Inference in First-Order Logic

- Proofs

- Unification
- Generalized modus ponens
- Forward and backward chaining

- Completeness

- Resolution

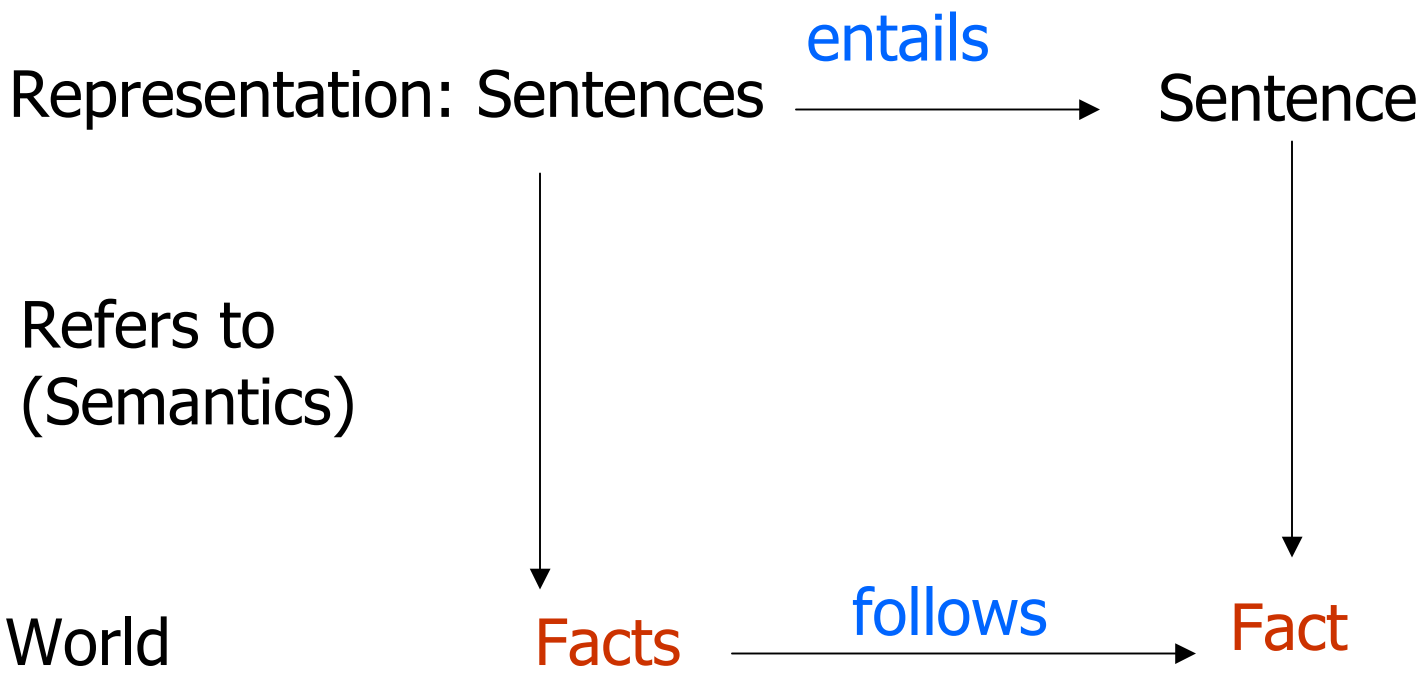
- Logic programming

Inference in First-Order Logic

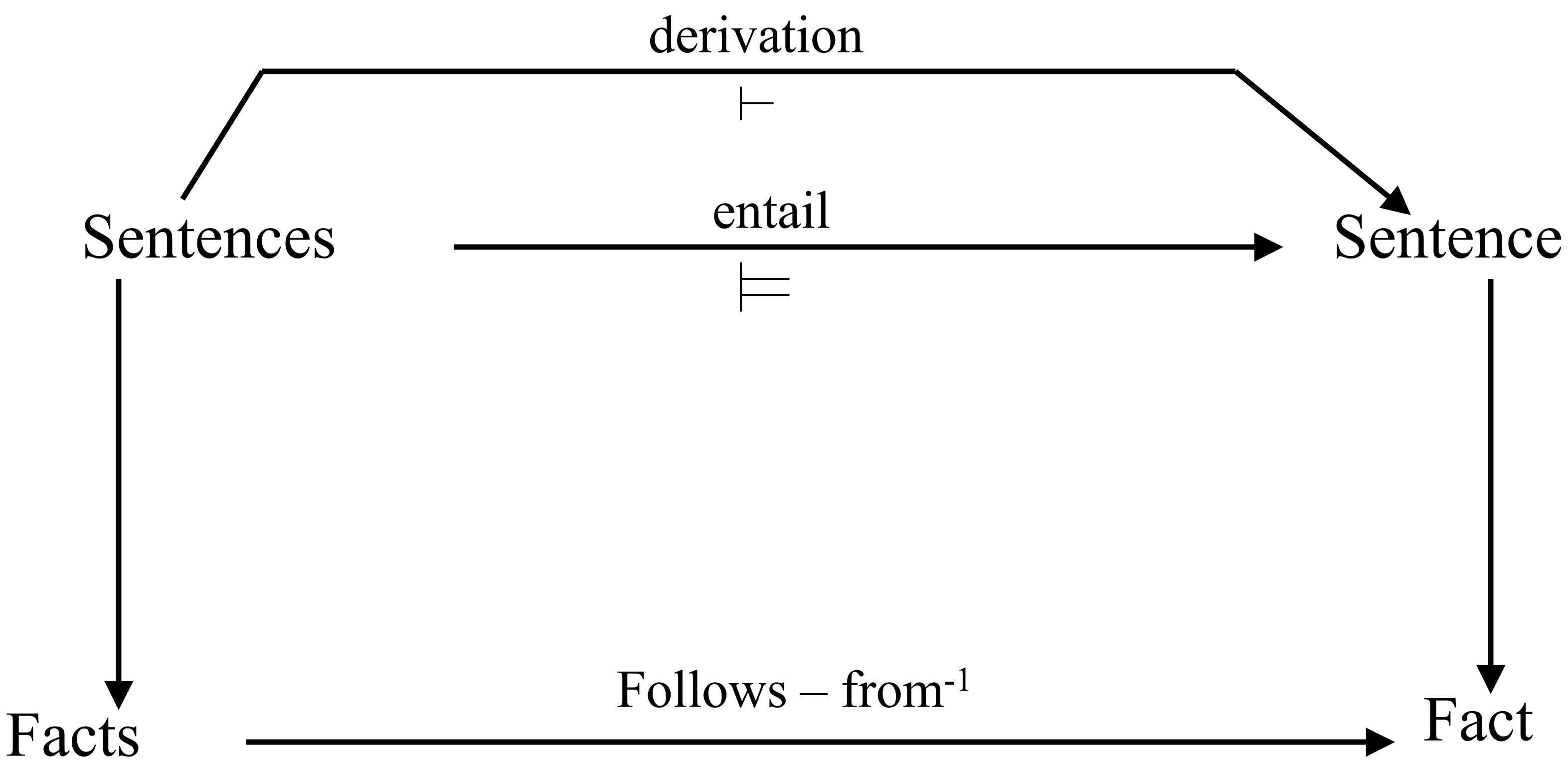
- Proofs – extend propositional logic inference to deal with quantifiers

- Unification
- Generalized modus ponens
- Forward and backward chaining – inference rules and reasoning program
- Completeness – Gödel's theorem: for FOL, any sentence entailed by another set of sentences can be proved from that set
- Resolution – inference procedure that is complete for any set of sentences
- Logic programming

Logic as a representation of the World



Desirable Properties of Inference Procedures



Remember: propositional logic

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

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

◇ **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.)

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

◇ **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.)

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

Reminder

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

- {x/a}: substitution/binding list

Proofs

Sound inference: find α such that $KB \models \alpha$.
Proof process is a search, operators are inference rules.

