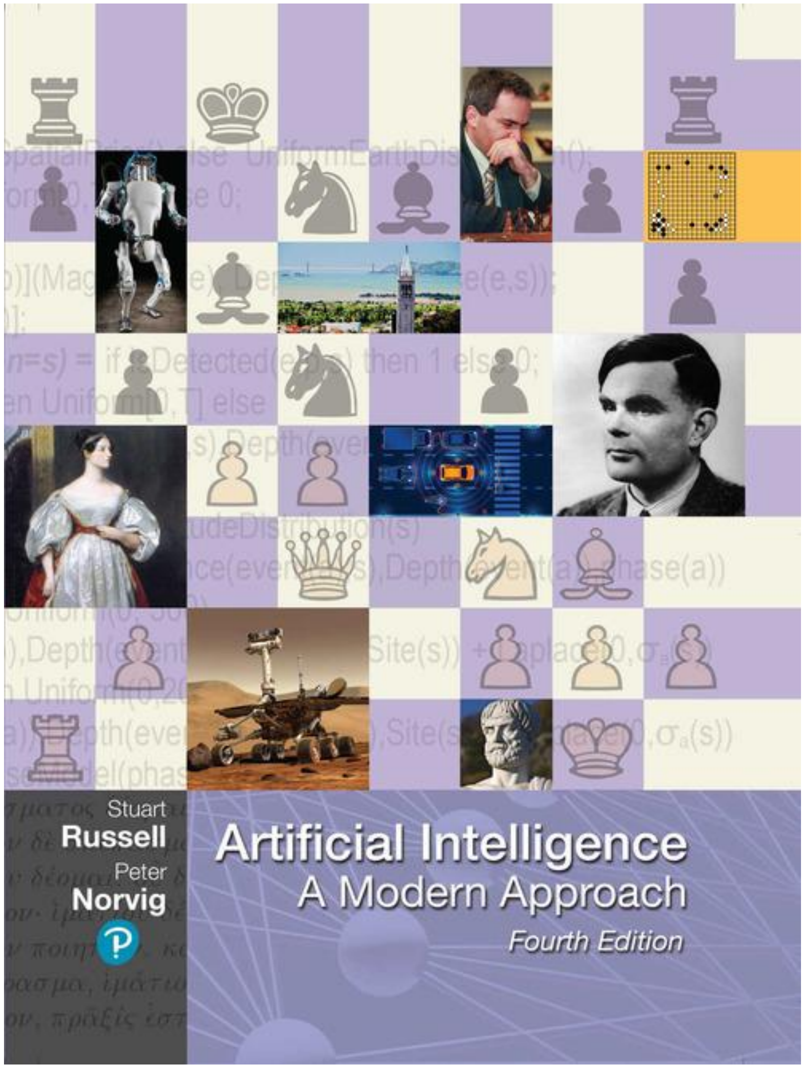
Artificial Intelligence Fundamentals

2024-2025



"The greatest progress that the human race has made lies in learning how to make correct inferences."

- Friedrich Nietzsche

AIMA Chapter 9

Inference in first-order logic

Outline

- Reducing first-order inference to propositional inference
- Unification
- Generalized Modus Ponens
- Forward and backward chaining
- Logic programming
- Resolution
- Real-world Knowledge Bases

Universal instantiation (UI)

Every instantiation of a universally quantified sentence is entailed by it:

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

for any variable v and ground term g

E.g., $\forall x \;\; King(x) \wedge Greedy(x) \;\Rightarrow\; Evil(x)$ yields

$$King(John) \wedge Greedy(John) \;\Rightarrow\; Evil(John)$$
$$King(Richard) \wedge Greedy(Richard) \;\Rightarrow\; Evil(Richard)$$
$$King(Father(John)) \wedge Greedy(Father(John)) \;\Rightarrow\; Evil(Father(John))$$

$$\vdots$$

Existential instantiation (EI)

For any sentence α, variable v, and constant symbol k that does not appear elsewhere in the knowledge base:

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

E.g., $\exists\, x \;\; Crown(x) \land OnHead(x, John)$ yields

$$Crown(C_1) \land OnHead(C_1, John)$$

provided C_1 is a new constant symbol, called a Skolem constant

Another example: from $\exists\, x \;\; d(x^y)/dy = x^y$ we obtain

$$d(e^y)/dy = e^y$$

provided e is a new constant symbol

Existential instantiation contd.

