Inference and Resolution

Introduction to Artificial Intelligence

CS/ECE 348

Lecture 15

October 11, 2001

Outline

Last Lecture

- Wumpus in FOL
- Inference rules of FOL
- Unification

This Lecture

- Generalized Modus Ponens
- Forward and backward chaining
- Resolution

Reading

- Chapter 7, 9

Upcoming Dealdines

- HW2 has been assigned
  - Due Tuesday, October 16
  - No late assignments
- Midterm:Thursday, October 18, in class.
- Projects for 1 Unit grad students: Available on web.
  - Proposal: Due October 25

Deducing Hidden Properties

- "Squares are breezy near a pit."
- **Diagnostic rule** – infer cause from effect

  $\forall y\ \text{Breezy}(y) \Rightarrow [\exists x\ \text{Pit}(x) \wedge \text{Adjacent}(x,y)]$

- **Causal rule** – infer effect from cause

  $\forall x,y\ \text{Pit}(x) \wedge \text{Adjacent}(x,y) \Rightarrow \text{Breezy}(y)$

- Neither of these is complete – e.g. diagnostic rule doesn't tell us that if there is no breeze then there isn't a pit nearby.

- Definition of Breezy predicate:

  $\forall y\ \text{Breezy}(y) \Leftrightarrow [\exists x\ \text{Pit}(x) \wedge \text{Adjacent}(x,y)]$

Combining Diagnostic & Causal Rules

Diagnostic: $\forall y\ \text{Breezy}(y) \Rightarrow [\exists x\ \text{Pit}(x) \wedge \text{Adjacent}(x,y)]$

Causal: $\forall x,y\ \text{Pit}(x) \wedge \text{Adjacent}(x,y) \Rightarrow \text{Breezy}(y)$

$= \forall y,x\ \text{Pit}(x) \wedge \text{Adjacent}(x,y) \Rightarrow \text{Breezy}(y)$

$= \forall y\ [\neg\ \exists x\ \ \neg(\text{Pit}(x) \wedge \text{Adjacent}(x,y) \Rightarrow \text{Breezy}(y))]$ Duality

$= \forall y\ [\neg\ \exists x\ \ \neg(\neg\ (\text{Pit}(x) \wedge \text{Adjacent}(x,y))\ \ \vee \text{Breezy}(y))]$ Rewrite $\Rightarrow$

$= \forall y\ [\neg\ \exists x\ \ (\text{Pit}(x) \wedge \text{Adjacent}(x,y)) \wedge \neg\ \text{Breezy}(y)]$ DeMorgan

$= \forall y\ [\neg\ (\ [\exists x\ \ (\text{Pit}(x) \wedge \text{Adjacent}(x,y))] \wedge \neg\ \text{Breezy}(y))]$

Breezy doesn't contain existential variable x

$= \forall y\ [\neg\ (\exists x\ \text{Pit}(x) \wedge \text{Adjacent}(x,y)) \vee \text{Breezy}(y)]$ DeMorgan

$= \forall y\ [\exists x\ (\text{Pit}(x) \wedge \text{Adjacent}(x,y))] \Rightarrow \text{Breezy}(y)]$ Rewrite as $\Rightarrow$

Take conjunction of diagnostic and causal rules to get definition:

$\forall y\ \text{Breezy}(y) \Leftrightarrow [\ \exists x\ \text{Pit}(x) \wedge \text{Adjacent}(x,y)\ ]$

$P \Rightarrow Q\ \equiv\ \neg\ P \vee Q$

$\neg(P \vee Q) \equiv (\neg P) \wedge (\neg Q)$

$(P \wedge Q) \vee (P \wedge R) \equiv P \wedge (Q \vee R)$

Validity and Satisfiability

- Valid: A sentence is **valid or a tautology** (necessarily true) iff it is TRUE in all possible worlds.
  - Example: $P \vee \neg P$
  - In propositional logic, truth table can be used to determine validity – sentence is TRUE for every row.
- Satisfiable: A sentence is **satisfiable** iff it is TRUE in some possible world.
  - Example: $P \vee Q$
  - In propositional logic, truth table can be used to determine satifiability sentence is TRUE for one row.
- Unsatisfiable: A sentence is **unsatisfiable** (self contradiction) iff it is FALSE in all possible worlds
  - Example: $P \wedge \neg P$
  - In propositional logic, truth table can be used to determine validity – sentence is FALSE for every row.

Inference Rules

- Sounds inference: Find α such that KB $\models \alpha$
- Proof process is a search, operators are inference rules
- All inference rules of propositional logic apply to FOL.

