Inference in first-order logic

Russell and Norvig Chapter 9

Outline
- Reducing first-order inference to propositional inference
- Generalized Modus Ponens
- Forward chaining
- Backward chaining
- Resolution

FOL to PL
- First order inference can be done by converting the knowledge base to PL and using propositional inference.
  - How to convert universal quantifiers?
    - Replace variable by ground term.
  - How to convert existential quantifiers?
    - Skolemization.

Universal instantiation (UI)
Every instantiation of a universally quantified sentence is entailed by it:
\[ \forall v \alpha \Rightarrow \text{Subst}({v/g}, \alpha) \]
for any variable \( v \) and ground term \( g \)

E.g., \( \forall x \text{King}(x) \land \text{Greedy}(x) \Rightarrow \text{Evil}(x) \) yields:
\[ \text{King}(\text{John}) \land \text{Greedy}(\text{John}) \Rightarrow \text{Evil}(\text{John}) \]
\[ \text{King}((\text{Father}(\text{John}))) \land \text{Greedy}((\text{Father}(\text{John}))) \Rightarrow \text{Evil}(\text{Father}(\text{John})) \]

Existential instantiation (EI)
For any sentence \( \alpha \), variable \( v \), and constant symbol \( k \) that does not appear elsewhere in the knowledge base:
\[ \exists v \alpha \Rightarrow \text{Subst}({v/k}, \alpha) \]

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

EI versus UI
- 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 if 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:
  \[ \text{King}(\text{John}) \land \text{Greedy}(\text{John}) \Rightarrow \text{Evil}(\text{John}) \]
  \[ \text{King}(\text{Richard}) \land \text{Greedy}(\text{Richard}) \Rightarrow \text{Evil}(\text{Richard}) \]

- The new KB is propositionalized

Reduction (cont)

- CLAIM: A ground sentence is entailed by the new KB iff entailed by the 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., \( \text{Father}(\text{Father}(\text{John})) \)

Reduction (cont)

- 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 \( n = \infty \) 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, does not halt if \( \alpha \) is not entailed
- THEOREM: Turing (1936), Church (1936). Entailment for FOL is semi-decidable.
  - algorithms exist that say yes to every entailed sentence, but no algorithm exists that says no to every non-entailed sentence.
  - With \( k \)-ary predicates and \( n \) constants, there are \( p^n \) instantiations!

Is there another way?

- Instead of translating the knowledge base to PL, we can make the inference rules work in FOL.
- For example, given
  \[ \forall x \text{King}(x) \land \text{Greedy}(x) \Rightarrow \text{Evil}(x) \]
  \[ \text{King}(\text{John}) \]
  \[ \forall y \text{Greedy}(y) \]

  It is intuitively clear that we can substitute \( \{ x/\text{John}, y/\text{John} \} \) and obtain that \( \text{Evil}(\text{John}) \)

Unification

- We can make the inference if we can find a substitution such that \( \text{King}(x) \) and \( \text{Greedy}(y) \) match \( \text{King}(\text{John}) \) and \( \text{Greedy}(y) \) (\( \{ x/\text{John}, y/\text{John} \} \) works)

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

| \( \text{Knows}(\text{John}, x) \) | \( \text{Knows}(\text{John}, \text{Jane}) \) | \( \text{Knows}(\text{John}, \text{Jane}) \) | \( \theta \text{ Subst} \) |
| \text{Knows}(\text{John}, x) | \text{Knows}(\text{John}, \text{Jane}) | \text{Knows}(\text{John}, \text{Jane}) | \theta \text{ Subst} |
| \text{Knows}(\text{John}, x) | \text{Knows}(\text{John}, \text{Mother}(\text{y})) | \text{Knows}(\text{John}, \text{Mother}(\text{y})) | \theta \text{ Subst} |
| \text{Knows}(\text{John}, x) | \text{Knows}(\text{John}, \text{OJ}) | \text{Knows}(\text{John}, \text{OJ}) | \theta \text{ Subst} |

Unification

- We can make the inference if we can find a substitution such that \( \text{Knows}(x) \) and \( \text{Knows}(y) \) match \( \text{Knows}(\text{John}) \) and \( \text{Knows}(\text{y}) \) (\( \{ x/\text{John}, y/\text{John} \} \) works)

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

| \( \text{Knows}(\text{John}, x) \) | \( \text{Knows}(\text{John}, \text{Jane}) \) | \( \text{Knows}(\text{John}, \text{Jane}) \) | \( \theta \text{ Subst} \) |
| \text{Knows}(\text{John}, x) | \text{Knows}(\text{John}, \text{Jane}) | \text{Knows}(\text{John}, \text{Jane}) | \theta \text{ Subst} |
| \text{Knows}(\text{John}, x) | \text{Knows}(\text{John}, \text{OJ}) | \text{Knows}(\text{John}, \text{OJ}) | \theta \text{ Subst} |

Turing (1936), Church (1936). Entailment for FOL is semi-decidable.