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

EI can be applied once to replace the existential sentence;
the new KB is not equivalent to the old,
but is satisfiable iff the old KB was satisfiable

Reduction to propositional inference

Suppose the KB contains just the following:

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

Instantiating the universal sentence in all possible ways, we have

$$King(John) \wedge Greedy(John) \;\Rightarrow\; Evil(John)$$
$$King(Richard) \wedge Greedy(Richard) \;\Rightarrow\; Evil(Richard)$$
$$King(John)$$
$$Greedy(John)$$
$$Brother(Richard, John)$$

The new KB is propositionalized: proposition symbols are

$King(John)$, $Greedy(John)$, $Evil(John)$, $King(Richard)$ etc.

Reduction contd.

Claim: a ground sentence is entailed by new KB iff entailed by original KB

Claim: every FOL KB can be propositionalized so as to preserve entailment

Idea: 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 α is entailed by an FOL KB,
it is entailed by a finite subset of the propositional KB

Idea: For $n = 0$ to ∞ do
create a propositional KB by instantiating with depth-n 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

Problems with propositionalization

Propositionalization seems to generate lots of irrelevant sentences.
E.g., from

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

it seems obvious that $Evil(John)$, but propositionalization produces lots of facts such as $Greedy(Richard)$ that are irrelevant

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

With function symbols, it gets nuch much worse!

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

$\mathrm{Unify}(\alpha, \beta) = \theta$ if $\alpha\theta = \beta\theta$

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

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

$\text{Unify}(\alpha, \beta) = \theta$ if $\alpha\theta = \beta\theta$

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

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

$\text{Unify}(\alpha, \beta) = \theta$ if $\alpha\theta = \beta\theta$

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

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

$\mathrm{Unify}(\alpha, \beta) = \theta$ if $\alpha\theta = \beta\theta$

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

Pearson

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

$\text{Unify}(\alpha, \beta) = \theta$ if $\alpha\theta = \beta\theta$

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

Standardizing apart eliminates overlap of variables, e.g., $Knows(z_{17}, OJ)$

The first algorithm given by Robinson (1965) was rather inefficient; Faster algorithm originated from Martelli, Montanari (1976, 1982)!

Generalized Modus Ponens (GMP)

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

where $p_i'\theta = p_i\theta$ for all i

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

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

provided that $p_i'\theta = p_i\theta$ for all i

Lemma: For any definite clause p, we have $p \models p\theta$ by UI

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

Example knowledge base

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 Col. West is a criminal

Example knowledge base contd.

. . . it is a crime for an American to sell weapons to hostile nations:

Example knowledge base contd.

. . . 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

Example knowledge base contd.

... it is a crime for an American to sell weapons to hostile nations:

$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$

Nono ... has some missiles, i.e., $\exists\, x\; Owns(Nono, x) \wedge Missile(x)$:

$Owns(Nono, M_1)$ and $Missile(M_1)$

... all of its missiles were sold to it by Colonel West

Example knowledge base contd.

. . . it is a crime for an American to sell weapons to hostile nations:

$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$

Nono . . . has some missiles, i.e., $\exists\, x\ Owns(Nono, x) \wedge Missile(x)$:

$Owns(Nono, M_1)$ and $Missile(M_1)$

. . . all of its missiles were sold to it by Colonel West

$\forall\, x\ \ Missile(x) \wedge Owns(Nono, x) \ \Rightarrow\ \ Sells(West, x, Nono)$

Missiles are weapons:

Example knowledge base contd.

. . . it is a crime for an American to sell weapons to hostile nations:

$$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$$

Nono . . . has some missiles, i.e., $\exists\, x\ Owns(Nono, x) \wedge Missile(x)$:

$Owns(Nono, M_1)$ and $Missile(M_1)$

. . . all of its missiles were sold to it by Colonel West

$$\forall\, x\ \ Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$$

Missiles are weapons:

$$Missile(x) \Rightarrow Weapon(x)$$

An enemy of America counts as “hostile”:

Example knowledge base contd.

. . . it is a crime for an American to sell weapons to hostile nations:

$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$

Nono . . . has some missiles, i.e., $\exists\, x\ Owns(Nono, x) \wedge Missile(x)$:

$Owns(Nono, M_1)$ and $Missile(M_1)$

. . . all of its missiles were sold to it by Colonel West

$\forall\, x\ \ Missile(x) \wedge Owns(Nono, x) \ \Rightarrow\ Sells(West, x, Nono)$

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

```
function FOL-FC-Ask(KB, α) returns a substitution or false
    repeat until new is empty
        new ← { }
        for each sentence r in KB do
            (p1 ∧ ... ∧ pn ⇒ q) ← Standardize-Apart(r)
            for each θ such that (p1 ∧ ... ∧ pn)θ = (p'1 ∧ ... ∧ p'n)θ
                            for some p'1, ..., p'n in KB
                q' ← Subst(θ, q)
                if q' is not a renaming of a sentence already in KB or new then do
                    add q' to new
                    φ ← Unify(q', α)
                    if φ is not fail then return φ
        add new to KB
    return false
```

