CSCI 561
Foundation for Artificial Intelligence

13-14: Inference in FOL

Professor Wei-Min Shen
University of Southern California

Inference in First-Order Logic

= solution in search

(1)

- Proofs – extend propositional logic inference to deal with quantifiers

(2)

- Unification
- Generalized modus ponens
- Forward and backward chaining – inference rules and reasoning program

(3)

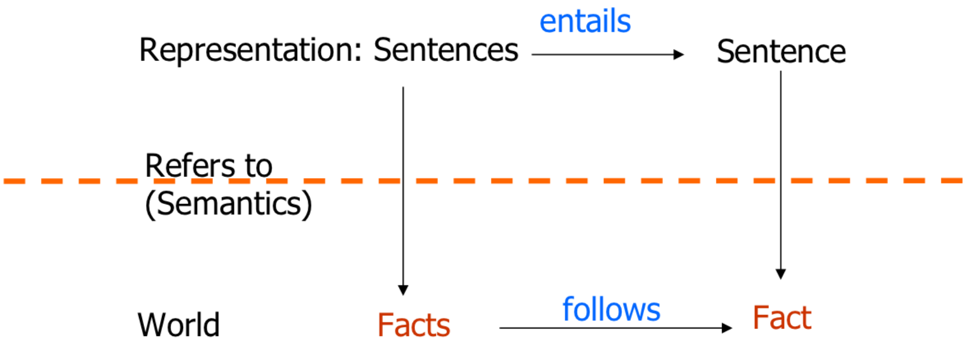
- Completeness – Gödel's theorem: for FOL, any sentence entailed by another set of sentences can be proved from that set

(4)

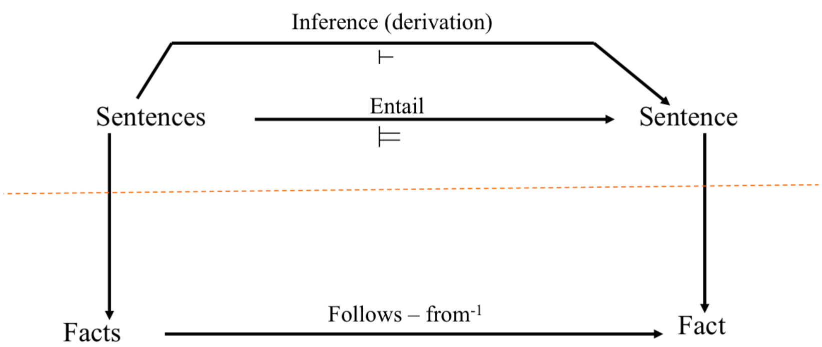
- Resolution – inference procedure that is complete for any set of sentences

Review

Logic as a representation of the World



Desirable Properties of Inference Procedures



Remember: Inference for Propositional Logic

◇ **Modus Ponens** or **Implication-Elimination**: (From an implication and the premise of the implication, you can infer the conclusion.)

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

◇ **And-Elimination**: (From a conjunction, you can infer any of the conjuncts.)

$$\frac{\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n}{\alpha_i}$$

◇ **And-Introduction**: (From a list of sentences, you can infer their conjunction.)

$$\frac{\alpha_1, \ \alpha_2, \quad \ldots, \quad \alpha_n}{\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n}$$

◇ **Or-Introduction**: (From a sentence, you can infer its disjunction with anything else at all.)

$$\frac{\alpha_i}{\alpha_1 \vee \alpha_2 \vee \ldots \vee \alpha_n}$$

◇ **Double-Negation Elimination**: (From a doubly negated sentence, you can infer a positive sentence.)

$$\frac{\neg\neg\alpha}{\alpha}$$

◇ **Unit Resolution**: (From a disjunction, if one of the disjuncts is false, then you can infer the other one is true.)

$$\frac{\alpha \vee \beta, \qquad \neg\beta}{\alpha}$$ or equivalently $$\frac{\neg\beta \Rightarrow \alpha, \quad \neg\beta}{\alpha}$$

◇ **Resolution**: (This is the most difficult. Because β cannot be both true and false, one of the other disjuncts must be true in one of the premises. Or equivalently, implication is transitive.)

$$\frac{\alpha \vee \beta, \qquad \neg\beta \vee \gamma}{\alpha \vee \gamma}$$ or equivalently $$\frac{\neg\alpha \Rightarrow \beta, \qquad \beta \Rightarrow \gamma}{\neg\alpha \Rightarrow \gamma}$$

(-)

Proofs (3 new inference rules for FOL)

1. The three new inference rules for FOL (compared to pro|

Ground term: A term that does not contain a variable.
- A constant symbol
- A function applies to some ground term

(1) **Universal Elimination (UE):**

for any sentence α, variable x and ground term τ,

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

{x/a}: substitution/binding list

e.g., from ∀x Likes(x, Candy) and {x/Joe} we can infer Likes(Joe, Candy)

(2) **Existential Elimination (EE):**

for any sentence α, variable x and a new constant symbol k not in KB,

$$\frac{\exists x\ \alpha}{\alpha\{x/k\}}$$

e.g., from ∃x Kill(x, Victim) we can infer Kill(Murderer, Victim), if Murderer is a new symbol

(3) **Existential Introduction (EI):** must exist one x makes α true

for any sentence α, variable x not in α and ground term g in α,

$$\frac{\alpha}{\exists x\ \alpha\{g/x\}}$$

e.g., from Likes(Joe, Candy) we can infer ∃x Likes(x, Candy) eg:

Example Proof

predicate

relation

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

2. Search with primitive example rules

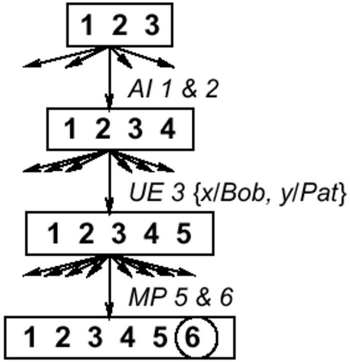
不能用DFS，因为 never come back

也不能用BFS. branching factor can be infinite.

