

FIRST-ORDER LOGIC

CHAPTER 7

CS 580, Jana Kosecka, Chapter 7 1

Syntax of FOL: Basic elements

Constants *KingJohn, 2, UCB, ...*

Predicates *Brother, >, ...*

Functions *Sqrt, LeftLegOf, ...*

Variables *x, y, a, b, ...*

Connectives $\wedge \vee \neg \Rightarrow \Leftrightarrow$

Equality =

Quantifiers $\forall \exists$

Complex sentences are made from atomic sentences using connectives

$$\neg S, \quad S_1 \wedge S_2, \quad S_1 \vee S_2, \quad S_1 \Rightarrow S_2, \quad S_1 \Leftrightarrow S_2$$

E.g. $Sibling(KingJohn, Richard) \Rightarrow Sibling(Richard, KingJohn)$

$$>(1, 2) \vee \leq(1, 2)$$

$$>(1, 2) \wedge \neg >(1, 2)$$

E.g., $Brother(KingJohn, RichardTheLionheart)$

$$>(Length(LeftLegOf(Richard)), Length(LeftLegOf(KingJohn)))$$

CS 580, Jana Kosecka, Chapter 7 2

Truth in first-order logic

Sentences are true with respect to a model and an interpretation

Model contains objects and relations among them Interpretation specifies referents for

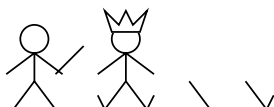
constant symbols → objects

predicate symbols → relations

function symbols → functional relations

An atomic sentence $predicate(term_1, \dots, term_n)$ is true iff the objects referred to by $term_1, \dots, term_n$ are in the relation referred to by $predicate$

Models for FOL: Example

objects 

relations: sets of tuples of objects

$\{ \langle \text{stick figure}, \text{crowned stick figure} \rangle, \langle \text{crowned stick figure}, \text{stick figure} \rangle, \dots \}$

functional relations: all tuples of objects + "value" object

$\{ \langle \text{stick figure}, \checkmark \rangle, \langle \text{crowned stick figure}, \checkmark \rangle, \dots \}$

Universal quantification

$\forall \langle \text{variables} \rangle \langle \text{sentence} \rangle$

Everyone at Berkeley is smart:

$\forall x \text{ At}(x, \text{Berkeley}) \Rightarrow \text{Smart}(x)$

$\forall x P$ is equivalent to the conjunction of instantiations of P

$\text{At}(\text{KingJohn}, \text{Berkeley}) \Rightarrow \text{Smart}(\text{KingJohn})$
 $\wedge \text{At}(\text{Richard}, \text{Berkeley}) \Rightarrow \text{Smart}(\text{Richard})$
 $\wedge \text{At}(\text{Berkeley}, \text{Berkeley}) \Rightarrow \text{Smart}(\text{Berkeley})$
 $\wedge \dots$

Typically, \Rightarrow is the main connective with \forall .

Common mistake: using \wedge as the main connective with \forall :

$\forall x \text{ At}(x, \text{Berkeley}) \wedge \text{Smart}(x)$

means “Everyone is at Berkeley and everyone is smart”

Existential quantification

$\exists \langle \text{variables} \rangle \langle \text{sentence} \rangle$

Someone at Stanford is smart:

$\exists x \text{ At}(x, \text{Stanford}) \wedge \text{Smart}(x)$

$\exists x P$ is equivalent to the disjunction of instantiations of P

$\text{At}(\text{KingJohn}, \text{Stanford}) \wedge \text{Smart}(\text{KingJohn})$
 $\vee \text{At}(\text{Richard}, \text{Stanford}) \wedge \text{Smart}(\text{Richard})$
 $\vee \text{At}(\text{Stanford}, \text{Stanford}) \wedge \text{Smart}(\text{Stanford})$
 $\vee \dots$

Typically, \wedge is the main connective with \exists .

Common mistake: using \Rightarrow as the main connective with \exists :

$\exists x \text{ At}(x, \text{Stanford}) \Rightarrow \text{Smart}(x)$

is true if there is anyone who is not at Stanford!

Properties of quantifiers

$\forall x \forall y$ is the same as $\forall y \forall x$ (why??)

$\exists x \exists y$ is the same as $\exists y \exists x$ (why??)