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

- We can make the inference if we can find a substitution such that King(x) and Greedy(y)
  match King(John) and Greedy(y)
  {x/John, y/John} works

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

Unify:

<table>
<thead>
<tr>
<th>( \alpha )</th>
<th>( \beta )</th>
<th>Subst</th>
</tr>
</thead>
<tbody>
<tr>
<td>Knows(John, x)</td>
<td>Knows(John, Jane)</td>
<td>{x/John}</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(Mother(y))</td>
<td>{y/John, x/Mother(John)}</td>
</tr>
<tr>
<td>Knows(John, x)</td>
<td>Knows(y, OJ)</td>
<td>{fail}</td>
</tr>
</tbody>
</table>

We can unify the last row if we keep the variable names unique.

Unification

- We can make the inference if we can find a substitution such that King(x) and Greedy(y)
  match King(John) and Greedy(y)
  \( \{x/John, y/John\} \) works

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

Unifiers of Knows(John, x) and Knows(y, z) are

\( \{y/John, x/z\} \) or \( \{y/John, x/John, z/John\} \)

The first unifier is more general than the second.

There is a single most general unifier (MGU) that is unique up to renaming of variables.

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

The unification algorithm

Function \( \text{Unify}(\alpha, \beta) \) returns a substitution to make \( \alpha \) and \( \beta \) identical

- \( \alpha \) a variable, constant, list, or compound
- \( \beta \) a variable, constant, list, or compound
- \( \text{Subst} \) the substitution built up so far

\[ \text{if } \alpha = \beta \text{ then return } \emptyset \]

\[ \text{else if } \alpha \text{ is a variable then return } \text{Unify-Var}(\alpha, \beta) \]

\[ \text{else if } \alpha \text{ is a variable and } \beta \text{ is a constant then return } \text{Unify-Var}(\alpha, \beta) \]

\[ \text{else if } \alpha \text{ is a variable and } \beta \text{ is a list then return } \text{Unify-Var}(\alpha, \beta) \]

\[ \text{else if } \alpha \text{ is a variable and } \beta \text{ is a compound then return } \text{Unify-Var}(\alpha, \beta) \]

\[ \text{else if } \alpha \text{ is a list and } \beta \text{ is a constant then return } \text{Unify-Lst}(\alpha, \beta) \]

\[ \text{else if } \alpha \text{ is a list and } \beta \text{ is a list then return } \text{Unify-Lst}(\alpha, \beta) \]

\[ \text{else if } \alpha \text{ is a compound and } \beta \text{ is a constant then return } \text{Unify-Lst}(\alpha, \beta) \]

\[ \text{else if } \alpha \text{ is a compound and } \beta \text{ is a list then return } \text{Unify-Lst}(\alpha, \beta) \]

\[ \text{else if } \alpha \text{ is a compound and } \beta \text{ is a compound then return } \text{Unify-Lst}(\alpha, \beta) \]

\[ \text{else if } \alpha \text{ is a constant and } \beta \text{ is a constant then return } \text{Unify-Const}(\alpha, \beta) \]

\[ \text{else if } \alpha \text{ is a constant and } \beta \text{ is a list then return } \text{Unify-Const}(\alpha, \beta) \]

\[ \text{else if } \alpha \text{ is a list and } \beta \text{ is a constant then return } \text{Unify-Const}(\alpha, \beta) \]

\[ \text{else if } \alpha \text{ is a list and } \beta \text{ is a list then return } \text{Unify-Const}(\alpha, \beta) \]

\[ \text{else return } \text{Unify-Var}(\alpha, \beta) \]

\[ \text{Unify-Var}(\alpha, \beta) \]

- \( \alpha \) a variable
- \( \beta \) a variable
- \( \text{Subst} \) the substitution

\[ \text{if } \text{Subst}(\alpha, \beta) \text{ then return } \text{Unify-Var}(\alpha, \beta) \]

\[ \text{else if } \text{varCheck}(\alpha, \beta) \text{ then return } \text{Unify-Var}(\alpha, \beta) \]

\[ \text{else if } \text{varCheck}(\alpha, \beta) \text{ and } \text{varCheck}(\beta, \alpha) \text{ then return } \text{Unify-Var}(\alpha, \beta) \]

\[ \text{else if } \text{varCheck}(\alpha, \beta) \text{ and } \text{varCheck}(\beta, \alpha) \text{ then return } \text{Unify-Var}(\alpha, \beta) \]

\[ \text{else return } \emptyset \]
Generalized Modus Ponens (GMP)

Suppose that \(\text{Subst}(n, p_i') = \text{Subst}(n, p_i)\) for all \(i\) then:

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

\(p_1'\) is \(\text{King}(\text{John})\)
\(p_2'\) is \(\text{Greedy}(y)\)
\(n\) is \((x, \text{John}, y, \text{John})\)
\(q\) is \(\text{Evil}(x)\)

\(\text{Subst}(n, q)\) is \(\text{Evil}(\text{John})\)

- All variables assumed universally quantified.

