First-Order Inference

Burr H. Settles
CS-540, UW-Madison
www.cs.wisc.edu/~cs540-1
Summer 2003

Announcements (7/1)

- Discussion list topic #2: pick a favorite saying and translate it to FOL
  - Must be well-formed
  - Have at least 1 quantified variable
  - Have at least 2 connectives (∧∨⇒)
- If someone post a translation that you disagree with, say so! (it is a *discussion* list, after all… you're not being graded on being wrong or right!)

Announcements (7/2)

- Excellent FOL translations going up on the discussion list! Feel free to post more than one!
- Plan for the next week and a half:
  - *No class* Friday (July 4)
  - Lectures Mon & Tues next week
  - Review session Wed (TAs will run it)
  - Midterm on Thursday (July 10, in class)
  - *No class next* Friday (July 11)
- Project papers/reports will be due Friday, August 1 (second to last week of classes)

Inference Rules for FOL

- All inference rules for PL also apply to FOL (MP, AE, AI, OI, DNE, UR, R, DML)
- **Universal Elimination, UE** $\dfrac{\forall v\ \alpha}{\text{SUBST}(\{v/g\}, \alpha)}$
  variable substituted with ground term
  `∀x Eats(Jim, x)` infer `Eats(Jim, Cake)`
- **Existential Elimination, EE** $\dfrac{\exists v\ \alpha}{\text{SUBST}(\{v/k\}, \alpha)}$
  variable substituted with *new* constant
  `∃x Eats(Jim, x)` infer `Eats(Jim, NewFood)`
- **Existential Introduction, EI** $\dfrac{\alpha}{\exists v\ \text{SUBST}(\{g/v\}, \alpha)}$
  ground term substituted with variable
  `Eats(Jim, Cake)` infer `∃x Eats(x, Cake)`

Proofs for FOL

- "Thom is a turtle."
  1. turtle(Thom)
- "Rob is a rabbit."
  2. rabbit(Rob)
- "Turtles outlast rabbits."
  3. ∀x,y turtle(x) ∧ rabbit(y) ⇒ outlasts(x,y)
- Prove: "Thom outlasts Rob."
  ? outlasts(Thom,Rob)

Proofs for FOL

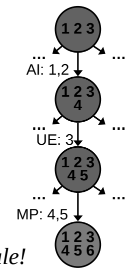
- And Introduction: 1, 2
  4. turtle(Thom) ∧ rabbit(Rob)
- Universal Elimination: 3 **{x/Thom, y/Rob}**
  5. turtle(Thom) ∧ rabbit(Rob) ⇒ outlasts(Thom,Rob)
- Modus Ponens: 4, 5
  6. outlasts(Thom,Rob)

∗ *AI, UE, MP is a common inference pattern*

FOL Proof as Search

- **States:** the current KB
- **Actions:** inference rules
- **Goal test:** see if query is in KB
- **Problem:** huge branching factor (especially for UE)
- **Idea:** Find a substitution that makes the rule premise match known facts

∗ *Make new, powerful inference rule!*



Automated Inference in FOL

∗ *Automated inference is harder for FOL than it is for PL*

- Variables can take on a potentially infinite number of possible values from the domain
- Thus… UE can be applied in a potentially infinite number of ways to KB!

Generalized Modus Ponens (GMP)

∗ *Unify rule premises with known facts and apply unifier to conclusion*

- Rule: `∀x,y turtle(x) ∧ rabbit(y) ⇒ outlasts(x,y)`
  - Known facts: `turtle(Thom)`, and `rabbit(Rob)`
  - Unifier: `{x/Thom, y/Rob}`
- Apply unifier to conclusion: `outlasts(Thom,Rob)`

Generalized Modus Ponens (GMP)

- Combines AI, UE, and MP into a single rule

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

(where $SUBST(\theta,p_i') = SUBST(\theta,p_i)$ for all i)

- $SUBST(\theta,\alpha)$ means apply substitutions in θ to sentence α
- Substitution list $\theta = \{v_1/t_1, v_2/t_2, \ldots, v_n/t_n\}$ means
  - Replace all occurrences of variable v_i with term t_i
  - Substitutions are made in left to right order

Generalized Modus Ponens (GMP)

