Inference in first-order logic

Chapter 9

Outline

• Reducing first-order inference to propositional inference
• Unification
• Generalized Modus Ponens
• Forward chaining
• Backward chaining
• 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., \( \forall x \ King(x) \land \text{Greedy}(x) \Rightarrow \text{Evil}(x) \)
    yields:
    \[
    \text{King}(\text{John}) \land \text{Greedy}(\text{John}) \Rightarrow \text{Evil}(\text{John})
    \]
    \[
    \text{King}(\text{Richard}) \land \text{Greedy}(\text{Richard}) \Rightarrow \text{Evil}(\text{Richard})
    \]
    \[
    \text{King}(\text{Father}(\text{John})) \land \text{Greedy}(\text{Father}(\text{John})) \Rightarrow \text{Evil}(\text{Father}(\text{John}))
    \]

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., \( \exists x \ Crown(x) \land \text{OnHead}(x, \text{John}) \)
    yields:
    \[
    \text{Crown}(C_1) \land \text{OnHead}(C_1, \text{John})
    \]
    provided \( C_1 \) is a new constant symbol, called a Skolem constant

Reduction to propositional inference

Suppose the KB contains just the following:

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

• Instantiating the universal sentence in all possible ways, we have:
  \[
  \text{King}(\text{John}) \land \text{Greedy}(\text{John}) \Rightarrow \text{Evil}(\text{John})
  \]
  \[
  \text{King}(\text{Richard}) \land \text{Greedy}(\text{Richard}) \Rightarrow \text{Evil}(\text{Richard})
  \]
  \[
  \text{King}(\text{Father}(\text{John})) \land \text{Greedy}(\text{Father}(\text{John})) \Rightarrow \text{Evil}(\text{Father}(\text{John}))
  \]

• The new KB is propositionalized: proposition symbols are
  \[
  \text{King}(\text{John}), \text{Greedy}(\text{John}), \text{Evil}(\text{John}), \text{King}(\text{Richard}), \text{etc.}
  \]

Reduction contd.

• Every FOL KB can be propositionalized so as to preserve entailment

  - (A ground sentence is entailed by new KB iff entailed by original KB)

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

  - Problem: with function symbols, there are infinitely many ground terms,
    - e.g., \( \text{Father}(\text{Father}(\text{Father}(\text{John}))) \)
**Reduction contd.**

Theorem: Herbrand (1930). If a sentence \( \alpha \) is entailed by an FOL KB, it is entailed by a finite subset of the propositionalized 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 for FOL is semidecidable (algorithms exist that say yes to every entailed sentence, but no algorithm exists that also says no to every nonentailed sentence.)

---

**Problems with propositionalization**

- Propositionalization seems to generate lots of irrelevant sentences.
- E.g., from:
  \[
  \forall x \text{ King}(x) \land \text{ Greedy}(x) \Rightarrow \text{ Evil}(x)
  \]
  King(John)
  \[
  \forall y \text{ Greedy}(y) \quad \text{Brother}(\text{Richard}, \text{John})
  \]
- It seems obvious that \( \text{ Evil}(\text{John}) \), but propositionalization produces lots of facts such as \( \text{ Greedy}(\text{Richard}) \) that are irrelevant.
- With \( p \)-ary predicates and \( n \) constants, there are \( p^n \) instantiations.

---

**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 \( \text{ Greedy}(\text{y}) \):

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

\[
\alpha \land \beta
\]

\[
\theta
\]

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

Knows(\text{John}, \text{x})
Knows(\text{John}, \text{y})
Knows(\text{y}, \text{Mother}(\text{y}))
Knows(\text{y}, \text{OJ})
Knows(\text{x}, \text{OJ})

Standardizing apart eliminates overlap of variables, e.g.,
Knows(\text{z}, \text{OJ})

---

**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 \( \text{ Greedy}(\text{y}) \):

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

\[
\alpha \land \beta
\]

\[
\theta
\]

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

Knows(\text{John}, \text{x})
Knows(\text{John}, \text{y})
Knows(\text{y}, \text{Mother}(\text{y}))
Knows(\text{y}, \text{OJ})
Knows(\text{x}, \text{OJ})

