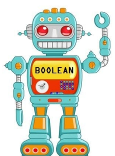
First-Order Logic

Bang Liu, Jian-Yun Nie

IFT3335: Introduction to Artificial Intelligence

Readings: AIMA 8.2∼8.3; 9.1∼9.5.



(Slides are adapted from **Tian-Li Yu**, National Taiwan University, Artificial Intelligence - Search & Logic)

Outline

1. First-Order Logic
   - Syntax and semantics
   - Using FOL
2. Inference
   - Instantiation
   - Propositionalization
   - Unification
   - Forward chaining
   - Backward chaining
   - Resolution

First-Order Logic (FOL)

Objects: People, houses, numbers, theories, Ronald McDonald, colors, baseball games, wars, wumpus ...

Relations: red, round, bogus, prime ..., brother of, bigger than, inside, part of, has color, occurred after, owns, comes between ...

Functions: father of, best friend, one more than, end of ...

- "One plus one equals two"

  Objects: one, one plus one, two; Relation: equals; Function: plus.
- "Squares neighboring the wumpus are smelly"

  Objects: wumpus, squares; Relation: smelly; Functions: neighbor of.
- One may, of course, use relations to express functions.

First-Order Logic (FOL)

Objects: e.g., in natural language, nouns and noun phrases

Relations: relations among objects. They can be unary relations or properties, or general n-ary relations

Functions: relations in which there is only one "value" for a given "input"

Language	Ontological Commitment (What exists in the world)	Epistemological Commitment (What an agent believes about facts)
Propositional logic	facts	true/false/unknown
First-order logic	facts, objects, relations	true/false/unknown
Temporal logic	facts, objects, relations, times	true/false/unknown
Probability theory	facts	degree of belief $\in [0, 1]$
Fuzzy logic	facts with degree of truth $\in [0, 1]$	known interval value

Figure 8.1 Formal languages and their ontological and epistemological commitments.

Model for First-Order Logic (FOL)

Model in FOL:

1. They have **objects** in them
2. The **domain** of a model is the set of objects or domain elements it contains.
3. The domain is required to be **nonempty**—every possible world must contain at least one object.
4. The objects in the model may be **related** in various ways.

First-Order Logic (FOL)

Model for First-Order Logic (FOL)

Recap: in propositional logic, a model contains facts and the true/false assertions.
In FOL, we have objects, relations/predicates, functions.

crown
on head
brother
brother
person
person
king
R
J
left leg
left leg

Figure 8.2 A model containing five objects, two binary relations (brother and on-head), three unary relations (person, king, and crown), and one unary function (left-leg).

Symbols and interpretations

The basic syntactic elements of first-order logic are the **symbols** that stand for **objects**, **relations**, and **functions**.

Constant symbols: stands for objects

Predicate symbols: stands for relations

Function symbols: stands for functions

Each model includes an **interpretation** that specifies exactly which objects, relations and functions are referred to by the constant, predicate, and function symbols.

Symbols and interpretations

In summary, a **model** in **first-order logic** consists of a set of **objects** and an **interpretation** that maps constant symbols to objects, function symbols to functions on those objects, and predicate symbols to relations.

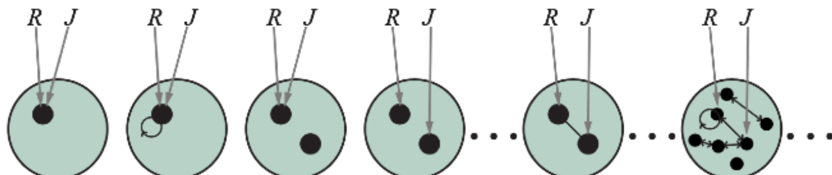


Figure 8.4 Some members of the set of all models for a language with two constant symbols, R and J, and one binary relation symbol. The interpretation of each constant symbol is shown by a gray arrow. Within each model, the related objects are connected by arrows.

Syntax of FOL

The basic syntactic elements of first-order logic are the symbols that stand for objects, relations, and functions.

Constants	A, 2, NTU, $John$, $\ldots$	stands for objects
Variables	a, b, x, y, $\ldots$	stands for ungrounded objects, can be the argument of a function
Functions	$Mother$, $LeftLeg$, $\ldots$	
Predicates	$After$, $Loves$, $\ldots$	stands for relations
Connectives	$\wedge$, $\vee$, $\neg$, $\Rightarrow$, $\Leftarrow$, $\Leftrightarrow$.	
Equality	$=$, ($\neq$ for $\neg =$).	
Quantifiers	$\forall$, $\exists$.	

- Check Figure 8.3 in AIMA for more details.