E.g., Modus Ponens (MP)

$$\frac{\alpha, \quad \alpha \Rightarrow \beta}{\beta} \qquad \frac{At(Joe, UCB) \quad At(Joe, UCB) \Rightarrow OK(Joe)}{OK(Joe)}$$

E.g., And-Introduction (AI)

$$\frac{\alpha \quad \beta}{\alpha \wedge \beta} \qquad \frac{OK(Joe) \quad CSMajor(Joe)}{OK(Joe) \wedge CSMajor(Joe)}$$

E.g., Universal Elimination (UE)

$$\frac{\forall x \;\; \alpha}{\alpha\{x/\tau\}} \qquad \frac{\forall x \;\; At(x, UCB) \Rightarrow OK(x)}{At(Pat, UCB) \Rightarrow OK(Pat)}$$

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

Proofs

The three new inference rules for FOL (compared to propositional logic) are:

- **Universal Elimination (UE):**
  for any sentence α, variable x and ground term τ,
  $$\frac{\forall x \quad \alpha}{\alpha\{x/\tau\}}$$

- **Existential Elimination (EE):**
  for any sentence α, variable x and constant symbol k not in KB,
  $$\frac{\exists x \quad \alpha}{\alpha\{x/k\}}$$

- **Existential Introduction (EI):**
  for any sentence α, variable x not in α and ground term g in α,
  $$\frac{\alpha}{\exists x \quad \alpha\{g/x\}}$$

Proofs

The three new inference rules for FOL (compared to propositional logic) are:

- **Universal Elimination (UE):**
  for any sentence α, variable x and ground term τ,

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

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

- **Existential Elimination (EE):**
  for any sentence α, variable x and constant symbol k not in KB,

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

  e.g., from $\exists$x Kill(x, Victim) we can infer
  Kill(Murderer, Victim), if Murderer new symbol

- **Existential Introduction (EI):**
  for any sentence α, variable x not in α and ground term g in α,

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

  e.g., from Likes(Joe, Candy) we can infer
  $\exists$x Likes(x, Candy)

Example Proof

Bob is a buffalo	1. $Buffalo(Bob)$
Pat is a pig	2. $Pig(Pat)$
Buffaloes outrun pigs	3. $\forall x, y \;\; Buffalo(x) \land Pig(y) \;\Rightarrow\; Faster(x, y)$
Bob outruns Pat	

Example Proof

AI 1 & 2	4. $Buffalo(Bob) \wedge Pig(Pat)$

Example Proof

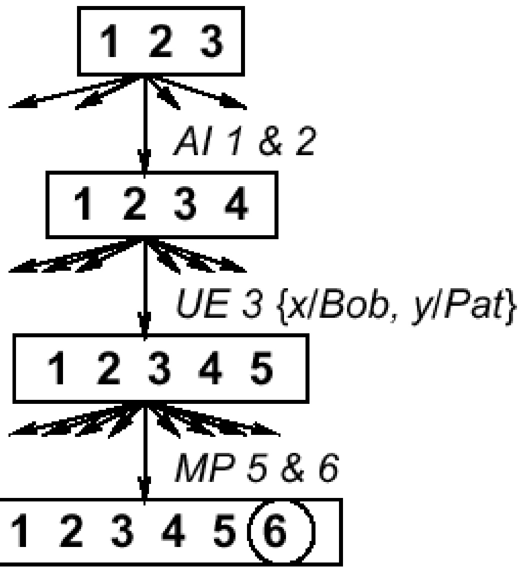
UE 3, $\{x/Bob, y/Pat\}$	5. $Buffalo(Bob) \wedge Pig(Pat) \Rightarrow Faster(Bob, Pat)$

Example Proof

MP 4 & 5	6. $Faster(Bob, Pat)$

Search with primitive example rules

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

Unification

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

p	q	σ
$Knows(John, x)$	$Knows(John, Jane)$	
$Knows(John, x)$	$Knows(y, OJ)$	
$Knows(John, x)$	$Knows(y, Mother(y))$	

Goal of unification: finding σ

Unification

		$\{x/Jane\}$
		$\{x/John, y/OJ\}$
		$\{y/John, x/Mother(John)\}$

<u>Idea</u>: 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 $Likes(John, Jane)$

$Likes(John, OJ)$

$Likes(John, Mother(John))$

Extra example for unification

P	Q	σ
Student(x)	Student(Bob)	{x/Bob}
Sells(Bob, x)	Sells(x, coke)	{x/coke, x/Bob} Is it correct?

Extra example for unification

P	Q	σ
Student(x)	Student(Bob)	{x/Bob}
Sells(Bob, x)	Sells(y, coke)	{x/coke, y/Bob}

More Unification Examples

VARIABLE term

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.

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

E.g. $p_1{}' =$ Faster(Bob,Pat)

$p_2{}' =$ Faster(Pat,Steve)

$p_1 \wedge p_2 \ \Rightarrow\ q\ =\ Faster(x, y) \wedge Faster(y, z)\ \Rightarrow\ Faster(x, z)$

$\sigma = \{x/Bob, y/Pat, z/Steve\}$

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

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

Soundness of GMP

Need to show that

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

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

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

Properties of GMP

- Why is GMP and 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 $(B \Rightarrow A) \land ((C \land D) \Rightarrow B)$

Horn form

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

- e.g., $\exists x$ Owns(Nono, x) $\wedge$ Missile(x) becomes

  Owns(Nono, M)
  Missile(M)

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

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

Add facts 1, 2, 3, 4, 5, 7 in turn.
Number in [] = unification literal; $\surd$ indicates 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. $Buffalo(Bob)$ [1a, $\times$]
5. $Pig(Pat)$ [1b, $\surd$] $\rightarrow$ 6. $Faster(Bob, Pat)$ [3a, $\times$], [3b, $\times$]
   [2a, $\times$]
7. $Slug(Steve)$ [2b, $\surd$]
   $\rightarrow$ 8. $Faster(Pat, Steve)$ [3a, $\times$], [3b, $\surd$]
   $\rightarrow$ 9. $Faster(Bob, Steve)$ [3a, $\times$], [3b, $\times$]

Example: Forward Chaining

Current available rules

- A ^ C => E
- D ^ C => F
- B ^ E => F
- B => C
- F => G

Example: Forward Chaining

Current available rules

- A ^ C => E (1)
- D ^ C => F (2)
- B ^ E => F (3)
- B => C (4)
- F => G (5)

Percept 1. A (is true)
Percept 2. B (is true)

then, from (4), C is true, then the premises of (1) will be satisfied, resulting to make E true, then the premises of (3) are going to be satisfied, thus F is true, and finally from (5) G is true.

Backward chaining

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)

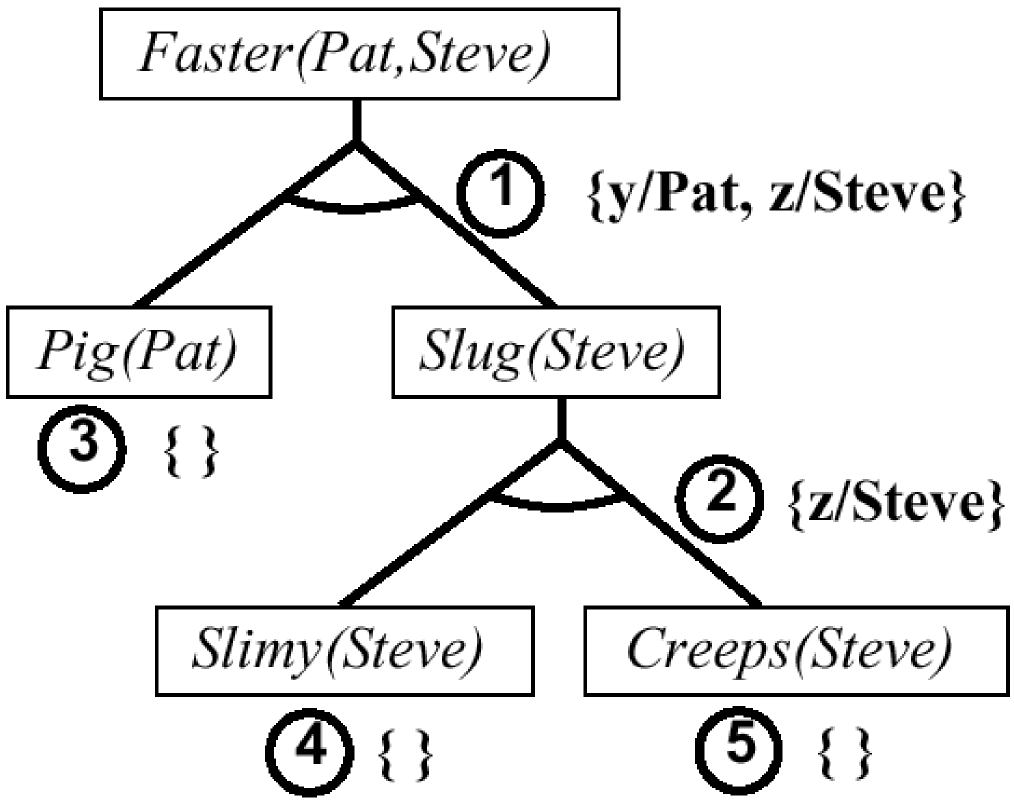
(More complications help to avoid infinite loops)

