CS 482/682 – AI: Inference in FOL

Fall 2021

Outline

- **FOL Inference**
  - Proof
  - Unification
  - Generalized Modus Ponens
  - Forward and backward chaining
- Industrial-strength Inference
  - Completeness
  - Resolution
  - Logic programming

Proof

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)

Example Proof

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

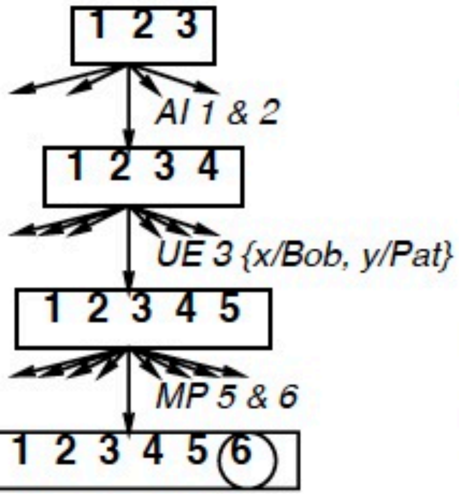
AI 1 & 2	4. $Buffalo(Bob) \wedge Pig(Pat)$
UE 3, $\{x/Bob, y/Pat\}$	5. $Buffalo(Bob) \wedge Pig(Pat) \; \Rightarrow \; Faster(Bob, Pat)$
MP 4.& 5.	6. $Faster(Bob, Pat)$

Search with Primitive Inference 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
$Knows(John, x)$	$Knows(John, Jane)$	$\{x/Jane\}$
$Knows(John, x)$	$Knows(y, OJ)$	$\{x/John, y/OJ\}$
$Knows(John, x)$	$Knows(y, Mother(y))$	$\{y/John, x/Mother(John)\}$

<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 $Likes(John, Jane)$

$Likes(John, OJ)$

$Likes(John, Mother(John))$

Generalized Modus Ponens (GMP)

$$\frac{p_1{}',\ p_2{}',\ \ldots,\ p_n{}',\ (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{q\sigma} \qquad \text{where } p_i{}'\sigma = p_i\sigma \text{ for all } i$$

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

GMP used with KB of <u>definite clauses</u> (*exactly* one positive literal):
either a single atomic sentence or

(conjunction of atomic sentences) $\Rightarrow$ (atomic sentence)

All variables assumed universally quantified

For Your Information (Google & Book)

- What is an atom? Give the definition and an example.

  Answer: An atom is a symbol starting with a lower case letter.
  Example: ai_is_fun

- What is a body? Give the definition and an example.

  Answer: A body is an atom or is of the form $b_1 \wedge b_2$ where b_1 and b_2 are bodies.
  Example: $students\_are\_motivated \wedge ai\_is\_fun$

- What is a definite clause? Give the definition and an example.

  Answer: A definite clause is an atom or is a rule of the form $h \leftarrow b$ where h is an atom and b is a body. (Read this as h if b.)
  Example:
  $students\_are\_successful \leftarrow ai\_is\_fun \wedge students\_are\_motivated$

Soundness of GMP

Need to show that

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

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

An Example Proof

It is a crime for an American to sell alcohol to a minor. Jimmy, a minor, has some beer. All of Jimmy's beer was sold to him by Nathan, an American.

1. $\forall x,y,z\ American(x) \land Alcohol(y) \land Minor(z) \land$
   $Sells(x,y,z) \Rightarrow Criminal(x)$
2. $Minor(Jimmy)$
3. $\exists x\ Owns(Jimmy,x) \land Beer(x)$
4. $\forall x\ Owns(Jimmy,x) \land Beer(x) \Rightarrow Sells(Nathan,x,Jimmy)$
5. $American(Nathan)$
6. $\forall x\ Beer(x) \Rightarrow Alcohol(x)$