Operators are inference rules

States are sets of sentences

Goal test checks state to see if it contains query sentence



AI, UE, MP is a common inference pattern

Problem: branching factor huge, esp. for UE

Idea: find a substitution that makes the rule premise match some known facts

$\Rightarrow$ a single, more powerful inference rule

(2) not always possible

1. Unification : unify two sentence with same structure but different variables, using substitution makes them same

A substitution σ unifies atomic sentences p and q if $p\sigma = q\sigma$

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

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

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

then we conclude $Likes(John, Jane)$

$Likes(John, OJ)$

$Likes(John, Mother(John))$

Goal of unification: finding σ, to make two things equal.

make us can apply rules

eg:

Extra Examples for Unification

(1)

P	Q	σ
Student(x)	Student(Bob)	{x/Bob}
Sells(Bob, x)	Sells(x, coke)	{x/coke, x/Bob} Is it correct? X
Sells(Bob, x)	Sells(y, coke) change to Y	{x/coke, y/Bob}

VARIABLE term

(2)

1 – unify(P(a,X), P(a,b)) σ = {X/b}

2 – unify(P(a,X), P(Y,b)) σ = {Y/a, X/b}

3 – unify(P(a,X), P(Y,f(a))) σ = {Y/a, X/f(a)}

4 – unify(P(a,X), P(X,b)) σ = failure

Note: If P(a,X) and P(X,b) are independent, then we can replace X with Y and get the unification to work.

2. goal of unification is to apply rules

Generalized Modus Ponens (GMP)

1)

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

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

$$\begin{aligned}
\text{E.g. } p_1{}' &= \text{Faster(Bob,Pat)} \\
p_2{}' &= \text{Faster(Pat,Steve)} \\
p_1 \wedge p_2 \ \Rightarrow\ q &= Faster(x,y) \wedge Faster(y,z) \ \Rightarrow\ Faster(x,z) \\
\sigma &= \{x/Bob, y/Pat, z/Steve\} \\
q\sigma &= Faster(Bob, Steve)
\end{aligned}$$

GMP used with KB of definite clauses (*exactly* one positive literal):
either a single atomic sentence or
(conjunction of atomic sentences) $\Rightarrow$ (atomic sentence)
All variables assumed universally quantified

(2) Soundness of GMP

即证明: $p_1', \ldots, p_n', (p_1 \wedge \ldots \wedge p_n) \Rightarrow q) \vdash_i q\sigma$ 时 ~ $\models q\sigma$

GMP

Need to show that

prove: $p_1', \ldots, p_n', \ (p_1 \wedge \ldots \wedge p_n \Rightarrow q) \models q\sigma$

provided that $p_i'\sigma = p_i\sigma$ for all i

Lemma: For any definite clause p, we have $p \models p\sigma$ by UE

do variable substitution(σ) on rule

distribute substitution to individual one

1. $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \models (p_1 \wedge \ldots \wedge p_n \Rightarrow q)\sigma = (p_1\sigma \wedge \ldots \wedge p_n\sigma \Rightarrow q\sigma)$

AI

2. $p_1', \ldots, p_n' \models p_1' \wedge \ldots \wedge p_n' \models p_1'\sigma \wedge \ldots \wedge p_n'\sigma$

3. From 1 and 2, $q\sigma$ follows by simple MP

(3)

Properties of GMP

- Why is GMP an efficient inference rule?

  - It takes bigger steps, combining several small inferences into one

  - It takes sensible steps: uses eliminations that are guaranteed to help (rather than random UEs)

  - It uses a precompilation step which converts the KB to canonical form (Horn sentences)

  Remember: sentence in Horn from is a conjunction of Horn clauses (clauses with at most one positive literal), e.g.,
  $(A \lor \neg B) \land (B \lor \neg C \lor \neg D)$, that is $(B \Rightarrow A) \land ((C \land D) \Rightarrow B)$

3、Horn Form

- We convert sentences to Horn form as they are entered into the KB
- Using Existential Elimination and And Elimination

$\frac{\exists x, \alpha}{\alpha\{x/k\}}$ $\frac{\alpha_1 \wedge \alpha_2 \wedge \cdots \wedge \alpha_n}{\alpha_i}$

- e.g., ∃x Owns(Nono, x) ∧ Missile(x) becomes

  Owns(Nono, M)
  Missile(M)

(where M is a new symbol that was not already in the KB)

4. search tenique:

Forward Chaining

KB中有sentence，一些是rules，一些是facts，代入fact若有rules能得facts之间生成new fact，则将new fact添加进KB，直到是想要的α

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

，然后可能生成目标sentence α

Forward chaining is data-driven
  e.g., inferring properties and categories from percepts

eg:

Add facts 1, 2, 3, 4, 5, 7 in turn.
Number in [] = unification literal; $\surd$ indicates rule firing

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. $Buffalo(Bob)$ [1a, ×] 对应可以应用的rule 1中的第a个(1a)，但不能生成new fact (×)
5. $Pig(Pat)$ [1b, $\surd$] → 6. $Faster(Bob, Pat)$ [3a, ×], [3b, ×] → rule 3中有2个相同的relation，所以同时检查3a, 3b
   第b个(1b)，可以生成new fact 6 (√)
   [2a, ×]
7. $Slug(Steve)$ [2b, $\surd$]
   → 8. $Faster(Pat, Steve)$ [3a, ×], [3b, $\surd$]
     → 9. $Faster(Bob, Steve)$ [3a, ×], [3b, ×]

