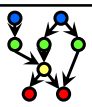


Lecture 14 of 42

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

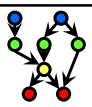
KSOL course page: http://snipurl.com/v9v3
Course web site: http://www.kddresearch.org/Courses/CIS730
Instructor home page: http://www.cis.ksu.edu/~bhsu

Reading for Next Class:

Section 9.5, p. 295 - 309, Russell & Norvig 2nd edition
Handout, Nilsson & Genesereth, *Logical Foundations of Artificial Intelligence*

Lecture Outline

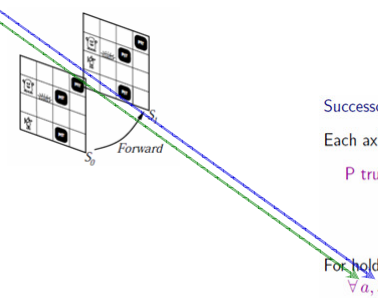
- **Reading for Next Class: Sections 9.5 (p. 295 – 309), R&N 2e**
- **Last Class: KR in FOL, 8.3-8.4 (p. 253-266), 9.1 (p. 272-274), R&N 2e**
  - **Frame problem: representational (frame axioms) vs. inferential**
  - **Related inference problems: ramification and qualification**
  - **Representing states, 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:
Adds a situation argument to each non-eternal predicate
E.g., Now in $Holding(Gold, Now)$ denotes a situation

Situations are connected by the $Result$ function
$Result(a, s)$ is the situation that results from doing a in s



Successor-state axioms solve the representational frame problem

Each axiom is "about" a **predicate** (not an action per se):

P true afterwards $\Leftrightarrow$ [an action made P true
$\vee$ P true already and no action made P false]

For holding the gold:

$$\forall a, s \;\; Holding(Gold, Result(a, s)) \Leftrightarrow [(a = Grab \wedge AtGold(s)) \vee (Holding(Gold, s) \wedge a \neq Release)]$$

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., $Father(Father(Father(John)))$

Theorem: Herbrand (1930). If a sentence α is entailed by an FOL KB, it is entailed by a **finite** subset of the propositional KB

Idea: For $n = 0$ to ∞ do
create a propositional KB by instantiating with depth-n terms
see if α is entailed by this KB

Problem: works if α is entailed, loops if α is not entailed

Theorem: Turing (1936), Church (1936), entailment in FOL is semidecidable

Automated Deduction [1]: Sequent Rules for FOL

Sound inference: find α such that $KB \models \alpha$.
Proof process is a search, operators are inference rules.

E.g., Modus Ponens (MP)

$$\frac{\alpha, \quad \alpha \Rightarrow \beta}{\beta} \qquad \frac{At(Joe,UCB) \quad At(Joe,UCB) \Rightarrow OK(Joe)}{OK(Joe)}$$

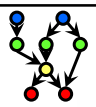
E.g., And-Introduction (AI)

$$\frac{\alpha \quad \beta}{\alpha \wedge \beta} \qquad \frac{OK(Joe) \quad CSMajor(Joe)}{OK(Joe) \wedge CSMajor(Joe)}$$

E.g., Universal Elimination (UE)

$$\frac{\forall x \ \alpha}{\alpha\{x/\tau\}} \qquad \frac{\forall x \ At(x,UCB) \Rightarrow OK(x)}{At(Pat,UCB) \Rightarrow OK(Pat)}$$

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



Automated Deduction [2]: Example Proof

Bob is a buffalo	1. $Buffalo(Bob)$
Pat is a pig	2. $Pig(Pat)$
Buffaloes outrun pigs	3. $\forall x, y \;\; Buffalo(x) \wedge Pig(y) \Rightarrow Faster(x, y)$
Bob outruns Pat	