An Example Proof

1. $\forall x,y,z$ American(x) ∧ Alcohol(y) ∧ Minor(z) ∧ Sells(x,y,z) ⇒ Criminal(x)
2. Minor(Jimmy)
3. $\exists x$ Owns(Jimmy,x) ∧ Beer(x)
4. $\forall x$ Owns(Jimmy,x) ∧ Beer(x) ⇒ Sells(Nathan,x,Jimmy)
5. American(Nathan)
6. $\forall x$ Beer(x) ⇒ Alcohol(x)
7. From 3 and existential elimination
   Owns(Jimmy, B1) ∧ Beer(B1)
8. From 7. and And-elimination
   Owns(Jimmy, B1)
   Beer(B1)
9. From 6 and universal elimination
   Beer(B1) ⇒ Alcohol(B1)
10. From 9, 8-2, and modus ponens
    Alcohol(B1)
11. From 4 and universal elimination
    Owns(Jimmy,B1) ∧ Beer(B1) ⇒ Sells(Nathan,B1,Jimmy)
12. From 11, 8-1, 8-2, and modus ponens
    Sells(Nathan,B1,Jimmy)
13. From 1 and universal elimination (3 times)
    American(Nathan) ∧ Alcohol(B1) ∧ Minor(Jimmy) ∧ Sells(Nathan,B1,Jimmy) ⇒ Criminal(Nathan)
14. From 5, 10, 2, 12, and and-introduction
    American(Nathan) ∧ Alcohol(B1) ∧ Minor(Jimmy) ∧ Sells(Nathan,B1,Jimmy)
15. From 13, 14, and modus ponens
    Criminal(Nathan)

Streamlined Proof with Generalized Modus Ponens…

1. American (x) ∧ Alcohol (y) ∧ Minor (z) ∧ Sells (x,y,z) ⇒ Criminal (x)
2. Minor (Jimmy)
3. Owns (Jimmy,B1)
4. Beer (B1)
5. Owns (Jimmy,x) ∧ Beer (x) ⇒ Sells (Nathan,x,Jimmy)
6. American (Nathan)
7. Beer (x) ⇒ Alcohol (x)
8. From 4 and 7 using GMP: Alcohol (B1)
9. From 3, 4, and 5 using GMP: Sells (Nathan,B1,Jimmy)
10. From 6, 8, 2, 9, and 1 using modus ponens: Criminal (Nathan)

Now only 3 induction steps!
(plus start-up time)

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

1. $\forall x,y,z\ American(x) \land Alcohol(y) \land Minor(z) \land Sells(x,y,z) \Rightarrow Criminal(x)$
2. $Minor(Jimmy)$
3. $\exists x\ Owns(Jimmy,x) \land Beer(x)$
4. $\forall x\ Owns(Jimmy,x) \land Beer(x) \Rightarrow Sells(Nathan,x,Jimmy)$
5. $American(Nathan)$
6. $\forall x\ Beer(x) \Rightarrow Alcohol(x)$

- Consider all the base terms: Nathan, Jimmy, B1, B2, ...
- Start instantiating #6
  - $Beer(Nathan) \Rightarrow Alcohol(Nathan)$
  - $Beer(Jimmy) \Rightarrow Alcohol(Jimmy)$
  - $Beer(B1) \Rightarrow Alcohol(B1)$
  - $Beer(B2) \Rightarrow Alcohol(B2)$
- Leads to a very disorganized (and full) knowledge base!

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, $\times$]

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

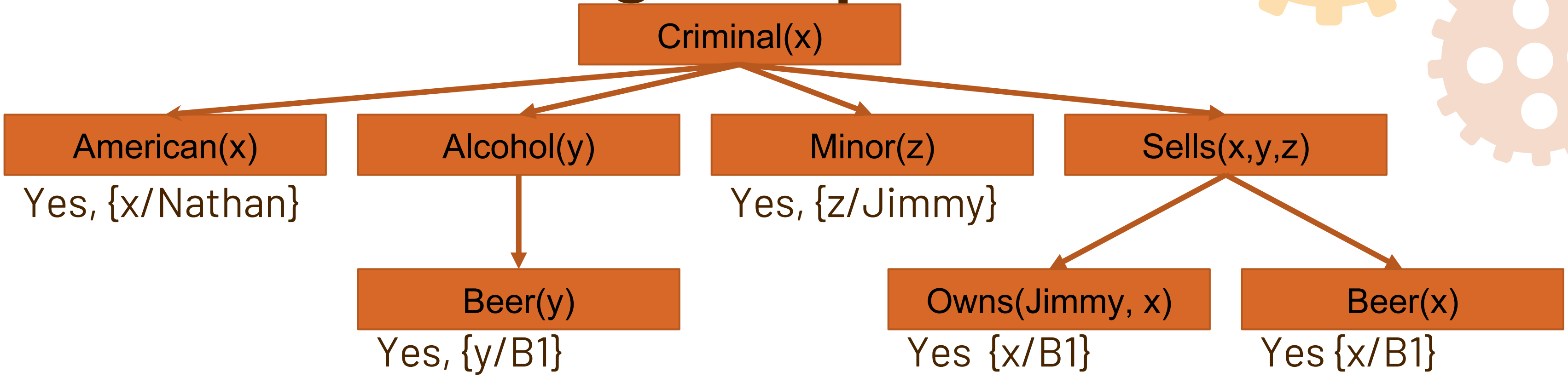
(Some added complications in keeping track of the unifiers)