$\exists x \forall y$ is not the same as $\forall y \exists x$

$\exists x \forall y \text{ Loves}(x, y)$

“There is a person who loves everyone in the world”

$\forall y \exists x \text{ Loves}(x, y)$

“Everyone in the world is loved by at least one person”

Quantifier duality: each can be expressed using the other

$\forall x \text{ Likes}(x, \text{IceCream}) \quad \neg \exists x \neg \text{Likes}(x, \text{IceCream})$

$\exists x \text{ Likes}(x, \text{Broccoli}) \quad \neg \forall x \neg \text{Likes}(x, \text{Broccoli})$

◇ Fun with sentences

Equality

$term_1 = term_2$ is true under a given interpretation

if and only if $term_1$ and $term_2$ refer to the same object

E.g., $1 = 2$ and $\forall x \times(\text{Sqrt}(x), \text{Sqrt}(x)) = x$ are satisfiable
 $2 = 2$ is valid

E.g., definition of (full) *Sibling* in terms of *Parent*:

$$\forall x, y \text{ Sibling}(x, y) \Leftrightarrow [\neg(x = y) \wedge \exists m, f \neg(m = f) \wedge \text{Parent}(m, x) \wedge \text{Parent}(f, x) \wedge \text{Parent}(m, y) \wedge \text{Parent}(f, y)]$$

Interacting with FOL KBs

Suppose a wumpus-world agent is using an FOL KB and perceives a smell and a breeze (but no glitter) at $t = 5$:

$TELL(KB, Percept([Smell, Breeze, None], 5))$

$ASK(KB, \exists a \text{ Action}(a, 5))$

I.e., does the KB entail any particular actions at $t = 5$?

Answer: *Yes*, $\{a/Shoot\}$ ← substitution (binding list)

Given a sentence S and a substitution σ ,

$S\sigma$ denotes the result of plugging σ into S ; e.g.,

$S = Smarter(x, y)$

$\sigma = \{x/Hillary, y/Bill\}$

$S\sigma = Smarter(Hillary, Bill)$

$ASK(KB, S)$ returns some/all σ such that $KB \models S\sigma$

Knowledge base for the wumpus world

“Perception”

$\forall b, g, t \text{ Percept}([Smell, b, g], t) \Rightarrow Smelt(t)$

$\forall s, b, t \text{ Percept}([s, b, Glitter], t) \Rightarrow AtGold(t)$

Reflex: $\forall t \text{ AtGold}(t) \Rightarrow \text{Action}(Grab, t)$

Reflex with internal state: do we have the gold already?

$\forall t \text{ AtGold}(t) \wedge \neg Holding(Gold, t) \Rightarrow \text{Action}(Grab, t)$

$Holding(Gold, t)$ cannot be observed

⇒ keeping track of change is essential

Deducing hidden properties

Properties of locations:

$$\forall l, t \text{ At}(\text{Agent}, l, t) \wedge \text{Smelt}(t) \Rightarrow \text{Smelly}(l)$$

$$\forall l, t \text{ At}(\text{Agent}, l, t) \wedge \text{Breeze}(t) \Rightarrow \text{Breezy}(l)$$

Squares are breezy near a pit:

Diagnostic rule—infer cause from effect

$$\forall y \text{ Breezy}(y) \Rightarrow \exists x \text{ Pit}(x) \wedge \text{Adjacent}(x, y)$$

Causal rule—infer effect from cause

$$\forall x, y \text{ Pit}(x) \wedge \text{Adjacent}(x, y) \Rightarrow \text{Breezy}(y)$$

Neither of these is complete—e.g., the causal rule doesn't say whether squares far away from pits can be breezy

Definition for the *Breezy* predicate:

$$\forall y \text{ Breezy}(y) \Leftrightarrow [\exists x \text{ Pit}(x) \wedge \text{Adjacent}(x, y)]$$

Keeping track of change

Facts hold in situations, rather than eternally

E.g., *Holding(Gold, Now)* rather than just *Holding(Gold)*

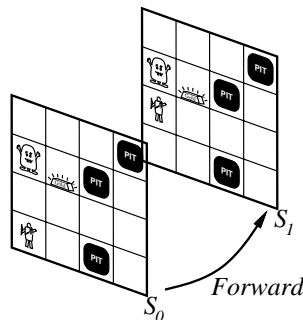
Situation calculus is one way to represent change in FOL:

Adds a situation argument to each non-eternal predicate

E.g., *Now* in *Holding(Gold, Now)* denotes a situation

Situations are connected by the *Result* function

Result(a, s) is the situation that results from doing *a* in *s*



Describing actions I

“Effect” axiom—describe changes due to action

$$\forall s \text{ AtGold}(s) \Rightarrow \text{Holding}(\text{Gold}, \text{Result}(\text{Grab}, s))$$

“Frame” axiom—describe non-changes due to action

$$\forall s \text{ HaveArrow}(s) \Rightarrow \text{HaveArrow}(\text{Result}(\text{Grab}, s))$$

Frame problem: find an elegant way to handle non-change

(a) representation—avoid frame axioms

(b) inference—avoid repeated “copy-overs” to keep track of state

Qualification problem: true descriptions of real actions require endless caveats—
what if gold is slippery or nailed down or . . .

