Inference in first-order logic

Propositional vs. FOL Inference

- First-order inference can be done by converting KB to propositional logic and using propositional inference

- Specifically, what to do with quantifiers?
  - Substitution: {*variable*/*Object*}
    - – Remove quantifier by substituting *variable* with specific object

Reduction to Propositional Inference

- Universal Quantifiers (∀)
  - Sentence must be true *for all* objects in the world (all values of variable)
  - So substituting any object must be valid (Universal Instantiation, UI)
- Example
  - 1) $\forall x \; Person(x) \rightarrow Likes(x, IceCream)$
  - Substituting: (1), $\{x/Jack\}$
  - 2) $Person(Jack) \rightarrow Likes(Jack, IceCream)$

Reduction to Propositional Inference

- Existential Quantifiers (∃)
  - Sentence must be true *for some* object in the world (or objects)
  - Assume we know this object and give it an arbitrary (unique!) name (Existential Instantiation, EI)
  - Known as Skolem constant (SK1, SK2, …)
- Example
  - 1) $\exists x \; Person(x) \wedge Likes(x, IceCream)$
  - Substituting: (1), $\{x/SK1\}$
  - 2) $Person(SK1) \wedge Likes(SK1, IceCream)$
- We don't know who "SK1" is (and usually can't), but we know they must exist

Reduction to Propositional Inference

- Multiple Quantifiers
  - No problem if same type ($\forall x,y$ or $\exists x,y$)

Reduction to Propositional Inference

- Multiple Quantifiers
  - No problem if same type ($\forall x,y$ or $\exists x,y$)
  - $\exists x\ \forall y$
    - There must be some x for which the sentence is true with every possible y
    - Skolem constant still works (for x)

Reduction to Propositional Inference

- Multiple Quantifiers
  - No problem if same type ($\forall x,y$ or $\exists x,y$)
  - $\exists x \; \forall y$
    - There must be some x for which the sentence is true with every possible y
    - Skolem constant still works (for x)
  - $\forall x \; \exists y$
    - For every possible x, there must be some y that satisfies the sentence
    - Could be different y value to satisfy for each x
    - The value we substitute for y <u>must depend on</u> x
    - Use a Skolem <u>function</u> instead

Reduction to Propositional Inference

- $\forall x\ \exists y$ *Skolem Substitution* Example

  1) $\forall x\ \exists y\ Person(x) \rightarrow Loves(x,y)$

  2) $\forall x\ Person(x) \rightarrow Loves(x,SK1(x))$ [Substitute, $\{y/SK1(x)\}$]

  3) $Person(Jack) \rightarrow Loves(Jack,SK1(Jack))$ [Then, $\{x/Jack\}$]

- SK1(x) is effectively a function which returns a person that x loves.
- But, again, we can't generally know the specific value it returns.

Reduction to Propositional Inference

- Internal Quantifiers
  - Previous rules only work if quantifiers are external (left-most)
  - Consider: "For all *x*, if there is some *y* that *x* loves, then *x* must be a person"
  - $\forall x\ (\exists y\ Loves(x,y)) \rightarrow Person(x)$
  - A Skolem function limits the values *y* could take (to one)
    - and we can't know what it is.

Reduction to Propositional Inference

- Internal Quantifiers
  - Need to move the quantifier outward
    - $\forall x\ (\exists y\ Loves(x,y)) \rightarrow Person(x)$
    - $\forall x\ \neg(\exists y\ Loves(x,y)) \vee Person(x)$ [convert to $\neg, \vee, \wedge$]
    - $\forall x\ \forall y\ \neg Loves(x,y) \vee Person(x)$ [move $\neg$ inward]
    - $\forall x\ \forall y\ Loves(x,y) \rightarrow Person(x)$
  - Now we can see that we can actually substitute *anything* for *y*
  - May need to rename variables before moving quantifier left

Reduction to Propositional Inference

- Once have non-quantified sentences it is possible to reduce first-order inference to propositional inference
- Suppose KB contains:

$$\forall x \ \ King(x) \land Greedy(x) \rightarrow Evil(x)$$

$$King(John)$$

$$Greedy(John)$$

$$Brother(Richard, John)$$

- We get

$$King(John) \land Greedy(John) \rightarrow Evil(John)$$

Reduction to Propositional Inference

- Now the KB is essentially propositional:

  King(John) ∧ Greedy(John) → Evil(John)

  King(John)

  Greedy(John)

  Brother(Richard, John)

- Then can use propositional inference algorithms to obtain conclusions
  - Modus Ponens yields Evil(John)

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

$$\frac{King(John) \wedge Greedy(John), \quad King(John) \wedge Greedy(John) \rightarrow Evil(John)}{Evil(John)}$$

Forward and Backward Chaining

Forward and Backward Chaining

- Generalized Modus Ponens can be used in two ways:

  #1) Start with sentences in KB and generate new conclusions (forward chaining)

  - "Used when a new fact is added to database and want to generate its consequences"

  or

  #2) Start with something want to prove, find implication sentences that allow to conclude it, then attempt to establish their premises in turn (backward chaining)

  - "Used when there is a goal to be proved"

Forward Chaining

- Forward chaining normally triggered by addition of new fact to KB (using TELL)
- When new fact p added to 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
  - Premise: Left-hand side of implication
    - Or, each term of conjunction on left hand side
  - Conclusion: Right-hand side of implication
- Forward chaining uses unification
  - Make two sentences (fact + premise) match by substituting variables (if possible)
- Forward chaining is data-driven
  - Inferring properties and categories from percepts

Example

Knowledge Base
$A \rightarrow B$
$A \rightarrow D$
$D \rightarrow C$
$A \rightarrow E$
$D \rightarrow F$
$E \rightarrow G$

Add A:

A, $A \rightarrow B$ gives B [done]

A, $A \rightarrow D$ gives D

  D, $D \rightarrow C$ gives C [done]

  D, $D \rightarrow F$ gives F [done]

A, $A \rightarrow E$ gives E

  E, $E \rightarrow G$ gives G [done]

[done]

Order of generation B, D, C, F, E, G

Backward Chaining

- Backward chaining designed to find all answers to a question posed to KB (using ASK)
- 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
- Added complications
  - Keeping track of unifiers, avoiding infinite loops
- Two versions
  - Find any solution
  - Find all solutions
- Backward chaining is basis of logic programming
  - Prolog

Backward Chaining Example

Given facts/rules 1-5 in KB:

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

Prove: *Faster(Pat, Steve)*

Backward Chaining Example

Given facts/rules 1-5 in KB:

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)

Prove: *Faster(Pat, Steve)*

$\boxed{Faster(Pat, Steve)}$

Start with what we want to prove.

Backward Chaining Example

Given facts/rules 1-5 in KB:

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)

Prove: *Faster(Pat, Steve)*

Faster(Pat, Steve)

1 {y/Pat, z/Steve}

Found Rule #1, which can prove that something is faster than something else.

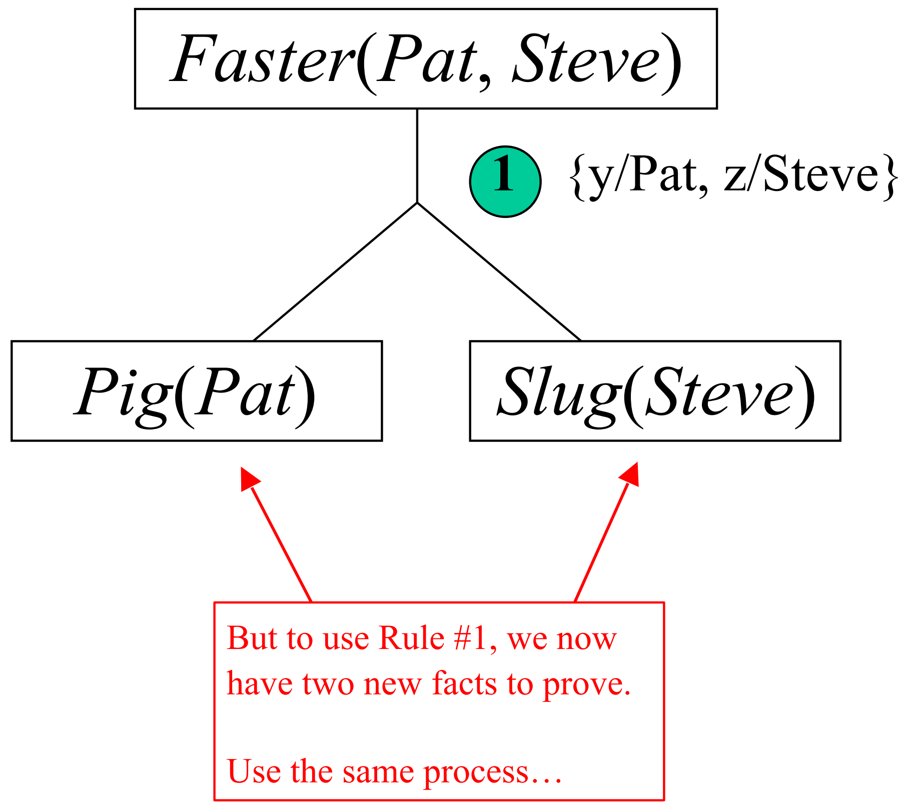
But for it to prove what we need, we have to substitute "Pat" for *y* and "Steve" for *z*.