(More complications help to avoid infinite loops)

Two versions: find <u>any</u> solution, find <u>all</u> solutions

Backward chaining is the basis for <u>logic programming</u>, e.g., Prolog

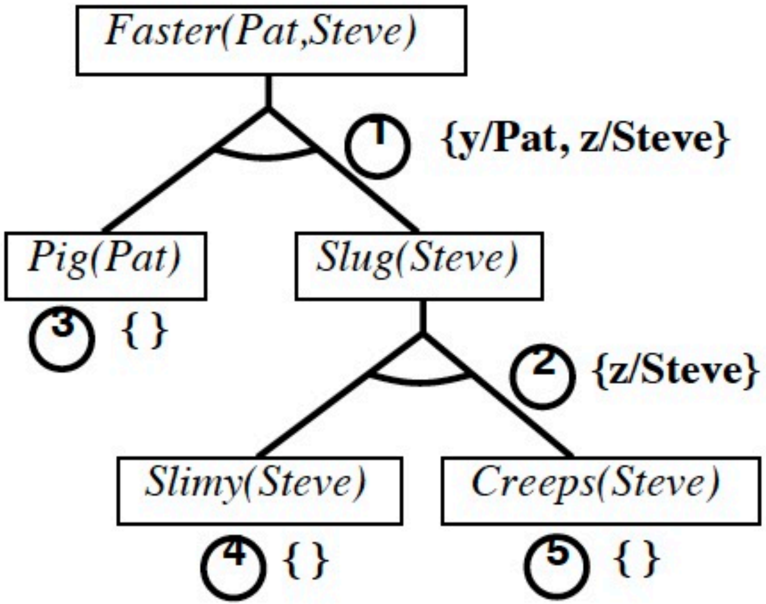
Backward Chaining Example



1. American(x) ∧ Alcohol(y) ∧ Minor(z) ∧ Sells(x,y,z) ⇒ Criminal(x)
2. American(Nathan)
3. Minor(Jimmy)
4. Beer(x)⇒Alcohol(x)
5. Owns(Jimmy,x) ∧ Beer(x) ⇒ Sells(Nathan,x,Jimmy)
6. Beer(B1)
7. Owns(Jimmy,B1)

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. $Creeps(Steve)$



Outline

- FOL Inference
  - Proof
  - Unification
  - Generalized Modus Ponens
  - Forward and backward chaining
- **Industrial-strength Inference**
  - Completeness
  - Resolution
  - Logic programming

Some Notes from the Past

- Sound or truth-preserving:
  - If KB is true in the real world, then any sentence α derived from KB by a sound inference procedure is also true in the real world
  - Whenever $KB \vdash_i \alpha$ it is also true that $KB \vDash \alpha$, i.e., $M(KB) \subseteq M(\alpha)$
- Completeness:
  - The inference algorithm can derive any sentence that is entailed
  - Whenever $KB \vDash \alpha$, it is also true that $KB \vdash_i \alpha$
- Horn clause is a clause (disjunct of literals) with at most one positive literal
  - $(A \lor \neg B) \land (\neg A \lor \neg C \lor D)$; which is equivalent to $(B \Rightarrow A) \land ((A \land C) \Rightarrow D)$

Completeness of 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?

- Problem is that ¬P(x) ⇒ R(x) cannot be converted into Horn form
- We need a more powerful inference rule!

