Title: Inference in First-Order Logic
AIMA: Chapter 9 (Sections 9.1—9.4)

Introduction to Artificial Intelligence
CSCE 476-876, Spring 2000
www.cse.unl.edu/~cse476

Berthe Y. Choueiry
choueiry@cse.unl.edu, (402)472-5444

Summary

First-order logic

— objects and relations are semantic primitives

— syntax: constants, functions, predicates, equality, quantifiers

Situation calculus

— conventions for describing actions and change in FOL

— can formulate planning as inference on a situation calculus KB

Inference in First-Order Logic?

Normal forms: some approaches to inference use syntactic operations on sentences, often expressed in standardized forms

Conjunctive Normal Form (CNF)

$\textit{conjunction}$ of $\underbrace{\textit{disjunctions} \text{ of } \textit{literals}}_{\textit{clauses}}$

E.g., $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$

Horn Form

conjunction of *Horn clauses* (clauses with ≤ 1 positive literal)

E.g., $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$

Often written as set of implications: $B \Rightarrow A$ and $(C \wedge D) \Rightarrow B$

Inference in Propositional Logic

Rules:

1. Modus Ponens
2. And-elimination
3. And-introduction
4. Or-introduction
5. Double-negation elimination
6. Unit-resolution
7. Resolution

Inference:

- sound and complete
- NP-complete in general
- polynomial for Horn clauses

Outline

- Proofs
- Unification
- Forward chaining
- Backward chaining

Important result:
Resolution with refutation is a complete inference procedure for FOL.

Inference rules for FOL

1. Modus Ponens (MP)
2. And-elimination
3. And-introduction (AI)
4. Or-introduction
5. Resolution
6. +3 additional rules to handle FOL sentences with quantifiers
   (a) Universal elimination (UE)
   (b) Existential elimination
   (c) Existential introduction

Substitution (a.k.a. binding list)

$$Subst(\{x/Sam, y/Pam\}, Likes(x, y)) = Likes(Sam, Pam)$$

Universal elimination (UE)

For any sentence α, variable v, and ground term g:

$$\frac{\forall\, v\ \alpha}{Subst(\{v, g\}), \alpha}$$

Example:

Sentence $= \forall\, x\ Likes(x, IceCream)$

Variable $= x$, constant = Ben, binding list= {x/Ben}

we can infer: $Likes(Ben, IceCream)$

(A universal sentence is the conjunction of all its possible instantiations)

Existential elimination

For any sentence α, variable v, and constant symbol term k:

$$\frac{\exists\, v\ \alpha}{Subst(\{v,k\}), \alpha}$$

Example:

Sentence $= \exists\, x\ Kill(x, Victim)$

Variable $= x$, constant $=$ Murdered, binding list= {x/Murderer}

we can infer: $Kill(Murderer, Victim)$

warning: Murderer does <u>not</u> appear in KB

e.g., $Kill(Victim, Victim)$ is not a logical consequence of the sentence

Existential introduction

For any sentence α, variable v, and ground term g,
warning: v does not appear α, g does appear in α

$$\frac{\alpha}{\exists\, v\ Subst(\{g, v\}), \alpha}$$

Example:

Sentence $= Likes(Jerry, IceCream)$
Variable $= x$, ground term = Jerry
we can infer: $\exists\, x\ Likes(x, IceCream)$

Proofs

Sound inference: find α such that $KB \models \alpha$.

Proof process is a search, operators are inference rules

Examples:

Modus Ponens (MP)

$$\frac{\alpha, \quad \alpha \Rightarrow \beta}{\beta} \qquad \frac{At(Lynn, UNL) \quad At(Lynn, UNL) \Rightarrow Smart(Lynn)}{Smart(Lynn)}$$

And-Introduction (AI)

$$\frac{\alpha \quad \beta}{\alpha \wedge \beta} \qquad \frac{Smart(Lynn) \quad CSMajor(Lynn)}{Smart(Lynn) \wedge CSMajor(Lynn)}$$

Universal Elimination (UE)

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

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

Proof procedure

Application of inference rules

— simply matches the premise patterns of the rules
to the sentences in the KB and

— adds their suitably instantiated conclusion patterns to the KB

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		
AI 1 & 2	4.	
UE 3, $\{x/Bob, y/Pat\}$	5.	
MP 6 & 7	6.	

Search with primitive inference rules

Search components:

- Operators: inference rules
- States: sets of sentences
- Goal tests: check if query sentence is in the state

Problems:

long proofs, with huge branching factors (especially for UE)

potentially explosive search space

Opportunity:

Pattern: And-Introduction, U-elimination, Modus-Ponens

$\longrightarrow$ define a single operator for 3-steps

Unification (I)

$Unify$:

— takes two atomic sentences p and q and

— returns a substitution in which p and q 'look the same'

— (if not possible, returns $fail$.)

$$Unify(p, q) = \theta \text{ where } Subst(\theta, p) = Subst(\theta, q)$$

Example 1:

$$p = Likes(John, x)$$

$$q = Likes(y, IceCream)$$

$$Unify(p, q) = \{John/y, x/IceCream\}$$

Example 2:

$$Unify(Likes(John, x), Likes(x, Jane)) = fail$$

Unification (II)

p	q	σ(substitution)
$Knows(John, x)$	$Knows(John, Jane)$	
$Knows(John, x)$	$Knows(y, OJ)$	
$Knows(John, x)$	$Knows(y, Mother(y))$	

Idea: Given a rule,
unify rule premises with known facts,
apply unifier to conclusion

E.g., if we know q and $Knows(John, x) \Rightarrow Likes(John, x)$

then we conclude $Likes(John, Jane)$

$Likes(John, OJ)$

$Likes(John, Mother(John))$

Unification (III) Most general unifier

MGU is the substitution that makes the least commitment about the bindings of the variables

$Unify(Knows(John, x), Knows(y, z))$ can return:

$\{y/John, x/z\}$,

$\{y/John, x/John, z/John\}$,

etc..