Backward Chaining Example

Given facts/rules 1-5 in KB:

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

Prove: *Faster(Pat, Steve)*

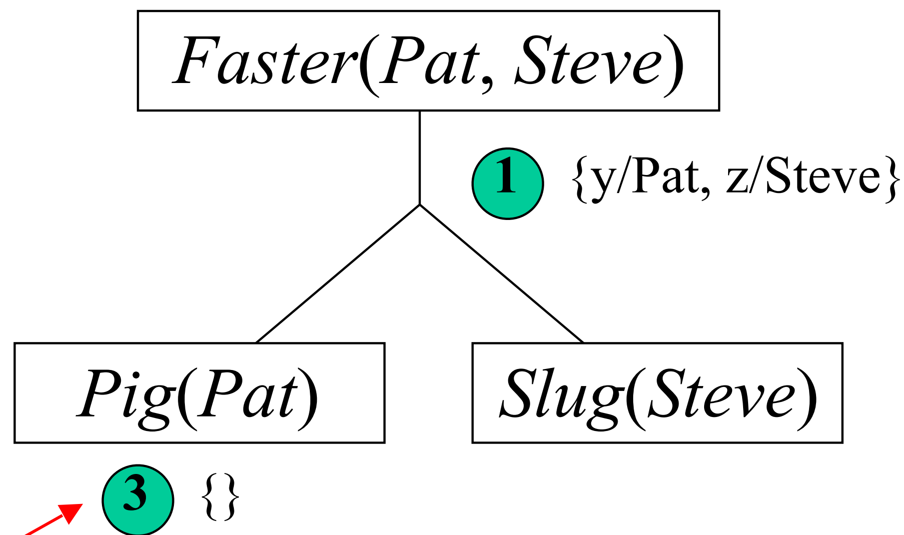


Backward Chaining Example

Given facts/rules 1-5 in KB:

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

Prove: *Faster(Pat, Steve)*



This fact we already know is true from #3 in our knowledge-base.