Two versions: find any solution, find all solutions

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

Backward chaining example

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. $Creeps(Steve)$



A simple example

- B^C=> G
- A^G=> I
- D^G=>J
- E=>   C
- D^C=>K
- F=>C
- Q: I?

A simple example

- B^C=> G
- A^G=> I
- D^G=>J
- E=>   C
- D^C=>K
- F=>C
- Q: I?

1. A^G
2. A?
    1. USER
3. G?
    1. B^C
        1. USER
        2. E v F

Another Example (from Konelsky)

- 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.

Forward Chaining

- The knowledge base initially contains:
  - Programmer(x) $\wedge$ Emulator(y) $\wedge$ People(z) $\wedge$ Provide(x,z,y) $\Rightarrow$ Criminal(x)
  - Use(friends, x) $\wedge$ Runs(x, N64 games) $\Rightarrow$ Provide(Reality Man, friends, x)
  - Software(x) $\wedge$ Runs(x, N64 games) $\Rightarrow$ Emulator(x)

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)

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

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)

- 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.

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)

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

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)

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

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)

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

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

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)

- 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.

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)

- Add:
  - FORWARD-CHAIN(Runs(U64, N64 games))

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)

Runs(U64, N64 games) (8)

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

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)
Runs(U64, N64 games) (8)

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

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)
Runs(U64, N64 games) (8)

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

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)

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) $\wedge$ 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)

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.

Backward Chaining

- Current knowledge:
  - hurts(x, head)

- What implications can lead to this fact?
  - kicked(x, head)
  - fell_on(x, head)
  - brain_tumor(x)
  - hangover(x)

- What facts do we need in order to prove these?

Backward Chaining

- The algorithm (available in detail in Fig. 9.2 on page 275 of the text):
  - a knowledge base KB
  - a desired conclusion c or question q
  - finds all sentences that are answers to q in KB *or* proves 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) $\Rightarrow$ Criminal(x)
  - Kill(x, y) $\Rightarrow$ Criminal(x)
  - Grow(x, y) $\wedge$ Illegal(y) $\Rightarrow$ Criminal(x)
  - HaveSillyName(x) $\Rightarrow$ Criminal(x)
  - Programmer(x) $\wedge$ Emulator(y) $\wedge$ People(z) $\wedge$ Provide(x,z,y) $\Rightarrow$Criminal(x)

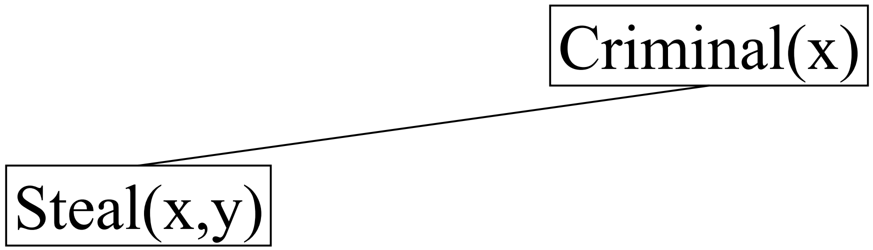
Backward Chaining

- Question: Has Reality Man done anything criminal?

Criminal(x)

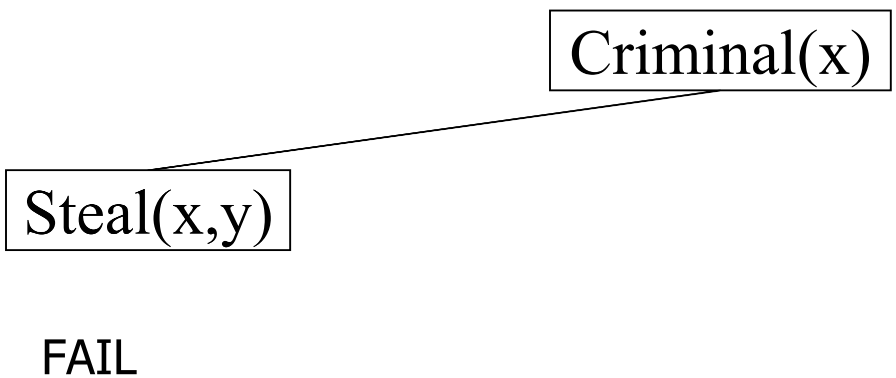
Backward Chaining

- Question: Has Reality Man done anything criminal?



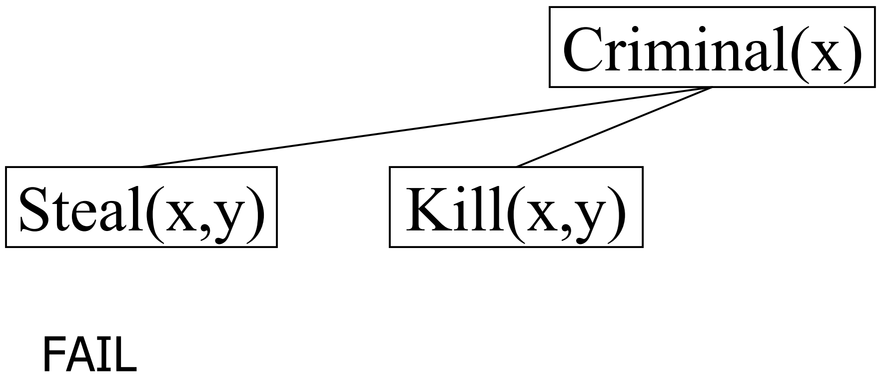
Backward Chaining

- Question: Has Reality Man done anything criminal?



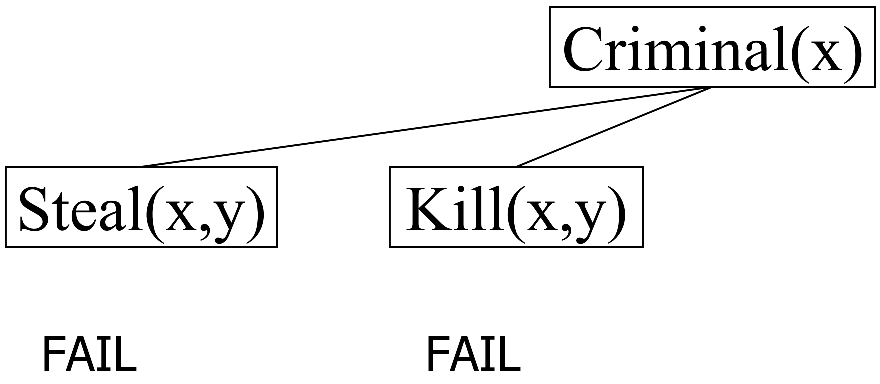
Backward Chaining

- Question: Has Reality Man done anything criminal?