- Apply Sequent Rules to Generate New Assertions

AI 1 & 2	4. $Buffalo(Bob) \wedge Pig(Pat)$
UE 3, $\{x/Bob, y/Pat\}$	5. $Buffalo(Bob) \wedge Pig(Pat) \Rightarrow Faster(Bob, Pat)$
MP 6 & 7	6. $Faster(Bob, Pat)$

$$\frac{\alpha, \quad \alpha \Rightarrow \beta}{\beta}$$

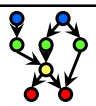
- Modus Ponens

$$\frac{\alpha \quad \beta}{\alpha \wedge \beta}$$

And Introduction

$$\frac{\forall x \;\; \alpha}{\alpha\{x/\tau\}}$$

Universal Elimination



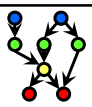
Generalized Modus Ponens (GMP): Review

$$\frac{p_1', \;\; p_2', \;\; \ldots, \;\; p_n', \;\; (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{q\theta}$$

where $p_i'\theta = p_i\theta$ for all i

p_1' is $King(John)$	p_1 is $King(x)$
p_2' is $Greedy(y)$	p_2 is $Greedy(x)$
θ is $\{x/John, y/John\}$	q is $Evil(x)$
$q\theta$ is $Evil(John)$	

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



Unification [1]: Algorithm – Main Function

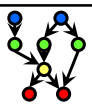
function UNIFY(x, y, θ) **returns** a substitution to make x and y identical
  inputs: x, a variable, constant, list, or compound
    y, a variable, constant, list, or compound
    θ, the substitution built up so far

  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

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

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

  if $\{var/val\} \in \theta$ **then return** UNIFY(val, x, θ)
  else if $\{x/val\} \in \theta$ **then return** UNIFY(var, val, θ)
  else if OCCUR-CHECK?(var, x) **then return** failure
  else return add $\{var/x\}$ to θ



Unification [3]: Examples

We can get the inference immediately if we can find a substitution θ such that $King(x)$ and $Greedy(x)$ match $King(John)$ and $Greedy(y)$

$\theta = \{x/John, y/John\}$ works

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

p	q	θ
$Knows(John, x)$	$Knows(John, Jane)$	$\{x/Jane\}$
$Knows(John, x)$	$Knows(y, OJ)$	$\{x/OJ, y/John\}$
$Knows(John, x)$	$Knows(y, Mother(y))$	$\{y/John, x/Mother(John)\}$
$Knows(John, x)$	$Knows(x, OJ)$	$fail$

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

Frame, Ramification, & Qualification Problems: Review

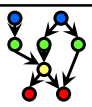
"Effect" axiom—describe changes due to action
$\forall s \;\; AtGold(s) \Rightarrow Holding(Gold, Result(Grab, s))$

"Frame" axiom—describe **non-changes** due to action
$\forall s \;\; HaveArrow(s) \Rightarrow HaveArrow(Result(Grab, s))$

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, . . .



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:

$$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$$

Nono . . . has some missiles, i.e., $\exists x\ Owns(Nono, x) \wedge Missile(x)$:

$Owns(Nono, M_1)$ and $Missile(M_1)$

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

$$\forall x\ \ Missile(x) \wedge Owns(Nono, x) \ \Rightarrow\ Sells(West, x, Nono)$$

Missiles are weapons:

$$Missile(x) \Rightarrow Weapon(x)$$

An enemy of America counts as "hostile":

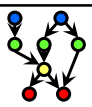
$$Enemy(x, America) \ \Rightarrow\ Hostile(x)$$

West, who is American . . .

$$American(West)$$

The country Nono, an enemy of America . . .

