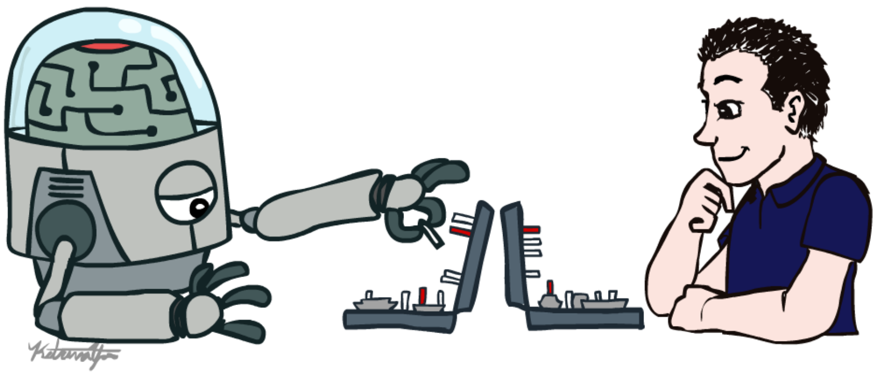
CSE 3521: Introduction to Artificial Intelligence



[Many slides are adapted from the UC Berkeley. CS188 Intro to AI at UC Berkeley and previous CSE 3521 course at OSU.]

THE OHIO STATE UNIVERSITY

FOPC

- First-order logic
    - Increased expressive power over Propositional Logic
    - Objects and relations are semantic primitives
    - Syntax: constants, functions, predicates, equality, quantifiers
        - Two standard quantifiers
            - Universal $\forall$
            - Existential $\exists$

Universal Quantifiers

- $\forall x \ \forall y$ is same as $\forall y \ \forall x \ (\ \forall x,y\ )$
- $\exists\, x \ \exists\, y$ is same as $\exists\, y \ \exists\, x \ (\ \exists\, x,y\ )$
- $\exists\, x \ \forall y$ is <u>not same</u> as $\forall y \ \exists\, x$
  - $\exists\, y\ Person(y) \wedge (\forall x\ Person(x) \Rightarrow Loves(x,y))$
    - "There is someone who is loved by everyone"
  - $\forall x\ Person(x) \Rightarrow \exists y\ Person(y) \wedge Loves(x,y)$
    - "Everybody loves somebody"
      (not guaranteed to be the same person)

How to do inference in FOPC

- Reduction of first-order inference to propositional inference
- First-order inference algorithms
  - Generalized Modus Ponens
  - Forward chaining ***
  - Backward chaining ***
  - Resolution-based theorem proving ***

Topics

- Reduction of first-order inference to propositional inference
- First-order inference algorithms
  - Generalized Modus Ponens
  - Forward chaining ***
  - Backward chaining ***
  - Resolution-based theorem proving ***

Propositional vs. FOL Inference

- First-order inference can be done by converting KB to propositional logic and using propositional inference
  - Using modus ponens, etc.
- Specifically, what to do with quantifiers?
- Substitution: {*variable*/*Object*}
  - Remove quantifier by substituting *variable* with specific object

Think about C or Python → assembly language!

Reduction to Propositional Inference

- Universal Quantifiers ($\forall$)
  - Recall: 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
  - $\forall x \;\; Person(x) \Rightarrow Likes(x, IceCream)$
    - Substituting: (1), $\{x/Jack\}$
  - $Person(Jack) \Rightarrow Likes(Jack, IceCream)$

Reduction to Propositional Inference (con't)

- Existential Quantifiers ($\exists$)
  - Recall: 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 <u>Skolem constant</u> (SK1, SK2, …)
- Example
  - $\exists x \;\; Person(x) \wedge Likes(x, IceCream)$
    - Substituting: (1), $\{x/SK1\}$
  - $Person(SK1) \wedge Likes(SK1, IceCream)$
- We don't know who "SK1" is (and usually can't), but we know they must exist

Reduction to Propositional Inference (con't)

- Multiple Quantifiers
  - No problem if same type ($\forall x,y$ or $\exists x,y$)
  - Also no problem if: $\exists x\ \forall y$
    - There must be some *x* for which the sentence is true with every possible *y*
    - Skolem constant still works (for *x*)
- Problem with $\forall x\ \exists y$
  - For every possible *x*, there must be some *y* that satisfies the sentence
  - Could be different *y* value to satisfy for each *x*!