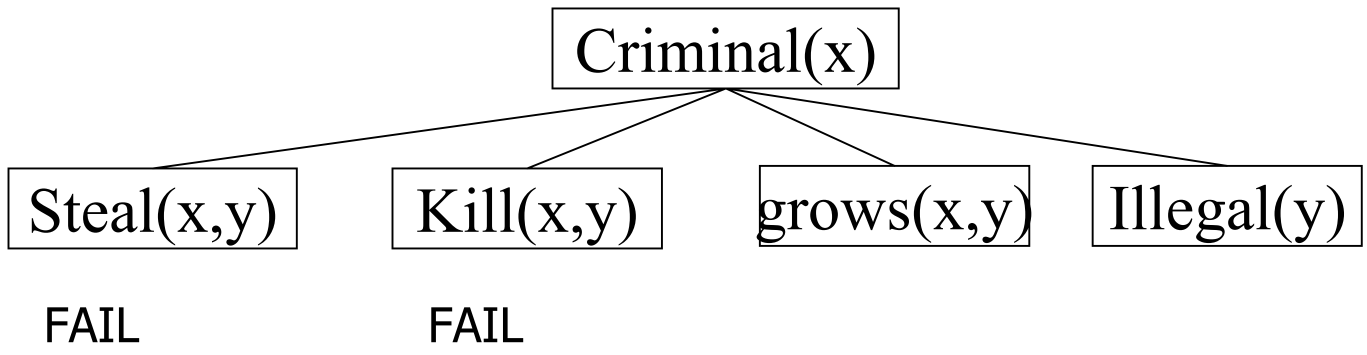
Backward Chaining

- Question: Has Reality Man done anything criminal?



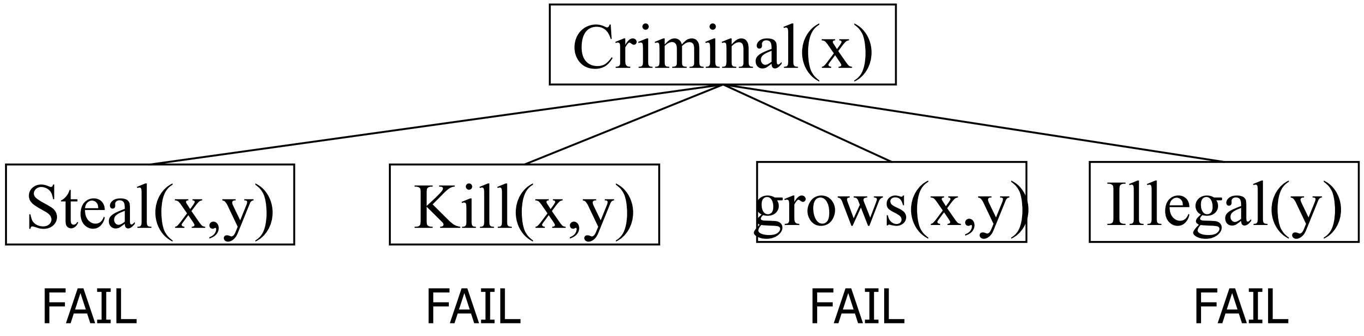
Backward Chaining

- Question: Has Reality Man done anything criminal?



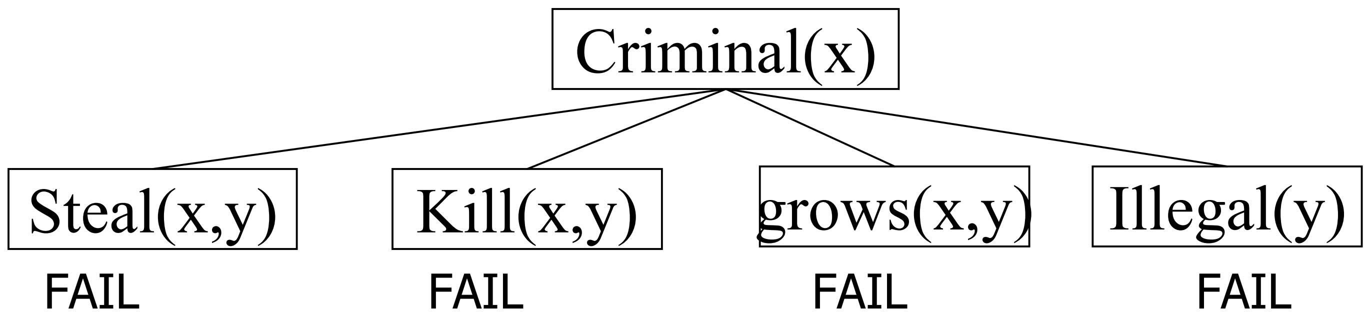
Backward Chaining

- Question: Has Reality Man done anything criminal?



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)

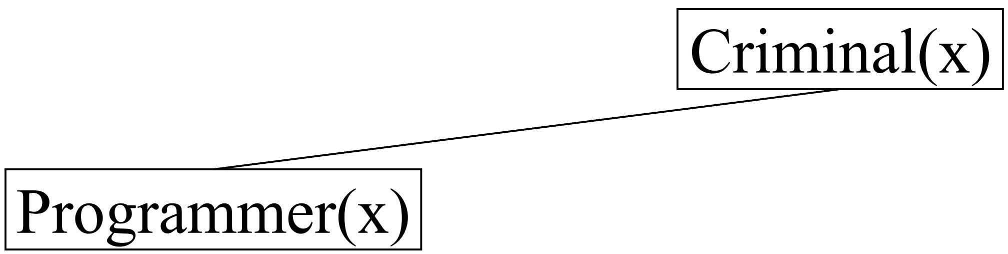
Backward Chaining

- Question: Has Reality Man done anything criminal?

Criminal(x)

Backward Chaining

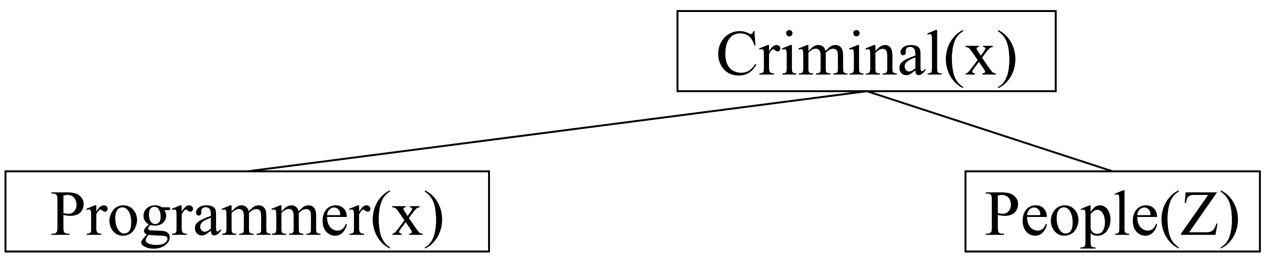
- Question: Has Reality Man done anything criminal?



Yes, {x/Reality Man}

Backward Chaining

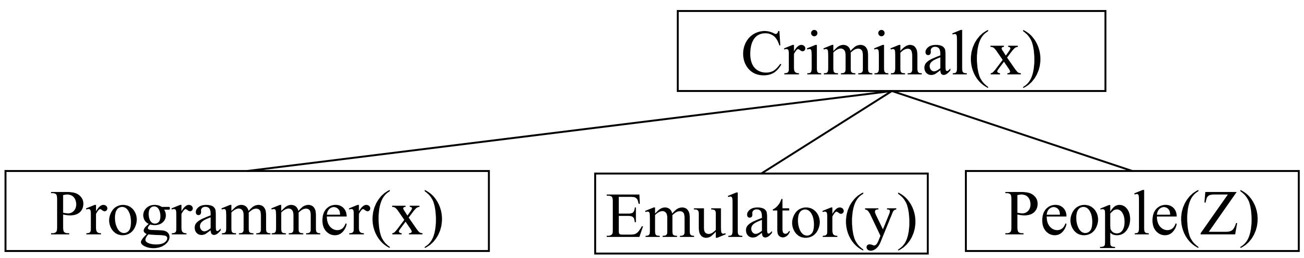
- Question: Has Reality Man done anything criminal?