Standardizing apart eliminates overlap of variables, e.g.,
Knows(\text{z}, \text{OJ})
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\((y)\)

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

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

\[\begin{array}{|c|c|}
\hline
p & q \\
\hline
\text{Knows}(John,x) & \text{Knows}(John,\text{Jane}) \\
\text{Knows}(John,x) & \text{Knows}(y,\text{OU}) \\
\text{Knows}(John,x) & \text{Knows}(y,\text{Mother}(y)) \\
\text{Knows}(John,x) & \text{Knows}(x,\text{OU}) \\
\hline
\end{array}\]

• Standardizing apart eliminates overlap of variables, e.g.,

\[ \text{Knows}(z,x,\text{OU}) \]

The unification algorithm

\[ \text{function } \text{UNIFY}(\alpha, \beta) \text{ returns a substitution} \]

\[ \text{inputs: } \alpha, \beta \text{ variable, } \]

\[ \text{set } \theta = \emptyset, \text{ the substitution built up so far} \]

\[ \text{if } (\alpha/\emptyset) \subseteq \emptyset \text{ then return } \text{UNIFY}(\alpha, \emptyset) \]

\[ \text{else if } (\emptyset/\emptyset) \subseteq \emptyset \text{ then return } \text{UNIFY}(\emptyset, \emptyset) \]

\[ \text{else return failure} \]

Generalized Modus Ponens (GMP)

\[ p_1', \ldots, p_n', (p_1 \wedge p_2 \wedge \ldots \wedge p_n \implies q) \]

\[ \text{where } p_i \theta = p_i \text{ for all } i \]

\[ \text{p}_1' \text{ is King}(John) \quad p_i \text{ is King}(x) \]

\[ \text{p}_2' \text{ is Greedy}(y) \quad p_i \text{ is Greedy}(x) \]

\[ \theta = \{ x/John, y/John \} \quad q \text{ is Evil}(x) \]

\[ q \theta \text{ is Evil}(John) \]

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

• All variables assumed universally quantified

The unification algorithm

\[ \text{function } \text{UNIFY}(x, \theta) \text{ returns a substitution to make } x \text{ and } y \text{ identical} \]

\[ \text{inputs: } x, y \text{ variable, constant, list, or compound} \]

\[ \text{set } \theta = \emptyset, \text{ the substitution built up so far} \]

\[ \text{if } \theta = \emptyset \text{ then return failure} \]

\[ \text{else if } x = y \text{ then return } \theta \]

\[ \text{else if VARIABLE}(x) \text{ then return } \text{UNIFY-Var}(x, \theta) \]

\[ \text{else if VARIABLE}(y) \text{ then return } \text{UNIFY-Var}(y, \theta) \]

\[ \text{else if COMPOUND}(x) \text{ and COMPOUND}(y) \text{ then return } \text{UNIFY-Var}(x, \theta) \]

\[ \text{else if } \text{COMPOUND}(x) \text{ then return failure} \]

\[ \text{else if } \text{COMPOUND}(y) \text{ then return failure} \]

\[ \text{else return } \text{UNIFY-Var}(x, \theta) \]

Soundness of GMP

• Need to show that

\[ p_1', \ldots, p_n', (p_1 \wedge p_2 \wedge \ldots \wedge p_n \implies q) \lor q \theta \]

provided that \( p_i \theta = p_i \text{ for all } i \)

• Lemma: For any sentence \( p \), we have \( p \lor \theta \) by UI

1. \((p_1 \wedge \ldots \wedge p_n \implies q) \lor (p_1 \wedge \ldots \wedge p_n \implies q) \theta = (p_1 \theta \wedge \ldots \wedge p_n \theta \implies q) \theta\)

2. \((p_1 \wedge \ldots \wedge p_n \implies q) \lor (p_1 \wedge \ldots \wedge p_n \implies q) \theta = (p_1 \theta \wedge \ldots \wedge p_n \theta \implies q) \theta\)

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:

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

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

\(\text{Owns}(\text{Nono},M)\) and \(\text{Missile}(M)\)