5. Backward Chaining

Starting with α, 找premise证明α, 放进KB
⇒ 重复找 premise' 证明 premise, 放进KB
⇒ 直到找到的premise已经在KB之中

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)

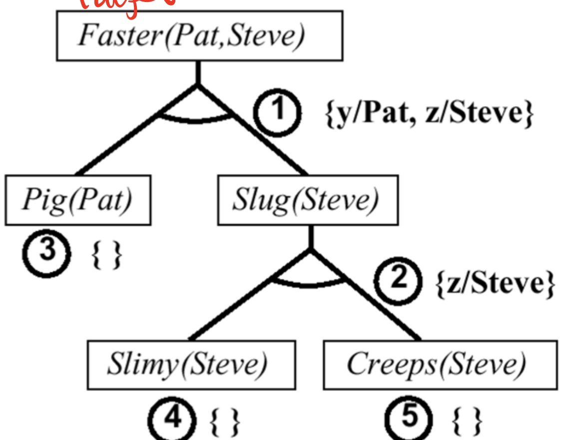
(More complications help to avoid infinite loops)

Two versions: find any solution, find all solutions

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

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

target



b.(1)

Forward Chaining

- Nintendo example.
  - Nintendo says it is Criminal for a programmer to provide emulators to people. My friends don't have a Nintendo 64, but they use software that runs N64 games on their PC, which is written by Reality Man, who is a programmer.

- The knowledge base initially contains:

has

$$\text{Programmer}(x) \wedge \text{Emulator}(y) \wedge \text{People}(z) \wedge \text{Provide}(x,z,y) \Rightarrow \text{Criminal}(x) \qquad (1)$$

$$\text{Use}(\text{friends}, x) \wedge \text{Runs}(x, \text{N64 games}) \Rightarrow \text{Provide}(\text{Reality Man}, \text{friends}, x) \qquad (2)$$

$$\text{Software}(x) \wedge \text{Runs}(x, \text{N64 games}) \Rightarrow \text{Emulator}(x) \qquad (3)$$

- Now we add atomic sentences to the KB sequentially, and call on the forward-chaining procedure:
  - FORWARD-CHAIN(KB, Programmer(Reality Man))

FORWARD-CHAIN(KB, Programmer(Reality Man))

Programmer(x) ∧ Emulator(y) ∧ People(z) ∧ Provide(x,z,y) ⇒ Criminal(x) (1)
Use(friends, x) ∧ Runs(x, N64 games) ⇒ Provide(Reality Man, friends, x) (2)
Software(x) ∧ Runs(x, N64 games) ⇒ Emulator(x) (3)

Programmer(Reality Man) (4)

- This new premise unifies with (1) with **subst(**{x/Reality Man}, Programmer(x)**)** but not all the premises of (1) are yet known, so nothing further happens.

- Continue adding atomic sentences:
  - FORWARD-CHAIN(KB, People(friends))

Forward Chaining

Programmer(x) ∧ Emulator(y) ∧ People(z) ∧ Provide(x,z,y) ⇒ Criminal(x) (1)
Use(friends, x) ∧ Runs(x, N64 games) ⇒ Provide(Reality Man, friends, x) (2)
Software(x) ∧ Runs(x, N64 games) ⇒ Emulator(x) (3)

Programmer(Reality Man) (4)
People(friends) (5)

- This also unifies with (1) with **subst(**{z/friends}, People(z)**)** but other premises are still missing.

- Add:
  - FORWARD-CHAIN(KB, Software(U64))

Forward Chaining

Programmer(x) ∧ Emulator(y) ∧ People(z) ∧ Provide(x,z,y) ⇒ Criminal(x) (1)
Use(friends, x) ∧ Runs(x, N64 games) ⇒ Provide(Reality Man, friends, x) (2)
Software(x) ∧ Runs(x, N64 games) ⇒ Emulator(x) (3)
Programmer(Reality Man) (4)
People(friends) (5)
Software(U64) (6)

- This new premise unifies with (3) but the other premise is not yet known.

- Add:
  - FORWARD-CHAIN(KB, Use(friends, U64))

Forward Chaining

$Programmer(x) \wedge Emulator(y) \wedge People(z) \wedge Provide(x,z,y) \Rightarrow Criminal(x)$ (1)
$Use(friends, x) \wedge Runs(x, N64\ games) \Rightarrow Provide(Reality\ Man, friends, x)$ (2)
$Software(x) \wedge Runs(x, N64\ games) \Rightarrow Emulator(x)$ (3)

Programmer(Reality Man) (4)
People(friends) (5)
Software(U64) (6)
Use(friends, U64) (7)

- This premise unifies with (2) but one still lacks.
  - Add:
    - FORWARD-CHAIN(Runs(U64, N64 games))

Forward Chaining

Programmer(x) ∧ Emulator(y) ∧ People(z) ∧ Provide(x,z,y)⇒ Criminal(x) (1)
Use(friends, x) ∧ Runs(x, N64 games) ⇒ Provide(Reality Man, friends, x) (2)
Software(x) ∧ Runs(x, N64 games) ⇒ Emulator(x) (3)

Programmer(Reality Man) (4)
People(friends) (5)
Software(U64) (6)
Use(friends, U64) (7)
Runs(U64, N64 games) (8)

- This new premise unifies with (2) and (3).

  Premises (6), (7) and (8) satisfy the implications fully.

So we can infer the consequents, which are now added to the knowledge base (this is done in two separate steps).

(9) (10)

Forward Chaining

Programmer(x) ∧ Emulator(y) ∧ People(z) ∧ Provide(x,z,y)⇒ Criminal(x) (1)
Use(friends, x) ∧ Runs(x, N64 games) ⇒ **Provide(Reality Man, friends, x)** (2)
Software(x) ∧ Runs(x, N64 games) ⇒ **Emulator(x)** (3)

Programmer(Reality Man) (4)
People(friends) (5)
Software(U64) (6)
Use(friends, U64) (7)
Runs(U64, N64 games) (8)
Provide(Reality Man, friends, U64) (9)
Emulator(U64) (10)

- Addition of these new facts triggers further forward chaining.