Reduction to Propositional Inference (con't)

- Problem with $\forall x\ \exists y$ (con't)
  - The value we substitute for *y* <u>must depend on</u> *x*
  - Use a Skolem <u>function</u> instead
- Example
  - $\forall x\ \exists y\ Person(x) \Rightarrow Loves(x,y)$
    - Substitute: (1), $\{y/SK1(x)\}$
  - $\forall x\ Person(x) \Rightarrow Loves(x,SK1(x))$
    - Then: (2), $\{x/Jack\}$
  - $Person(Jack) \Rightarrow Loves(Jack,SK1(Jack))$
- *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 (con't)

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

- Once have non-quantified sentences (from quantified sentences using UI, EI), possible to reduce first-order inference to propositional inference
- Suppose KB contains:

$$\forall x \;\; King(x) \wedge Greedy(x) \Rightarrow Evil(x)$$

$$King(John)$$

$$Greedy(John)$$

$$Brother(Richard, John)$$

- Using UI with $\{x/John\}$ and $\{x/Richard\}$, we get

$$King(John) \wedge Greedy(John) \Rightarrow Evil(John)$$

$$King(Richard) \wedge Greedy(Richard) \Rightarrow Evil(Richard)$$

Reduction to Propositional Inference (con't)

- Now the KB is essentially propositional:

$$King(John) \wedge Greedy(John) \Rightarrow Evil(John)$$

$$King(Richard) \wedge Greedy(Richard) \Rightarrow Evil(Richard)$$

$$King(John)$$

$$Greedy(John)$$

$$Brother(Richard, John)$$

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

$$\frac{\alpha, \quad \alpha \rightarrow \beta}{\beta} \qquad \frac{King(John) \wedge Greedy(John), \ King(John) \wedge Greedy(John) \Rightarrow Evil(John)}{Evil(John)}$$

Topics

- Reduction of first-order inference to propositional inference
- First-order inference algorithms
  - Generalized Modus Ponens
  - Forward chaining ***
  - Backward chaining ***
  - Resolution-based theorem proving ***

Forward and Backward Chaining

- Have language representing knowledge (FOL) and inference rules (Generalized Modus Ponens)
  - Now study how a reasoning program is constructed
- Generalized Modus Ponens can be used in two ways:
  - 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
  - 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

Forward Chaining Example

- Add sentences gradually

1. $\forall x,y\ Buffalo(x) \wedge Pig(y) \Rightarrow Faster(x, y)$
2. $\forall y,z\ Pig(y) \wedge Slug(z) \Rightarrow Faster(y, z)$
3. $\forall x,y,z\ Faster(x, y) \wedge Faster(y, z) \Rightarrow Faster(x, z)$

Now we need to find rule(s) that can use this fact…

Forward Chaining Example

- Add sentences gradually

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. $Buffalo(Bob)$

Note: $\forall x,y,z$ dropped

Add new facts one at a time

Now we need to find rule(s) that can use this fact…

Forward Chaining Example

- Add facts 1, 2, 3, 4, 5, 7 in turn
  - Number in [] is unification literal

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. $Buffalo(Bob)$ [1]

Check each rule in turn…

Rule 1 can make use of the fact that something (x) is a Buffalo

Forward Chaining Example

- Add facts 1, 2, 3, 4, 5, 7 in turn
  - Number in [] is unification literal; √ rule firing

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. $Buffalo(Bob)$ [1]

And it's the first (a) term that our fact (#4) can satisfy.

Which is important because...

Forward Chaining Example

- Add facts 1, 2, 3, 4, 5, 7 in turn
  - Number in [] is unification literal; √ rule firing

1. *Buffalo*(*x*) ∧ *Pig*(*y*) ⇒ *Faster*(*x*, *y*)
2. *Pig*(*y*) ∧ *Slug*(*z*) ⇒ *Faster*(*y*, *z*)
3. *Faster*(*x*, *y*) ∧ *Faster*(*y*, *z*) ⇒ *Faster*(*x*, *z*)
4. *Buffalo*(*Bob*) [1]

…we need to check to see if the rule can be satisfied and fired.

BUT we are missing a fact to fill in the second (b) part of the rule, so NO, we fail to fire the rule.

Forward Chaining Example

- Add facts 1, 2, 3, 4, 5, 7 in turn
  - Number in [] is unification literal; √ rule firing

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. $Buffalo(Bob)$ [1]
5. $Pig(Pat)$ [1]

From (#4) we also can satisfy (1), so we can fire the rule!