Ramification problem: real actions have many secondary consequences—
what about the dust on the gold, wear and tear on gloves, . . .

Describing actions II

Successor-state axioms solve the representational frame problem

Each axiom is “about” a predicate (not an action per se):

$$\begin{aligned} \text{P true afterwards} &\Leftrightarrow [\text{an action made P true} \\ &\vee \text{P true already and no action made P false}] \end{aligned}$$

For holding the gold:

$$\begin{aligned} \forall a, s \text{ Holding}(\text{Gold}, \text{Result}(a, s)) &\Leftrightarrow \\ &[(a = \text{Grab} \wedge \text{AtGold}(s)) \\ &\vee (\text{Holding}(\text{Gold}, s) \wedge a \neq \text{Release})] \end{aligned}$$

Making plans

Initial condition in KB:

$At(Agent, [1, 1], S_0)$

$At(Gold, [1, 2], S_0)$

Query: $ASK(KB, \exists s \text{ Holding}(Gold, s))$

i.e., in what situation will I be holding the gold?

Answer: $\{s / \text{Result}(Grab, \text{Result}(Forward, S_0))\}$

i.e., go forward and then grab the gold

This assumes that the agent is interested in plans starting at S_0 and that S_0 is the only situation described in the KB

Making plans: A better way

Represent plans as action sequences $[a_1, a_2, \dots, a_n]$

$PlanResult(p, s)$ is the result of executing p in s

Then the query $ASK(KB, \exists p \text{ Holding}(Gold, PlanResult(p, S_0)))$

has the solution $\{p / [Forward, Grab]\}$

Definition of $PlanResult$ in terms of $Result$:

$\forall s \text{ PlanResult}([], s) = s$

$\forall a, p, s \text{ PlanResult}([a|p], s) = \text{Result}(a, s)$

Planning systems are special-purpose reasoners designed to do this type of inference more efficiently than a general-purpose reasoner

Summary

First-order logic:

- objects and relations are semantic primitives
- syntax: constants, functions, predicates, equality, quantifiers

Increased expressive power: sufficient to define wumpus world

Situation calculus:

- conventions for describing actions and change in FOL
- can formulate planning as inference on a situation calculus KB

Inference in first-order logic

- ◇ Chapter 9, Sections 1–4
- ◇ Proofs
- ◇ Unification
- ◇ Generalized Modus Ponens
- ◇ Forward and backward chaining

Proofs

Sound inference: find α such that $KB \models \alpha$.

Proof process is a search, operators are inference rules.

E.g., Modus Ponens (MP)

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

E.g., And-Introduction (AI)

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

E.g., Universal Elimination (UE)

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

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

Example proof

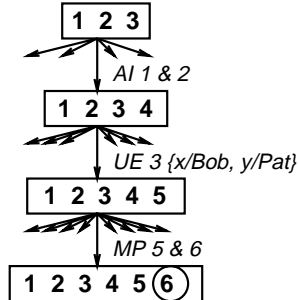
Bob is a buffalo	1. <i>Buffalo</i> (Bob)
Pat is a pig	2. <i>Pig</i> (Pat)
Buffaloes outrun pigs	3. $\forall x, y \text{ Buffalo}(x) \wedge \text{Pig}(y) \Rightarrow \text{Faster}(x, y)$
Bob outruns Pat	

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

⇒ a single, more powerful inference rule

Unification

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

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

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 $Likes(John, Jane)$

$Likes(John, OJ)$

$Likes(John, Mother(John))$

Generalized Modus Ponens (GMP)

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

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

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

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

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

$q\sigma = \text{Faster}(\text{Bob}, \text{Steve})$

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

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

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

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

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

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; \checkmark 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, \checkmark] \rightarrow 6. $Faster(Bob, Pat)$ [3a, \times], [3b, \times]
[2a, \times]
7. $Slug(Steve)$ [2b, \checkmark]
 \rightarrow 8. $Faster(Pat, Steve)$ [3a, \times], [3b, \checkmark]
 \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 any solution, find all solutions

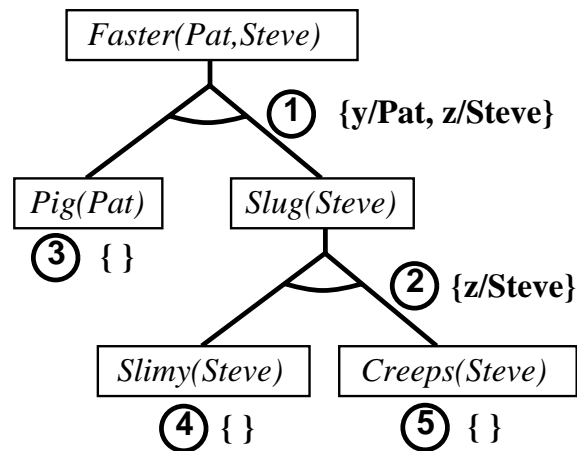
Backward chaining is the basis for logic programming, e.g., Prolog