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

(where $SUBST(\theta,p_i') = SUBST(\theta,p_i)$ for all i)

- All variables assumed to be universally quantified
- Used with a KB in Horn normal form (HNF): Horn sentence: disjunction with one positive literal
  - single atomic sentence: `P(x)`
  - (conj. of atoms) ⇒ atom: `¬P(x) ∨ ¬Q(x) ∨ R(x)`

Generalized Modus Ponens (GMP)

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

(where $SUBST(\theta,p_i') = SUBST(\theta,p_i)$ for all i)

Example:

p_1' = `taller(Larry,Curly)`
p_2' = `taller(Curly,Moe)`
$p_1 \wedge p_2 \Rightarrow q$ = `taller(x,y) ∧ taller(y,z) ⇒ taller(x,z)`
θ = `{x/Larry, y/Curly, z/Moe}`
$SUBST(\theta,q)$ = `taller(Larry,Moe)`

Unification

- Substitution θ is said to unifiy p and q if $SUBST(\theta,p) = SUBST(\theta,q)$

p	q	$\boldsymbol{\theta}$
`turtle(y)`	`turtle(Thom)`	`{y/Thom}`
`loves(Burr,x)`	`loves(Burr,Nat)`	`{x/Nat}`
`friends(Burr,x)`	`friends(y,Mark)`	`{y/Burr, x/Mark}`
`obeys(Ron,x)`	`obeys(z,mother(z))`	`{z/Ron,x/mother(Ron)}`
`eats(y,y)`	`eats(z,Fish)`	`{y/z, z/Fish}`
`sees(JD,x,y)`	`sees(z,DJ,home(z))`	`{z/JD,x/DJ,y/home(JD)}`
`sees(x,id(x), home(JD))`	`sees(DJ,id(y),home(y))`	`failure, assuming home(JD) ≠ home(DJ)`

Unification Algorithm

```
// p.278 has a more detailed version of the algorithm
unify (P, Q, THETA) {
  if no differences then return THETA
  else {
    R = mismatch term in P      // where R ≠ S
    S = mismatch term in Q
  }
  if R is a variable then {
    if R is in S then return FAILURE
    add {R/S} to THETA
    return unify(P-sub-THETA, Q-sub-THETA, THETA)
  }
  else if S is a variable {
    if R is in S then return FAILURE
    add {S/R} to THETA
    return unify(P-sub-THETA, Q-sub-THETA, THETA)
  }
  else return FAILURE           // no unifier found
}
```

Unification Algorithm

- θ is a most general unifier (MGU)
  - Shortest length substitution list to make a match
  - In general, more than one MGU
- Our algorithm recursively explores the two expressions and simultaneously builds θ
- We want to prevent replacing variables with terms that contains that variable (*e.g.* `{x/F(x)}`)
  - This slows down the algorithm
- Unification with this variable-substitution check has a time complexity of $O(n^2)$, where n is the number of terms in the expressions

Soundness of GMP

- **Note:** $\alpha\theta$ is the same as $SUBST(\theta,\alpha)$
- We want to show:
  $p_1', p_2', \ldots, p_n', (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q) \models q\theta$
  provided $p_i'\theta = p_i\theta$ for all i
- Lemma: for any Horn clause p: $p \models p\theta$ by UE

1. $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \models (p_1 \wedge \ldots \wedge p_n \Rightarrow q)\theta = (p_1\theta \wedge \ldots \wedge p_n\theta \Rightarrow q\theta)$
2. $p_1', \ldots, p_n' \models p_1' \wedge \ldots \wedge p_n' \models p_1'\theta \wedge \ldots \wedge p_n'\theta$
3. $q\theta$ (MP: 1,2) since $p_i'\theta = p_i\theta$ for all i

Completeness of FOL Inference

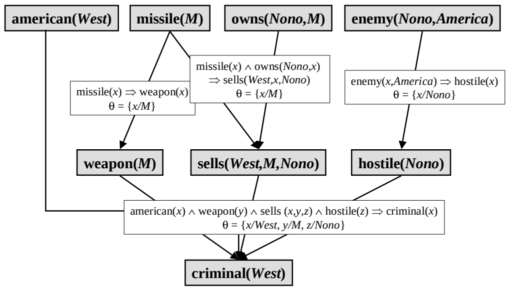
- Truth table enumeration: incomplete for FOL (table may be infinite in size)
- Natural Deduction: complete for FOL (but impractical… branching factor is too large)
- GMP: incomplete for FOL (not all sentences can be converted to Horn form)
- GMP: complete for FOL KB in HNF
  - Forward chaining: move from KB to query
  - Backward chaining: move from query to KB

Inference Example

"The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, and enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is an American."