6. $Faster(Bob, Pat)$

Firing the rule gets us a new fact! But we treat it the same, so check against all rules…

Forward Chaining Example

- Add facts 1, 2, 3, 4, 5, 7 in turn
  - Number in [] is unification literal; √ rule firing

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. $Buffalo(Bob)$ [1]
5. $Pig(Pat)$ [1]
   6. $Faster(Bob, Pat)$ [3]

Forward Chaining Example

- Add facts 1, 2, 3, 4, 5, 7 in turn
  - Number in [] is unification literal; √ rule firing

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. $Buffalo(Bob)$ [1]
5. $Pig(Pat)$ [1]
    6. $Faster(Bob, Pat)$ [3]
7. $Slug(Steve)$ [2]
    8. $Faster(Pat, Steve)$ [3]
        9. $Faster(Bob, Steve)$ [3]

Another 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

Topics

- Reduction of first-order inference to propositional inference
- First-order inference algorithms
  - Generalized Modus Ponens
  - Forward chaining ***
  - Backward chaining ***
  - Resolution-based theorem proving ***

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 <u>any</u> solution
  - Find <u>all</u> solutions
- Backward chaining is basis of <u>logic programming</u>
  - Prolog

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)*

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

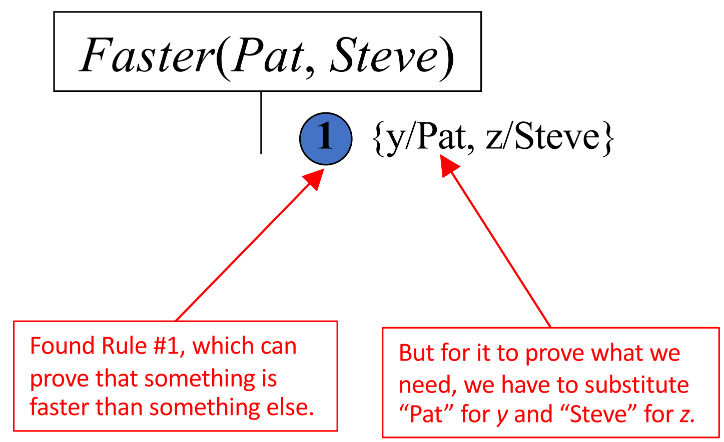
Start with what we want to prove.

