Vorlesung

Grundlagen der Künstlichen Intelligenz

Reinhard Lafrenz / Prof. A. Knoll

Robotics and Embedded Systems
Department of Informatics – I6
Technische Universität München

www6.in.tum.de
lafrenz@in.tum.de
089-289-18136
Room 03.07.055

Wintersemester 2012/13

30.11.2012
Chapter 8/9 (3rd ed.)

First-Order Logic and Inference
Knowledge engineering in FOL

1. Identify the task
2. Assemble the relevant knowledge
3. Decide on a vocabulary of predicates, functions, and constants
4. Encode general knowledge about the domain
5. Encode a description of the specific problem instance
6. Pose queries to the inference procedure and get answers
7. Debug the knowledge base
The electronic circuits domain

One-bit full adder
The electronic circuits domain

1. Identify the task
   - Does the circuit actually add properly? (circuit verification)

2. Assemble the relevant knowledge
   - Composed of wires and gates; Types of gates (AND, OR, XOR, NOT)
   - Irrelevant: size, shape, color, cost of gates

3. Decide on a vocabulary
   - Type(X1) = XOR (or Type(X1, XOR) or XOR(X1))
The electronic circuits domain

4. Encode general knowledge of the domain
   Assume t, t₁,t₂ are terminals, i.e. Terminal (t₁) ...

   - \( \forall t₁,t₂ \) Connected\( (t₁, t₂) \) \( \Rightarrow \) Signal\( (t₁) = \) Signal\( (t₂) \)
   - \( \forall t \) Signal\( (t) = 1 \lor \) Signal\( (t) = 0 \)
   - \( 1 \neq 0 \)

   - \( \forall t₁,t₂ \) Connected\( (t₁, t₂) \) \( \Rightarrow \) Connected\( (t₂, t₁) \)

   - \( \forall g \) Type\( (g) = \text{OR} \) \( \Rightarrow \) Signal\( (\text{Out}(1,g)) = 1 \iff \exists n \) Signal\( (\text{In}(n,g)) = 1 \)
   - \( \forall g \) Type\( (g) = \text{AND} \) \( \Rightarrow \) Signal\( (\text{Out}(1,g)) = 0 \iff \exists n \) Signal\( (\text{In}(n,g)) = 0 \)

   - \( \forall g \) Type\( (g) = \text{XOR} \) \( \Rightarrow \) Signal\( (\text{Out}(1,g)) = 1 \iff \) Signal\( (\text{In}(1,g)) \neq \) Signal\( (\text{In}(2,g)) \)

   - \( \forall g \) Type\( (g) = \text{NOT} \) \( \Rightarrow \) Signal\( (\text{Out}(1,g)) \neq \) Signal\( (\text{In}(1,g)) \)
The electronic circuits domain

5. Encode the specific problem instance

Type($X_1$) = XOR
Type($X_2$) = XOR
Type($A_1$) = AND
Type($A_2$) = AND
Type($O_1$) = OR

Connected(Out(1,$X_1$),In(1,$X_2$))
Connected(Out(1,$X_1$),In(2,$A_2$))
Connected(Out(1,$A_2$),In(1,$O_1$))
Connected(Out(1,$A_1$),In(2,$O_1$))
Connected(Out(1,$X_2$),Out(1,$C_1$))
Connected(Out(1,$O_1$),Out(2,$C_1$))

Connected(In(1,$C_1$),In(1,$X_1$))
Connected(In(1,$C_1$),In(1,$A_1$))
Connected(In(2,$C_1$),In(2,$X_1$))
Connected(In(2,$C_1$),In(2,$A_1$))
Connected(In(3,$C_1$),In(2,$X_2$))
Connected(In(3,$C_1$),In(1,$A_2$))
The electronic circuits domain

6. Pose queries to the inference procedure
   Which input values lead to “sum bit of $C_1 = 0$ and carry bit of $C_1 = 1$”?
   \[
   \exists i_1, i_2, i_3 \text{ Signal}((1, C_1)) = i_1 \land \text{Signal}((2, C_1)) = i_2 \\
   \land \text{Signal}((3, C_1)) = i_3 \\
   \land \text{Signal}((\text{Out}(1, C_1))) = 0 \land \text{Signal}((\text{Out}(2, C_1))) = 1
   \]

What are the possible sets of values of all the terminals for the adder circuit?
   \[
   \exists i_1, i_2, i_3, o_1, o_2 \text{ Signal}((1, C_1)) = i_1 \land \text{Signal}((2, C_1)) = i_2 \land \\
   \text{Signal}((3, C_1)) = i_3 \land \text{Signal}((\text{Out}(1, C_1))) = o_1 \land \\
   \text{Signal}((\text{Out}(2, C_1))) = o_2
   \]
The electronic circuits domain