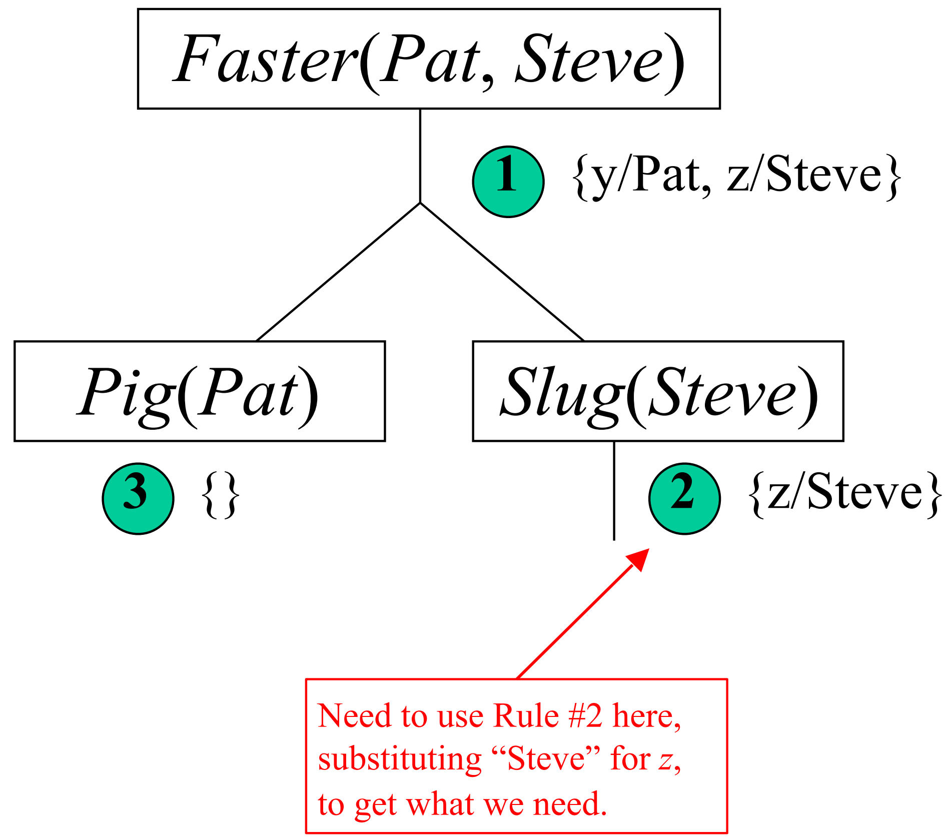
(And no substitution needed, so empty.)

Backward Chaining Example

Given facts/rules 1-5 in KB:

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

Prove: *Faster(Pat, Steve)*

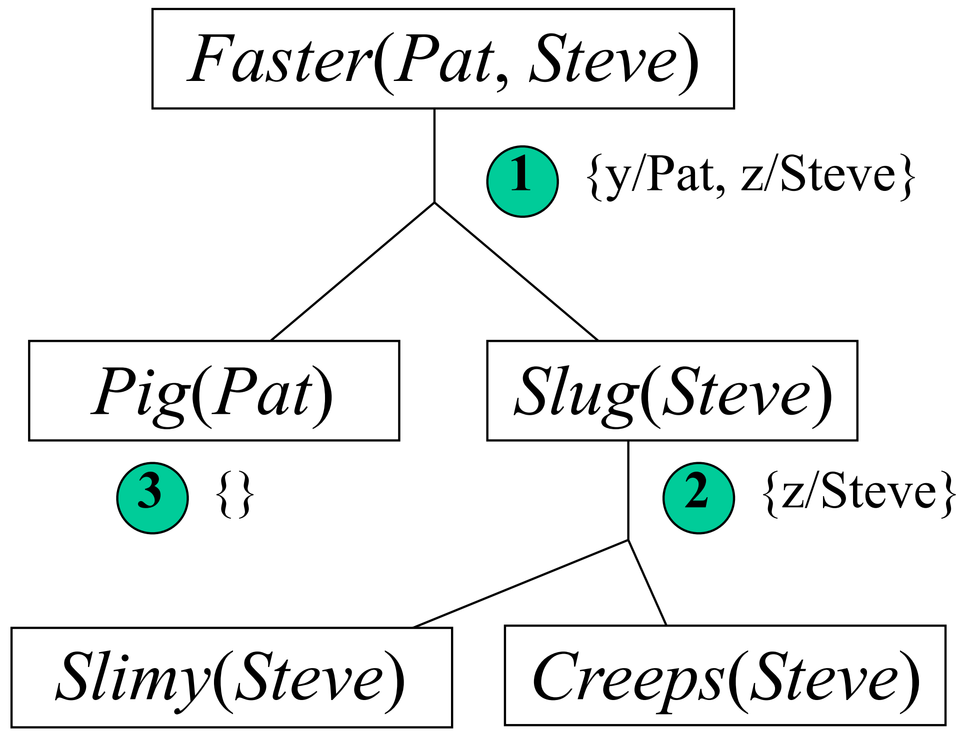


Backward Chaining Example

Given facts/rules 1-5 in KB:

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

Prove: *Faster(Pat, Steve)*



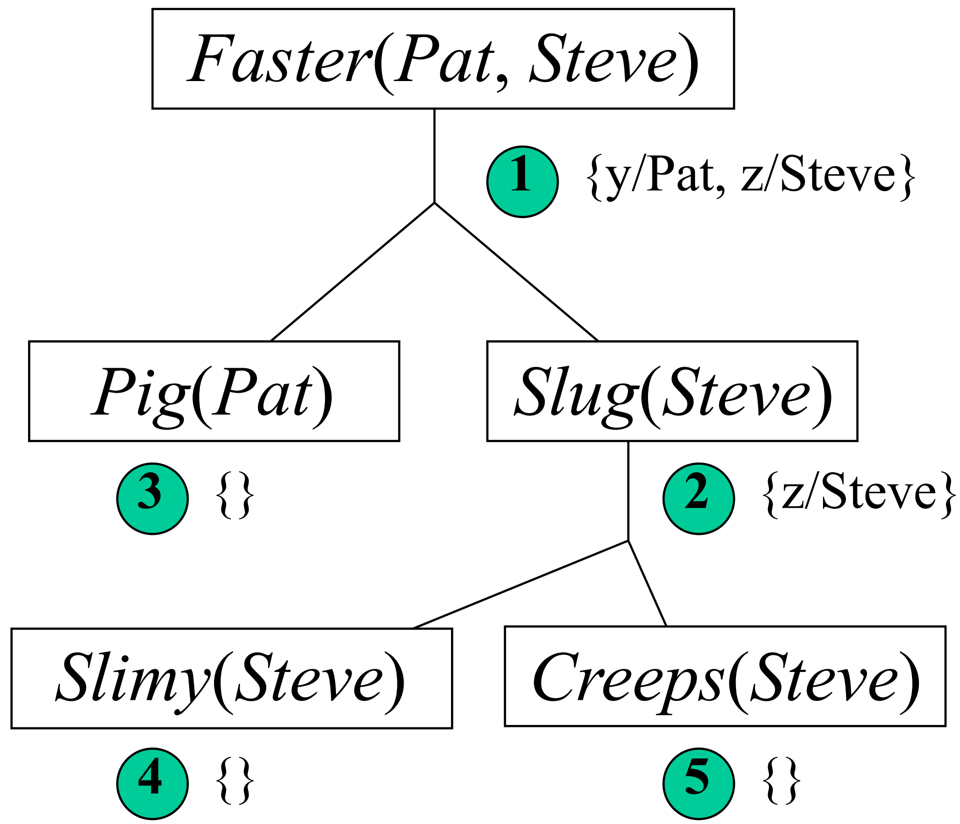
And Rule #2 requires these two facts…

Backward Chaining Example

Given facts/rules 1-5 in KB:

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

Prove: *Faster(Pat, Steve)*



Which we know are true directly from our knowledge-base.

Resolution

Resolution

- Uses proof by contradiction
  - Referred to by other names
    - Refutation
    - Reductio ad absurdum
- Inference procedure using resolution
  - To prove *P*:
    - Assume *P* is FALSE
    - Add $\neg P$ to KB
    - Prove a contradiction
  - Given that the KB is known to be True, we can believe that the negated goal is in fact False, meaning that the original goal must be True

Simple Example

- Given: "All birds fly", "Peter is a bird"
- Prove: "Peter does not fly"

- Step #1: have in FOL

$$\forall x \quad Bird(x) \rightarrow Flies(x)$$
$$Bird(Peter)$$

- Step #2: put in normal form

$$\neg Bird(x) \vee Flies(x)$$
$$Bird(Peter)$$

Simple Example (con't)

- Step #3: Assume contradiction of goal

  GOAL TO TEST: $\neg Flies(Peter)$

KB:

$\neg Bird(x) \vee Flies(x)$

$Bird(Peter)$

Simple Example (con't)

- Step #3: Assume contradiction of goal

  GOAL TO TEST: $\neg Flies(Peter)$

- Step #4: Unification $\{x/Peter\}$

  $$\neg Bird(Peter) \vee Flies(Peter)$$

KB:

$\neg Bird(x) \vee Flies(x)$

$Bird(Peter)$

Simple Example (con't)

- Step #3: Assume contradiction of goal

  GOAL TO TEST: $\neg Flies(Peter)$

- Step #4: Unification $\{x/Peter\}$

  $$\neg Bird(Peter) \vee Flies(Peter)$$

- Step #5: Resolution (unit)

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

KB:

$\neg Bird(x) \vee Flies(x)$

$Bird(Peter)$

Simple Example (con't)

KB:

$\neg Bird(x) \vee Flies(x)$

$Bird(Peter)$

- Step #3: Assume contradiction of goal

  GOAL TO TEST: $\neg Flies(Peter)$
- Step #4: Unification $\{x/Peter\}$

  $$\neg Bird(Peter) \vee Flies(Peter)$$
- Step #5: Resolution (unit)

$$\frac{\alpha,\ \neg\alpha \vee \beta}{\beta} \qquad \frac{\neg Flies(Peter),\ Flies(Peter) \vee \neg Bird(Peter)}{\neg Bird(Peter)}$$

Simple Example (con't)

KB:

$\neg Bird(x) \vee Flies(x)$

$Bird(Peter)$

- Step #3: Assume contradiction of goal

  GOAL TO TEST: $\neg Flies(Peter)$

- Step #4: Unification $\{x/Peter\}$

  $$\neg Bird(Peter) \vee Flies(Peter)$$

- Step #5: Resolution (unit)

$$\frac{\alpha,\ \neg\alpha \vee \beta}{\beta} \qquad \frac{\neg Flies(Peter),\ Flies(Peter) \vee \neg Bird(Peter)}{\neg Bird(Peter)}$$

- Step #6: Contradiction
  - The result of Step #5 says that "Peter is not a bird", but this is in contrast to KB containing *Bird*(*Peter*)
  - Therefore, we can conclude that "Peter does indeed fly"

Another Example

KB:

kb-1: $A(x,\text{bar}) \vee B(x) \vee C(x)$

kb-2: $D(y,\text{foo}) \vee \neg B(y)$

kb-3: $E(z) \vee \neg A(z,\text{bar})$

kb-4: $\neg D(\text{Minsky},\text{foo})$

kb-5: $\neg A(\text{Minsky},\text{bar})$

Goal: prove C(Minsky)

Another Example

0: ¬C(Minsky)

KB:

kb-1: A(x,bar) ∨ B(x) ∨ C(x)

kb-2: D(y,foo) ∨ ¬B(y)

kb-3: E(z) ∨ ¬A(z,bar)

kb-4: ¬D(Minsky,foo)

kb-5: ¬A(Minsky,bar)

Goal: prove C(Minsky)

Start off using our negated goal (proof by contradiction)

Another Example

KB:

kb-1: A(x,bar) ∨ B(x) ∨ C(x)

kb-2: D(y,foo) ∨ ¬B(y)

kb-3: E(z) ∨ ¬A(z,bar)

kb-4: ¬D(Minsky,foo)

kb-5: ¬A(Minsky,bar)

Goal: prove C(Minsky)

0: ¬C(Minsky)

1: A(Minsky,bar) ∨ B(Minsky) ∨ C(Minsky) *[kb-1] {x/Minsky}*

Look for a rule that has C(Minsky) *to oppose* ¬C(Minsky) *from #0.*
This rule (kb-1) needed a substitution for it to work, giving us the new sentence #1.

Another Example

KB:

kb-1: $A(x,\text{bar}) \vee B(x) \vee C(x)$

kb-2: $D(y,\text{foo}) \vee \neg B(y)$

kb-3: $E(z) \vee \neg A(z,\text{bar})$

kb-4: $\neg D(\text{Minsky},\text{foo})$

kb-5: $\neg A(\text{Minsky},\text{bar})$

Goal: prove C(Minsky)

0: ¬C(Minsky)

1: A(Minsky,bar) ∨ B(Minsky) ∨ C(Minsky) *[kb-1]{x/Minsky}*

2: ¬C(Minsky), A(Minsky,bar) ∨ B(Minsky) ∨ C(Minsky)

2.a: A(Minsky,bar) ∨ B(Minsky) *[resolution: 0,1]*

Now that we have #0 and #1 with opposing terms, use resolution to eliminate them.