Yes, {x/Reality Man}　　　*Yes, {z/friends}*

Backward Chaining

- Question: Has Reality Man done anything criminal?

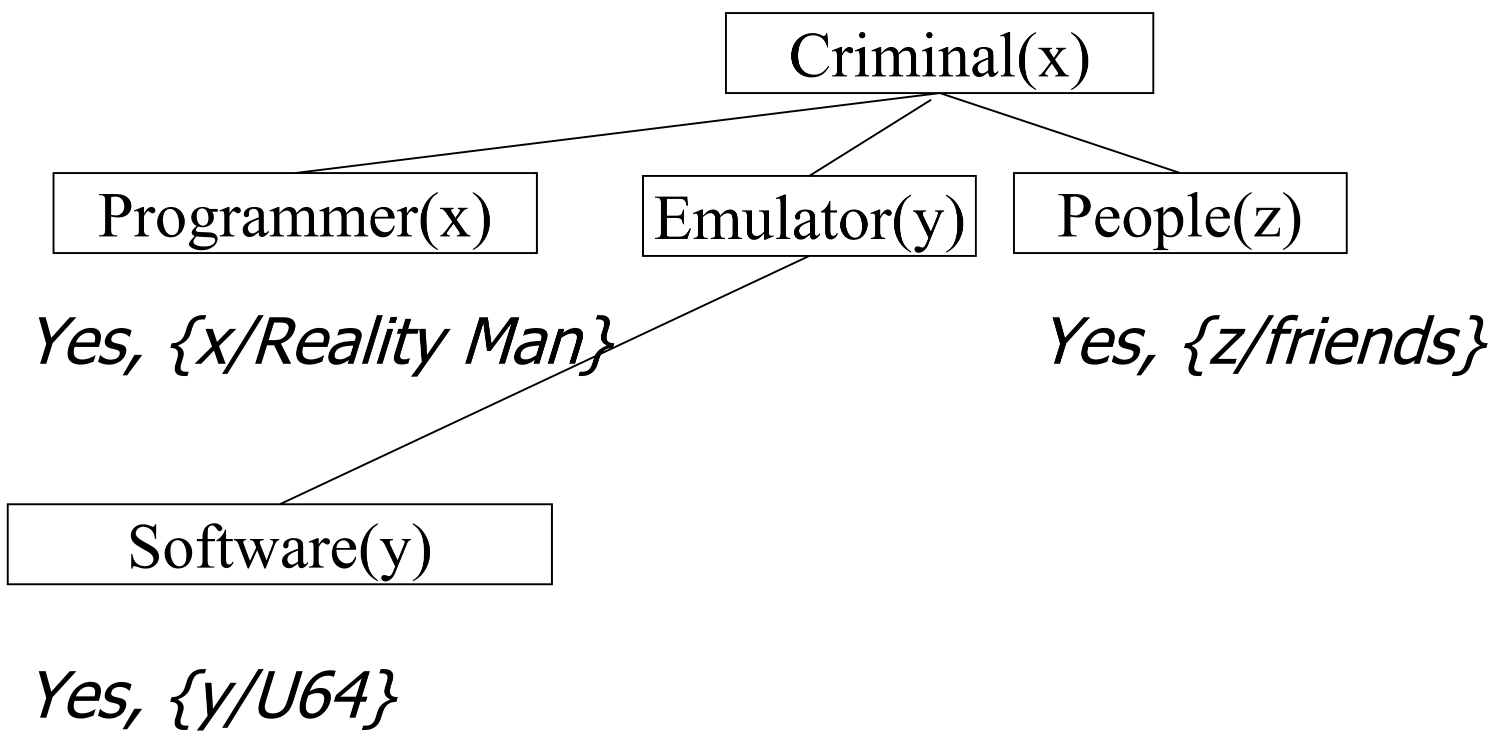


Yes, {x/Reality Man}

Yes, {z/friends}

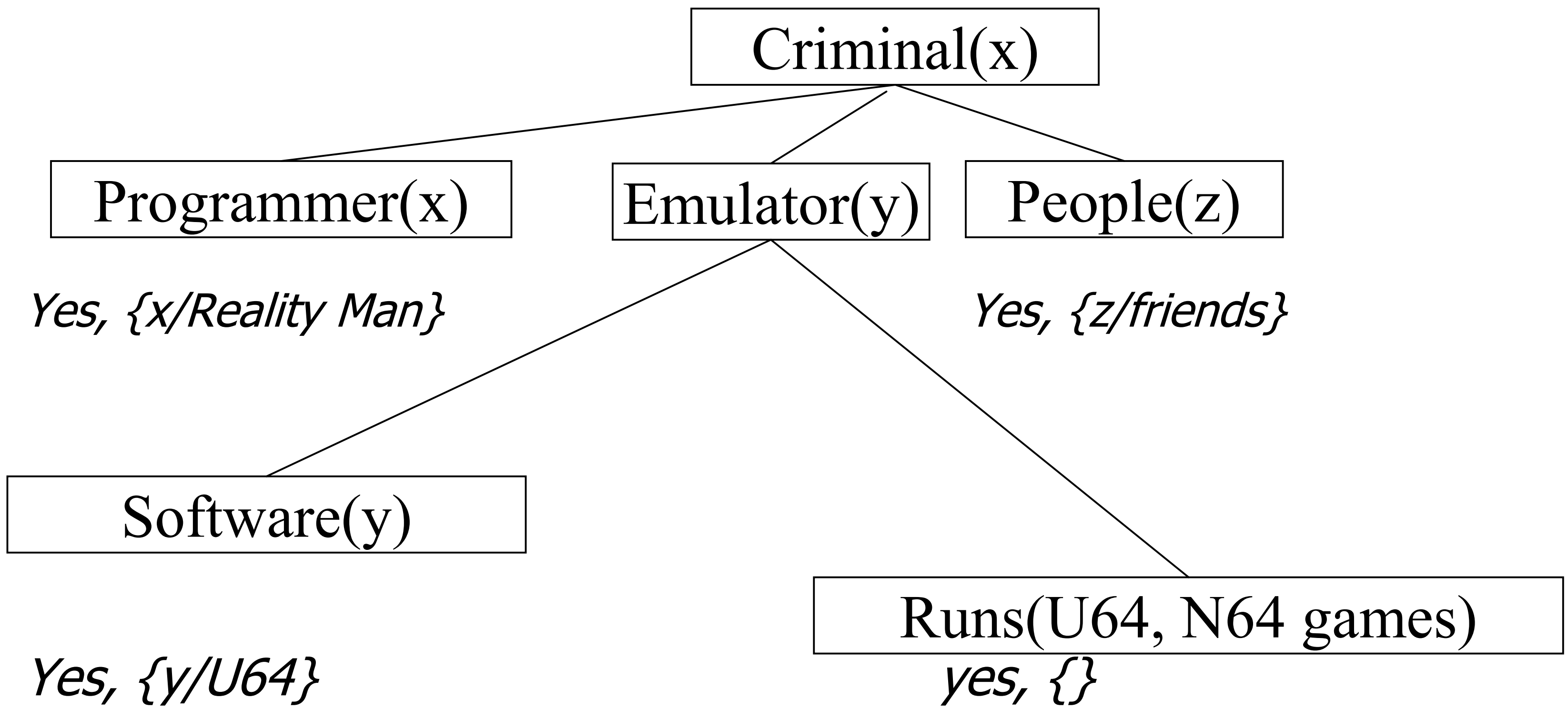
Backward Chaining

- Question: Has Reality Man done anything criminal?



