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

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 $\exists x \ d(x^0)/dy = x^0$ we obtain

$$d(e^0)/dy = e^0$$

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??

A brief history of reasoning

450 b.c. Stoics propositional logic, inference (maybe)
322 b.c. Aristotle “syllogisms” (inference rules), quantifiers
1565 Cardano probability theory (propositional logic + uncertainty)
1847 Boole propositional logic (again)
1879 Frege first-order logic
1922 Wittgenstein proof by truth tables
1930 Gödel $\exists$ 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
**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.

**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 \( Evil(John) \), but propositionalization produces lots of facts such as \( Greedy(Richard) \) that are irrelevant.

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

With function symbols, it gets much much worse!

**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

\( King(John), Greedy(John), Evil(John), King(Richard) \) etc.

**Unification**

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

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

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

**Unification contd.**

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

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

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

**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., \( \text{Father(Father(Father(John)))} \)

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.
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 \( \text{Greedy}(\text{y}) \)
\[ \theta = \{ x/\text{John}, y/\text{John} \} \text{ works} \]
\[ \text{UNIFY}(\alpha, \beta) = \theta \text{ if } \alpha \theta = \beta \theta \]

Generalized Modus Ponens (GMP)

\[ p_1, p_2, \ldots, p_n, \{ p_1 \land p_2 \land \ldots \land p_n \Rightarrow q \} \]
\[ q^\theta \]

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

\[ p_1 \text{ is } \text{King}(\text{John}) \]
\[ p_1 \text{ is } \text{King}(\text{John}) \]
\[ p_2 \text{ is } \text{Greedy}(\text{y}) \]
\[ p_2 \text{ is } \text{Greedy}(\text{y}) \]
\[ \theta \text{ is } \{ x/\text{John}, y/\text{John} \} \]
\[ q \text{ is } \text{Evil}(\text{y}) \]
\[ q^\theta \text{ is } \text{Evil}(\text{y}) \]

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. \( \{ p_1, \ldots, p_n \} \models p_1^\theta \land \ldots \land p_n^\theta = p_1^\theta \land \ldots \land p_n^\theta \)
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:

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

\[ \text{Nono} \ldots \text{has some missiles} \]

\[ \exists x \ \text{Owns}(\text{Nono}, x) \land \text{Missile}(x); \]

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

\[ \ldots \text{all of its missiles were sold to it by Colonel West} \]

\[ \forall x \ \text{Missile}(x) \land \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono}) \]

\[ \text{Missiles are weapons:} \]

\[ \text{Missile}(x) \Rightarrow \text{Weapon}(x) \]

\[ \text{An enemy of America counts as "hostile":} \]

\[ \text{Enemy}(x; \text{America}) \Rightarrow \text{Hostile}(x) \]

\[ \text{West, who is American} \ldots \]

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

\[ \text{The country Nono, an enemy of America} \ldots \]

\[ \text{Enemy}(\text{Nono}; \text{America}) \]
Forward chaining algorithm

\[
\text{function FOL-FC-Ask}(KB, \alpha) \text{ returns a substitution or } \text{false}
\]

\[
\text{repeat until } \text{new is empty}
\]

\[
\text{new} \leftarrow \{ \}
\]

\[
\text{for each sentence } r \text{ in } KB \text{ do}
\]

\[
(p_1 \land \ldots \land p_n \Rightarrow q) \leftarrow \text{STANDARDIZE-APART}(r)
\]

\[
\text{for each } \theta \text{ such that } (p_1 \land \ldots \land p_n)\theta = (p_1' \land \ldots \land p_n')\theta
\]

\[
\text{for some } p_1', \ldots, p_n' \text{ in } KB
\]

\[
q' \leftarrow \text{SUBST}(\theta, q)
\]

\[
\text{if } q' \text{ is not a renaming of a sentence already in } KB \text{ or } \text{new then do}
\]

\[
\text{add } q' \text{ to } \text{new}
\]

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

\[
\text{if } \phi \text{ is not } \text{fail} \text{ then return } \phi
\]

\[
\text{add } \text{new} \text{ to } KB
\]

\[
\text{return } \text{false}
\]

Forward chaining proof

<table>
<thead>
<tr>
<th>Criminal(West)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weapon(M1)</td>
</tr>
<tr>
<td>Sells(West,M1,Nono)</td>
</tr>
<tr>
<td>Hostile(Nono)</td>
</tr>
</tbody>
</table>

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 \( k-1 \)