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

\(\text{Missile}(y) \land \text{Owns}(\text{Nono},y) \implies \text{Sells}(\text{West},y,\text{Nono})\)

Missiles are weapons:

\(\text{Missile}(x) \implies \text{Weapon}(x)\)

An enemy of America counts as "hostile":

\(\text{Enemy}(x,\text{America}) \implies \text{Hostile}(x)\)

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-Alt(KB, α) returns a substitution or false:

repeat until new is empty

for each sentence r in KB do

for each q such that \((p_1 \land \ldots \land p_n \land q) \in \text{STANDARDIZE-APART}(r)\) for some \(p_1, \ldots, p_n, q\) in KB

\(q' = \text{UNIFY}(q, \alpha)\)

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

add \(q'\) to new

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

add new to KB

return false

Forward chaining proof

American(West)
Hostile(M1)
Owns(Nono,M1)
Missile(M1)

American(Nono)
Hostile(M1)

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

Enemy(Nono, America)

Forward chaining proof

American(West)
Hostile(M1)

American (Nono)

American (West)

American (Nono)
Missile(M1)

Enemy(Nono, America)
Properties of forward chaining

- Sound and complete for first-order definite clauses
- **Datalog** = first-order definite clauses + no functions
- FC terminates for Datalog in finite number of iterations
- May not terminate in general if α is not entailed
- This is unavoidable: entailment with definite clauses is semidecidable

Efficiency of forward chaining

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 positive literal

Matching itself can be expensive:

Database indexing allows O(1) retrieval of known facts

- e.g., query Missile(x) retrieves Missile(M1)

Forward chaining is widely used in deductive databases

Hard matching example

- **Colorable()** 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-Ans(LKB goal, F) returns a set of substitutions
inputs: LKB, a knowledge base
        goal, a list of conjuncts forming a query
        F, the current substitution, initially the empty substitution {}
local variables: α, a set of substitutions, initially empty
if goal is empty then return [F]
F = FOL-BC-Ans(F, goal)
for each α in F when SEMANTIC-ASSIGN(α) = (\(p_1 \land \ldots \land p_n \Rightarrow q\))
        α = FOL-BC-Ans(LKB, (\(p_1 \ldots p_n\) \{REST(\{goal\})\} ∪ α))
return α

SUBST(COMPOSE(\(θ_1\), \(θ_2\)), p) = SUBST(\(θ_2\),
        SUBST(\(θ_1\), p))
```

Backward chaining example

```
Near(Wa, Nt) \land Near(Wa, Sa) \land Near(Nt, Q)
\land Near(Nt, Sa)
\land Near(Q, Nsw)
\land Near(Q, Sa)
\land Near(Nsw, V)
\land Near(Nsw, Sa)
\land Near(V, Sa)
⇒ Colorable()
Diff(Red, Blue) \land Diff(Red, Green)
Diff(Green, Red) \land Diff(Green, Blue)
Diff(Blue, Red) \land Diff(Blue, Green)
```

Backward chaining example
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 subgoals (both success and failure)
  - \( \Rightarrow \) fix using caching of previous results (extra space)
- Widely used for logic programming

Logic programming: Prolog

- Algorithm = Logic + Control
- Basis: backward chaining with Horn clauses + bells & whistles
  Widely used in Europe, Japan (basis of 5th Generation project)
  Compilation techniques \( \Rightarrow \) 60 million LIPS
- Program = set of clauses = head :- literal, .. literal.
  - e.g., \( X \) is \( Y \)?
- Built-in predicates for arithmetic etc., e.g., \( X \) is \( Y \)?
  - Built-in predicates that have side effects (e.g., input and output
  - predicates, assert/retract predicates)
- Depth-first, left-to-right backward chaining
- Closed-world assumption ("negation as failure")
  - e.g., \( \\text{alive(joe)} \) succeeds if \( \text{dead(joe)} \) fails

Conversion to CNF

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

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

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

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

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

6. Distribute \( \lor \) over \( \land \):
   \( [\text{Animal}(F(x)) \lor \text{Loves}(G(x),x)] \lor [\neg \text{Loves}(F(x),x) \lor \text{Loves}(G(x),x)] \)
Resolution proof: definite clauses