Syntax of FOL

$$
\begin{aligned}
\textit{Sentence} &\rightarrow \textit{AtomicSentence} \mid \textit{ComplexSentence} \\
\textit{AtomicSentence} &\rightarrow \textit{Predicate} \mid \textit{Predicate}(\textit{Term},\ldots) \mid \textit{Term} = \textit{Term} \\
\textit{ComplexSentence} &\rightarrow (\,\textit{Sentence}\,) \\
&\mid\ \neg\,\textit{Sentence} \\
&\mid\ \textit{Sentence} \land \textit{Sentence} \\
&\mid\ \textit{Sentence} \lor \textit{Sentence} \\
&\mid\ \textit{Sentence} \Rightarrow \textit{Sentence} \\
&\mid\ \textit{Sentence} \Leftrightarrow \textit{Sentence} \\
&\mid\ \textit{Quantifier Variable},\ldots\ \textit{Sentence} \\
\\
\textit{Term} &\rightarrow \textit{Function}(\textit{Term},\ldots) \\
&\mid\ \textit{Constant} \\
&\mid\ \textit{Variable} \\
\\
\textit{Quantifier} &\rightarrow \forall \mid \exists \\
\textit{Constant} &\rightarrow A \mid X_1 \mid \textit{John} \mid \cdots \\
\textit{Variable} &\rightarrow a \mid x \mid s \mid \cdots \\
\textit{Predicate} &\rightarrow \textit{True} \mid \textit{False} \mid \textit{After} \mid \textit{Loves} \mid \textit{Raining} \mid \cdots \\
\textit{Function} &\rightarrow \textit{Mother} \mid \textit{LeftLeg} \mid \cdots \\
\textsc{Operator Precedence} &: \neg, =, \land, \lor, \Rightarrow, \Leftrightarrow
\end{aligned}
$$

Figure 8.3 The syntax of first-order logic with equality, specified in Backus–Naur form (see page 1030 if you are not familiar with this notation). Operator precedences are specified, from highest to lowest. The precedence of quantifiers is such that a quantifier holds over everything to the right of it.

An atomic sentence (or atom for short) is formed from a predicate symbol optionally followed by a parenthesized list of terms

Once we have a logic that allows objects, it is only natural to want to express **properties of entire collections of objects**, instead of enumerating the objects by name. Quantifiers let us do this.

Quantifiers

- Typically, $\Rightarrow$ is the main connective with $\forall$.
- Typically, $\wedge$ is the main connective with $\exists$.

- Be careful
  - $\forall x\ In(x, \text{UdeM}) \Rightarrow Smart(x)$: Everyone in UdeM is smart.
  - $\forall x\ In(x, \text{UdeM}) \wedge Smart(x)$: Everyone is in UdeM and everyone is smart.
  - $\exists x\ In(x, \text{UdeM}) \Rightarrow Smart(x)$: This is TRUE when no one is in UdeM !
  - $\exists x\ In(x, \text{UdeM}) \wedge Smart(x)$: Someone in UdeM is smart.

Properties of Quantifiers

- $\forall x \forall y$ is equivalent to $\forall y \forall x$.
- $\exists x \exists y$ is equivalent to $\exists y \exists x$.
- $\exists x \forall y$ is NOT equivalent to $\forall y \exists x$.
  - $\exists x \forall y\ Loves(x, y)$
    There exists someone who loves everyone.
  - $\forall y \exists x\ Loves(x, y)$
    Everyone is loved by at least one person.
- Quantifier duality
  - $\forall x\ Likes(x, IceCream)$ is equivalent to $\neg \exists x\ \neg Likes(x, IceCream)$.
  - $\exists x\ Likes(x, Studying)$ is equivalent to $\neg \forall x\ \neg Likes(x, Studying)$.

Expressing with FOL

- John has two brothers, Mark and David.

  How to express the above meaning with FOL?

Expressing with FOL

- John has two brothers, Mark and David.

  $Brother(John, Mark) \wedge Brother(John, David)$?

- It only says Mark and David are John's brothers. We need to make sure John has no other brothers.

  $Brother(John, Mark) \wedge Brother(John, David) \wedge$
  $(\forall x\ Brother(John, x) \Rightarrow (x = Mark \vee x = Davide))$?

- John might have only one brother with two names....orz

  $Brother(John, Mark) \wedge Brother(John, David) \wedge$
  $(\forall x\ Brother(John, x) \Rightarrow (x = Mark \vee x = Davide)) \wedge (Mark \neq David)$

Database Semantics

One proposal that is very popular in database systems works as follows:

1. Unique-names assumption — every constant symbol refer to a distinct object.
2. Closed-world assumption — atomic sentences not known to be true are in fact false.
3. Domain closure — each model contains no more domain elements than those named by the constant symbols.