Forward Chaining

Programmer(x) ∧ Emulator(y) ∧ People(z) ∧ Provide(x,z,y)⇒ Criminal(x) (1)
Use(friends, x) ∧ Runs(x, N64 games) ⇒ **Provide(Reality Man, friends, x)** (2)
Software(x) ∧ Runs(x, N64 games) ⇒ **Emulator(x)** (3)

Programmer(Reality Man) (4)
People(friends) (5)
Software(U64) (6)
Use(friends, U64) (7)
Runs(U64, N64 games) (8)
Provide(Reality Man, friends, U64) (9)
Emulator(U64) (10)
Criminal(Reality Man) (11)

- Which results in the final conclusion: Criminal(Reality Man)

Forward Chaining

- Forward Chaining acts like a breadth-first search at the top level, with depth-first sub-searches.
- Since the search space spans the entire KB, a large KB must be organized in an intelligent manner in order to enable efficient searches in reasonable time.

(2).

Backward Chaining

- The algorithm (available in detail in textbook):
  - a knowledge base KB
  - a desired conclusion c or question q
  - finds all sentences that are answers to q in KB *or* prove c
    - if q is directly provable by premises in KB, infer q and remember how q was inferred (building a list of answers).
    - find all implications that have q as a consequent.
    - for each of these implications, find out whether all of its premises are now in the KB, in which case infer the consequent and add it to the KB, remembering how it was inferred. If necessary, attempt to prove the implication also via backward chaining
    - premises that are conjuncts are processed one conjunct at a time

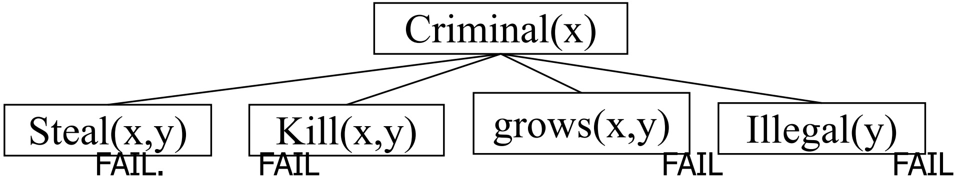
Backward Chaining

- Question: Has Reality Man done anything criminal?
  - Criminal(Reality Man)

- Possible answers:
  - Steal(x, y) ⇒ Criminal(x)
  - Kill(x, y) ⇒ Criminal(x)
  - Grow(x, y) ∧ Illegal(y) ⇒ Criminal(x)
  - HaveSillyName(x) ⇒ Criminal(x)
  - Programmer(x) ∧ Emulator(y) ∧ People(z) ∧ Provide(x,z,y) ⇒Criminal(x)

Backward Chaining

- Question: Has Reality Man done anything criminal?



- Backward Chaining is a depth-first search: in any knowledge base of realistic size, many search paths will result in failure.

Backward Chaining

- Question: Has Reality Man done anything criminal?
- We will use the same knowledge as in our forward-chaining version of this example:

Programmer(x) ∧ Emulator(y) ∧ People(z) ∧ Provide(x,z,y)⇒ Criminal(x)

Use(friends, x) ∧ Runs(x, N64 games) ⇒ Provide(Reality Man, friends, x)

Software(x) ∧ Runs(x, N64 games) ⇒ Emulator(x)

Programmer(Reality Man)

People(friends)

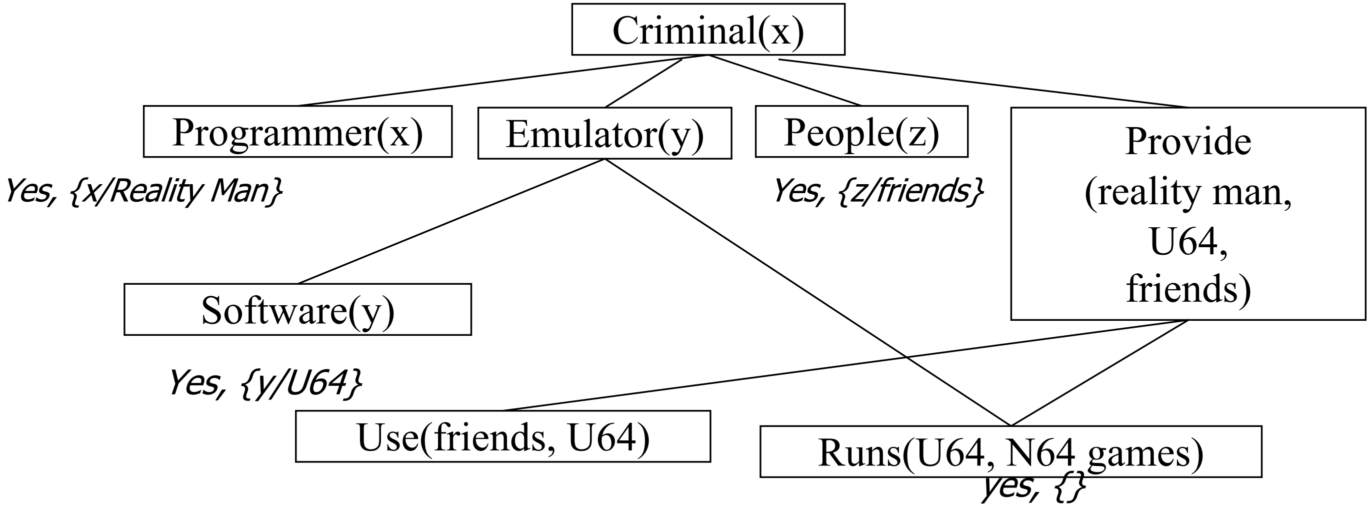
Software(U64)

Use(friends, U64)

Runs(U64, N64 games)

Backward Chaining

- Question: Has Reality Man done anything criminal?



