text{Weapon}(y) \lor \neg \text{Sells}(West,y,z) \lor \neg \text{Hostile}(z) \]

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

\[ \neg \text{Weapon}(y) \lor \neg \text{Sells}(West,y,z) \lor \neg \text{Hostile}(z) \]

\[ \neg \text{Missile}(x) \lor \text{Weapon}(x) \]

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

\[ \neg \text{Missile}(x) \lor \neg \text{Owns}(Nono,x) \lor \text{Sells}(West,x,Nono) \]

\[ \neg \text{Sells}(West,M1,z) \lor \neg \text{Hostile}(z) \]

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

\[ \neg \text{Missile}(M1) \lor \neg \text{Owns}(Nono,M1) \lor \neg \text{Hostile}(Nono) \]

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

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

\[ \neg \text{Enemy}(x,America) \lor \text{Hostile}(x) \]

\[ \text{Enemy}(Nono,America) \]

\[ \neg \text{Enemy}(Nono,America) \]

\[ \neg \text{Hostile}(Nono) \]

\[ \neg \text{Hostile}(Nono) \]
Resolution: Implicative Form

Resolution can also be stated for Implicative Normal Form:

\[
\begin{align*}
  p_1 \land p_2 \ldots \land p_m & \Rightarrow r_1 \lor r_2 \ldots r_n, \\
  s_1 \land s_2 \ldots \land s_o & \Rightarrow q_1 \lor q_2 \ldots q_o
\end{align*}
\]

\[
(p_1 \land \ldots \land p_{j-1} \land p_{j+1} \ldots \land p_m \land s_1 \land \ldots \land s_o \Rightarrow r_1 \lor r_2 \ldots \lor r_n \lor q_1 \lor \ldots \lor q_{k-1} \lor q_{k+1} \ldots \lor q_o)\sigma
\]

where \( p_j \sigma = q_k \sigma \)

For example,

\[
\begin{align*}
  \text{Rich}(x) \land \text{Famous}(x) & \Rightarrow \text{Unhappy}(x) \\
  \text{HighlyEducated}(y) & \Rightarrow \text{Rich}(y) \lor \text{Academic}(x)
\end{align*}
\]

\[
\begin{align*}
  \text{HighlyEducated}(x) \land \text{Famous}(x) & \Rightarrow \text{Unhappy}(x) \lor \text{Academic}(x)
\end{align*}
\]

with \( \sigma = \{x/y\} \)
Conjunctive Normal Form vs. Implicative Normal Form

CNF expressions can be easily converted to Implicative Normal Form:

\[ P(x) \] converts to ?

\[ \neg P(x) \] converts to ?
Conjunctive Normal Form vs. Implicative Normal Form

CNF expressions can be easily converted to Implicative Normal Form:

\[ P(x) \text{ converts to } True \Rightarrow P(x) \]

\[ \neg P(x) \text{ converts to } P(x) \Rightarrow False \]
Resolution proof with Implicative Normal Forms

\[ P(w) \Rightarrow Q(w) \]
\[ Q(y) \Rightarrow S(y) \]
\[ \{y/w\} \]
\[ P(w) \Rightarrow S(w) \]
\[ \text{True} \Rightarrow P(x) \lor R(x) \]
\[ \{w/x\} \]
\[ \text{True} \Rightarrow S(x) \lor R(x) \]
\[ \text{True} \Rightarrow S(A) \]
\[ \{x/A, z/A\} \]
\[ R(z) \Rightarrow S(z) \]
Refutation with Implicative Normal Form

\[ P(w) \Rightarrow Q(w) \]
\[ Q(y) \Rightarrow S(y) \]
\[ P(w) \Rightarrow S(w) \]
\[ Q(y) \Rightarrow S(y) \]
\[ P(w) \Rightarrow S(w) \]
\[ Q(y) \Rightarrow S(y) \]
\[ R(z) \Rightarrow S(z) \]
\[ S(A) \Rightarrow False \]
\[ True \Rightarrow False \]
Resolution in practice

Resolution is complete and usually necessary for mathematics

Automated theorem provers are starting to be useful to mathematicians and have proved several new theorems
Prolog systems

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

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

\[
\text{criminal}(X) :- \text{american}(X), \text{weapon}(Y), \text{sells}(X,Y,Z), \text{hostile}(Z).
\]

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., \(X \text{ is } Y \times Z + 3\)
Closed-world assumption ("negation as failure")
  e.g., given \(\text{alive}(X) :- \text{not}\ \text{dead}(X)\).
  \(\text{alive}(joe)\) succeeds if \(\text{dead}(joe)\) fails
Depth-first search from a start state X:

dfs(X) :- goal(X).
dfs(X) :- successor(X,S),dfs(S).

No need to loop over S: successor succeeds for each

Appending two lists to produce a third:

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

query: append(A,B,[1,2]) ?
answers: A=[] B=[1,2]
         A=[1,2] B=[]
Skip [301-308]
Skim [Thm.provers(pp.308),309]