1. american(x) $\wedge$ weapon(y) $\wedge$ sells (x,y,z) $\wedge$ hostile(z) $\Rightarrow$ criminal(x)
2. enemy(*Nono*,*America*)
$\exists x$ owns(*Nono*,x) $\wedge$ missile(x) ← ***Must be in HNF! Use EE and generate 2 sentences***
3. owns(*Nono*,*M*)
4. missile(*M*)
5. missile(x) $\wedge$ owns(*Nono*,x) $\Rightarrow$ sells(*West*,x,*Nono*)
6. american(*West*)
7. enemy(x,*America*) $\Rightarrow$ hostile(x)
8. missile(x) $\Rightarrow$ weapon(x)

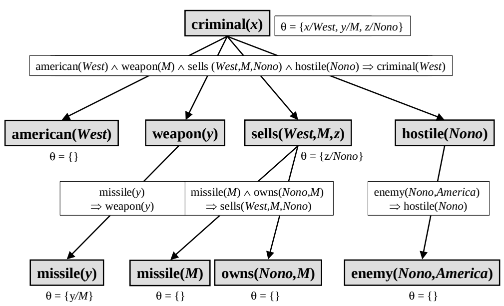
Forward Chaining with GMP



Forward Chaining with GMP

- For full FC Algorithm, see page 282
- Sound and complete for first-order definite clauses
  - Proof similar to PL proof
- Datalog: FOL clauses with no functions (*e.g.* crime KB)
  - FC terminates for Datalog in at most pn^k literals
- FC typically adds all sentences that can be inferred
  - Matching premises against known facts in NP-Hard
- Still, FC is used widely in deductive databases

Backward Chaining with GMP



Backward Chaining with GMP

- For full BC Algorithm, see page page 288
- DFS recursive proof search
  - Space is linear in the size of the proof
- Incomplete due to possible infinite loops
  - Fix by checking current goal against every goal on the stack
- Inefficient due to repeated subgoals
  - Complications added to keep track of unifiers
  - Fix by caching previous results (but this uses up space!)
- Two versions: find *any* solution, find *all* solutions
- BC still used (without improvements!) extensively for logic programming systems (e.g. Prolog)

Completeness of General FOL

- FC and BC are complete for KBs in Horn form, but incomplete for general FOL:
  `owns(Burr,x) ∧ shoe(x) ⇒ stinky(x)`
  `shoe(x) ∧ stinky(x) ⇒ ¬allowed(x)`
  `?- shoe(x) ∧ owns(Burr,x) ∧ allowed(x)`
- Can't prove query with FC or BC… why?
- Does a complete algorithm for FOL exist?

Brief History of Reasoning

- 450BC **Stoics** **PL, inference (?)**
- 32BC **Aristotle** **inference rules (syllogisms), quantifiers**
- 1565 **Cardano** **PL + uncertainty (probability theory)**
- 1847 **Boole** **PL (again)**
- 1879 **Frege** **FOL**
- 1922 **Wittgenstein** **proof using truth table**
- 1930 **Gödel** **complete algo for FOL exists**
- 1930 **Herbrand** **complete algo for FOL (reduce to PL)**
- 1931 **Gödel** **no complete algo for number theory**
- 1960 **Davis/Putnam** **practical algo for PL**
- 1965 **Robinson** **practical algo for FOL (resolution)**

Resolution

- Entailment in general FOL is only semi-decidable:
  - Can prove α if **KB** $\models \alpha$
  - Cannot always prove that **KB** doesn't $\models \alpha$ (halting)
- Resolution is a refutation technique:
  - To prove **KB** $\models \alpha$ show that **KB** $\wedge \neg\alpha$ is unsatisfiable
- Resolution uses **KB** and $\neg\alpha$ in CNF:
  - Conjunction of clauses that are disjunction of literals