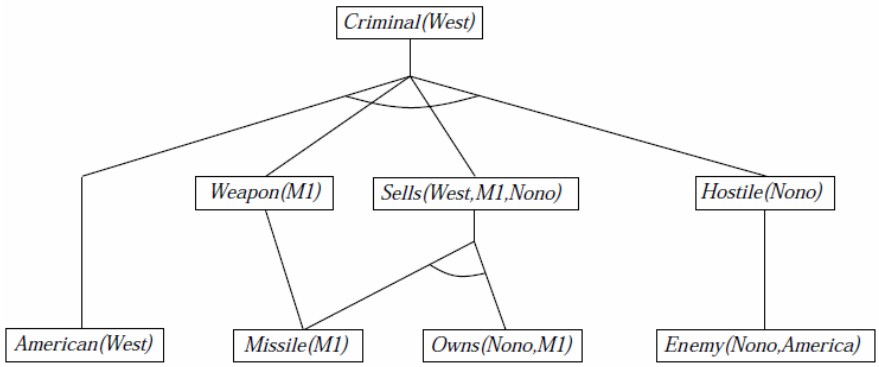
$$Enemy(Nono, America)$$

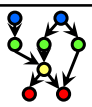


Forward Chaining in FOL [1]: Algorithm

```
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 ∧ ... ∧ pn)θ = (p'1 ∧ ... ∧ p'n)θ
                  for some p'1, ..., p'n 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





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 α is not entailed

This is unavoidable: entailment with definite clauses is semidecidable

Forward Chaining in FOL [4]: Efficiency

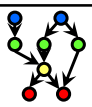
Simple observation: no need to match a rule on iteration k
if a premise wasn't added on iteration $k-1$
$\Rightarrow$ match each rule whose premise contains a newly added literal

Matching itself can be expensive

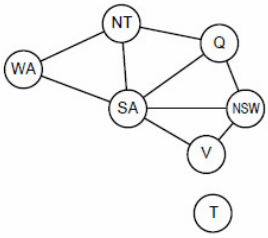
Database indexing allows $O(1)$ retrieval of known facts
e.g., query $Missile(x)$ retrieves $Missile(M_1)$

Matching conjunctive premises against known facts is NP-hard

Forward chaining is widely used in deductive databases



Hard Matching Example: Constraint Logic Programming



$$Diff(wa, nt) \wedge Diff(wa, sa) \wedge$$
$$Diff(nt, q) Diff(nt, sa) \wedge$$
$$Diff(q, nsw) \wedge Diff(q, sa) \wedge$$
$$Diff(nsw, v) \wedge Diff(nsw, sa) \wedge$$
$$Diff(v, sa) \Rightarrow Colorable()$$

$$Diff(Red, Blue) \quad Diff(Red, Green)$$
$$Diff(Green, Red) \quad Diff(Green, Blue)$$
$$Diff(Blue, Red) \quad Diff(Blue, Green)$$

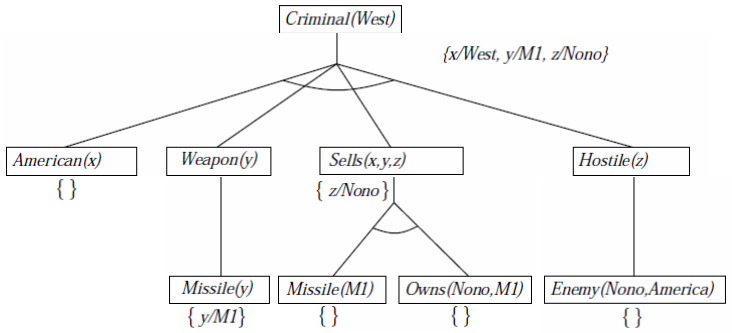
$Colorable()$ is inferred iff the CSP has a solution
CSPs include 3SAT as a special case, hence matching is NP-hard

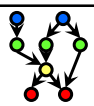
Backward Chaining in FOL [1]: Algorithm

function FOL-BC-Ask($KB, goals, \theta$) **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** $\{\theta\}$
  $q' \leftarrow$ Subst(θ, First($goals$))
  for each sentence r **in** KB
    where Standardize-Apart(r) = $(p_1 \wedge \ldots \wedge p_n \Rightarrow q)$
    and $\theta' \leftarrow$ Unify(q, q') succeeds
    $new\_goals \leftarrow [p_1, \ldots, p_n | \text{Rest}(goals)]$
    $answers \leftarrow$ FOL-BC-Ask(KB, new_goals, Compose(θ', θ)) $\cup$ $answers$
  return $answers$

Backward Chaining in FOL [2]: Example Proof





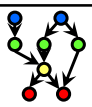
Backward Chaining in FOL [3]: Properties

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

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

Should be easier to debug $Capital(NewYork, US)$ 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`$_n$`.`

```
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 `alive(X) :- not dead(X).`
`alive(joe)` succeeds if `dead(joe)` fails

Prolog Examples

Depth-first search from a start state X:

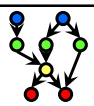
```
dfs(X) :- goal(X).
dfs(X) :- successor(X,S),dfs(S).
```