Forward chaining proof

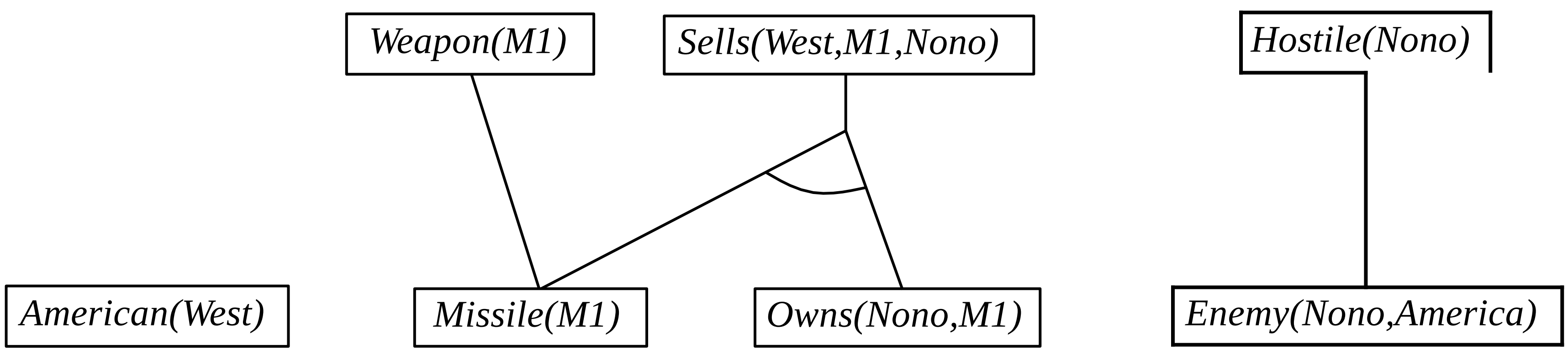
American(West)

Missile(M1)

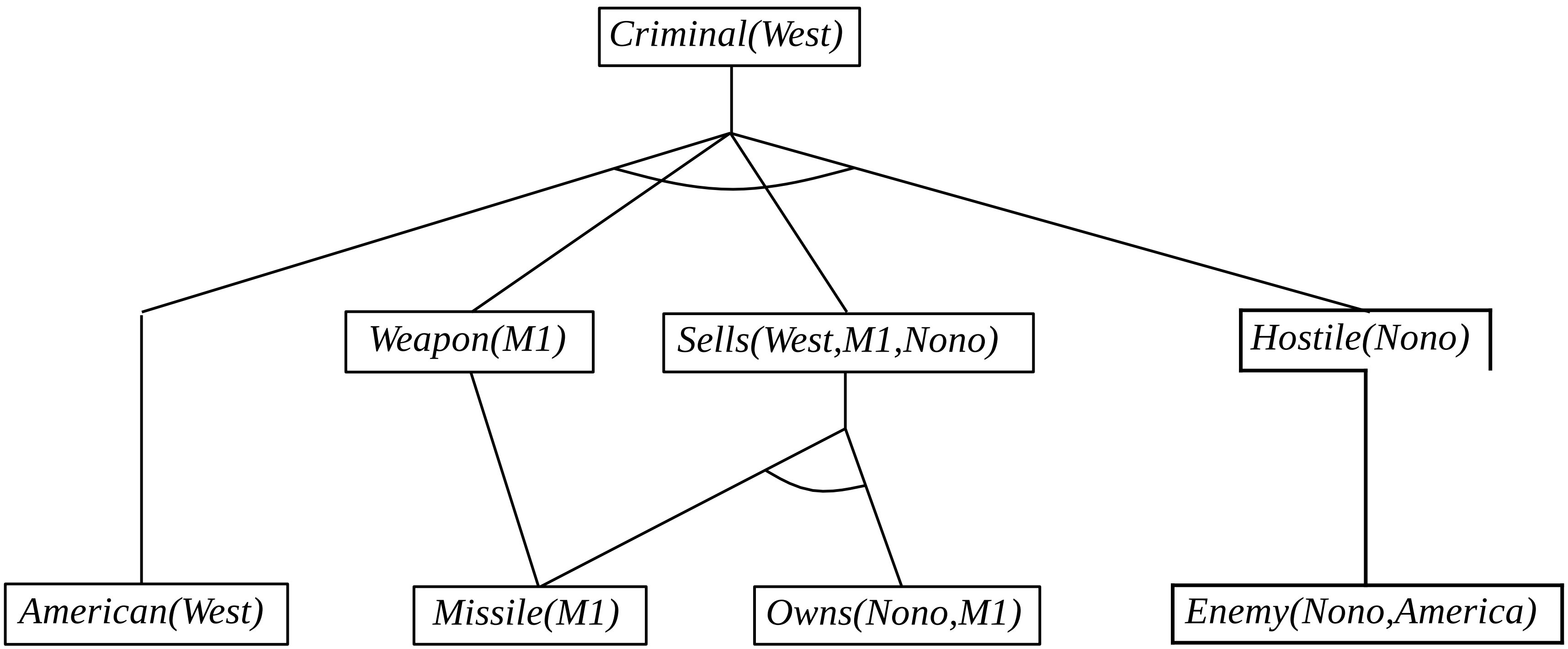
Owns(Nono,M1)