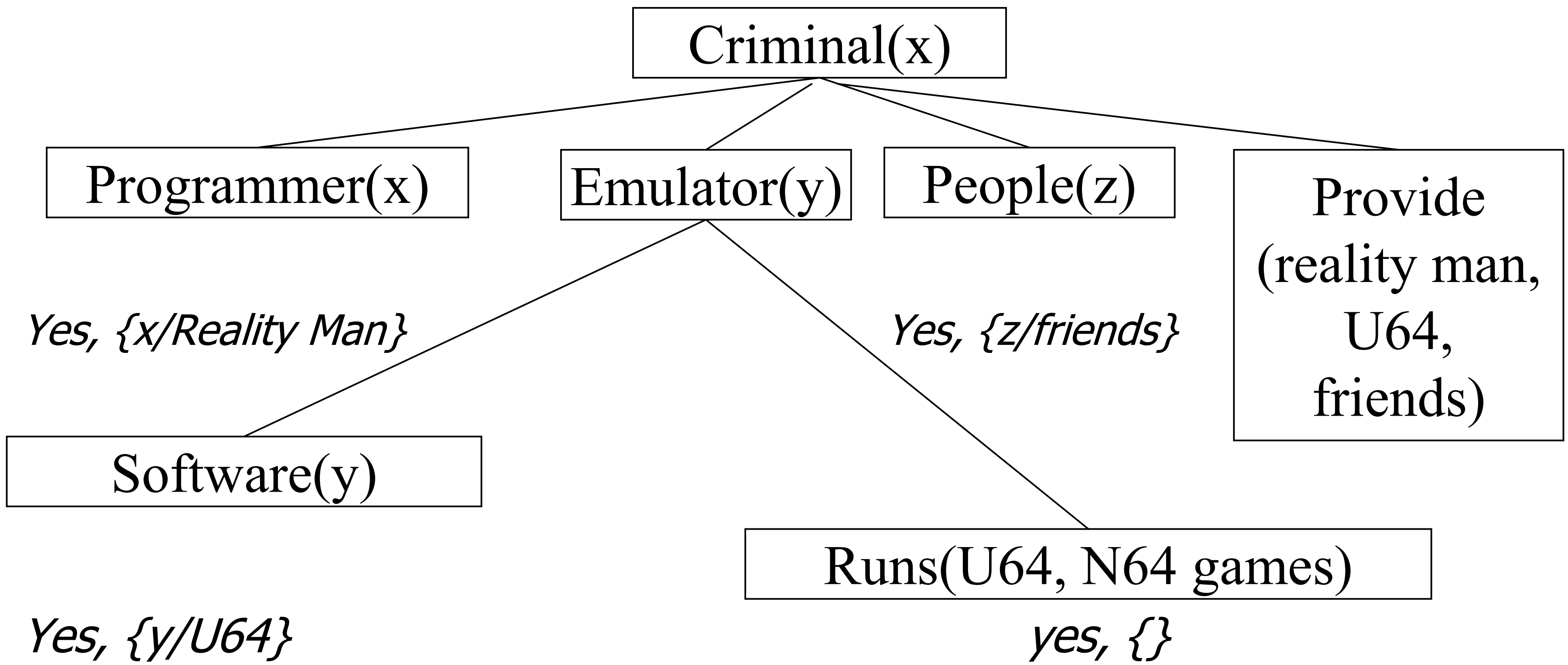
Backward Chaining

- Question: Has Reality Man done anything criminal?



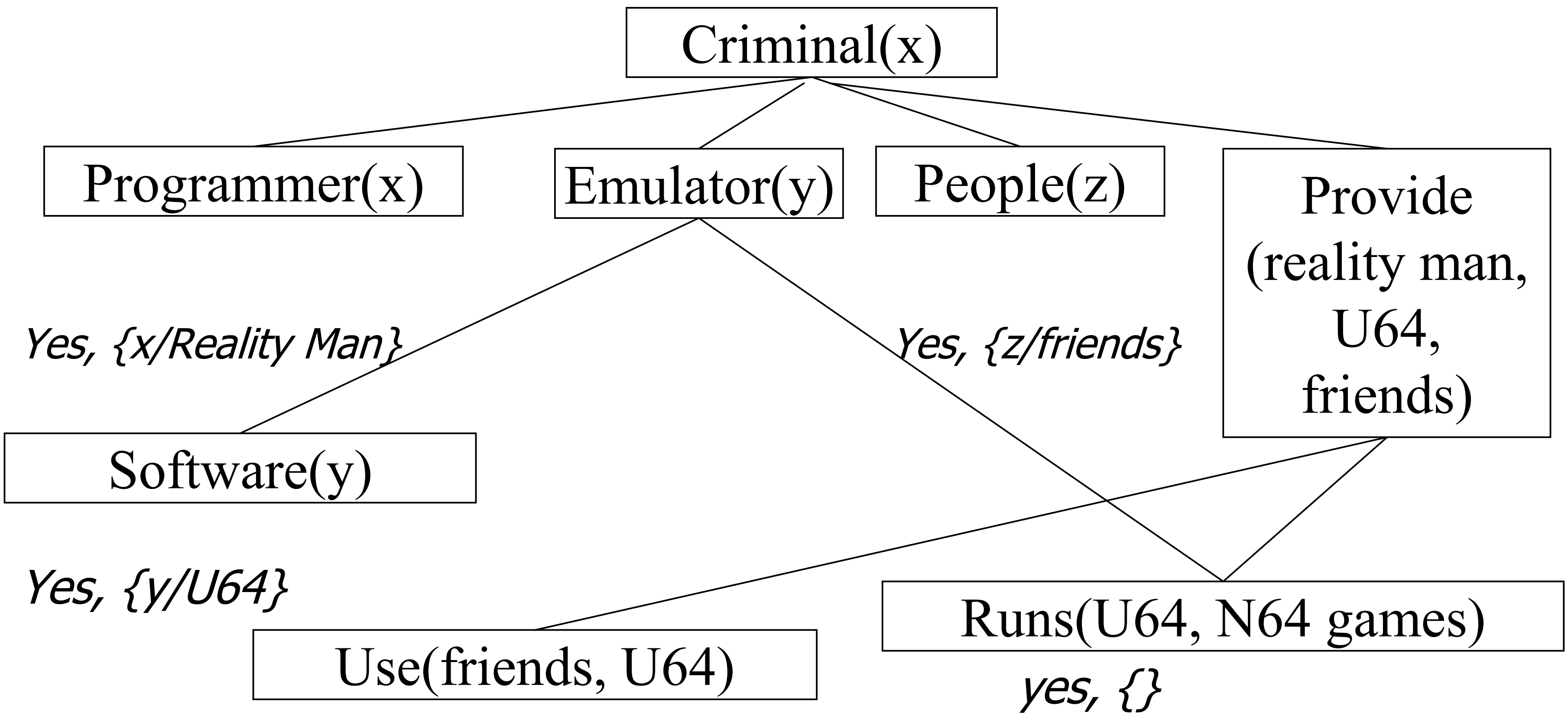
Backward Chaining

- Question: Has Reality Man done anything criminal?



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.

Completeness

- 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)
  - Cannot be expressed in Horn form 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.***

Completeness

- 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.

Completeness in FOL

Procedure i is complete if and only if

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

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?