Unify must return the most general unifier: $\{y/John, x/z\}$

Generalized Modus Ponens (GMP) (I)

Idea: a generalization of MP to do in a single blow AI+UE+MP

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

$$\begin{aligned}
\text{Example: } p_1{}' =& \quad Faster(Bob, Pat)\\
p_2{}' =& \quad Faster(Pat, Steve)\\
p_1 \wedge p_2 \Rightarrow q \;\; =& \quad Faster(x,y) \wedge Faster(y,z) \Rightarrow Faster(x,z)\\
\theta =& \quad \{x/Bob, y/Pat, z/Steve\}\\
Subst(\theta, q) =& \quad Faster(Bob, Steve)
\end{aligned}$$

Generalized Modus Ponens (II)

All sentences in KB must be in a form that matches one of the premises of the Modus-Ponens rule:

— a single atomic sentence, or

— (conjunction of atomic sentences) $\Rightarrow$ (atomic sentence)

$\longrightarrow$ KB must be of Horn Normal Form

(*i.e.*, conjunction of Horn clauses)

Reminder: Horn Normal Form

conjunction of *Horn clauses* (clauses with ≤ 1 positive literal)

e.g., $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$

Often written as set of implications: $B \Rightarrow A$ and $(C \wedge D) \Rightarrow B$

Generalized Modus Ponens (III)

$\longrightarrow$ KB must be of Horn Normal Form
(*i.e.*, conjunction of Horn clauses)

- Sentences converted to Horn clauses as entered to KB using $\exists$-elimination and And-elimination
  Example: $\exists x Dog(x) \wedge Likes(John, x)$ becomes
  $Dog(Rebel)$ and $Likes(John, Rebel)$
- All variables assumed universally quantified:
  Example: $\forall\, x\ Likes(x, IceCream)$ is written $Likes(x, IceCream)$

GMP is a sound procedure, basis for:
— forward chaining and
— backward chaining

Forward chaining

When a new fact p is added to the KB

for each rule such that p unifies with a premise

if the other premises are known

then add the conclusion to the KB and continue chaining

Forward chaining is data-driven

e.g., inferring properties from percepts

inference is not directed toward solving a particular task

Forward chaining: example

Add facts 1, 2, 3, 4, 5, 7 in turn.

1. $Buffalo(x) \wedge Pig(y) \Rightarrow Faster(x, y)$
2. $Pig(y) \wedge Snail(z) \Rightarrow Faster(y, z)$
3. $Faster(x, y) \wedge Faster(y, z) \Rightarrow Faster(x, z)$
4. $Buffalo(Bob)$
5. $Pig(Pat)$
6. ...
7. $Snail(Lynn)$
8. ...
9. ...

Backward chaining:

When a query q is asked

if a matching fact q' is known, return the unifier

for each rule whose consequent q' matches q

attempt to prove each premise of the rule

by backward chaining

→ Some added complications in keeping track of the unifiers

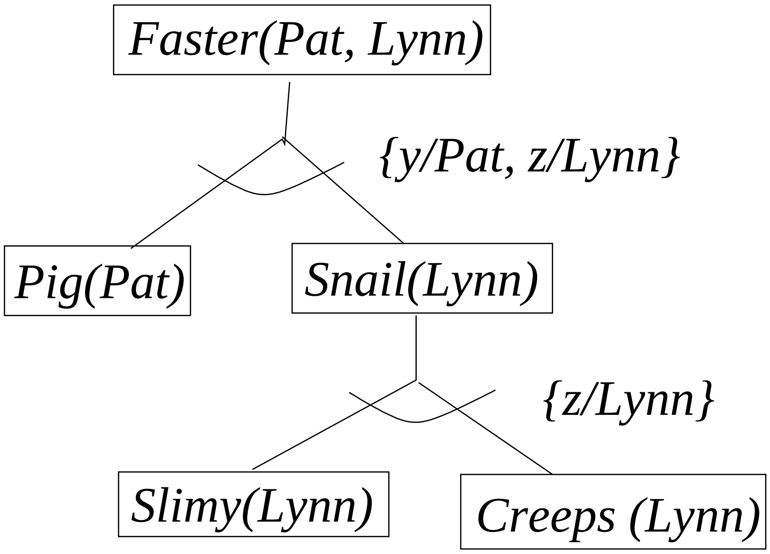
→ More complications help to avoid infinite loops

→ Two versions: find <u>any</u> solution, find <u>all</u> solutions

Backward chaining is the basis for <u>logic programming</u>, e.g., Prolog

Backward chaining: example

1. $Pig(y) \wedge Snail(z) \Rightarrow Faster(y, z)$
2. $Slimy(z) \wedge Creeps(z) \Rightarrow Snail(z)$
3. $Pig(Pat)$
4. $Slimy(Lynn)$
5. $Creeps(Lynn)$



List of terms for glossary:

Due date: Mon, March 20, 2000
Words in margins in AIMA