Enemy(Nono,America)

Forward chaining proof



Forward chaining proof



Properties of forward chaining

Sound and complete for first-order definite clauses
(proof similar to propositional proof)

Datalog = first-order definite clauses + no functions (e.g., crime KB)
FC terminates for Datalog in poly iterations: at most $p \cdot n^k$ literals

May not terminate in general if a is not entailed

This is unavoidable: entailment with definite clauses is semidecidable

Pearson

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

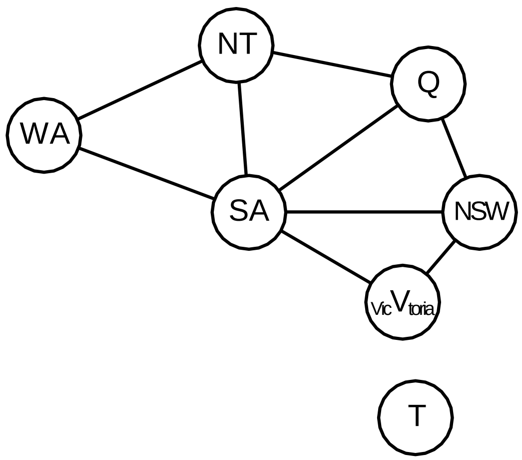
Database indexing allows $O(1)$ retrieval of known facts

e.g., query $Missile(x)$ retrieves $Missile(M_1)$

Matching conjunctive premises against known facts is NP-hard

Forward chaining is widely used in deductive databases

Hard matching example



$$Diff(wa, nt) \land Diff(wa, sa) \land$$
$$Diff(nt, q) \land Diff(nt, sa) \land$$
$$Diff(q, nsw) \land Diff(q, sa) \land$$
$$Diff(nsw, v) \land Diff(nsw, sa) \land$$
$$Diff(v, sa) \Rightarrow Colorable()$$

$Diff(Red, Blue)$ $Diff(Red, Green)$
$Diff(Green, Red)$ $Diff(Green, Blue)$
$Diff(Blue, Red)$ $Diff(Blue, Green)$

$Colorable()$ is inferred iff the CSP has a solution

Backward chaining algorithm

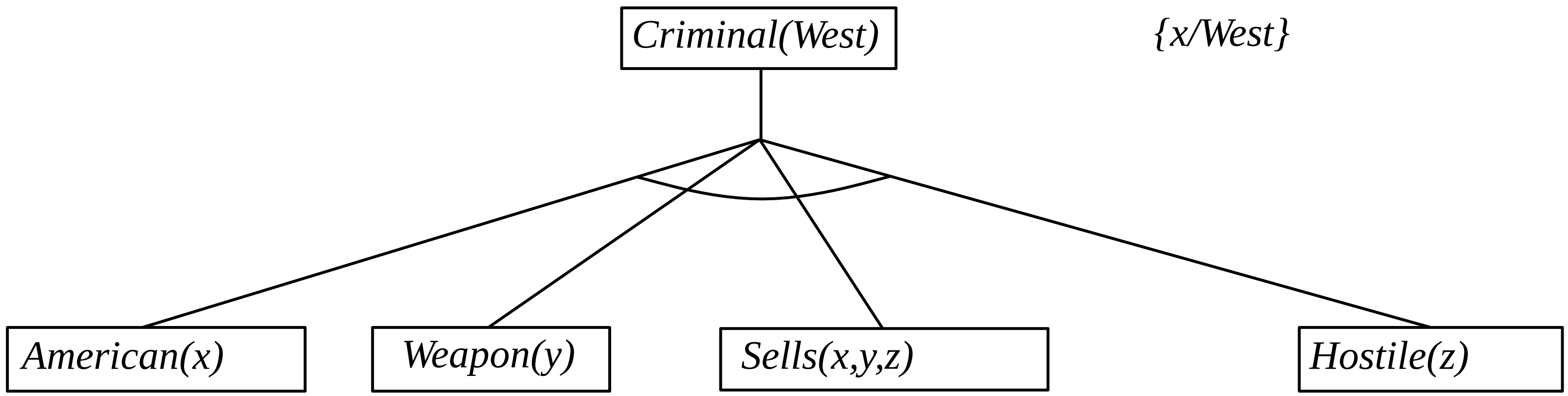
```
function FOL-BC-Ask(KB, goals, θ) returns a set of substitutions
   inputs: KB, a knowledge base
           goals, a list of conjuncts forming a query (θ already applied)
           θ, the current substitution, initially the empty substitution { }
   local variables: answers, a set of substitutions, initially empty

   if goals is empty then return {θ}
   q′ ← Subst(θ, First(goals))
   for each sentence r in KB
           where Standardize-Apart(r) = (p1 ∧ ... ∧ pn ⇒ q)
           and θ′ ← Unify(q, q′) succeeds
      new_goals ← [p1, ..., pn|Rest(goals)]
      answers ← FOL-BC-Ask(KB, new_goals, Compose(θ′, θ)) ∪ answers
   return answers
```