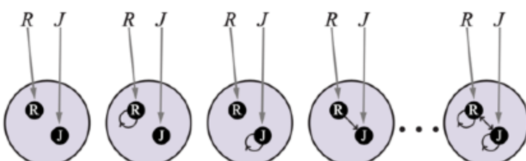
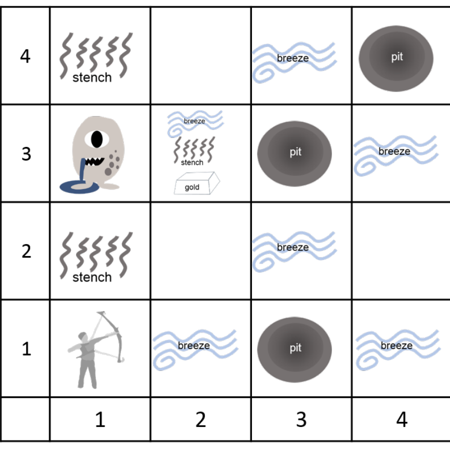


Figure 8.5 Some members of the set of all models for a language with two constant symbols, R and J, and one binary relation symbol, under database semantics. The interpretation of the constant symbols is fixed, and there is a distinct object for each constant symbol.

Wumpus World



Back to the Wumpus World, Again

Bump (Walks into a wall)
Scream (wumps is killed)

- $\text{PERCEPT}([\text{STENCH}, \text{BREEZE}, \text{GLITTER}, \text{NONE}, \text{NONE}], 5)$, where 5 is the step number.
- Actions: $\text{TURN}(\text{RIGHT})$, $\text{TURN}(\text{LEFT})$, FORWARD, SHOOT, GRAB, CLIMB.
- Interaction with *KB*: $\text{ASKVARS}(\exists a\ BestAction(a, 5))$

  Returns a substitution (binding list) $\{a/\text{GRAB}\}$.
- Define raw percept data:

  $\forall t, s, g, m, c\ \text{PERCEPT}([s, \text{BREEZE}, g, m, c], t) \Rightarrow Breeze(t).$

  $\forall t, s, b, m, c\ \text{PERCEPT}([s, b, \text{GLITTER}, m, c], t) \Rightarrow Glitter(t).$
- Simple reflex best action:

  $\forall t\ Glitter(t) \Rightarrow BestAction(\text{GRAB}, t).$

Back to the Wumpus World, Again

- Define adjacency:

  $\forall x, y, a, b\ Adjacent([x, y], [a, b]) \Leftrightarrow$
  $(x = a \land (y = b-1 \lor y = b+1)) \lor (y = b \land (x = a-1 \lor x = a+1))$.

- Location predicator, x is at square s at time t:

  $\forall t\ At(\text{WUMPUS}, [2, 2], t)$. Fix Wumpus at a specific location

  $\forall x, s_1, s_2, t\ At(x, s_1, t) \land At(x, s_2, t) \Rightarrow s_1 = s_2$. can be at one location at a time

- Define property for squares:

  $\forall s, t\ At(\text{AGENT}, s, t) \land Breeze(t) \Rightarrow Breezy(s)$.

  $\forall s, t\ At(\text{PIT}, s, t) \Rightarrow Pit(s)$.

  $\forall s, t\ At(\text{WUMPUS}, s, t) \Rightarrow Wumpus(s)$.

- Rules of the wumpus world can be defined.

  $\forall s\ Breezy(s) \Leftrightarrow \exists r\ Adjacent(r, s) \land Pit(r)$.

  $\forall t\ HaveArrow(t+1) \Leftrightarrow (HaveArrow(t) \land \neg Action(Shoot, t))$.

Instantiation

- $\textsc{Subst}(\theta, \alpha)$: apply the substitution θ to the sentence α.

Universal Instantiation(UI)

$$\frac{\forall v\ \alpha}{\textsc{Subst}(\{v/g\}, \alpha)},$$

where g is a ground term.

Existence Instantiation(EI)

$$\frac{\exists v\ \alpha}{\textsc{Subst}(\{v/k\}, \alpha)},$$

where k was not in the KB.

- $\forall King(x) \wedge Greedy(x) \Rightarrow Evil(x)$ yields
  $King(John) \wedge Greedy(John) \Rightarrow Evil(John)$
  $King(Father(Tom)) \wedge Greedy(Father(Tom)) \Rightarrow Evil(Father(Tom))$
- $\exists x\ Crown(x) \wedge OnHead(x, John)$ yields
  $Crown(C_1) \wedge OnHead(C_1, John)$, where C_1 is a Skolem constant.

Applying the EI rule just gives a name to some object, which can't already belong to another object. So it cannot in the KB.

Instantiation

- UI can be applied several times to add new sentences; the new KB is logically equivalent to the old.

- EI can be applied only once to replace the existential sentence.
- No longer need $\exists x\ Kill(x, Victim)$ once we have $Kill(Murderer, Victim)$.
- Strictly speaking, the new KB is **not** logically equivalent to the old.
- However, the new KB is satisfiable **iff** the old was satisfiable.
- We call them inferentially equivalent.

Reduction to Propositional Inference

- Suppose the KB contains only the following:
  $\forall x \; King(x) \land Greedy(x) \Rightarrow Evil(x)$
  $King(John)$
  $Greedy(John)$
  $Brother(Richard, John)$
- Instantiating the universal sentence in all possible ways, we have
  $King(John) \land Greedy(John) \Rightarrow Evil(John)$
  $King(Richard) \land Greedy(Richard) \Rightarrow Evil(Richard)$
  $King(John)$
  $Greedy(John)$
  $Brother(Richard, John)$
- The new KB is propositionalized.