Another Example

KB:

kb-1: $A(x,\text{bar}) \vee B(x) \vee C(x)$

kb-2: $D(y,\text{foo}) \vee \neg B(y)$

kb-3: $E(z) \vee \neg A(z,\text{bar})$

kb-4: ¬D(Minsky,foo)

kb-5: ¬A(Minsky,bar)

Goal: prove C(Minsky)

0: ¬C(Minsky)

1: A(Minsky,bar) ∨ B(Minsky) ∨ C(Minsky) *[kb-1]{x/Minsky}*

2: ¬C(Minsky), A(Minsky,bar) ∨ B(Minsky) ∨ C(Minsky)

2.a: A(Minsky,bar) ∨ B(Minsky) *[resolution: 0,1]*

3: D(Minsky,foo) ∨ ¬B(Minsky) *[kb-2]*
{y/Minsky}

4: A(Minsky,bar) ∨ B(Minsky), D(Minsky,foo) ∨ ¬B(Minsky)

4.a: A(Minsky,bar) ∨ D(Minsky,foo) *[resol: 2a,3]*

And repeat to find and eliminate other opposing terms.

Another Example

KB:

kb-1: A(x,bar) ∨ B(x) ∨ C(x)

kb-2: D(y,foo) ∨ ¬B(y)

kb-3: E(z) ∨ ¬A(z,bar)

kb-4: ¬D(Minsky,foo)

kb-5: ¬A(Minsky,bar)

Goal: prove C(Minsky)

0: ¬C(Minsky)

1: A(Minsky,bar) ∨ B(Minsky) ∨ C(Minsky) *[kb-1] {x/Minsky}*

2: ¬C(Minsky), A(Minsky,bar) ∨ B(Minsky) ∨ C(Minsky)

2.a: A(Minsky,bar) ∨ B(Minsky) *[resolution: 0,1]*

3: D(Minsky,foo) ∨ ¬B(Minsky) *[kb-2]*
{y/Minsky}

4: A(Minsky,bar) ∨ B(Minsky), D(Minsky,foo) ∨ ¬B(Minsky)

4.a: A(Minsky,bar) ∨ D(Minsky,foo) *[resol: 2a,3]*

5: ¬A(Minsky,bar), A(Minsky,bar) ∨ D(Minsky,foo)

5.a: D(Minsky,foo) *[resol: 4a,kb-5]*

And again…

Another Example

KB:

kb-1: $A(x,\text{bar}) \lor B(x) \lor C(x)$

kb-2: $D(y,\text{foo}) \lor \neg B(y)$

kb-3: $E(z) \lor \neg A(z,\text{bar})$

kb-4: $\neg D(\text{Minsky},\text{foo})$

kb-5: $\neg A(\text{Minsky},\text{bar})$

Goal: prove C(Minsky)

0: ¬C(Minsky)

1: A(Minsky,bar) ∨ B(Minsky) ∨ C(Minsky) *[kb-1]{x/Minsky}*

2: ¬C(Minsky), A(Minsky,bar) ∨ B(Minsky) ∨ C(Minsky)

2.a: A(Minsky,bar) ∨ B(Minsky) *[resolution: 0,1]*

3: D(Minsky,foo) ∨ ¬B(Minsky) *[kb-2]* *{y/Minsky}*

4: A(Minsky,bar) ∨ B(Minsky), D(Minsky,foo) ∨ ¬B(Minsky)

4.a: A(Minsky,bar) ∨ D(Minsky,foo) *[resol: 2a,3]*

5: ¬A(Minsky,bar), A(Minsky,bar) ∨ D(Minsky,foo)

5.a: D(Minsky,foo) *[resol: 4a,kb-5]*

6: D(Minsky,foo) ∧ ¬D(Minsky,foo)

FALSE, CONTRADICTION!!!

must be C(Minsky)

Summary

- Reduction of first-order inference to propositional inference
  - Universal and Existential Instantiation
- Forward chaining
  - Infer properties in data-driven manner
- Backward chaining
  - Proving query of a consequent by proving premises
- Resolution using proof by contradiction