Modus Ponens (MP)

$$\frac{\alpha \Rightarrow \beta \qquad \alpha}{\beta}$$

Example

$$\frac{\text{Fish(George)} \Rightarrow \text{Swims(George)} \qquad \text{Fish(George)}}{\text{Swims(George)}}$$

And Elimination (AE)

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

$$\frac{\text{Tired} \wedge \text{Hungry}}{\text{Tired}}$$

$$\frac{}{\beta}$$

$$\frac{}{\text{Hungry}}$$

Inference Rules - cont

And Introduction (AI)

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

$$\frac{\text{Relaxed} \quad \text{Thirsty}}{\text{Relaxed} \wedge \text{Thirsty}}$$

Modus Tolens

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

$$\frac{\neg\text{Snows} \Rightarrow \text{Rains} \quad \neg\, \text{Rains}}{\text{Snows}}$$

Substitution

Given a sentence α and binding list θ, the result of applying the substitution θ to α is denoted by subst(θ, α)

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

Inference with Universal Quantifiers

Universal Elimination (UE)

$$\frac{\forall x \ \alpha}{\text{Subst}(\{x/\tau\}, \alpha)}$$

substitution list

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

Example

Imagine Universe of Discourse is

{Rodrigo, Sharon, David, Jonathan}

$$\frac{\forall x \ \text{At}(x, \text{UIUC}) \Rightarrow \text{OK}(x)}{\text{At}(\text{Rodrigo}, \text{UIUC}) \Rightarrow \text{OK}(\text{Rodrigo})}$$

Inference with Existential Quantifiers

Existential Elimination

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

where constant symbol k is introduced into the Universe of Discourse (i.e., doesn't appear elsewhere)

Example

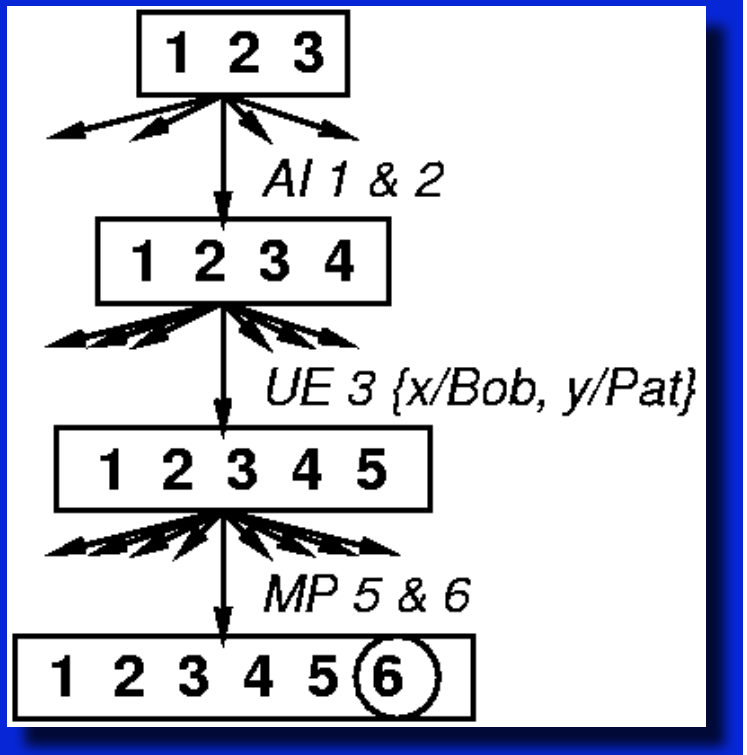
From $\exists x$ Teaches(x, CS348)

we can infer that Teaches(PorkyPig, CS348)

if the symbol PorkyPig is not in the UofD.

Search with primitive inference rules

- Operators are inference rules
- States are sets of sentences
- Goal test checks state to see if it contains query sentence



- Problem: branching factor huge, esp. for UE
- AI, UE, MP is a common inference pattern
- Idea: find a substitution that makes the rule premise match some known facts

  → a single, more powerful inference rule

Unification

A substitution σ unifies atomic sentences p and q if

subst(σ, p)= subst(σ, q) (also denoted pσ=qσ)

p	q	σ
Knows(John,x)	Knows(John, Jane)	{ x/Jane }
Knows(John,x)	Knows(y, Amy)	{ y/John, x/Amy }
Knows(John,x)	Knows(y, Mother(y))	{ y/John, x/Mother(John) }

Idea: Unify rule premises with known facts, apply unifier to conclusion

E.g., if we know both q and Knows(John,x) ⇒ Likes(John,x)

then we conclude

Likes(John,Jane)