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. $Creeps(Steve)$



Industrial-strength inference

Chapter 9.5–6, Chapters 8.1 and 10.2–3

- ◇ Completeness
- ◇ Resolution
- ◇ Logic programming

Completeness in FOL

Procedure i is complete if and only if

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

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?

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

CS 580, Jana Kosecka, Chapter 7 31

Resolution

Entailment in first-order logic is only semidecidable:

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

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

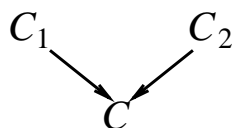
Cf. Halting Problem: proof procedure may be about to terminate with success or failure, or may go on for ever

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)

CS 580, Jana Kosecka, Chapter 7 32

Resolution inference rule

Basic propositional version:

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

Full first-order version:

$$\frac{p_1 \vee \dots \vee p_j \dots \vee p_m, \quad q_1 \vee \dots \vee q_k \dots \vee q_n}{(p_1 \vee \dots \vee p_{j-1} \vee p_{j+1} \dots \vee p_m \vee q_1 \dots \vee q_{k-1} \vee q_{k+1} \dots \vee q_n)\sigma}$$

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

For example,

$$\frac{\neg Rich(x) \vee Unhappy(x), \quad Rich(Me)}{Unhappy(Me)}$$

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 $(P \vee R) \wedge (Q \vee R)$

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

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

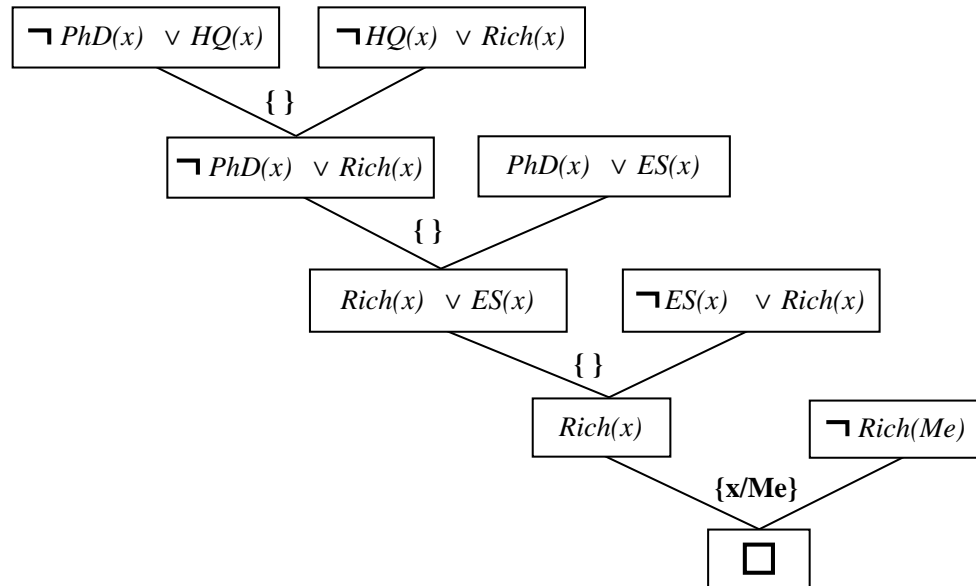
$\neg PhD(x) \vee HighlyQualified(x)$

$PhD(x) \vee EarlyEarnings(x)$

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

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

Resolution proof



CS 580, Jana Kosecka, Chapter 7 37

Logic programming

Sound bite: computation as inference on logical KBs

<u>Logic programming</u>	<u>Ordinary programming</u>
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 $x := x + 2 !$

CS 580, Jana Kosecka, Chapter 7 38

Prolog systems

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 `:- literal1, ... literaln.`
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

Prolog examples

Depth-first search from a start state X:

```
dfs(X) :- goal(X).  
dfs(X) :- successor(X,S),dfs(S).
```

No need to loop over S: successor succeeds for each

Appending two lists to produce a third:

```
append([],Y,Y).  
append([X|L],Y,[X|Z]) :- append(L,Y,Z).
```

```
query:   append(A,B,[1,2]) ?  
answers: A=[]      B=[1,2]  
         A=[1]     B=[2]  
         A=[1,2]   B=[]
```