Reduction to Propositional Inference

- A ground sentence is entailed by new KB **iff** entailed by original KB.
- Every FOL KB can be propositionalized so as to preserve entailment.
- Propositionalize KB and query, apply resolution, return result.
- Problem: with function symbols, there are infinitely many ground terms, e.g., *Father*(*Father*(*Father*(*John*)))
- **Theorem:** Herbrand (1930). If a sentence is entailed by an FOL KB, it is entailed by a finite subset of the propositional KB.
- Idea: For $d = 0$ **to** ∞ **do**

  Think about Iterative deepening search

  create a propositional KB by instantiating with depth-d terms

  see if α is entailed by this KB.
- Problem: works if α is entailed, loops if α is not entailed.
- Theorem: Turing (1936), Church (1936), entailment in FOL is semidecidable.

semidecidable—that is, algorithms exist that say yes to every entailed sentence, but no algorithm exists that also says no to every nonentailed sentence.

Problems with Propositionalization

- Propositionalization generates lots of irrelevant sentences and can be inefficient.
  E.g., from the KB,

  $\forall x\ King(x) \wedge Greedy(x) \Rightarrow Evil(x)$
  $King(John)$
  $\forall y\ Greedy(y)$
  $Brother(Richard, John)$

  It seems obvious that query $Evil(x)$ yields $x = John$, but propositionalization produces lots of irrelevant facts such as $Greedy(Richard)$.

- With p k-ary predicates and n constants, there are $p \cdot n^k$ instantiations

- With function symbols, it gets much worse!

Generalized Modus Ponens (GMP)

GMP

$$\frac{p_1{}', p_2{}', \ldots, p_n{}',\ (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{\textsc{Subst}(\theta, q)},$$

where $\forall i\ \ \textsc{Subst}(\theta, p_i') = \textsc{Subst}(\theta, p_i)$

$p_1{}'$ is $King(John)$ $\quad$ p_1 is $King(x)$
$p_2{}'$ is $Greedy(y)$ $\quad$ p_2 is $Greedy(x)$
θ is $\{x/John, y/John\}$ $\quad$ q is $Evil(x)$
$\textsc{Subst}(\theta, q)$ is $Evil(John)$

- GMP used with KB of definite clauses (exactly one positive literal).
- All variables assumed universally quantified.

Soundness of GMP

- Need to show that

$$p_1', \ \ldots, \ p_n', \ (p_1 \wedge \ldots \wedge p_n \Rightarrow q) \models \textsc{Subst}(\theta, q)$$

provided that $\forall i \ \textsc{Subst}(\theta, p_i') = \textsc{Subst}(\theta, p_i)$

- Lemma: For any definite clause p, we have $p \models \textsc{Subst}(\theta, p)$ by UI.

Proof.

1. $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \models \textsc{Subst}(\theta, p_1 \wedge \ldots \wedge p_n \Rightarrow q) = \textsc{Subst}(\theta, p_1) \wedge \ldots \wedge \textsc{Subst}(\theta, p_n) \Rightarrow \textsc{Subst}(\theta, q)$
2. $p_1', \ \ldots, \ p_n' \models p_1' \wedge \ldots \wedge p_n' \models \textsc{Subst}(\theta, p_1') \wedge \ldots \wedge \textsc{Subst}(\theta, p_n')$
3. From 1 and 2, $\textsc{Subst}(\theta, q)$ follows by ordinary Modus Ponens.

□

Unification

- We can get the inference immediately if we can find a substitution θ such that $King(x)$ and $Greedy(x)$ match $King(John)$ and $Greedy(y)$

  $\theta = \{x/John, y/John\}$ works

- UNIFY takes two sentences and returns a unifier for them:
  $\text{UNIFY}(p, q) = \theta$ where $\text{SUBST}(\theta, p) = \text{SUBST}(\theta, q)$.

p	q	θ
$Knows(John, x)$	$Knows(John, Jane)$	$\{x/Jane\}$
$Knows(John, x)$	$Knows(y, Bill)$	$\{x/Bill, y/John\}$
$Knows(John, x)$	$Knows(y, Mother(y))$	$\{y/John, x/Mother(John)\}$
$Knows(John, x)$	$Knows(x, Eliza)$	*fail* (*why?*)

Unification

- Standardizing apart eliminates overlap of variables
  $\textsc{Unify}(Knows(John, x), Knows(x_{17}, Eliza)) = \{x/Eliza, x_{17}/John\}$

- $\textsc{Unify}$ returns the most general unifier (MGU) if there are several.
  E.g., $\textsc{Unify}(Knows(John, x), Knows(y, z)) =$
  - $\{y/John, x/z\}$ (MGU)
  - $\{y/John, x/John, z/John\}$

- To retrieve MGU, the algorithm recursively explore the expressions simultaneously. See Fig. 9.1 in AIMA
  Need to perform occur check so that $S(x)$ doesn't unify with $S(S(x))$.
  Occur check: when matching a variable against a complex term, one must check whether the variable itself occurs inside the term; if it does, the match fails because no consistent unifier can be constructed.