Backward Chaining

- Backward Chaining benefits from the fact that it is directed toward proving one statement or answering one question.
- In a focused, specific knowledge base, this greatly decreases the amount of superfluous work that needs to be done in searches.
- However, in broad knowledge bases with extensive information and numerous implications, many search paths may be irrelevant to the desired conclusion.
- Unlike forward chaining, where all possible inferences are made, a strictly backward chaining system makes inferences only when called upon to answer a query.

(三) completeness ~ 31등

Field Trip – Russell's Paradox (Bertrand Russell, 1901)

- Your life has been simple up to this point, lets see how logical negation and self-referencing can totally ruin our day.
- Russell's paradox is the most famous of the logical or set-theoretical paradoxes. The paradox arises within naive set theory by considering the set of all sets that are not members of themselves. Such a set appears to be a member of itself if and only if it is not a member of itself, hence the paradox.
- Negation and self-reference naturally lead to paradoxes, but are necessary for FOL to be universal.

- Published in *Principles of Mathematics* (1903).

Field Trip – Russell's Paradox

- **Basic example:**
  - Librarians are asked to make catalogs of all the books in their libraries.
  - Some librarians consider the catalog to be a book in the library and list the catalog in itself.
  - The library of congress is asked to make a master catalog of all library catalogs which do **not** include themselves.
  - Should the master catalog in the library of congress include **itself**?
- Keep this tucked in you brain as we talk about logic today. Logical systems can easily tie themselves in knots.
- See also: http://plato.stanford.edu/entries/russell-paradox/
- For additional fun on paradoxes check out "I of Newton" from: *The New Twilight Zone* (1985) http://en.wikipedia.org/wiki/I_of_Newton

2.

Completeness

- (1) As explained earlier, Generalized Modus Ponens requires sentences to be in Horn form:
  - atomic, or
  - an implication with a conjunction of atomic sentences as the antecedent and an atom as the consequent.

- However, some sentences cannot be expressed in Horn form.
  - e.g.: $\forall x \neg$ bored_of_this_lecture (x) (not a definite Horn clause)
  - Cannot be expressed as a definite Horn clause (exactly 1 positive literal) due to presence of negation.

Completeness

- A significant problem since Modus Ponens cannot operate on such a sentence, and thus cannot use it in inference.
- Knowledge exists but cannot be used.
- Thus inference using Modus Ponens is ***incomplete.***
- (2) However, Kurt Gödel in 1930-31 developed the **completeness theorem**, which shows that it is possible to find **complete** inference rules.
- The theorem states:
  - any sentence entailed by a set of sentences can be proven from that set.

=> **Resolution Algorithm** which is a complete inference method.

Completeness

- The completeness theorem says that a sentence can be proved *if* it is entailed by another set of sentences.
- This is a big deal, since arbitrarily deeply nested functions combined with universal quantification make a potentially infinite search space.
- But entailment in first-order logic is only **semi-decidable**, meaning that if a sentence is *not entailed* by another set of sentences, it cannot necessarily be proven.
    - This is to a certain degree an *exotic* situation, but a *very real* one - for instance the *Halting Problem*.
    - Much of the time, in the real world, you can decide if a sentence is not entailed if by no other means than exhaustive elimination.

yes no question，如果这个问题是yes的话，我可以在finite有限的时间，跟你回答yes，但是你如果本来答案是no的话我没办法决定它是no，我没办法肯定它是no。这个是semidecidable。也就是说这个我们刚刚讲虽然

3.

Completeness in FOL

450B.C.	Stoics	propositional logic, inference (maybe)
322B.C.	Aristotle	"syllogisms" (inference rules), quantifiers
1565	Cardano	probability theory (propositional logic + uncertainty)
1847	Boole	propositional logic (again)
1879	Frege	first-order logic
1922	Wittgenstein	proof by truth tables
1930	Gödel	$\exists$ complete algorithm for FOL
1930	Herbrand	complete algorithm for FOL (reduce to propositional)
1931	Gödel	$\neg\exists$ complete algorithm for arithmetic
1960	Davis/Putnam	"practical" algorithm for propositional logic
1965	Robinson	"practical" algorithm for FOL—resolution

Procedure i is complete if and only

$$KB \vdash_i \alpha \quad \text{whenever} \quad KB \models \alpha$$

Forward and backward chaining are complete for Horn KBs
but incomplete for general first-order logic

E.g., from

$$\begin{aligned} PhD(x) &\Rightarrow HighlyQualified(x) \\ \neg PhD(x) &\Rightarrow EarlyEarnings(x) \\ HighlyQualified(x) &\Rightarrow Rich(x) \\ EarlyEarnings(x) &\Rightarrow Rich(x) \end{aligned}$$

should be able to infer $Rich(Me)$, but FC/BC won't do it

Does a complete algorithm exist?

Kinship Example

KB:

(1) father (art, jon)

(2) father (bob, kim)

(3) father (X, Y) ⇒ parent (X, Y)

Goal: parent (art, jon)?