Historical note

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

Kinship Example

KB:

(1) father (art, jon)

(2) father (bob, kim)

(3) father (X, Y) $\Rightarrow$ 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)
                      \        /
                         []
```

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

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:

Inference continues until an empty clause is derived (contradiction)

Resolution inference rule

Basic propositional version:

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

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

Remember: normal forms

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

Conjunctive Normal Form (CNF—universal)

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

E.g., $(A \lor \neg B) \land (B \lor \neg C \lor \neg D)$

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

Disjunctive Normal Form (DNF—universal)

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

E.g., $(A \land B) \lor (A \land \neg C) \lor (A \land \neg D) \lor (\neg B \land \neg C) \lor (\neg B \land \neg D)$

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

Horn Form (restricted)

$conjunction$ of $Horn\ clauses$ (clauses with ≤ 1 positive literal)

E.g., $(A \lor \neg B) \land (B \lor \neg C \lor \neg D)$

Often written as set of implications:

$B \Rightarrow A$ and $(C \land D) \Rightarrow B$

Conjunctive normal form

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 $\vee$, e.g., $(P \wedge Q) \vee R$ becomes $(P \vee Q) \wedge (P \vee R)$

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

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

4. Standardize variables
$(\forall x)(\neg P(x) \vee ((\forall y)(\neg P(y) \vee P(f(x,y))) \wedge (\exists z)(Q(x,z) \wedge \neg P(z))))$

Examples: Converting FOL sentences to clause form…

5. Eliminate existential quantification