Likes(John,Amy)

Likes(John,Mother(John))

A bit more on substitutions

A substitution σ is a set of pairings of variables with terms [symbol | variable | function(term, …)]

$$\sigma = \{ v_1/term_1, v_2/term_2, \ldots \}$$

- Each variable is paired at most once
- A variable's pairing term may not contain the variable directly or indirectly.
  - e.g. can't have substitution { x/f(y), y/f(x) }
- When unifying expressions P and Q, the variable names in P and the variable names in Q should be disjoint.
  - No: UNIFY(Loves (John, x), Loves (x, Jane)) -- No unifier
  - Yes: UNIFY(Loves (John, x), Loves (y, Jane)) σ = { x/Jane, y/John }

Most General Unifier

- Unification is not unique

  Unify (Loves (John, y), Loves (x,y))

  σ = {x/John, y/Jane} or σ = {x/John, y/z }

- Informally, the most general unifier (MGU) imposes the fewest constraints on the terms (contains the most variables).

- Formally, a substitution σ is more general than τ iff there is a substitution δ such that $\sigma\delta=\tau$.

  e.g. σ = { z/ F(w) } is more general than τ = { z/F(C) } since δ = { w/C }

  A most general unifier ρ of ϕ and ψ is such that, for any unifier κ of ϕ and ψ, there exists a substitution $\phi\kappa = \psi\kappa = (\phi\rho)\kappa$.

- MGU's are unique up to variable reordering and changes of naming of variables in terms.

- The unification procedure on pg. 303 gives the MGU

Generalized Modus Ponens (GMP)

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

where $\mathbf{p_i'\sigma = p_i\sigma}$ for all i

p_i and q atomic sentences

Universally quantified variables

For example, let

p_1' = Faster (Bob, Pat)

p_2' = Faster (Pat, Steve)

Faster (x, y) $\wedge$ Faster (y, z) $\Rightarrow$ Faster (x, z)

Unify p_1' and p_2' with the premise

σ = { x/Bob, y/Pat, z/Steve }

Apply substitution to the conclusion

$q\sigma$ = Faster(Bob, Steve)

A GMP Inference Engine

- Perform inference search using only one inference rule, Generalized Modus Ponens

- Every sentence in the database should be put in canonical form
  - an atomic sentence, or
  - $p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q$ where p_i and q are atomic sentences

- These are just Horn sentences

  Can convert other sentences to Horn sentences using Existential Elimination and And-Elimination
- All variables are assumed to be universally quantified
- Previous proof is now one step.

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 and categories from percepts

Forward Chaining Example

- White: Facts added in turn
- Yellow: The result of implication of rules.

1. Buffalo(x) $\wedge$Pig(y) $\Rightarrow$ Faster(x, y)
2. Pig(y) $\wedge$Slug(z) $\Rightarrow$ Faster(y, z)
3. Faster(x, y) $\wedge$Faster(y, z) $\Rightarrow$ Faster(x, z)
4. Buffalo(Bob) [Unifies with 1-a]
5. Pig(Pat) [Unifies with 1-b, GMP Fires]
   [Unifies with 2-a]
6. Faster(Bob,Pat) [Unifies with 3-a, 3-b]
7. Slug(Steve) [Unifies with 2-b, GMP Fires]
8. Faster(Pat, Steve) [Unifies with 3-b and with 6,GMP Fires]
9. Faster (Bob, Steve)
10. …

