LECTURE 14 OF 42

FOL: Unification, Forward and Backward Chaining, and Resolution Theorem Proving
Discussion: GMP, Constraint Logic Programming

William H. Hsu
Department of Computing and Information Sciences, KSU

KSOL course page: http://snipurl.com/v9v3
Course web site: http://www.kddresearch.org/Courses/CIS730
Instructor home page: http://www.cis.ksu.edu/~bhsu

Reading for Next Class:
Section 9.5, p. 295 - 309, Russell & Norvig 2nd edition
Handout, Nilsson & Genesereth, Logical Foundations of Artificial Intelligence

LECTURE OUTLINE

• Reading for Next Class: Sections 9.5 (p. 295 – 309), R&N 2e
• Last Class: KR in FOL, 8.3-8.4 (p. 253-266), 9.1 (p. 272-274), R&N 2e
  ❖ Frame problem: representational (frame axioms) vs. inferential
  ❖ Related inference problems: ramification and qualification
  ❖ Representing states, actions: situation calculus, successor-state axioms
  ❖ Towards planning systems
  ❖ First-order inference: Generalized Modus Ponens (GMP)
• Today: Unification and Resolution, 9.2 – 9.4 (p. 275 – 294), R&N 2e
  ❖ Unification (previewed last time)
  ❖ GMP implemented
    ☺ forward chaining and the Rete algorithm for production systems
    ☺ backward chaining
  ❖ Resolution theorem proving
  ❖ Constraint logic programming
  ❖ Preview: backward chaining in logic programming
• Next Class: Logic Programming (Prolog)
Successor-State Axioms and Situation Calculus: Review

Situation calculus is one way to represent change in FOL:
- Add a situation argument to each non-external predicate
- E.g., Now in HoldingGold, New) denotes a situation

Successor-state axioms solve the representational frame problem
- Each axiom is "about" a predicate (not an action per se):
  - \( P \) true afterwards \( \iff \) [an action made \( P \) true
  - \( P \) true already and no action made \( P \) false]

- For holding the gold:
  \[ \forall a, s. \text{HoldingGold}(a, s) \iff (a = \text{Grab} \land \neg \text{Gold}(a)) \]
  \[ \lor (\text{HoldingGold}(s) \land a \neq \text{Release}) \]

Adapted from slides © 2004 S. Russell & P. Norvig. Reused with permission.

Chapter 9 Topics: Review

- Reducing first-order inference to propositional inference
- Unification
- Generalized Modus Ponens
- Forward and backward chaining
- Logic programming
- Resolution

**HERBRAND’S LIFTING LEMMA: REVIEW**

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

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

---

**AUTOMATED DEDUCTION [1]: SEQUENT RULES FOR FOL**

Sound inference: find \( \alpha \) such that \( KB \models \alpha \).

Proof process is a search, operators are inference rules.

E.g., Modus Ponens (MP)

\[
\frac{\alpha \quad \alpha \Rightarrow \beta}{\beta} \quad \text{At}(\text{Joe}, \text{UCB}) \quad \text{At}(\text{Joe}, \text{UCB}) \Rightarrow \text{OK}(\text{Joe})
\]

\[
\text{OK}(\text{Joe})
\]

E.g., And-Introduction (AI)

\[
\alpha \land \beta \quad \text{OK}(\text{Joe}) \quad \text{CSMajor}(\text{Joe})
\]

\[
\text{OK}(\text{Joe}) \land \text{CSMajor}(\text{Joe})
\]

E.g., Universal Elimination (UE)

\[
\forall x \alpha \quad \forall x \text{ At}(x, \text{UCB}) \Rightarrow \text{OK}(x)
\]

\[
\alpha[x/\tau] \quad \text{At}(\text{Pat}, \text{UCB}) \Rightarrow \text{OK}(\text{Pat})
\]

\( \tau \) must be a ground term (i.e., no variables)

---

Based on slide © 2004 S. Russell & P. Norvig. Reused with permission.