Refutation Proof/Graph

```
¬parent(art, jon)      father(X, Y) => parent(X, Y)
              \           /
               \         /
      ¬ father (art, jon)      father (art,  jon)
                     \       /
                        []
```

反证: 将 $\neg\alpha$ put in with rest of KB
若 exist contradiction 则 $\neg\alpha$ is bad $\Rightarrow$ α good

4. Resolution

Entailment in first-order logic is only semidecidable:

can find a proof of α if $KB \models \alpha$

cannot always prove that $KB \not\models \alpha$

Cf. Halting Problem: proof procedure may be about to terminate with success or failure, or may go on for ever

(1) Resolution is a refutation procedure:

to prove $KB \models \alpha$, show that $KB \wedge \neg\alpha$ is unsatisfiable

Resolution uses KB, $\neg\alpha$ in CNF (conjunction of clauses)

Resolution inference rule combines two clauses to make a new one:

$$\begin{tikzcd} C_1 \arrow[dr] & & C_2 \arrow[dl] \\ & C & \end{tikzcd}$$

Inference continues until an empty clause is derived (contradiction)

12) Resolution Inference Rule

unit resolution 和 mp is identical

Basic propositional version:

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

相反的cancel

or equivalently

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

相同的cancel

Full first-order version:

$$\frac{\begin{array}{c} p_1 \vee \ldots \; p_j \; \ldots \vee p_m, \\ q_1 \vee \ldots \; q_k \; \ldots \vee q_n \end{array}}{(p_1 \vee \ldots \; p_{j-1} \vee p_{j+1} \; \ldots p_m \vee q_1 \ldots \; q_{k-1} \vee q_{k+1} \; \ldots \vee q_n)\sigma}$$

where $p_j\sigma = \neg q_k\sigma$

For example,

$$\frac{\begin{array}{l} \neg Rich(x) \vee Unhappy(x) \\ Rich(Me) \end{array}}{Unhappy(Me)}$$

with $\sigma = \{x/Me\}$

To prove $KB \models \alpha$

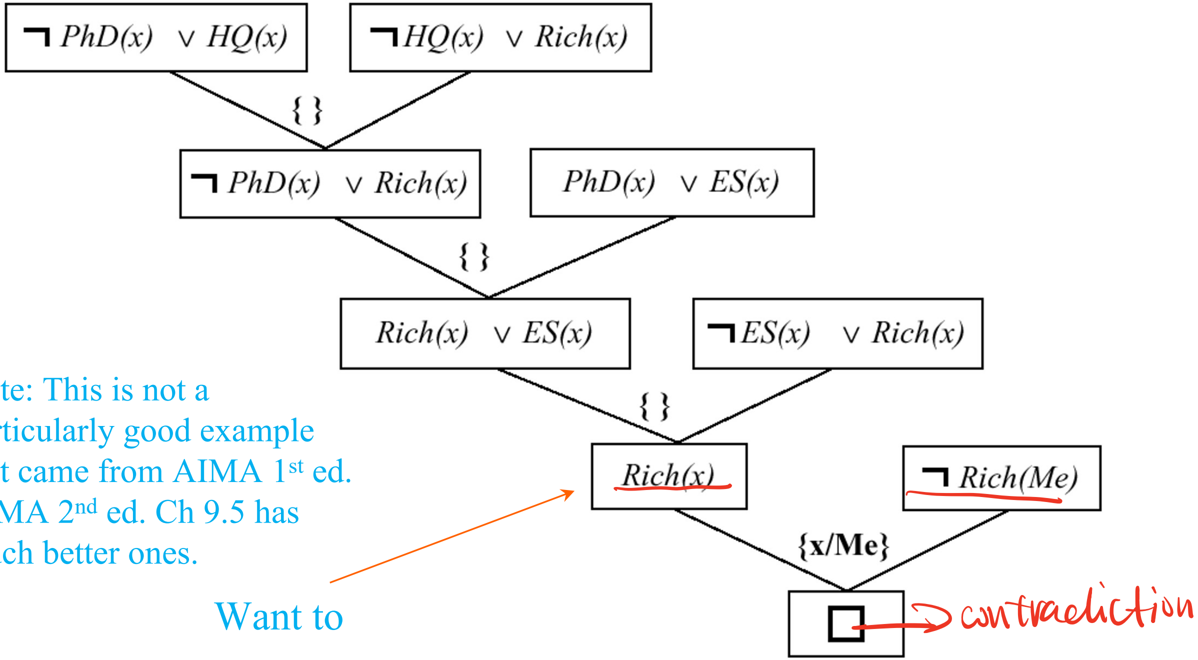
steps

- negate α
- convert to CNF
- add to CNF KB
- infer contradiction

E.g., to prove $Rich(me)$, add $\neg Rich(me)$ to the CNF KB

$$
\begin{array}{l}
\neg PhD(x) \lor HighlyQualified(x) \\
PhD(x) \lor EarlyEarnings(x) \\
\neg HighlyQualified(x) \lor Rich(x) \\
\neg EarlyEarnings(x) \lor Rich(x)
\end{array}
$$

Resolution Proof



Note: This is not a particularly good example that came from AIMA 1st ed. AIMA 2nd ed. Ch 9.5 has much better ones.

Want to prove

5. # Inference in First-Order Logic

- Two Canonical Forms for Resolution

most time

(1) Conjunctive Normal Form (CNF)

$$\neg P(w) \vee Q(w)$$
$$P(x) \vee R(x)$$
$$\neg Q(y) \vee S(y)$$
$$\neg R(z) \vee S(z)$$

(2) Implicative Normal Form (INF)

$$P(w) \Rightarrow Q(w)$$
$$True \Rightarrow P(x) \vee R(x)$$
$$Q(y) \Rightarrow S(y)$$
$$R(z) \Rightarrow S(z)$$

6. **Example of Refutation Proof**
(in conjunctive normal form)

(1)

cat(x) → like(x, fish)

(1) Cats like fish ¬cat (x) ∨ likes (x,fish) cat(y) ∧ like(y, z) → eat(y, z)

(2) Cats eat everything they like ¬cat (y) ∨ ¬likes (y,z) ∨ eats (y,z)

(3) Josephine is a cat. cat (jo)

(4) Prove: Josephine eats fish. eats (jo,fish)

Refutation

Negation of goal wff: $\neg$ eats(jo, fish)