Backward chaining

$$p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q$$

- 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

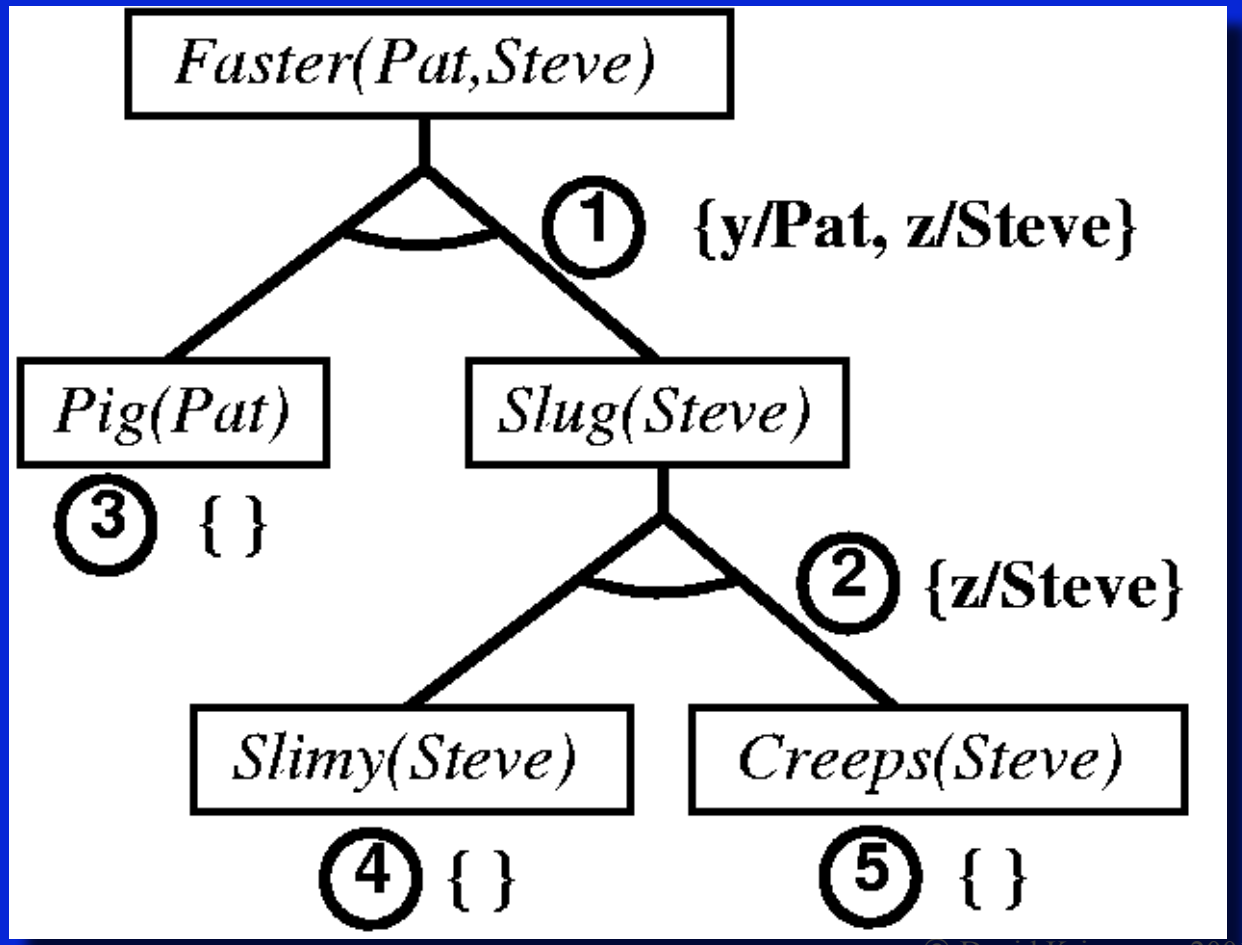
- (Some added complications in keeping track of the unifiers)
- (More complications help to avoid infinite loops)

- Two versions: (1) find **any** solution, (2) find **all** solutions

- Backward chaining is the basis for "logic programming," e.g., Prolog

Backward Chaining Example

1. Pig(y)∧ Slug(z) ⇒ Faster (y, z)
2. Slimy(a)∧ Creeps(a) ⇒ Slug(a)
3. Pig(Pat)
4. Slimy(Steve)
5. Creeps(Steve)



Soundness & Completeness

- An inference procedure is sound if the conclusions is true in all cases where the premises are true.
- Easy to show the **Generalized Modus Ponens (GMP) is sound.**
- Completeness: If KB entails α, then GMP can be used to infer α.

$$PhD(x) \Rightarrow HighlyQualified(x)$$

$$\neg PhD(x) \Rightarrow EarlyEarnings(x)$$

$$HighlyQualified(x) \Rightarrow Rich(x)$$

$$EarlyEarnings(x) \Rightarrow Rich(x)$$

Should be able to infer Rich(Me) [or Rich(Rodrigo) for that matter], but can't since $\neg PhD(x) \Rightarrow EarlyEarnings(x)$ is not a Horn sentence

- Therefore, GMP is NOT complete!

Resolution Refutation

- Given
  - a knowledge base KB (collection of true sentences)
  - a proposition P

  We wish to prove that P is true

- Proof by contradiction:
  - Assume that P is FALSE (i.e., that ¬P is TRUE).
  - Show that a contradiction arises

    - Start with KB
    - Add ¬P to KB
    - Apply resolution rule to KB, adding results to KB
    - If result of resolution rule is FALSE, and we try to add FALSE to KB, then there is a contradiction since KB should only contain true sentences.