Adapted from slides © 2004 S. Russell & P. Norvig. Reused with permission.
AUTOMATED DEDUCTION [2]:
EXAMPLE PROOF

- Bob is a buffalo
  Pat is a pig
  Buffaloes outrun pigs
  Bob outruns Pat

  1. Buffalos(Bob)
  2. Pig(Pat)
  3. ∀x,y Buffalos(x) ∧ Pig(y) ⇒ Faster(x, y)

- Apply Sequent Rules to Generate New Assertions

  A1 & 2  4. Buffalos(Bob) ∧ Pig(Pat)
  UE 3, {x/Bob,y/Pat}  5. Buffalos(Bob) ∧ Pig(Pat) ⇒ Faster(Bob, Pat)
  MP 6 & 7  6. Faster(Bob, Pat)

  \[ \frac{\alpha, \alpha \Rightarrow \beta}{\beta} \]
  \[ \frac{\alpha \land \beta}{\alpha \land \beta} \]
  \[ \frac{\forall x \alpha}{\alpha[x/\tau]} \]

- Modus Ponens
  And Introduction
  Universal Elimination

Adapted from slides © 2004 S. Russell & P. Norvig. Reused with permission.

GENERALIZED MODUS PONENS (GMP):
REVIEW

\[ p_1', p_2', \ldots, p_n', (p_1 ∧ p_2 ∧ \ldots ∧ p_n ⇒ q) \]
\[ q\theta \]

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

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

GMP used with KB of definite clauses (exactly one positive literal)
All variables assumed universally quantified

Unification [1]: Algorithm — Main Function

function \textsc{Unify}(x, y, \theta) returns a substitution to make \( x \) and \( y \) identical
inputs: \( x \), a variable, constant, list, or compound
\hspace{1em} \( y \), a variable, constant, list, or compound
\hspace{1em} \( \theta \), the substitution built up so far
\hspace{1em} if \( \theta = \text{failure} \) then return failure
\hspace{1em} else if \( x = y \) then return \( \theta \)
\hspace{1em} else if \( \text{Variable}(x) \) then return \textsc{Unify-Var}(x, y, \theta)
\hspace{1em} else if \( \text{Variable}(y) \) then return \textsc{Unify-Var}(y, x, \theta)
\hspace{1em} else if \( \text{Compound}(x) \) and \( \text{Compound}(y) \) then
\hspace{2em} return \textsc{Unify}([\text{Args}[x]], [\text{Args}[y]], \textsc{Unify}([\text{Op}[x]], [\text{Op}[y]], \theta))
\hspace{1em} else if \( \text{List}(x) \) and \( \text{List}(y) \) then
\hspace{2em} return \textsc{Unify}([\text{Rest}[x]], [\text{Rest}[y]], \textsc{Unify}([\text{First}[x]], [\text{First}[y]], \theta))
\hspace{1em} else return failure

Based on slide © 2004 S. Russell & P. Norvig. Reused with permission.

Unification [2]: Co-Recursive Function for Unifying Variables

function \textsc{Unify-Var}(\texttt{var}, x, \theta) returns a substitution
inputs: \( \texttt{var} \), a variable
\hspace{1em} \( x \), any expression
\hspace{1em} \( \theta \), the substitution built up so far
\hspace{1em} if \{\texttt{var}/\texttt{val}\} \in \theta \) then return \textsc{Unify}(\texttt{val}, x, \theta)
\hspace{1em} else if \{x/\texttt{val}\} \in \theta \) then return \textsc{Unify}(\texttt{var}, \texttt{val}, \theta)
\hspace{1em} else if \texttt{Occur-Check}?(\texttt{var}, x) then return failure
\hspace{1em} else return add \{\texttt{var}/x\} to \( \theta \)

Based on slide © 2004 S. Russell & P. Norvig. Reused with permission.
**Unification [3]: Examples**

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

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

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

<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, OJ)</td>
<td>{ x/OJ, 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, OJ)</td>
<td>fail</td>
</tr>
</tbody>
</table>

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

---

**Frame, Ramification, & Qualification Problems: Review**

“Effect” axiom—describe changes due to action
\[ \forall s. \text{AtGold}(s) \Rightarrow \text{Holding(Gold, Result(Grab, s))} \]

“Frame” axiom—describe non-changes due to action
\[ \forall s. \text{HaveArrow}(s) \Rightarrow \text{HaveArrow(Result(Grab, s))} \]

Frame problem: find an elegant way to handle non-change
(a) representation—avoid frame axioms
(b) inference—avoid repeated “copy-overs” to keep track of state

Qualification problem: true descriptions of real actions require endless caveats—what if gold is slippery or nailed down or . . .