$\neg$ eats(jo, fish) (2) $\neg$ cat(y) $\vee$ $\neg$likes(y, z) $\vee$ eats(y, z)

θ = {y/jo, z/fish}

$\neg$ cat(jo) $\vee$ $\neg$likes(jo, fish) (3) cat(jo)

$\theta = \emptyset$

(1) $\neg$ cat(x) $\vee$ likes(x, fish) $\neg$ likes(jo, fish)

θ = {x/jo}

$\neg$ cat(jo) (3) cat(jo)

$\perp$ (contradiction)

Forward Chaining (Growing the tree from top down)

```
cat (jo)            ¬cat (X) ∨ likes (X,fish)
        \          /
likes (jo,fish)        ¬cat (Y) ∨ ¬likes (Y,Z) ∨ eats (Y,Z)
            \                /
          ¬cat (jo) ∨ eats (jo,fish)       cat (jo)
                               \          /
                              eats (jo,fish)
```

Example Resolution Proof

(2)

Jack owns a dog.
Every dog owner is an animal lover.
No animal lover kills an animal.
Either Jack or Curiosity killed Tuna the cat.
Did Curiosity kill the cat?

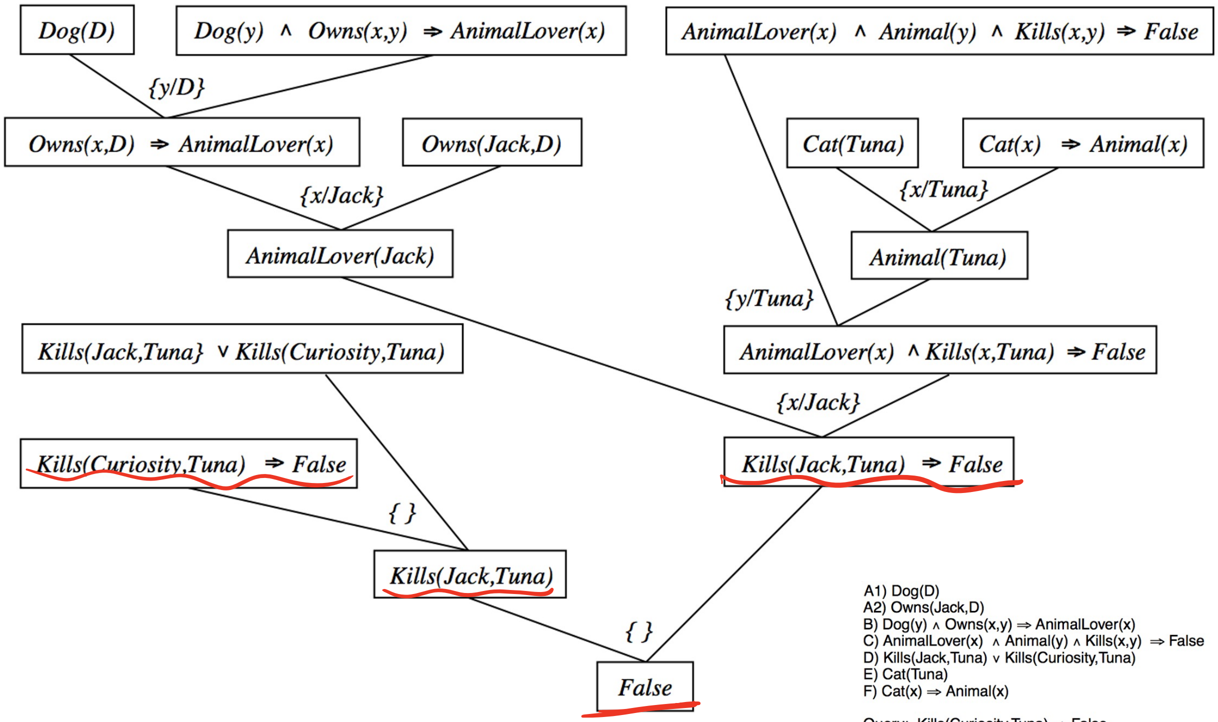
A) $\exists x\ Dog(x) \wedge Owns(Jack,x)$
B) $\forall x\ (\exists y\ Dog(y) \wedge Owns(x,y)) \Rightarrow AnimalLover(x))$
C) $\forall x\ AnimalLover(x) \Rightarrow (\forall y\ Animal(y) \Rightarrow \neg Kills(x,y))$
D) $Kills(Jack,Tuna) \vee Kills(Cursiosity,Tuna)$
E) $Cat(Tuna)$
F) $\forall x(Cat(x) \Rightarrow Animal(x))$

Query: $Kills(Curiosity,Tuna)$

A1) $Dog(D)$
A2) $Owns(Jack,D)$
B) $Dog(y) \wedge Owns(x,y) \Rightarrow AnimalLover(x)$
C) $AnimalLover(x) \wedge Animal(y) \wedge Kills(x,y) \Rightarrow False$
D) $Kills(Jack,Tuna) \vee Kills(Curiosity,Tuna)$
E) $Cat(Tuna)$
F) $Cat(x) \Rightarrow Animal(x)$

Query: $Kills(Curiosity,Tuna) \Rightarrow False$

Example Resolution Proof



A1) Dog(D)
A2) Owns(Jack,D)
B) Dog(y) ∧ Owns(x,y) ⇒ AnimalLover(x)
C) AnimalLover(x) ∧ Animal(y) ∧ Kills(x,y) ⇒ False
D) Kills(Jack,Tuna) ∨ Kills(Curiosity,Tuna)
E) Cat(Tuna)
F) Cat(x) ⇒ Animal(x)

Query: Kills(Curiosity,Tuna) ⇒ False

http://www.cs.utsa.edu/~bylander/cs5233/notes/logichandout.pdf

Another Example Resolution Proof

- Knowledge Base

```
-parent(x,y) | -ancestor(y,z) | ancestor(x,z)
-parent(x,y) | ancestor(x,y)
-mother(x,y) | parent(x,y)
-father(x,y) | parent(x,y)
mother(Liz,Charley)
father(Charley,Billy)
```