Resolution Inference Rule

- Idea: If β is true or α is true
  and β is false or γ is true
  then α or γ must be true
- Basic resolution rule from propositional logic:

$$\frac{\alpha \vee \beta, \quad \neg\beta \vee \gamma}{\alpha \vee \gamma}$$

- Can be expressed in terms of implications

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

- Note that Resolution rule is a generalization of Modus Ponens

$$\frac{\beta, \quad \beta \Rightarrow \gamma}{\gamma} \qquad \text{is equivalent to} \qquad \frac{\text{TRUE} \Rightarrow \beta, \beta \Rightarrow \gamma}{\text{TRUE} \Rightarrow \gamma}$$

Generalized Resolution

Generalized resolution rule for first order logic (with variables)

If p_j can be unified with $\neg q_k$, then we can apply the resolution rule:

$$\frac{p_1 \vee \ldots \vee p_j \vee \ldots \vee p_m \qquad q_1 \vee \ldots \vee q_k \vee \ldots \vee q_n}{\text{Subst}(\theta, (p_1 \vee \ldots \vee p_{j-1} \vee p_{j+1} \vee \ldots \vee p_m \vee q_1 \vee \ldots \vee q_{k-1} \vee q_{k+1} \vee \ldots \vee q_n))}$$

where $\theta = \text{Unify}(p_j, \neg q_k)$

- Can also be expressed as an implication (See Text)
- Example:

KB:	$\neg$ Rich(x) $\vee$ Unhappy(x)
	Rich(Me)
Substitution:	θ = { x/Me }
Conclusion:	Unhappy(Me)

Canonical Form

- For generalized Modus Ponens, entire knowledge base is represented as Horn Sentences.
- For resolution, entire database will be represented using Conjunctive Normal Form (CNF)
- Any first order logic sentence can be converted to a Canonical CNF form.
- Note: Can also do resolution with implicative form, but let's stick to CNF.

Converting any FOL to CNF

- Literal = (possibly negated) atomic sentence, e.g., ¬ Rich(Me)
- Clause = disjunction of literals, e.g., ¬ Rich(Me) ∨ Unhappy(Me)
- The KB is a conjunction of clauses

- Any FOL sentence can be converted to CNF as follows:
  1. Replace $P \Rightarrow Q$ by $\neg P \vee Q$
  2. Move ¬ inwards to literals, e.g., $\neg\forall x\ P$ becomes $\exists x\ \neg P$
  3. Standardize variables, e.g., $(\forall x\ P) \vee (\exists x\ Q)$ becomes $(\forall x\ P) \vee (\exists y\ Q)$
  4. Move quantifiers left in order, e.g., $\forall x\ P \vee \exists y\ Q$ becomes $\forall x\ \exists y\ P \vee Q$
  5. Eliminate ∃ by Skolemization (next slide)
  6. Drop universal quantifiers
  7. Distribute ∧ over ∨ , e.g., $(P \wedge Q) \vee R$ becomes $(P \vee R) \wedge (Q \vee R)$
  8. Flatten nested conjunctions & disjunctions, e.g. $(P \vee Q) \vee R \rightarrow P \vee Q \vee R$

Moving ¬ Inward

- $\neg(P \vee Q)$ becomes $\neg P \wedge \neg Q$
- $\neg(P \wedge Q)$ becomes $\neg P \vee \neg Q$
- $\neg \forall x\ P$ becomes $\exists x\ \neg P$
- $\neg \exists x\ P$ becomes $\forall x\ \neg P$
- $\neg\ \neg\ P$ becomes P

Skolemization

(Thoralf Skolem 1920)

- The process of removing existential quantifiers by elimination.
- Simple case: No universal quantifiers

  → Existential Elimination Rule
- For example:

  $\exists x \; Rich(x)$

  becomes

  $Rich(G1)$

  where $G1$ is a new ``Skolem constant''

- More tricky when $\exists$ is inside $\forall$

Skolemization – cont.

- More tricky when $\exists$ is inside $\forall$

  E.g., ``Everyone has a heart''

  $$\forall x \; Person(x) \Rightarrow \exists y \; Heart(y) \wedge Has(x,y)$$

- Incorrect:

  $$\forall x \; Person(x) \Rightarrow Heart(H1) \wedge Has(x,H1)$$

  This means everyone has the same heart called $H1$

- Problem is that for each person, we need another "heart" – i.e., consider the "heart" to be a function of the person.

- Correct:

  $$\forall \; Person(x) \Rightarrow Heart(H(x)) \wedge Has(x,H(x))$$

  where H is a new symbol (``Skolem function'')

- Skolem function arguments: all enclosing universally quantified variables