Ramification problem: real actions have many secondary consequences—what about the dust on the gold, wear and tear on gloves, . . .

Based on slides © 2004 S. Russell & P. Norvig. Reused with permission.
Example Knowledge Base [1]:

**English Statement of KB and Query**

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 [2]:

**Rules and Facts**

... 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., \( \exists x \ \text{Owns}(\text{Nono}, x) \land \text{Missile}(x) \):

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

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

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

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 ...

\[ \text{Enemy}(\text{Nono, America}) \]

---

Forward Chaining in FOL [1]: Algorithm

function FOL-FC-ASK(KB, α) returns a substitution or false

repeat until new is empty
    new ← {} 
    for each sentence r in KB do
        (p₁ ∧ ... ∧ pₙ → q) ← STANDARDIZE-APART(r)
        for each θ such that (p₁ θ ∧ ... ∧ pₙ θ) = (p₁' θ ∧ ... ∧ pₙ' θ)
            for some p₁', ..., pₙ' in KB
                q' ← SUBST(θ, q)
                if q' is not a renaming of a sentence already in KB or new then do
                    add q' to new
                    φ ← UNIFY(q', α)
                    if φ is not fail then return φ
                add new to KB
return false


Forward Chaining in FOL [2]: Example Proof

Adapted from slides © 2004 S. Russell & P. Norvig. Reused with permission.
Forward Chaining in FOL [3]:
Properties

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

Forward Chaining in FOL [4]:
Efficiency

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: Constraint Logic Programming

\[
\begin{align*}
\text{Diff}(wa, nt) & \land \text{Diff}(wa, sa) \land \\
\text{Diff}(nt, q) & \land \text{Diff}(nt, sa) \land \\
\text{Diff}(q, nsw) & \land \text{Diff}(q, sa) \land \\
\text{Diff}(nsw, v) & \land \text{Diff}(nsw, sa) \land \\
\text{Diff}(v, sa) & \Rightarrow \text{Colorable}() \\
\text{Diff}(Red, Blue) & \land \text{Diff}(Red, Green) \\
\text{Diff}(Green, Red) & \land \text{Diff}(Green, Blue) \\
\text{Diff}(Blue, Red) & \land \text{Diff}(Blue, Green)
\end{align*}
\]

Colorable() is inferred iff the CSP has a solution.
CSPs include 3SAT as a special case, hence matching is NP-hard

Backward Chaining in FOL [1]: 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 already applied)
        θ, the current substitution, initially the empty substitution { }
local variables: answers, a set of substitutions, initially empty
if goals is empty then return {θ}
q' ← SUBST(θ, FIRST(goals))
for each sentence r in KB
    where STANDARDIZE-APART(r) = ( p_1 \land \ldots \land p_n \rightarrow q)
    and θ' ← UNIFY(q, q') succeeds
    new_goals ← [p_1, \ldots, p_n] \cup \text{REST(goals)}
    answers ← FOL-BC-Ask(KB, new_goals, COMPOSE(θ', θ)) ∪ answers
return answers

**Backward Chaining in FOL [2]: Example Proof**

- **Criminal** (West)
  - 
  - **Weapon** (y MI)
    - **Shoot** (x, y, z) → z (Army)
      - **Massive** (x)
        - x (Army)
      - **Massive** (MI)
        - **Ownership** (x, MI)
          - **Enemy** (x, Army)


---

**Backward Chaining in FOL [3]: Properties**

- 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

Logic Programming

Sound bite: computation as inference on logical KBs

<table>
<thead>
<tr>
<th>Logic programming</th>
<th>Ordinary programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Identify problem</td>
<td>Identify problem</td>
</tr>
<tr>
<td>2. Assemble information</td>
<td>Assemble information</td>
</tr>
<tr>
<td>3. Tea break</td>
<td>Figure out solution</td>
</tr>
<tr>
<td>4. Encode information in KB</td>
<td>Program solution</td>
</tr>
<tr>
<td>5. Encode problem instance as facts</td>
<td>Encode problem instance as data</td>
</tr>
<tr>
<td>6. Ask queries</td>
<td>Apply program to data</td>
</tr>
<tr>
<td>7. Find false facts</td>
<td>Debug procedural errors</td>
</tr>
</tbody>
</table>

Should be easier to debug $\text{Capital(\text{New York, US})}$ than $x := x + 2$!