7. Debug the knowledge base, example XOR

\[ \exists i_1, i_2, o \quad \text{Signal}(\text{In}(1, C_1)) = i_1 \land \text{Signal}(\text{In}(2, C_1)) = i_2 \land \text{Signal}(\text{Out}(1, C_1)) = X_1 \]

Consider definition of XOR
\[ \text{Signal}(\text{Out}(1, X_1)) = 1 \Leftrightarrow \text{Signal}(\text{Out}(1, X_1)) \neq \text{Signal}(\text{Out}(2, X_1)) \]

Example Input is 0 and 1:
\[ \text{Signal}(\text{Out}(1, X_1)) = 1 \Leftrightarrow 1 \neq 0 \]

May have omitted assertions like \( 1 \neq 0 \), then the statement
\[ \text{Signal}(\text{Out}(1, X_1)) = 1 \] cannot be inferred
Inference in First-Order Logic (chap. 9)

- 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:

\[ \forall v \alpha \Rightarrow \text{Subst}\{\{v/g\}, \alpha\} \]

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:

\[
\exists v \alpha \quad \frac{\text{Subst}\{v/k, \alpha\}}}
\]

- E.g., $\exists x \ Crown(x) \land OnHead(x, John)$ yields:

\[
\text{Crown}(C_1) \land OnHead(C_1, John)
\]

provided $C_1$ is a new constant symbol, called a **Skolem constant**
Reduction to propositional inference

Suppose the KB contains just the following:

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

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.
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., Father(Father(Father(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.

- Example:

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

\[ King(John) \]

\[ \forall y \ Greedy(y) \]

\[ Brother(Richard,John) \]

- it seems obvious that Evil(John), but propositionalization produces lots of facts such as Greedy(Richard) that are irrelevant

- With \( p \) \( k \)-ary predicates and \( n \) constants, there are \( p \cdot n^k \) 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 $\text{Greedy}(y)$

  $\theta = \{x/\text{John}, y/\text{John}\}$ works: $\text{Subst}(\theta, \text{King}(x) \land \text{Greedy}(x) \Rightarrow \text{Evil}(x), \text{Greedy}(y))$

- $\text{Unify}(\alpha, \beta) = \theta$ if $\text{Subst}(\theta, \alpha) = \text{Subst}(\theta, \beta)$

<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>${x/Jane}$</td>
</tr>
<tr>
<td>Knows(John,x)</td>
<td>Knows(y, Bill)</td>
<td>${x/Bill, y/\text{John}}$</td>
</tr>
<tr>
<td>Knows(John,x)</td>
<td>Knows(y, Mother(y))</td>
<td>${y/\text{John}, x/Mother(\text{John})}$</td>
</tr>
<tr>
<td>Knows(John,x)</td>
<td>Knows(x, Elisabeth)</td>
<td>${\text{fail}}$</td>
</tr>
</tbody>
</table>

- **Standardizing apart** eliminates overlap of variables, e.g., Knows($z_{17}$, Elisabeth)
Unification

- To unify $\text{Knows}(\text{John}, x)$ and $\text{Knows}(y, z)$,

$$\theta = \{y/\text{John}, x/z \} \text{ or } \theta = \{y/\text{John}, x/\text{John}, z/\text{John}\}$$

- The first unifier is more general than the second, because there are fewer constraints on the variables.

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

$$\text{MGU} = \{ y/\text{John}, x/z \}$$
The unification algorithm

function UNIFY(x, y, θ) returns a substitution to make x and y identical
inputs: x, a variable, constant, list, or compound
        y, a variable, constant, list, or compound
        θ, the substitution built up so far

if θ = failure then return failure
else if x = y then return θ
else if VARIABLE?(x) then return UNIFY-VAR(x, y, θ)
else if VARIABLE?(y) then return UNIFY-VAR(y, x, θ)
else if COMPOUND?(x) and COMPOUND?(y) then
    return UNIFY(ARGS[x], ARGs[y], UNIFY(OP[x], OP[y], θ))
else if LIST?(x) and LIST?(y) then
    return UNIFY(REST[x], REST[y], UNIFY(FIRST[x], FIRST[y], θ))
else return failure
The unification algorithm

function UNIFY-VAR(var, x, θ) returns a substitution
inputs: var, a variable
        x, any expression
        θ, the substitution built up so far

if {var/val} ∈ θ then return UNIFY(val, x, θ)
else if {x/val} ∈ θ then return UNIFY(var, val, θ)
else if OCCUR-CHECK?(var, x) then return failure
else return add {var/x} to θ
Generalized Modus Ponens (GMP)

\[ \begin{array}{c}
p_1', p_2', \ldots, p_n', (p_1 \land p_2 \land \ldots \land p_n \Rightarrow q) \\
q \theta
\end{array} \]

where \( \text{Subst}(\theta, p_i') = \text{Subst}(\theta, p_i) \) for all \( i \)