First-Order Definite Clauses

- The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.
- Prove that Colonel West is a criminal.
- For efficient inference, we use first-order definite clauses:
  - Exactly one positive literal.
  - May include variables (universally quantified).

Knowledge Base Using First-Order Definite Clauses

- "... it is a crime for an American to sell weapons to hostile nations":
  $American(x) \land Weapon(y) \land Sells(x, y, z) \land Hostile(z) \Rightarrow Criminal(x)$
- "Nono ... has some missiles", i.e., $\exists x\ Owns(Nono, x) \land Missile(x)$:
  $Owns(Nono, M_1)$ and $Missile(M_1)$
- "... all of its missiles were sold to it by Colonel West":
  $\forall x\ Missile(x) \land Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$
- Missiles are weapons:
  $Missile(x) \Rightarrow Weapon(x)$
- An enemy of America counts as "hostile":
  $Enemy(x, America) \Rightarrow Hostile(x)$
- "West, who is American ...":
  $American(West)$
- "The country Nono, an enemy of America ...":
  $Enemy(Nono, America)$

Forward Chaining Algorithm

FOL-FC-ASK(KB, α)

```
1   repeat until new is empty
2       new = {}
3       for each rule in KB
4           (p1 ∧ ... ∧ pn ⇒ q) = STANDARDIZE-VARIABLES(rule)
5           for each θ such that SUBST(θ, p1 ∧ ... ∧ pn) ==
                    SUBST(θ, p1' ∧ ... ∧ pn') for some p1', ..., pn' in KB
6               q' = SUBST(θ, q)
7               if q' does not unify with some sentence already in KB or new
8                   add q' to new
9                   φ = UNIFY(q', α)
10                  if φ is not fail
11                      return φ
12      add new to KB
13  return FALSE
```

Forward Chaining Proof

Facts

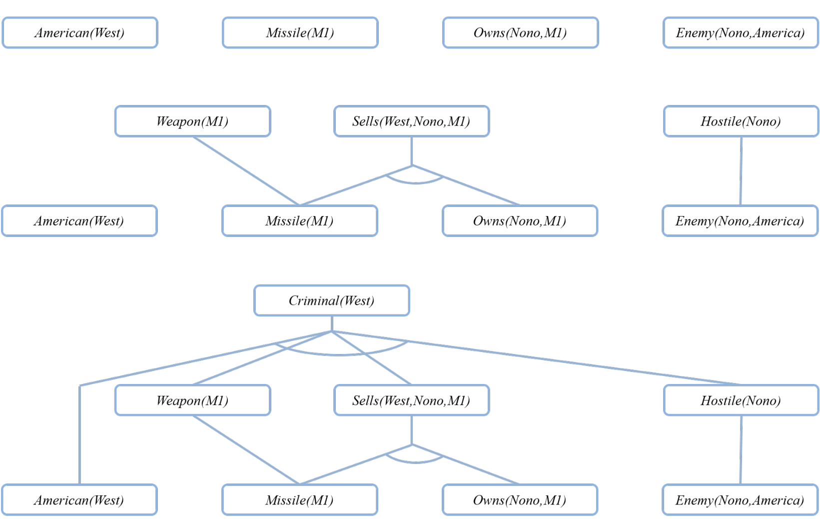
1. $Owns(Nono, M_1)$
2. $Missile(M_1)$
3. $American(West)$
4. $Enemy(Nono, America)$

Implications

5. $American(x) \land Weapon(y) \land Sells(x, y, z) \land Hostile(z) \Rightarrow Criminal(x)$
6. $\forall x\ Missile(x) \land Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$
7. $Missile(x) \Rightarrow Weapon(x)$
8. $Enemy(x, America) \Rightarrow Hostile(x)$

- 1st iteration, R_5 has unsatisfied premises.
  R_6 is satisfied with $\{x/M_1\}$, $Sells(West, M_1, Nono)$ is added.
  R_7 is satisfied with $\{x/M_1\}$, $Weapon(M_1)$ is added.
  R_8 is satisfied with $\{x/Nono\}$, $Hostile(Nono)$ is added.
- 2nd iteration, R_5 is satisfied with $\{x/West, y/M_1, z/Nono\}$, $Criminal(West)$ is added.

Forward Chaining Proof



Properties of Forward Chaining

- Sound and complete for first-order definite clauses.
  Sound because of generalized Modus Ponens.
  Complete proof similar to propositional proof by introducing the concept of fixed point.
- Datalog = first-order definite clauses + no functions (e.g., crime KB)
  FC terminates for Datalog in poly iterations: at most $p \cdot n^k$ distinct ground facts.
- May not terminate in general if α is not entailed.
- This is unavoidable: entailment with definite clauses is semi-decidable.