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

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…

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
   - $\neg P(x) \vee \neg P(y) \vee P(f(x,y))$
   - $\neg P(z) \vee Q(z,g(z))$
   - $\neg P(w) \vee \neg P(g(w))$

Resolution proof

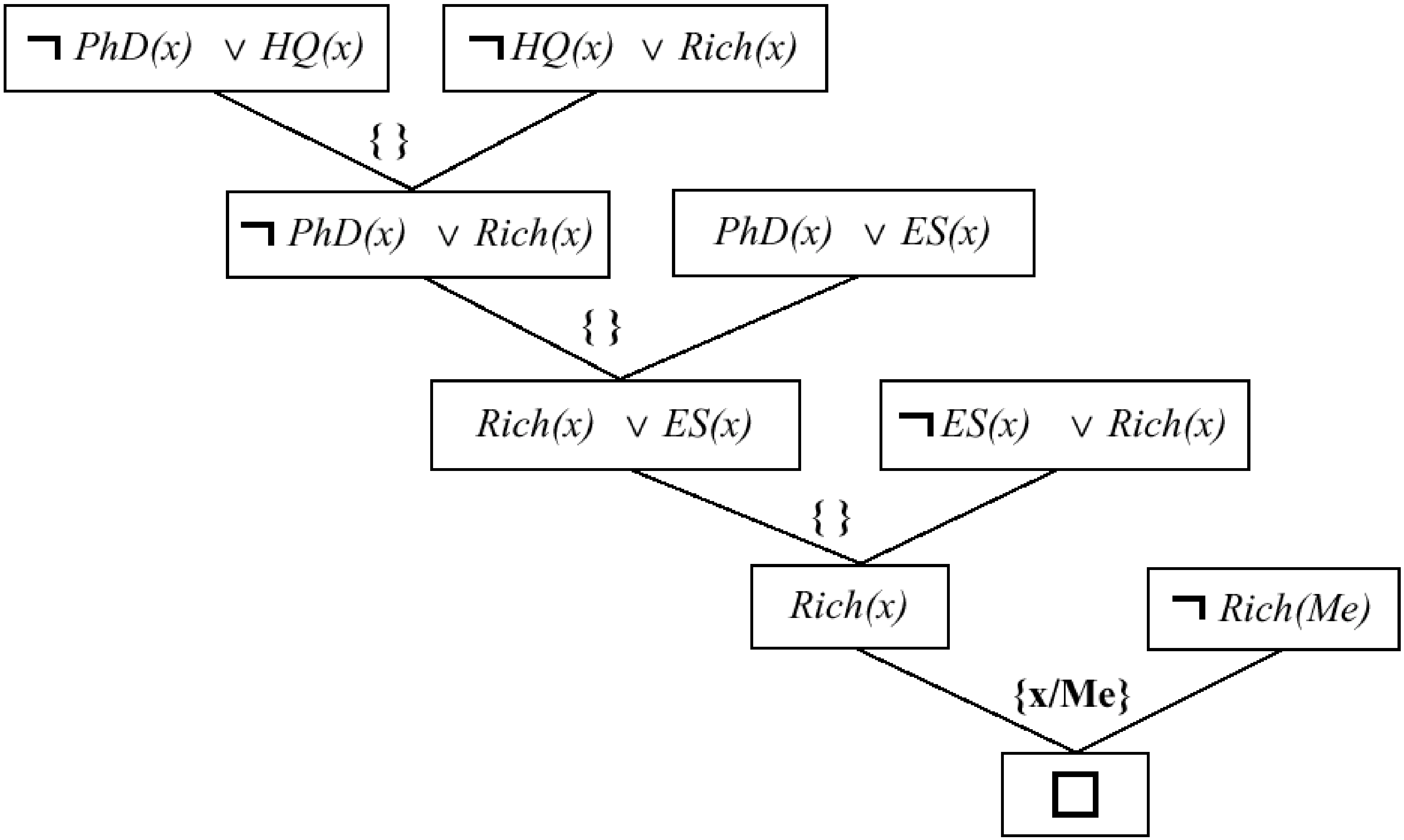
To prove α:

– negate it
– convert to CNF
– add to CNF KB
– infer contradiction

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

$$\neg PhD(x) \vee HighlyQualified(x)$$
$$PhD(x) \vee EarlyEarnings(x)$$
$$\neg HighlyQualified(x) \vee Rich(x)$$
$$\neg EarlyEarnings(x) \vee Rich(x)$$

Resolution proof



Inference in First-Order Logic

- Canonical forms for resolution

Conjunctive Normal Form (CNF)

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

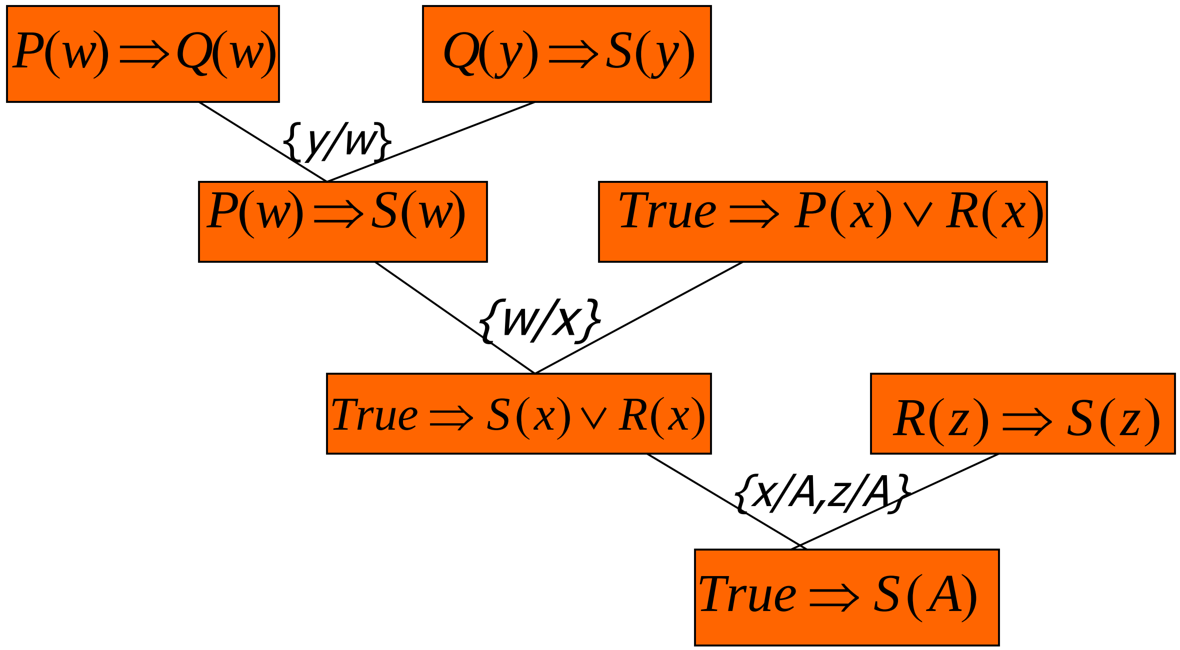
Implicative Normal Form (INF)

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

Reference in First-Order Logic

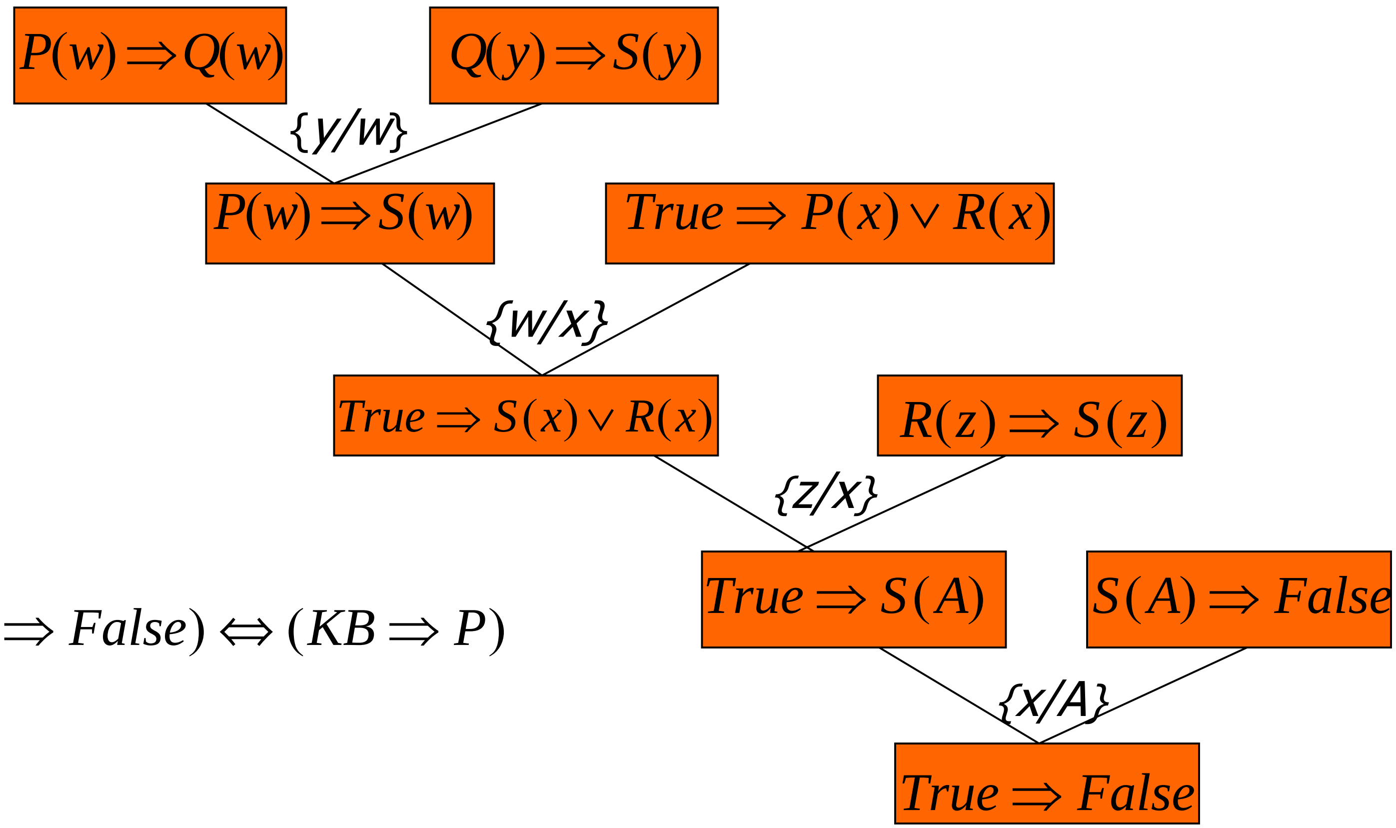
- Resolution Proofs

In a forward- or backward-chaining algorithm, just as Modus Ponens.



Inference in First-Order Logic

- Refutation



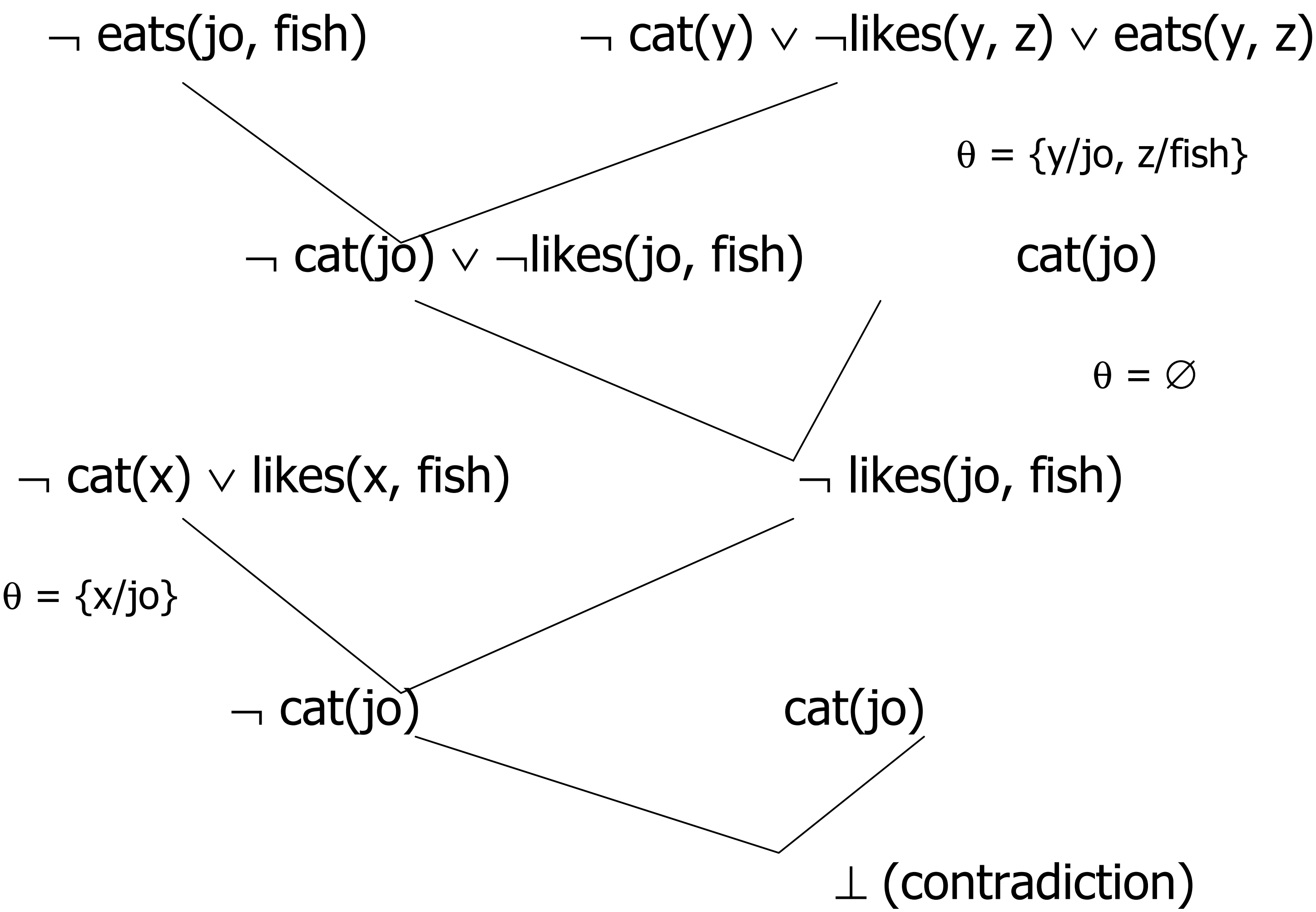
$$(KB \wedge \neg P \Rightarrow False) \Leftrightarrow (KB \Rightarrow P)$$

Example of Refutation Proof (in conjunctive normal form)

(1)	Cats like fish	$\neg cat(x) \vee likes(x, fish)$
(2)	Cats eat everything they like	$\neg cat(y) \vee \neg likes(y, z) \vee eats(y, z)$
(3)	Josephine is a cat.	$cat(jo)$
(4)	Prove: Josephine eats fish.	$eats(jo, fish)$

Backward Chaining

Negation of goal wff: ¬ eats(jo, fish)



Forward chaining

```
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)      ¬ eats (jo,fish)
                                     \       /
                                        []
```

Question:

- When would you use forward chaining? What about backward chaining?

- A:
  - FC: If expert needs to gather information before any inferencing
  - BC: If expert has a hypothetical solution