∗ *Resolution repeatedly combines two clauses to make a new one until an empty clause is derived (a contradiction)*

Resolution

- **Resolution in PL**

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

equivalently:

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

- **Generalized Resolution (GR) for FOL:**

where p_i and q_i are literals for all i
where $UNIFY(p_j, q_k) = \theta$, and q_k is the negation of p_j

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

Resolution Refutation

$$\frac{\texttt{well-fed(Me)},\ \neg\texttt{well-fed(x)} \vee \texttt{happy(x)}}{SUBST(\theta,\ \texttt{happy(x)})}$$

- p_j is **`well-fed(Me)`**
  q_k is $\neg$**`well-fed(x)`**
- $UNIFY(p_j, q_k)$ result in θ **`= {x/Me}`**
  $SUBST(\theta,$ **`happy(x)`**$)$ result in **`happy(Me)`**

  Inferred sentence: `happy(Me)`

∗ ***GMP is a special case of generalized resolution (for KBs in HNF)***

Resolution Refutation

- Can be thought of as search
  - Reversed construction of search tree (leaves to root)
  - Leaves are KB clauses and ¬query
  - Resolvent is new node with arcs to parent clauses
  - Root is a clause containing false
- A search is complete if it guarantees that the empty clause (*i.e.* false) can be derived whenever **KB** $\models$ **q**

∗ *Goal is to design a complete search that efficiently finds a contradiction (i.e. empty clause)*

Resolution Refutation Example

Recycling the "West is a criminal" example, let's begin by making sure that all the facts and rules in our KB are in CNF. The following are already in CNF:

enemy(*Nono*,*America*) **owns(*Nono*,*M*)**
missile(*M*) **american(*West*)**

The remaining four need to be converted to CNF:

american(*x*) ∧ weapon(*y*) ∧ sells (*x*,*y*,*z*) ∧ hostile(*z*) ⇒ criminal(*x*)
¬american(*x*) ∨ ¬weapon(*y*) ∨ ¬sells (*x*,*y*,*z*) ∨ ¬hostile(*z*) ∨ criminal(*x*)

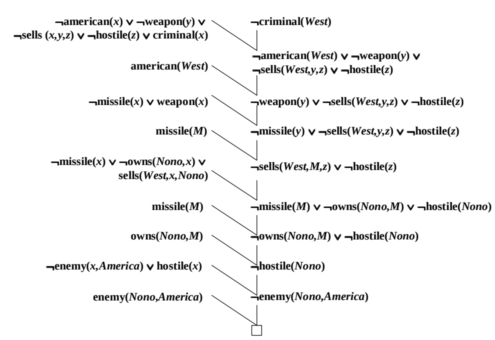
missile(*x*) ∧ owns(*Nono*,*x*) ⇒ sells(*West*,*x*,*Nono*)
¬missile(*x*) ∨ ¬owns(*Nono*,*x*) ∨ sells(*West*,*x*,*Nono*)

enemy(*x*,*America*) ⇒ hostile(*x*)
¬enemy(*x*,*America*) ∨ hostile(*x*)

missile(*x*) ⇒ weapon(*x*)
¬missile(*x*) ∨ weapon(*x*)

And we also need to negate our query: **¬criminal(*West*)**

Resolution Refutation Example



Conjunctive Normal Form

- The KB needs to be a conjunction of CNF clauses and/or literals
- CNF clause: disjunction of literals
  - *e.g.* `hot(x)` ∨ `warm(x)` ∨ `cold(x)`
- Literal: atom (may be negated)
  - *e.g.* ¬`happy(Sally)`

✱ *Any FOL KB can be converted into CNF*

Converting FOL to CNF

1. Replace ⇔ with equivalent: **P ⇔ Q** **becomes P ⇒ Q ∧ Q ⇒ P**
2. Replace ⇒ with equivalent: **P ⇒ Q** **becomes ¬P ∨ Q**
3. Reduce scope of ¬ to single literals:
   - **¬¬P** **becomes P** (DNE)
   - **¬(P ∨ Q)** **becomes ¬P ∧ ¬Q** (de Morgan's)
   - **¬(P ∧ Q)** **becomes ¬P ∨ ¬Q** (de Morgan's)
   - **¬∀x P** **becomes ∃x ¬P**
   - **¬∃x P** **becomes ∀x ¬P**
4. Standardize variables apart:
   - Each quantifier must have a unique variable name
   - Avoids confusion in steps 5 and 6
   - *e.g.* **[∀x P] ∨ [∃x Q]** **becomes ∀x P ∨ ∃y Q**