- \( p_1' \) is \( \text{King}(John) \) \hspace{1cm} \( p_1 \) is \( \text{King}(x) \)
- \( p_2' \) is \( \text{Greedy}(y) \) \hspace{1cm} \( p_2 \) is \( \text{Greedy}(x) \)
- \( \theta \) is \( \{x/John, y/John\} \) \hspace{1cm} q is \( \text{Evil}(x) \)
- q \( \theta \) is \( \text{Evil}(John) \)

- GMP used with KB of \textit{definite clauses} (\textit{exactly} one positive literal)
- All variables assumed universally quantified
Soundness of GMP

- Need to show that

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

provided that Subst(\(\theta\), \(p_i'\)) = Subst(\(\theta\), \(p_i\)) for all \(i\)

- Lemma:
  For any sentence \(p\), we have \(p \vdash \text{Subst}(\theta, p)\) by UI

1. \((p_1 \land \ldots \land p_n \Rightarrow q) \vdash \text{Subst}(\theta, p_1 \land \ldots \land p_n \Rightarrow q) = \text{Subst}(\theta, p_1) \land \ldots \land \text{Subst}(\theta, p_n) \Rightarrow \text{Subst}(\theta, q)\)
2. \(p_1', \ldots, p_n' \vdash p_1' \land \ldots \land p_n' \vdash \text{Subst}(\theta, p_1') \land \ldots \land \text{Subst}(\theta, p_n')\)
3. From 1 and 2, \(q_\theta\) follows by ordinary Modus Ponens
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:

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

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

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

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 chaining algorithm

```latex
\textbf{function FOL-FC-Ask}(KB, \alpha) \textbf{returns} a substitution or false

\textbf{repeat until} new is empty

new \leftarrow \{\}

\textbf{for each sentence } r \textbf{ in } KB \textbf{ do}

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

\textbf{for each } \theta \textbf{ such that } (p_1 \land \ldots \land p_n)\theta = (p'_1 \land \ldots \land p'_n)\theta

\textbf{for some } p'_1, \ldots, p'_n \textbf{ in } KB

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

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

add q' to new

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

\textbf{if } \phi \textbf{ is not fail then return } \phi

add new to KB

\textbf{return} false
```
Forward chaining proof

- American(West)
- Missile(M1)
- Owns(Nono,M1)
- Enemy(Nono,America)
Forward chaining proof
Forward chaining proof

[Diagram showing a hierarchical structure with relationships between concepts such as Criminal(West), Weapon(M1), Sells(West,M1,Nono), Hostile(Nono), American(West), Missile(M1), Owns(Nono,M1), and 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 $\alpha$ is not entailed
Efficiency of forward chaining

Incremental forward chaining: no need to match a rule on iteration $k$ if a premise wasn't added on iteration $k-1$

$\Rightarrow$ 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 $\text{Missile}(x)$ retrieves $\text{Missile}(M_1)$

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

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

\[ \begin{align*}
\text{Diff}(\text{Red},\text{Blue}) & \quad \text{Diff} (\text{Red},\text{Green}) \\
\text{Diff}(\text{Green},\text{Red}) & \quad \text{Diff}(\text{Green},\text{Blue}) \\
\text{Diff}(\text{Blue},\text{Red}) & \quad \text{Diff}(\text{Blue},\text{Green})
\end{align*} \]
Backward chaining algorithm

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: ans, a set of substitutions, initially empty
    if goals is empty then return {θ}
    q' ← SUBST(θ, FIRST(goals))
    for each r in KB where STANDARDIZE-APART(r) = (p₁ ∧ ... ∧ pₙ ⇒ q)
        and θ' ← UNIFY(q, q') succeeds
            ans ← FOL-BC-Ask(KB, [p₁, ..., pₙ | REST(goals)], COMPOSE(θ, θ')) ∪ ans
    return ans

SUBST(COMPOSE(θ₁, θ₂), p) = SUBST(θ₂, SUBST(θ₁, p))
Backward chaining example

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

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

\( Owns(Nono, M_1) \) and \( Missile(M_1) \)
Backward chaining example
Backward chaining example
Backward chaining example
Properties of backward chaining

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

- Incomplete due to infinite loops
  \[ \Rightarrow \text{fix by checking current goal against every goal on stack} \]

- Inefficient due to repeated subgoals (both success and failure)
  \[ \Rightarrow \text{fix using caching of previous results (extra space)} \]

- Widely used for logic programming
Summary

- Inference in First-Order Logic
- Reduction to propositional inference
- Generalized Modus Ponens
- Unification
- Forward and Backward Chaining
- Prolog