Efficiency of Forward Chaining

- Simple observation: no need to match a rule on iteration k if a premise wasn't added on iteration $k-1$.
  $\Rightarrow$ match each rule whose premise contains a newly added literal.

- Matching itself can be expensive.

- Database indexing allows $O(1)$ retrieval of known facts
  e.g., query $Missile(x)$ retrieves $Missile(M_1)$

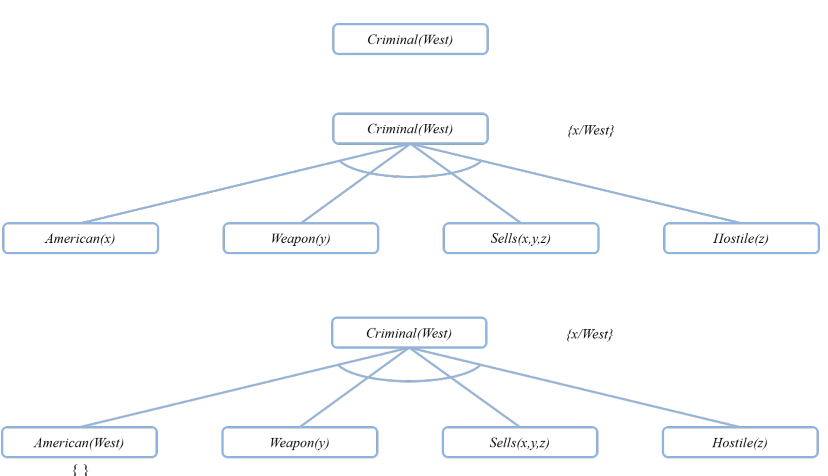
NP-hard: at least as hard as the hardest problems in NP

- Matching conjunctive premises against known facts is $\mathcal{NP}$-hard.
  e.g., $Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$
  Conjunction ordering problem: find an ordering to minimize the cost.
  Nevertheless, good heuristics are available.

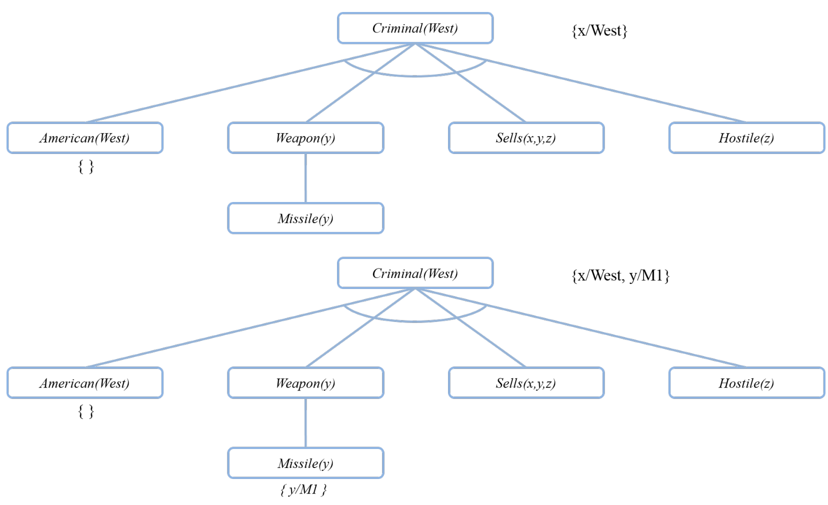
- Forward chaining is widely used in deductive databases

A deductive database is a database system that can make deductions based on rules and facts stored

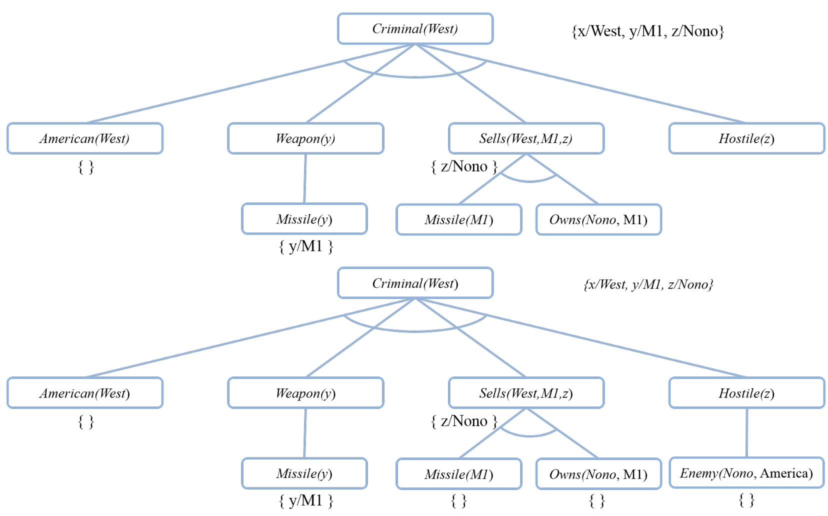
Backward Chaining Example



Backward Chaining Example



Backward Chaining Example



Properties of Backward Chaining

- Depth-first recursive proof search: space is linear in size of proof.
- AND-OR search: AND for all premises; OR since the goal query can be proved by any rules.
- Incomplete due to infinite loops
  $\Rightarrow$ fix by checking current goal against every goal on stack
- Inefficient due to repeated subgoals (both success and failure)
  $\Rightarrow$ fix using caching of previous results (extra space!)
- Widely used (without improvements!) for logic programming