A Brief History of Reasoning

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

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:

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

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

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"

$$\forall\, x\ \ Person(x)\ \Rightarrow\ \exists\, y\ \ Heart(y) \land Has(x, y)$$

Incorrect:

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

Correct:

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

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

Skolem function arguments: all enclosing universally quantified variables

Resolution Proof

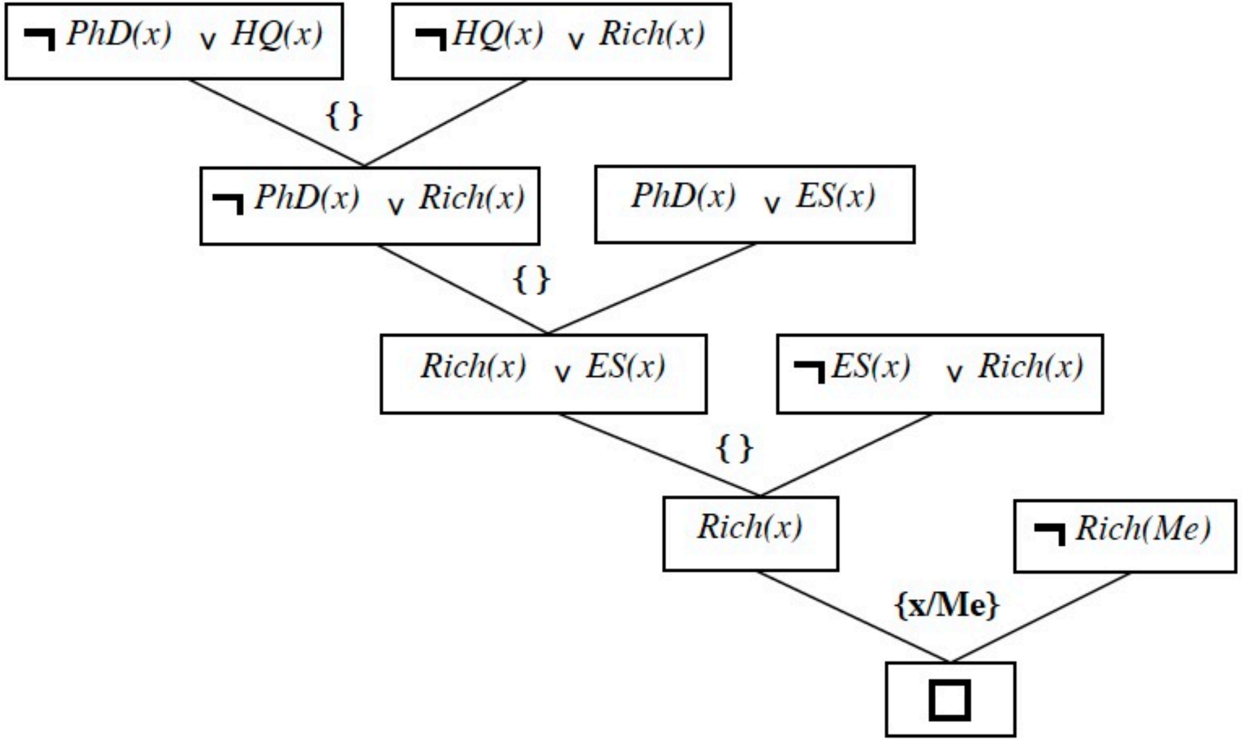
To prove α:

- 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



Logic Programming

Sound bite: computation as inference on logical KBs

	Logic programming	Ordinary programming
1.	Identify problem	Identify problem
2.	Assemble information	Assemble information
3.	Tea break	Figure out solution
4.	Encode information in KB	Program solution
5.	Encode problem instance as facts	Encode problem instance as data
6.	Ask queries	Apply program to data
7.	Find false facts	Debug procedural errors

Should be easier to debug $Capital(NewYork, US)$ than $x := x + 2$!

PROLOG

Basis: backward chaining with Horn clauses + bells & whistles
Widely used in Europe, Japan (basis of 5th Generation project)
Compilation techniques $\Rightarrow$ 10 million LIPS

Program = set of clauses = `head :- literal`$_1$`, ... literal`$_n$`.`
Efficient unification by open coding
Efficient retrieval of matching clauses by direct linking
Depth-first, left-to-right backward chaining
Built-in predicates for arithmetic etc., e.g., `X is Y*Z+3`
Closed-world assumption ("negation as failure")
e.g., `not PhD(X)` succeeds if `PhD(X)` fails