⇒ 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
Hard matching example

\[\text{Diff}(wa, nt) \land \text{Diff}(wa, sa) \land \]
\[\text{Diff}(at, q) \land \text{Diff}(at, sa) \land \]
\[\text{Diff}(q, nsw) \land \text{Diff}(q, sa) \land \]
\[\text{Diff}(nsw, v) \land \text{Diff}(nsw, sa) \land \]
\[\text{Diff}(v, sa) \Rightarrow \text{Colorable}()\]

\[\text{Diff}(\text{Red, Blue}) \land \text{Diff}(\text{Red, Green})\]
\[\text{Diff}(\text{Green, Red}) \land \text{Diff}(\text{Green, Blue})\]
\[\text{Diff}(\text{Blue, Red}) \land \text{Diff}(\text{Blue, Green})\]

\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

\[\text{function FOL-BC-Ask}(KB, goals, \theta) \text{ returns a set of substitutions}\]
\[\text{inputs: } KB, \text{ a knowledge base}\]
\[\text{goals, a list of conjuncts forming a query } (\theta \text{ already applied})\]
\[\theta, \text{ the current substitution, initially the empty substitution } ()\]
\[\text{local variables: } \text{answers}, \text{ a set of substitutions, initially empty}\]

\[\text{if goals is empty then return } (\theta)\]
\[\theta' \leftarrow \text{Subst}(\theta, \text{First}(\text{goals}))\]
\[\text{for each sentence } r \text{ in } KB\]
\[\quad \text{where } \text{Standardize-Apart}(r) = (p_1 \land \ldots \land p_n \Rightarrow q)\]
\[\quad \text{and } \theta' \leftarrow \text{UNIFY}(q, \theta) \text{ succeeds}\]
\[\quad \text{new_goals} \leftarrow \left\{ p_1, \ldots, p_n | \text{REST}(\text{goals}) \right\}\]
\[\quad \text{answers} \leftarrow \text{FOL-BC-Ask}(KB, \text{new_goals}, \text{COMPOSE}(\theta', \theta)) \cup \text{answers}\]
\[\text{return } \text{answers}\]
### Backward chaining example

- **Criminal(West)**
  - **American(West)**
    - **Weapon(y)**
      - **Sells(West,M1,z)**
        - **Hostile(z)**
      - **Missile(y)**
  - **Missile(M1)**

### 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**
- **Ordinary 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

Should be easier to debug `Capital[NewYork,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 \( \Rightarrow \text{roughly a billion LIPS on workstation} \)
- Program = set of clauses = head :- literal1, ... literaln.
  
  ```prolog
  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., \( X \ldots Y+Z+3 \)
  - 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 $\alpha, \beta, \gamma$:
\[
\alpha \models \gamma \quad \text{then} \quad \alpha \land \beta \models \gamma
\]
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, $\text{alive}(joe)$ is a consequence
- with $\text{dead}(joe)$ in KB, $\text{alive}(joe)$ 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

**Digression: monotonicity**

**Conversion to CNF**

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

1. Eliminate biconditionals and implications
\[
\forall x \ (\neg \forall y \ (-\text{Animal}(y) \lor \text{Loves}(x,y)) \lor (\exists y \ \text{Loves}(y,x))
\]

2. Move $\neg$ inwards: $\neg \forall x, p \equiv \exists x \neg p, \neg \exists x, p \equiv \forall x \neg p$
\[
\forall x \ (\exists y \ (\neg \text{Animal}(y) \lor \text{Loves}(x,y)) \lor (\exists y \ \text{Loves}(y,x))
\]

3. Standardize variables: each quantifier should use a different one
\[
\forall x \ (\exists y \ (\neg \text{Animal}(y) \lor \text{Loves}(x,y)) \lor (\exists y \ \text{Loves}(y,x))
\]

4. Skolemize: a more general form of existential instantiation.
Each existential variable is replaced by a Skolem function of the enclosing universally quantified variables:
\[
\forall x \ (\text{Animal}(F(x)) \land \neg \text{Loves}(x,F(x)) \lor \text{Loves}(G(x),x)
\]

5. Drop universal quantifiers:
\[
\forall x \ (\text{Animal}(F(x)) \land \neg \text{Loves}(x,F(x)) \lor \text{Loves}(G(x),x)
\]

6. Distribute $\land$ over $\lor$:
\[
\forall x \ (\text{Animal}(F(x)) \land \neg \text{Loves}(x,F(x)) \lor \text{Loves}(G(x),x)
\]

**Conversion to CNF contd.**

By standard methods:
\[
\exists y \ (\text{Animal}(y) \land \neg \text{Loves}(x,y) \lor (\exists y \ \text{Loves}(y,x))
\]

Full first-order version:
\[
\ell_1 \lor \cdots \lor \ell_k, \ m_1 \lor \cdots \lor m_n
\]

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

For example:
\[
\neg \text{Silent}(x) \lor \text{Supreme}(x)
\]
\[
\text{Silent(Nono)} \land \text{Supreme(Nono)}
\]

with $\emptyset = \{x/\text{Nono}\}$

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