Logic Programming

Logic programming is a technology that comes close to embodying the declarative ideal: that systems should be constructed by expressing knowledge in a formal language and that problems should be solved by running inference processes on that knowledge.
The ideal is summed up in Robert Kowalski's equation:

- $Algorithm = Logic + Control$

	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 as facts	Encode problem as data
6.	Ask queries	Apply program to data
7.	Find false facts	Debug procedural errors

See AIMA 8.4

Prolog Systems

Prolog is the most widely used logic programming language.
Many expert systems have been written in Prolog for legal, medical, financial, and other domains.
Prolog programs are sets of definite clauses written in a notation somewhat different from standard first-order logic:

- Lowercases for constants; uppercases for variables (opposite of the textbook).
- Program = set of definite clauses.
  $A \wedge B \Rightarrow C$ in Prolog is `C :- A, B.`
  `criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).`
- `[E|L]` is a list whose first element is `E` and rest is `L`.

Prolog Examples

- Depth-first search from a start state X:

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

- Appending two lists (X and Y) to produce a third (Z):

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

A Prolog program for append(X,Y,Z), which succeeds if list Z is the result of appending lists X and Y

query: `append(X,Y,[1,2]) ?`

answers: `X=[] Y=[1,2];`
`X=[1] Y=[2];`
`X=[1,2] Y=[]`

Prolog Systems

- Unification without the occur check, may results in unsound inferences. But almost never a problem in practice.

- Depth-first, left-to-right backward chaining search with no checks for infinite recursion.

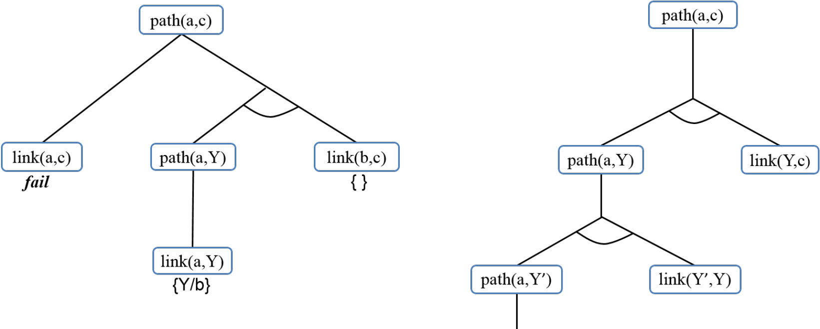
When matching a variable against a complex term, one must check whether the variable itself occurs inside the term. This so-called occur check makes the complexity of the entire algorithm quadratic in the size of the expressions being unified. Some systems, including many logic programming systems, simply omit the occur check and put the onus on the user to avoid making unsound inferences as a result

- Database semantics instead of first-order semantics.

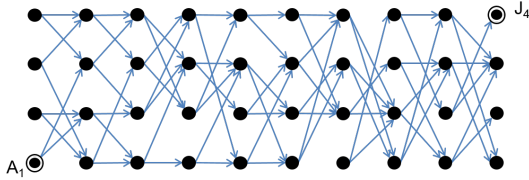
Redundant Inference and Infinite Loops in Prolog

```
path(X,Z) :- link(X,Z).
path(X,Z) :- path(X,Y), link(Y,Z).
```

Query: `path(a,c) ?`



Redundant Inference and Infinite Loops in Prolog



- Backward chaining (Prolog) takes 877 inferences.
- Forward chaining (similar to dynamic programming) takes only 62 inferences.
- To make backward chaining more efficient, memoization can be adopted, but extra memory is needed.

Database Semantics of Prolog

- Closed-world assumption — anything not known to be true is false.
- Unique-names assumption — different names refer to distinct objects.
- Domain closure — only those mentioned exist in the domain.

Prolog assertions:

```
Course(CS,101), Course(CS,102), Course(CS,106), Course(EE,101).
```

FOL:

at most 4 courses:

$$Course(d,n) \Leftrightarrow (d = CS \wedge n = 101) \vee (d = CS \wedge n = 102) \vee (d = CS \wedge n = 106) \vee (d = EE \wedge n = 101).$$

at least 4 courses:

$$x = y \Leftrightarrow (x = CS \wedge y = CS) \vee (x = EE \wedge y = EE) \vee (x = 101 \wedge y = 101) \vee (x = 102 \wedge y = 102) \vee (x = 106 \wedge n = 106).$$

Resolution

- Full first-order version:

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \qquad m_1 \vee \cdots \vee m_n}{\textsc{Subst}(\theta, \ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n)},$$

where $\textsc{Unify}(\ell_i, \neg m_j) = \theta$.