Converting FOL to CNF

5. Eliminate existential quantifiers (Skolemize):
   ∀x P(x) **becomes P(K)** (EE)
   K is some new constant (Skolem constant)
   - *e.g.* **∀x∃y P(x,y) becomes ∀x P(x,F(x))**
     F() must be a new function (Skolem function) with arguments that are all enclosing universlly quantified variables
   - Everyone has a name.
     ∀x person(x) ⇒ ∃y name(y) ∧ has(x,y)
     wrong: ∀x person(x) ⇒ name(K)∧has(x,K)
     Everyone has the same name K!!
     We want everyone to have a name based on who they are
     right: ∀x person(x) ⇒ name(F(x))∧has(x,F(x))

Converting FOL to CNF

6. Drop universal quantifiers:
   - All variables are only universally quantified after step 5
   - *e.g.* **∀x P(x) ∨ ∀y Q(y) becomes P(x) ∨ Q(y)**
   - All variables in KB will be assumed to be universally quantified
7. Distribute ∨ over ∧ :
   (P ∧ Q)∨ R becomes (P ∨ R)∧(Q ∨ R)
8. Group conjunctions/disjunctions together:
   (P ∧ Q)∧ R becomes (P ∧ Q ∧ R)
   (P ∨ Q)∨ R becomes (P ∨ Q ∨ R)

FOL-CNF Conversion Example

"Everyone who loves all animals is loved by someone."
∀*x* [∀*y* animal(*y*) ⇒ loves(*x*,*y*)] ⇒ [∃*y* loves(*y*,*x*)]

1&2. Eliminate biconditionals and implications
∀*x* ¬[∀*y* ¬animal(*y*) ∨ loves(*x*,*y*)] ∨ [∃*y* loves(*y*,*x*)]

3. Reduce scope of ¬ to single literals
∀*x* [∃*y* ¬{¬animal(*y*) ∨ loves(*x*,*y*)}] ∨ [∃*y* loves(*y*,*x*)]
∀*x* [∃*y* ¬¬animal(*y*) ∧ ¬loves(*x*,*y*)] ∨ [∃*y* loves(*y*,*x*)]
∀*x* [∃*y* animal(*y*) ∧ ¬loves(*x*,*y*)] ∨ [∃*y* loves(*y*,*x*)]

FOL-CNF Conversion Example

4. Standardize variables apart
∀*x* [∃*y* animal(*y*) ∧ ¬loves(*x*,*y*)] ∨ [∃*z* loves(*z*,*x*)]

5. Eliminate Existentials by skolemizing
∀*x* [animal(*F(x)*) ∧ ¬loves(*x*,*F(x)*)] ∨ loves(*G(x)*,*x*)

6. Drop universals
[animal(*F(x)*) ∧ ¬loves(*x*,*F(x)*)] ∨ loves(*G(x)*,*x*)

7&8. Distribute ∨ over ∧ , group conjunctions/disjunctions:
[animal(*F(x)*) ∨ loves(*G(x)*,*x*)] ∧ [¬loves(*x*,*F(x)*) ∨ loves(*G(x)*,*x*)]

Summary

- First-Order logic is a language that is very expressive, but difficult to perform inference with
  - One inference method is removing all variables/ quantifiers (*i.e.* propositionalizing), which is slow
  - We can also use unification to identify appropriate substitutions with generalized Modus Ponens (GMP)
  - The Forward Chaining and Backward Chaining algorithms use GMP to KBs in HNF
  - GMP complete for definite clauses (HNF), but not general FOL domains

Summary

- Generalized resolution inference provides a complete proof system for all of FOL
  - KBs must be converted to CNF
  - Resolution refutation is an efficient strategy for proving a query q by showing that it's negation $\neg q$ is inconsistent with the KB
  - *i.e.* show that KB $\wedge \neg q$ is unsatisfiable