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)*

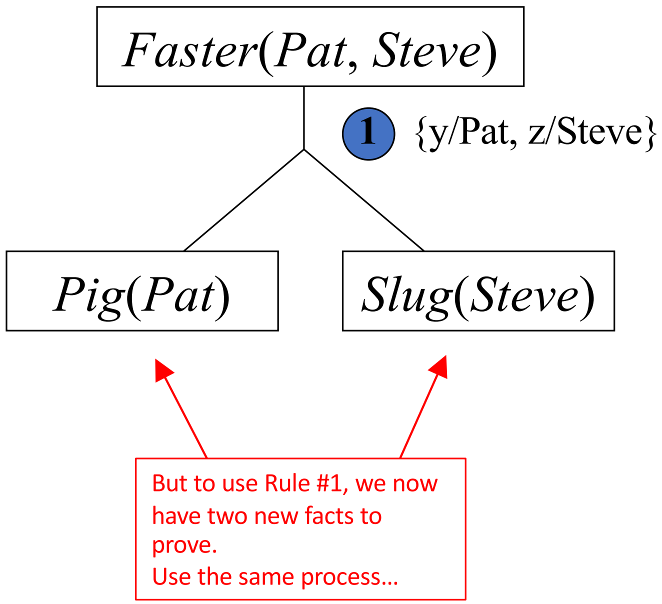


Backward Chaining Example

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: $Faster(Pat, Steve)$

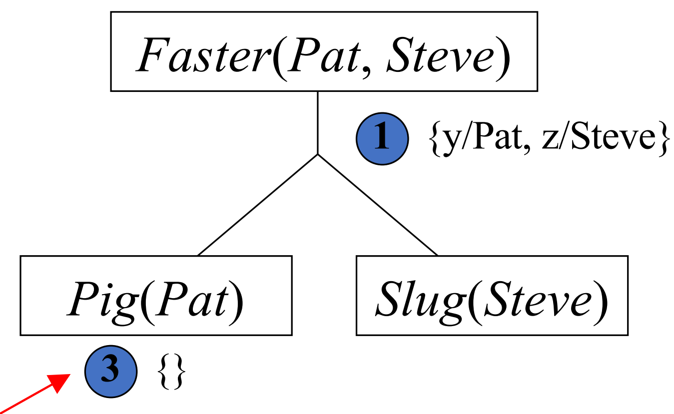


Backward Chaining Example

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: $Faster(Pat, Steve)$



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

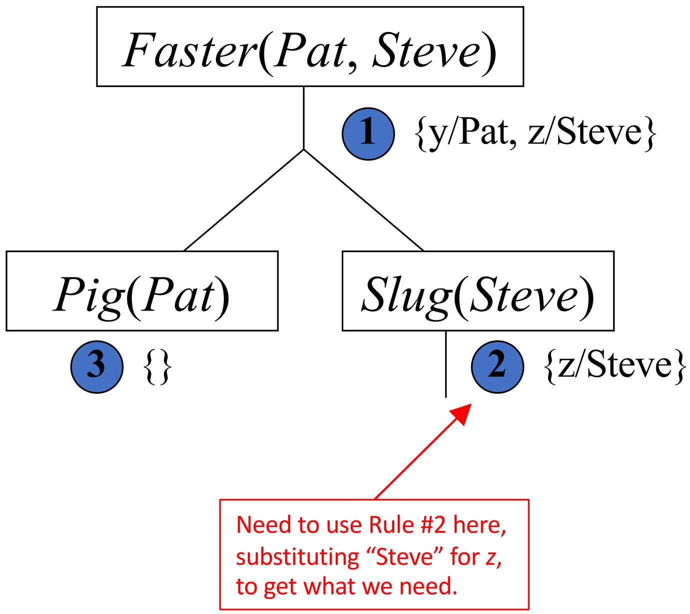
(And no substitution needed, so empty.)

Backward Chaining Example

Given facts/rules 1-5 in KB:

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

Prove: *Faster(Pat, Steve)*

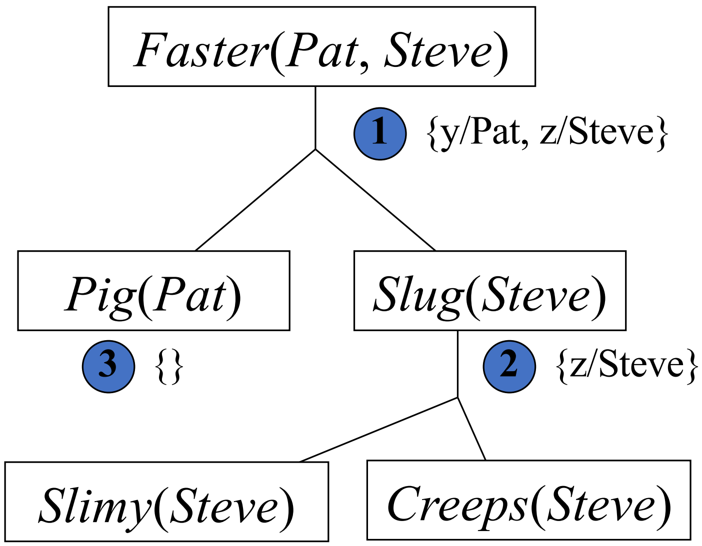


Backward Chaining Example

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)*



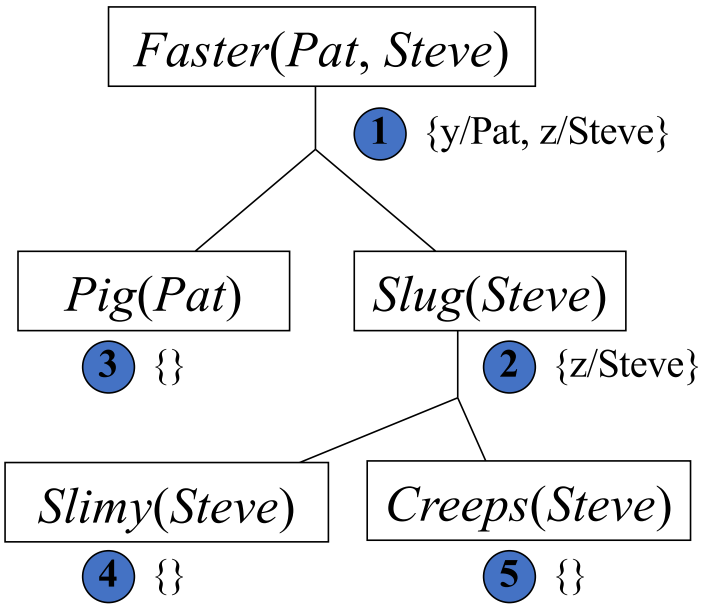
And Rule #2 requires these two facts…

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)*



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

Topics

- Reduction of first-order inference to propositional inference
- First-order inference algorithms
    - Generalized Modus Ponens
    - Forward chaining ***
    - Backward chaining ***
    - Resolution-based theorem proving ***

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 flies”
- 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)