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]   B=[2]
         A=[1,2] B=[]
```

Resolution: Brief Summary

Full first-order version:

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \qquad m_1 \vee \cdots \vee m_n}{(\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n)\theta}$$

where $\text{UNIFY}(\ell_i, \neg m_j) = \theta$.

For example,

$$\frac{\neg Rich(x) \vee Unhappy(x) \quad Rich(Ken)}{Unhappy(Ken)}$$

with $\theta = \{x/Ken\}$

Apply resolution steps to $CNF(KB \wedge \neg\alpha)$; complete for FOL

Conversion of FOL to Clausal Form (CNF) [1]

Everyone who loves all animals is loved by someone:

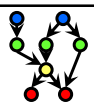
$$\forall x \ [\forall y \ \ Animal(y) \Rightarrow Loves(x, y)] \Rightarrow [\exists y \ \ Loves(y, x)]$$

1. Eliminate biconditionals and implications

$$\forall x \ [\neg \forall y \ \neg Animal(y) \vee Loves(x, y)] \vee [\exists y \ \ Loves(y, x)]$$

2. Move $\neg$ inwards: $\neg \forall x, p \equiv \exists x \ \neg p$, $\neg \exists x, p \equiv \forall x \ \neg p$:

$$\forall x \ [\exists y \ \neg(\neg Animal(y) \vee Loves(x, y))] \vee [\exists y \ \ Loves(y, x)]$$
$$\forall x \ [\exists y \ \neg\neg Animal(y) \wedge \neg Loves(x, y)] \vee [\exists y \ \ Loves(y, x)]$$
$$\forall x \ [\exists y \ \ Animal(y) \wedge \neg Loves(x, y)] \vee [\exists y \ \ Loves(y, x)]$$



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

3. Standardize variables: each quantifier should use a different one

$$\forall x \ [\exists y \ \ Animal(y) \wedge \neg Loves(x, y)] \vee [\exists z \ \ Loves(z, x)]$$

4. Skolemize: a more general form of existential instantiation.
   Each existential variable is replaced by a Skolem function of the enclosing universally quantified variables:

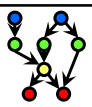
$$\forall x \ [Animal(F(x)) \wedge \neg Loves(x, F(x))] \vee Loves(G(x), x)$$

5. Drop universal quantifiers:

$$[Animal(F(x)) \wedge \neg Loves(x, F(x))] \vee Loves(G(x), x)$$

6. Distribute $\wedge$ over $\vee$:

$$[Animal(F(x)) \vee Loves(G(x), x)] \wedge [\neg Loves(x, F(x)) \vee Loves(G(x), x)]$$



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

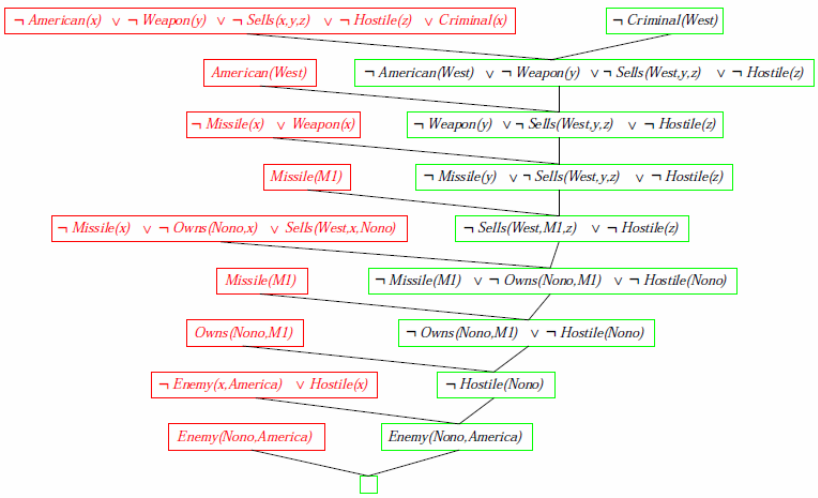
Captain Picard:

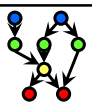
I'll Notify Spock's Eminent Underground Dissidents On Romulus

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

Adapted from: Nilsson and Genesereth (1987). *Logical Foundations of Artificial Intelligence.*
http://bit.ly/45Cmqq

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