- For example,

$$\frac{\begin{array}{l}\neg Rich(x) \vee Unhappy(x) \\ Rich(Ken)\end{array}}{Unhappy(Ken)}$$

with $\theta = \{x/Ken\}$

- Apply resolution steps to $CNF(KB \wedge \neg\alpha)$; complete for FOL

Conversion to CNF

- Everyone who loves all animals is loved by someone:

$$\forall x\ [\forall y\ Animal(y) \Rightarrow Loves(x,y)] \Rightarrow [\exists y\ Loves(y,x)]$$

1. **Eliminate biconditionals and implications:**

$$\forall x\ [\neg\forall y\ \neg Animal(y) \lor Loves(x,y)] \lor [\exists y\ Loves(y,x)]$$

2. **Move $\neg$ inwards:** $\neg\forall x\ p \equiv \exists x\ \neg p, \quad \neg\exists x\ p \equiv \forall x\ \neg p$:

$$\forall x\ [\exists y\ \neg(\neg Animal(y) \lor Loves(x,y))] \lor [\exists y\ Loves(y,x)]$$

$$\forall x[\exists y\ \neg\neg Animal(y) \land \neg Loves(x,y)] \lor [\exists y\ Loves(y,x)]$$

$$\forall x[\exists y\ Animal(y) \land \neg Loves(x,y)] \lor [\exists y\ Loves(y,x)]$$

Conversion to CNF

3. **Standardize variables:** each quantifier should use a different one

$$\forall x\ [\exists y\ Animal(y) \wedge \neg Loves(x, y)] \vee [\exists z\ Loves(z, x)]$$

4. **Skolemize:** a more general form of existential instantiation.
   Each existential variable is replaced by a Skolem function
   of the enclosing universally quantified variables:

$$\forall x\ [Animal(F(x)) \wedge \neg Loves(x, F(x))] \vee Loves(G(x), x)$$

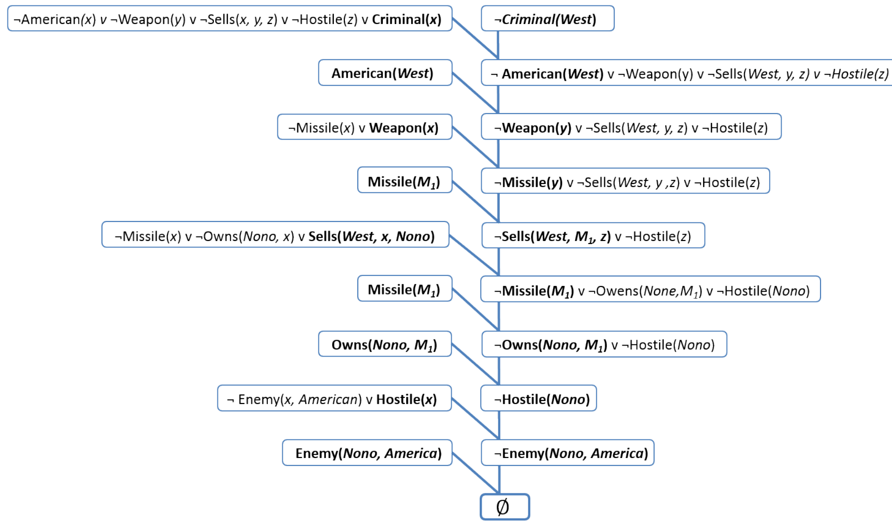
5. Drop universal quantifiers:

$$[Animal(F(x)) \ \wedge \neg Loves(x, F(x))] \vee Loves(G(x), x)$$

6. Distribute $\wedge$ over $\vee$:

$$[Animal(F(x)) \ \vee Loves(G(x), x)] \wedge [\neg Loves(x, F(x)) \vee Loves(G(x), x)]$$

Resolution Proof: Definite Clauses



Completeness of Resolution

- Resolution is refutation-complete. If a set of sentences is unsatisfiable, resolution always derives a contradiction.
- It cannot generate all logical consequences.
- It can find all answers of a given question, $Q(x)$, by proving that $KB \wedge \neg Q(x)$ is unsatisfiable.
- Check out AIMA for the (brief) proof:

> *if S is an unsatisfiable set of clauses, then the application of a finite number of resolution steps to S will yield a contradiction.*

Summary

- For small domains, we can use UI and EI to propositionalize the problem.
- Unification identifies proper substitutions, more efficient than instantiation.
- Forward- and backward-chaining uses the generalized Modus Ponens on a sets of definite clauses.
- GMP is complete for definite clauses, where the entailment is semi-decidable; for Datalog KB (function-less definite clauses), entailment can be decided in $\mathcal{P}$-time (with forward-chaining).
- Backward chaining is used in logic programming systems; inferences are fast but may be unsound or incomplete.
- Resolution is sound and (refutation-)complete for FOL, using CNF KB.