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

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: $\neg C(\text{Minsky})$

Start off using our negated goal (proof by contradiction)

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: $\neg C(\text{Minsky})$

1: $A(\text{Minsky},\text{bar}) \vee B(\text{Minsky}) \vee C(\text{Minsky})$ *[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: $\neg C(\text{Minsky})$

1: $A(\text{Minsky},\text{bar}) \vee B(\text{Minsky}) \vee C(\text{Minsky})$ *[kb-1]*
{x/Minsky}

2: $\neg C(\text{Minsky})$, $A(\text{Minsky},\text{bar}) \vee B(\text{Minsky}) \vee C(\text{Minsky})$
2.a: $A(\text{Minsky},\text{bar}) \vee B(\text{Minsky})$ *[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: $\neg D(\text{Minsky},\text{foo})$
- kb-5: $\neg A(\text{Minsky},\text{bar})$

Goal: prove C(Minsky)

0: $\neg C(\text{Minsky})$

1: $A(\text{Minsky},\text{bar}) \vee B(\text{Minsky}) \vee C(\text{Minsky})$ *[kb-1]*
{x/Minsky}

2: $\neg C(\text{Minsky})$, $A(\text{Minsky},\text{bar}) \vee B(\text{Minsky}) \vee C(\text{Minsky})$
2.a: $A(\text{Minsky},\text{bar}) \vee B(\text{Minsky})$ *[resolution: 0,1]*

3: $D(\text{Minsky},\text{foo}) \vee \neg B(\text{Minsky})$ *[kb-2]*
{y/Minsky}

4: $A(\text{Minsky},\text{bar}) \vee B(\text{Minsky})$, $D(\text{Minsky},\text{foo}) \vee \neg B(\text{Minsky})$
4.a: $A(\text{Minsky},\text{bar}) \vee D(\text{Minsky},\text{foo})$ *[resol: 2a,3]*

And repeat to find and eliminate other opposing terms.

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: $\neg C(\text{Minsky})$

1: $A(\text{Minsky},\text{bar}) \vee B(\text{Minsky}) \vee C(\text{Minsky})$ *[kb-1]*
{x/Minsky}

2: $\neg C(\text{Minsky})$, $A(\text{Minsky},\text{bar}) \vee B(\text{Minsky}) \vee C(\text{Minsky})$
2.a: $A(\text{Minsky},\text{bar}) \vee B(\text{Minsky})$ *[resolution: 0,1]*

3: $D(\text{Minsky},\text{foo}) \vee \neg B(\text{Minsky})$ *[kb-2]*
{y/Minsky}

4: $A(\text{Minsky},\text{bar}) \vee B(\text{Minsky})$, $D(\text{Minsky},\text{foo}) \vee \neg B(\text{Minsky})$
4.a: $A(\text{Minsky},\text{bar}) \vee D(\text{Minsky},\text{foo})$ *[resol: 2a,3]*

5: $\neg A(\text{Minsky},\text{bar})$, $A(\text{Minsky},\text{bar}) \vee D(\text{Minsky},\text{foo})$
5.a: $D(\text{Minsky},\text{foo})$ *[resol: 4a,kb-5]*

And again…

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: $\neg C(\text{Minsky})$

1: $A(\text{Minsky},\text{bar}) \vee B(\text{Minsky}) \vee C(\text{Minsky})$ *[kb-1]*
{x/Minsky}

2: $\neg C(\text{Minsky})$, $A(\text{Minsky},\text{bar}) \vee B(\text{Minsky}) \vee C(\text{Minsky})$
2.a: $A(\text{Minsky},\text{bar}) \vee B(\text{Minsky})$ *[resolution: 0,1]*

3: $D(\text{Minsky},\text{foo}) \vee \neg B(\text{Minsky})$ *[kb-2]*
{y/Minsky}

4: $A(\text{Minsky},\text{bar}) \vee B(\text{Minsky})$, $D(\text{Minsky},\text{foo}) \vee \neg B(\text{Minsky})$
4.a: $A(\text{Minsky},\text{bar}) \vee D(\text{Minsky},\text{foo})$ *[resol: 2a,3]*

5: $\neg A(\text{Minsky},\text{bar})$, $A(\text{Minsky},\text{bar}) \vee D(\text{Minsky},\text{foo})$
5.a: $D(\text{Minsky},\text{foo})$ *[resol: 4a,kb-5]*

6: $D(\text{Minsky},\text{foo}) \wedge \neg D(\text{Minsky},\text{foo})$

FALSE, CONTRADICTION!!!
must be C(Minsky)

FOPC Infrerence

- 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