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

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 \(g\)

E.g., \(\forall \, x \, \text{King}(x) \land \text{Greedy}(x) \Rightarrow \text{Evil}(x)\) yields

\[
\text{King}(\text{John}) \land \text{Greedy}(\text{John}) \Rightarrow \text{Evil}(\text{John}) \\
\text{King}(\text{Richard}) \land \text{Greedy}(\text{Richard}) \Rightarrow \text{Evil}(\text{Richard}) \\
\text{King}(\text{Father}(\text{John})) \land \text{Greedy}(\text{Father}(\text{John})) \Rightarrow \text{Evil}(\text{Father}(\text{John}))
\]

Chapter 9

Existential instantiation (EI)

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

\[
\exists \, v \, \alpha \\
\text{Subst}(\{v/k\}, \alpha)
\]

E.g., \(\exists \, x \, \text{Crown}(x) \land \text{OnHead}(x, \text{John})\) yields

\[
\text{Crown}(\text{C}_1) \land \text{OnHead}(\text{C}_1, \text{John})
\]

provided \(\text{C}_1\) is a new constant symbol, called a Skolem constant

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

\[
d(e^v)/dy = e^v
\]

provided \(e\) is a new constant symbol

Chapter 9

A brief history of reasoning

450s. 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

Chapter 9

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
Reduction to propositional inference

Suppose the KB contains just the following:

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

Instantiating the universal sentence in all possible ways, we have

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

The new KB is propositionalized: proposition symbols are

\[ \text{King}(\text{John}), \text{Greedy}(\text{John}), \text{Evil}(\text{John}), \text{King}(\text{Richard}) \]

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

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

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

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 \( p \cdot n^k \) instantiations

With function symbols, it gets much much worse!
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 \( \alpha\theta = \beta\theta \)

```
K1n(x)   K1n(y)
K1n(x)   G1d(y)
K1n(y)   K1n(y)
K1n(y)   G1d(y)
K1n(x)   G1d(x)
K1n(x)   G1d(x)
```

\( \text{K1n}(x) \) fail

\( \text{Standardizing apart} \) eliminates overlap of variables, e.g., \( \text{K1n}(z_{17}, OJ) \)

Generalized Modus Ponens (GMP)

\( p_1', p_2', \ldots, p_n', (p_1 \land \ldots \land p_n \Rightarrow q) \Rightarrow q\theta \)

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

\( p_1' \) is \( \text{K1n}(\text{John}) \)
\( p_2' \) is \( \text{K1n}(\text{John}) \)
\( \theta \) is \( \{ x/\text{John}, y/\text{John} \} \)
\( q\theta \) is \( \text{Evil}(\text{John}) \)

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

All variables assumed universally quantified

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

Soundness of GMP

Need to show that

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

provided that \( p_i' = p_i\theta \) 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. \( p_1', \ldots, p_n' \models p_1' \land \ldots \land p_n' \models p_1' \land \ldots \land p_n' \theta \)
3. From 1 and 2, \( q\theta \) follows by ordinary Modus Ponens

Example knowledge base contd.

\( \ldots \) 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
\[ \exists x \ \text{Owns}(\text{Nono}, x) \land \text{Missile}(x) \]
\[ \text{Missiles are weapons:} \ \\
\forall x \ \text{Missile}(x) \Rightarrow \text{Weapon}(x) \]
An enemy of America counts as “hostile”:
\[ \text{Enemy}(\text{Nono}, \text{America}) \Rightarrow \text{Hostile}(x) \]
West, who is American ...
\[ \text{American}(\text{West}) \]
The country Nono, an enemy of America ...
\[ \text{Enemy}(\text{Nono}, \text{America}) \]

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., \( \exists x \ \text{Owns}(\text{Nono}, x) \land \text{Missile}(x) \)
\[ \text{Missiles are weapons:} \ \\
\forall x \ \text{Missile}(x) \Rightarrow \text{Weapon}(x) \]
An enemy of America counts as “hostile”:
\[ \text{Enemy}(\text{Nono}, \text{America}) \Rightarrow \text{Hostile}(x) \]
West, who is American ...
\[ \text{American}(\text{West}) \]
The country Nono, an enemy of America ...
\[ \text{Enemy}(\text{Nono}, \text{America}) \]

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., \( \exists x \ \text{Owns}(\text{Nono}, x) \land \text{Missile}(x) \)
\[ \text{Missiles are weapons:} \ \\
\forall x \ \text{Missile}(x) \Rightarrow \text{Weapon}(x) \]
An enemy of America counts as “hostile”:
\[ \text{Enemy}(\text{Nono}, \text{America}) \Rightarrow \text{Hostile}(x) \]
West, who is American ...
\[ \text{American}(\text{West}) \]
The country Nono, an enemy of America ...
\[ \text{Enemy}(\text{Nono}, \text{America}) \]

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., \( \exists x \ \text{Owns}(\text{Nono}, x) \land \text{Missile}(x) \)
\[ \text{Missiles are weapons:} \ \\
\forall x \ \text{Missile}(x) \Rightarrow \text{Weapon}(x) \]
An enemy of America counts as “hostile”:
\[ \text{Enemy}(\text{Nono}, \text{America}) \Rightarrow \text{Hostile}(x) \]
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 \text{FOL-FC-Ask}(\text{KB}, \alpha) \ returns a substitution or \text{false} \n
repeat until \ new \ is empty \n
\ new \ = \ \ \{\ \} \n
for each sentence \( r \) in \( \text{KB} \) do \n
\( (p_1 \land \ldots \land p_n) \Rightarrow q \) \ 

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 \( \text{KB} \) \ 

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

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

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

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

if \( \phi \) is not \text{false} then return \( \phi \) \ 

add \( \ new \) to \( \text{KB} \) \ 

return \text{false}
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
- May not terminate in general 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

Hard matching example

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

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

$\text{Colorable}()$ is inferred iff the CSP has a solution
CSPs include 3SAT as a special case, hence matching is NP-hard
function $\text{FOL-BC-Ask}(KB, goals, \theta)$ 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

new goals $\leftarrow [p_1, \ldots, p_n, \text{Rest}(goals)]$

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

return answers

---

### Backward chaining example

Introduction to algorithm with example sentences:

- Criminal(West)
- Weapon(y)
- Sells(x, y, z)
- Hostile(z)

Applying substitutions:

- $\{x/\text{West}\}$
- $\{y/M1\}$

Diagram illustrative of the chaining process.
Backward chaining example

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

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 :- literal, ... literal.

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).
alive(joe) succeeds if dead(joe) fails

Logic programming

Sound bite: computation as inference on logical KBs
Logic programming

1. Identify problem
2. Assemble information
3. Tree 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!

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=[]
Resolution: brief summary

Full first-order version:
\[ \ell_1 \lor \cdots \lor \ell_k \lor m_1 \lor \cdots \lor m_n \theta \]
where \( \text{UNIFY}(\ell_i, m_j) = \theta \).

For example,
\[ \neg \text{Rich}(x) \lor \text{Unhappy}(x) \]
\[ \text{Rich}(\text{Ken}) \]
\[ \text{Unhappy}(\text{Ken}) \]
with \( \theta = \{x/\text{Ken}\} \)

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:
\[ \forall x [ \forall y \text{Animal}(y) \Rightarrow \text{Loves}(x, y)] \Rightarrow [ \exists y \text{Loves}(y, x)] \]

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

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

Conversion to CNF contd.

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

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

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

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