Example

- 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

- 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

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

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

... it is a crime for an American to sell weapons to hostile nations:
American(x) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) ⇒ Criminal(x)
Nono ... has some missiles, i.e., ∃x Owns(Nono,x) ∧ Missile(x):
Owns(Nono,M1) and Missile(M1)
... all of its missiles were sold to it by Colonel West
Missiles are weapons:
Missile(x) ⇒ Weapon(x)

An enemy of America counts as "hostile":
Enemy(No,America) ⇒ Hostile(No)
Example

... 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., \( \exists x \text{Owns}(Nono,x) \land \text{Missile}(x) \):
\[ \text{Owns}(Nono,M_1) \land \text{Missile}(M_1) \]
... all of its missiles were sold to it by Colonel West
\[ \text{Missile}(x) \land \text{Owns}(Nono,x) \Rightarrow \text{Sells}(West,x,Nono) \]
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 ...

Forward chaining algorithm

Function FOL-FC-Allow(KB) returns a substitution or false:
repeat until \( K \) is empty
for each sentence \( r \) in \( K \) do
for each \( x \) such that \( r \) contains \( x \) do
rename variables so there are no collisions
\[ \text{Simplify}(r,x) \]
if \( r \) is not a rewriting of a sentence already in \( K \) then do
add \( r \) to \( K \)
if \( r \) is not a rewriting of \( r \) then return \( c \)
add \( x \) to \( K \)
return false

Forward chaining example
Forward chaining example

Forward chaining for FOL

- Sound and complete for first-order definite clauses.
- **Datalog** = first-order definite clauses with no functions (e.g., crime KB)
  - FC terminates for Datalog in finite number of iterations
  - May not terminate in general definite clauses with functions if sentence is not entailed
  - This is unavoidable: entailment with definite clauses is semidecidable

Increasing the efficiency of forward chaining

- Incremental forward chaining: no need to match a rule on iteration \( t \)
  if a premise wasn’t added on iteration \( t-1 \)
  - Match each rule whose premise contains a newly added positive literal.

- May not terminate in general definite clauses with functions if sentence is not entailed
  - This is unavoidable: entailment with definite clauses is semidecidable

Connection with CSP

- **Colorable()** is inferred iff the CSP has a solution
- Hardness of inference problem related to the difficulty of the corresponding CSP.

Backward chaining algorithm

```
function FOL-BC-Analyze(KB, goals) return a set of substitutions

Input:
KB: a knowledge base
goals: a list of goals having a query

1. if goals is empty then return \( \emptyset \)
2. for each \( \sigma \) in KB
   3. if \( \text{Forward-Knowledge-Analysis}(\sigma) = \{ \sigma_1, \ldots, \sigma_n \} \)
      4. if \( \text{Unify}(\sigma, \sigma_i) \) succeeds
         5. foreach \( \sigma_j \) in \( \text{FOL-BC-Analyze}(KB, \text{goals}) \) return \( \text{Compose}(\sigma, \sigma_j) \)
```

\( \{ \sigma_1, \ldots, \sigma_n \} \) is the list of new goals (FIRST(goals) already addressed)
Compose is the effect of sequentially applying two substitutions
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
  - fix by caching previous results (extra space!!)
- Widely used for logic programming: problem solving by inference.
  - Example: Prolog

Logic programming: Prolog
- BASIS: backward chaining with Horn clauses + bells & whistles
- Program = set of clauses of the form
  \( \text{head} : \text{literal}_1, \ldots, \text{literal}_n \)
  - criminal(X) \( \iff \) american(X), weapon(Y), sells(X,Y,Z), hostile(Z).

Resolution
- Full first-order version:
  \( \lnot \varphi \lor \varphi \lor \cdots \lor \varphi \lor \phi \)
  \( \text{where } \theta = \text{Unify}(\varphi, \neg \phi) \)
- The two clauses are assumed to be standardized apart so that they share no variables.
- For example,
  \( \neg \text{Rich}(x) \lor \text{Unhappy}(x), \text{Rich}(x) \text{Unhappy}(x) \)
  with \( \theta = \{x/Ken\} \)
- Apply resolution steps to CNF(KB \( \land \neg \alpha \)); complete for FOL

Conversion to CNF
- Need to convert KB to CNF, eliminating existential quantifiers
- Consider the sentence “There is someone who is loved by everyone.”
  \( \exists y \forall x \text{ Loves}(x, y) \)
  Let’s name that someone using a constant that does not appear elsewhere in the KB (Skolem constant):
  \( y \forall x \text{ Loves}(x, Sk1) \)
- Let’s try now the sentence “Everyone is loved by someone”
  \( y \exists x \text{ Loves}(x, y) \)
  Skolemization using a constant:
  \( y \text{ Loves}(Sk2, y) \)
- This doesn’t work, but skolemization using a function does:
  \( y \text{ Loves}(Sk2(y), y) \)
**Conversion to CNF**

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

- Skolemize: Each existential variable is replaced by a Skolem function of the enclosing universally quantified variables:
  \[ \forall x \left[ \text{Animal}(F(x)) \land \neg \text{Loves}(x,F(x)) \right] \lor \text{Loves}(G(x),x) \]

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

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

**Theorem provers**

- Theorem prover – a system that does full first order logic inference (using resolution)

- Uses:
  - Prove mathematical theorems (known successes!)
  - Hardware/software verification
    - Verify that circuit/software produces correct output for all possible inputs (RSA algorithm was verified this way)