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$_1$, ..., literal$_n$.

$\text{criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z)}.$

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., $X$ is $Y*Z+3$
Closed-world assumption ("negation as failure")
  e.g., given $\text{alive(X)}$ :- not $\text{dead(X)}$
  $\text{alive(joe)}$ succeeds if $\text{dead(joe)}$ fails
Prolog Examples

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:

\[
\ell_1 \lor \cdots \lor \ell_k, \quad m_1 \lor \cdots \lor m_n
\]
\[
\frac{(\ell_1 \lor \cdots \lor \ell_{i-1} \lor \ell_{i+1} \lor \cdots \lor \ell_k \lor m_1 \lor \cdots \lor m_{j-1} \lor m_{j+1} \lor \cdots \lor m_n)\theta}{\text{UNIFY}(\ell_i, -m_j) = \theta}
\]

where UNIFY(\ell_i, -m_j) = \theta.

For example,

```
\neg \text{Rich}(x) \lor \text{Unhappy}(x)
\text{Rich}(Ken)
\text{Unhappy}(Ken)
```

with \( \theta = \{x/\text{Ken}\} \)

Apply resolution steps to CNF\((KB \land \neg \alpha)\); complete for FOL
Conversion of FOL to Clausal Form (CNF) [1]

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

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

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

Conversion of FOL to Clausal Form (CNF) [2]

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

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

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

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

Based on slide © 2004 S. Russell & P. Norvig. Reused with permission.
Resolution Mnemonic: INSEUDOR

- Implications Out (Replace with Disjunctive Clauses)
- Negations Inward (DeMorgan’s Theorem)
- Standardize Variables Apart (Eliminate Duplicate Names)
- Existentials Out (Skolemize)
- Universals Made Implicit
- Distribute And Over Or (i.e., Disjunctions Inward)
- Operators Made Implicit (Convert to List of Lists of Literals)
- Rename Variables (Independent Clauses)
- A Memonic for Star Trek: The Next Generation Fans

Captain Picard:

I'll Notify Spock's Eminent Underground Dissidents On Romulus
I'll Notify Sarek's Eminent Underground Descendant On Romulus


Resolution Proof: Definite Clauses

**TERMINOLOGY**

- **Generalized Modus Ponens (GMP)**
  - Sound and complete rule for first-order inference (reasoning in FOL)
  - Requires pattern matching by unification

- **Unification: Algorithm for Matching Patterns**
  - Used in type inference, first-order inference
  - Matches well-formed formulas (WFFs), atoms, terms
  - **Terms**: variables, constants, functions and arguments
  - **Arguments**: nested terms

- **Resolution: Sound and Complete Inference Rule/Procedure for FOL**
  - **Antecedent** (aka precedent): sentences “above line” in sequent rule
  - **Resolvent** (aka consequent): sentences “below line” in sequent rule

- **Forward Chaining: Systematic Application of Rule to Whole KB**
  - Rete algorithm in production systems for expert systems development
  - Susceptible to high fan-out (branch factor)

- **Backward Chaining: Goal-Directed**

---

**SUMMARY POINTS**

- Last Class: From Propositional Logic to FOL
  - **Herbrand theory**
    - ground terms: full instantiation (propositionalization)
    - lifting lemma: can generalize from propositional inference
  - Need to convert FOL to clausal form (CNF)
  - **Generalized Modus Ponens (GMP)**
  - Sound and complete rule for first-order inference (reasoning in FOL)
  - Requires pattern matching by unification

- **Unification: Algorithm for Matching Patterns**
  - Used in type inference, first-order inference
  - Matches well-formed formulas (WFFs), atoms, terms
  - **Terms**: variables, constants, functions and arguments
  - **Arguments**: nested terms

- **Resolution: Sound and Complete Inference Rule/Procedure for FOL**

- **Forward Chaining: Systematic Application of Sequent Rule to KB**