- To prove `ancestor(Liz,Billy)`
  Refute `-ancestor(Liz,Billy)`

```
-parent(x,y) | -ancestor(y,z) | ancestor(x,z)
-ancestor(Liz,Billy)
----------------------------------------------
-parent(Liz,y) | -ancestor(y,Billy)

-mother(x,y) | parent(x,y)
-parent(Liz,y) | -ancestor(y,Billy)
----------------------------------------------
-mother(Liz,y) | -ancestor(y,Billy)

mother(Liz,Charley)
-mother(Liz,y) | -ancestor(y,Billy)
----------------------------------------------
-ancestor(Charley,Billy)

-parent(x,y) | ancestor(x,y)
-ancestor(Charley,Billy)
----------------------------------------------
-parent(Charley,Billy)

-father(x,y) | parent(x,y)
-parent(Charley,Billy)
----------------------------------------------
-father(Charley,Billy)

father(Charley,Billy)
-father(Charley,Billy)
---------------------------------------------- contradiction
```

contradiction

Remember: Three Normal Forms

Other approaches to inference use syntactic operations on sentences, often expressed in standardized forms

Conjunctive Normal Form (CNF—universal)

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

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

"product of sums of simple variables or negated simple variables"

Disjunctive Normal Form (DNF—universal)

$\textit{disjunction}$ of $\underbrace{\textit{conjunctions} \text{ of } \textit{literals}}_{\textit{terms}}$

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

"sum of products of simple variables or negated simple variables"

Horn Form (restricted)

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 $(C \wedge D) \Rightarrow B$

present clause in CNF

Conjunctive Normal Form - (how-to is coming in discussion...)

Literal = (possibly negated) atomic sentence, e.g., $\neg Rich(Me)$

Clause = disjunction of literals, e.g., $\neg Rich(Me) \vee Unhappy(Me)$

The KB is a conjunction of clauses

Any FOL KB can be converted to CNF as follows:

1. Replace $P \Rightarrow Q$ by $\neg P \vee Q$
2. Move $\neg$ inwards, e.g., $\neg \forall x\, P$ becomes $\exists x\, \neg P$
3. Standardize variables apart, 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., $\forall x\, P \vee \exists x\, Q$ becomes $\forall x \exists y\, P \vee Q$
5. Eliminate $\exists$ by Skolemization (next slide)
6. Drop universal quantifiers
7. Distribute $\wedge$ over $\vee$

Skolemization

$\exists x\, Rich(x)$ becomes $Rich(G1)$ where $G1$ is a new "Skolem constant"

$\exists\, k\ \ \frac{d}{dy}(k^y) = k^y$ becomes $\frac{d}{dy}(e^y) = e^y$

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:

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

Correct:

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

where H is a new symbol ("Skolem function")

Skolem function arguments: all enclosing universally quantified variables

If x has a y, then we can infer that y exists. However, its existence is contingent on x, thus y is a function of x as $H(x)$.

Examples: Converting FOL sentences to clause form…

Convert the sentence

1. $(\forall x)(P(x) \Rightarrow ((\forall y)(P(y) \Rightarrow P(f(x,y))) \wedge \neg(\forall y)(Q(x,y) \Rightarrow P(y))))$

```
(like A => B ^ C)
```

2. Eliminate =>

$(\forall x)(\neg P(x) \vee ((\forall y)(\neg P(y) \vee P(f(x,y))) \wedge \neg(\forall y)(\neg Q(x,y) \vee P(y))))$

3. Reduce scope of negation

$(\forall x)(\neg P(x) \vee ((\forall y)(\neg P(y) \vee P(f(x,y))) \wedge \boldsymbol{(\exists y)(Q(x,y) \wedge \neg P(y))}))$

4. Standardize variables

$(\forall x)(\neg P(x) \vee ((\forall y)(\neg P(y) \vee P(f(x,y))) \wedge \boldsymbol{(\exists z)(Q(x,z) \wedge \neg P(z))}))$

Examples: Converting FOL sentences to clause form…

$(\forall x)(\neg P(x) \vee ((\forall y)(\neg P(y) \vee P(f(x,y))) \wedge (\exists z)(Q(x,z) \wedge \neg P(z))))$ …

5. Eliminate existential quantification

   $(\forall x)(\neg P(x) \vee ((\forall y)(\neg P(y) \vee P(f(x,y))) \wedge (Q(x,\boxed{g(x)}) \wedge \neg P(\boxed{g(x)}))))$

6. Drop universal quantification symbols

   $(\neg P(x) \vee ((\neg P(y) \vee P(f(x,y))) \wedge (Q(x,g(x)) \wedge \neg P(g(x)))))$

7. Convert to conjunction of disjunctions

   $(\neg P(x) \vee \neg P(y) \vee P(f(x,y))) \wedge (\neg P(x) \vee Q(x,g(x))) \wedge (\neg P(x) \vee \neg P(g(x)))$

Examples: Converting FOL sentences to clause form…

$(\neg P(x) \vee \neg P(y) \vee P(f(x,y))) \wedge (\neg P(x) \vee Q(x,g(x))) \wedge (\neg P(x) \vee \neg P(g(x)))$ …

8. Create separate clauses

   $\neg P(x) \vee \neg P(y) \vee P(f(x,y))$

   $\neg P(x) \vee Q(x,g(x))$

   $\neg P(x) \vee \neg P(g(x))$

9. Standardize variables

   $\neg P(x) \vee \neg P(y) \vee P(f(x,y))$

   $\neg P(z) \vee Q(z,g(z))$

   $\neg P(w) \vee \neg P(g(w))$