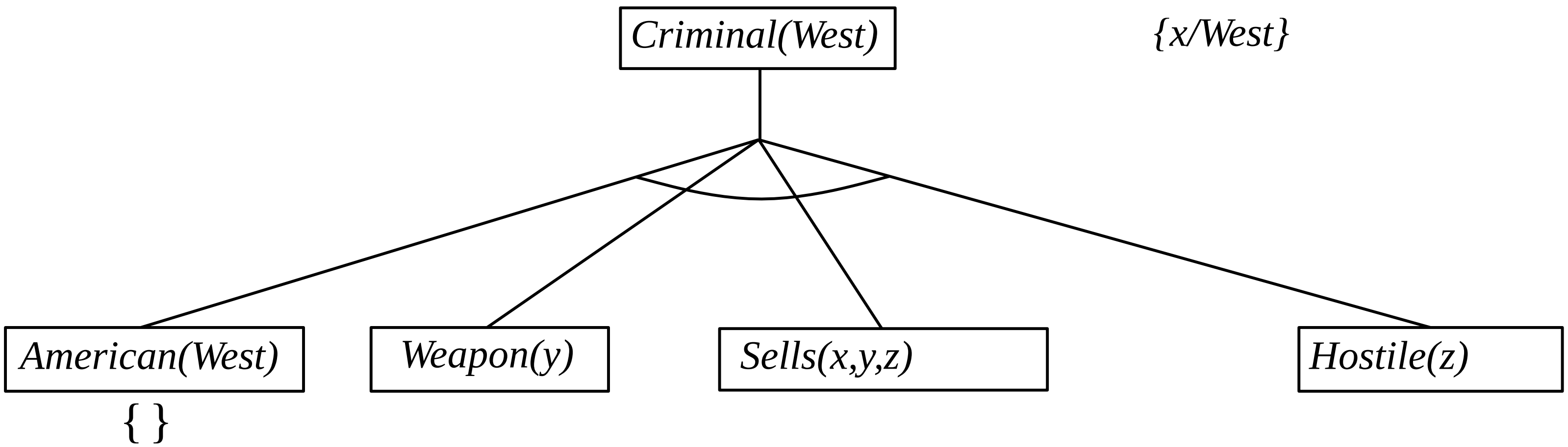
Backward chaining example

$\boxed{Criminal(West)}$

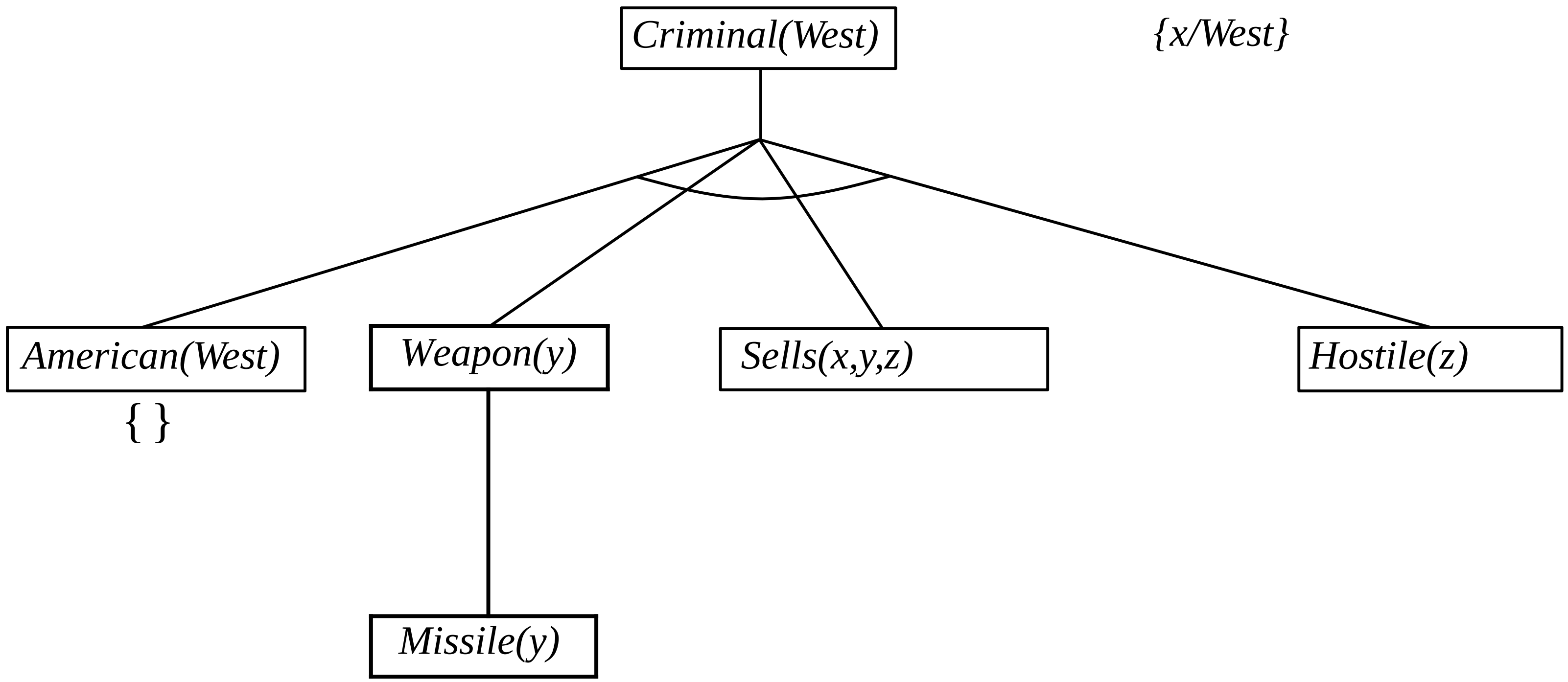
Backward chaining example



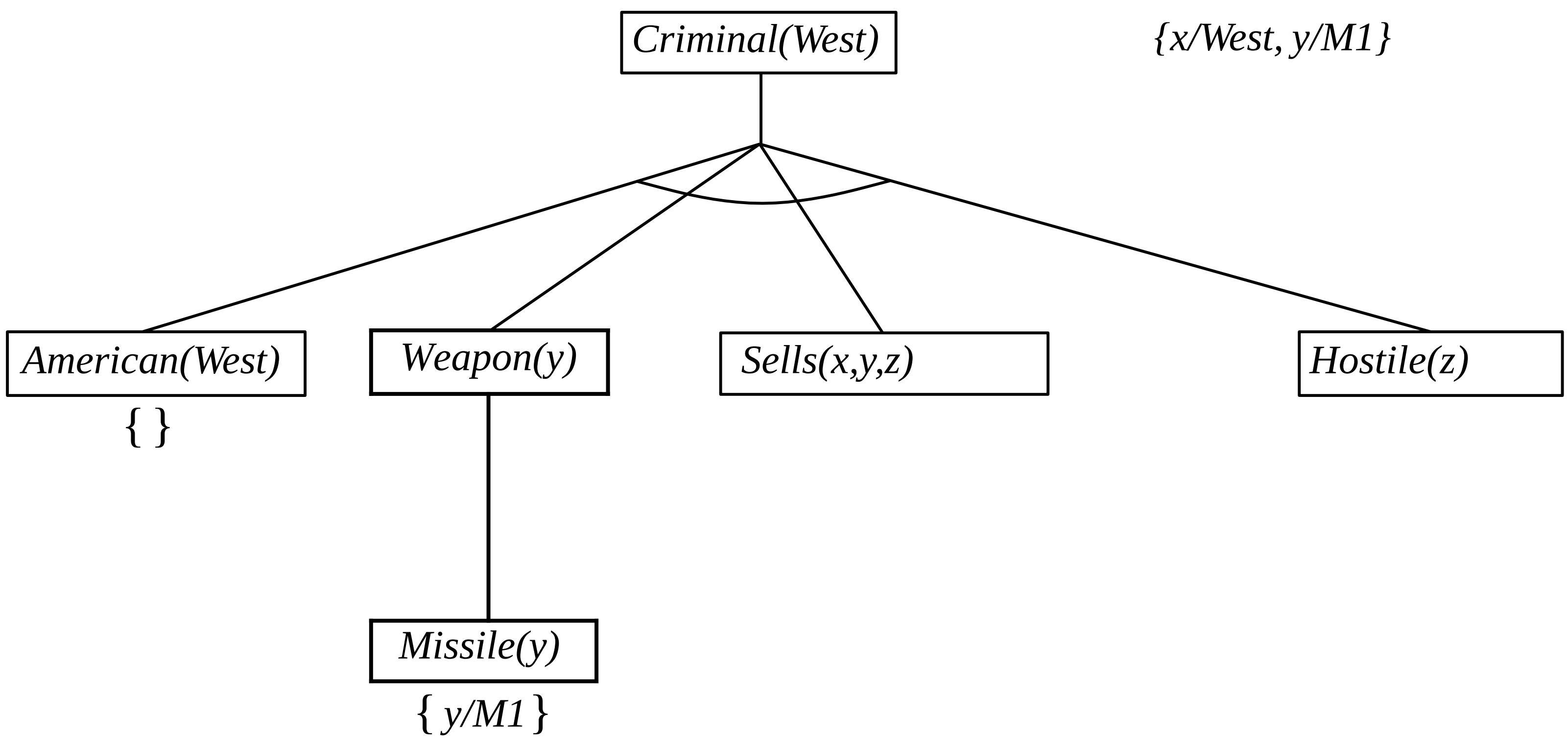
Backward chaining example



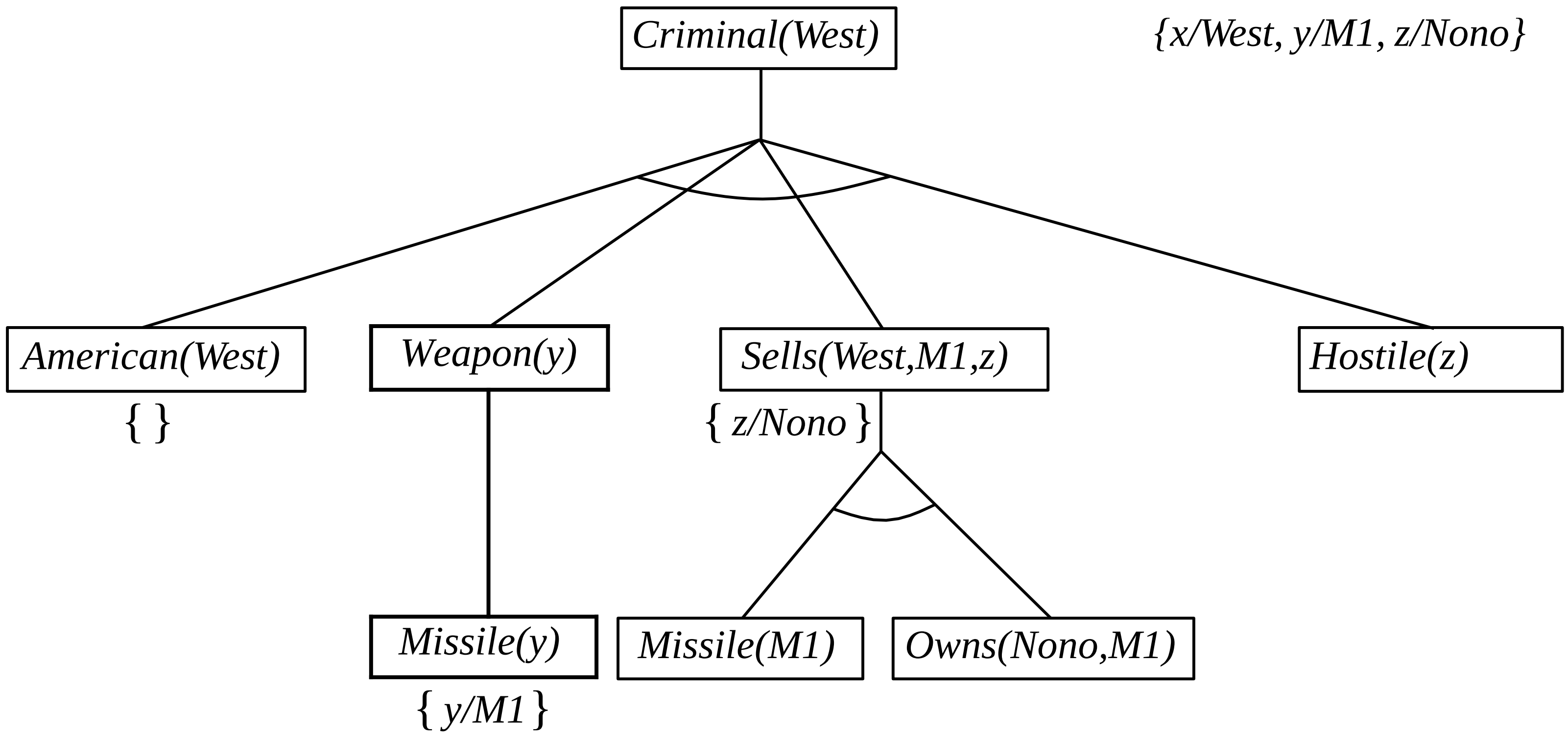
Backward chaining example



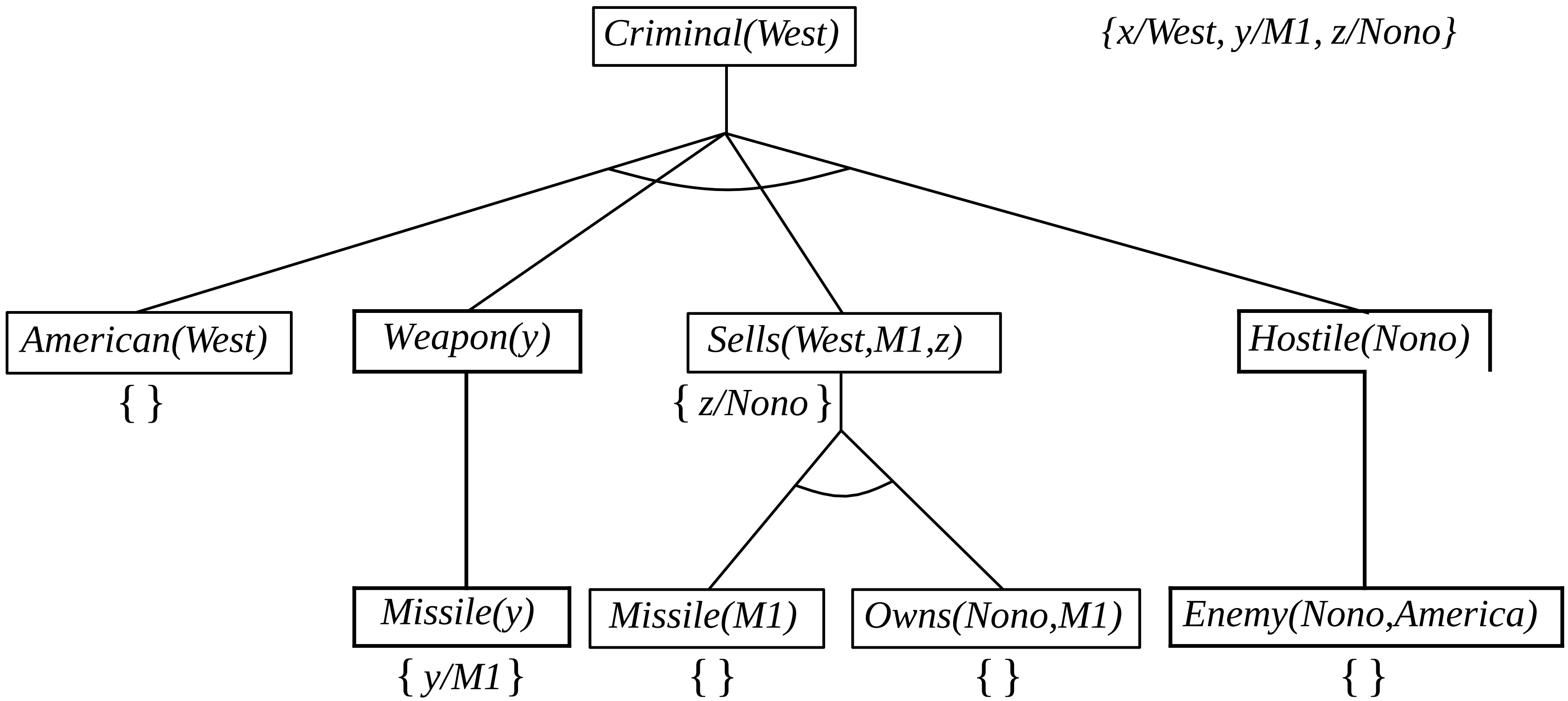
Backward chaining example



Backward chaining example



Backward chaining example



Properties of backward chaining

Depth-first recursive proof search: space is linear in size of proof

Incomplete due to infinite loops

⇒ fix by checking current goal against every goal on stack

Inefficient due to repeated subgoals (both success and failure)

⇒ fix using caching of previous results (extra space!)

Widely used (without much improvements!) for logic programming

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 systems

Basis: backward chaining with Horn clauses + bells & whistles

Widely used in Europe, **Japan** (basis of 5th Generation project)

Program = set of clauses = head :- $\mathtt{literal}_1, \ldots, \mathtt{literal}_n$.

e.g. *criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).*

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., given

alive(X) :- not dead(X).

alive(joe) succeeds if dead(joe) fails

Prolog examples

Depth-first search from a start state X:

```
% Define edges of the graph

edge(a, b).
edge(a, c).
edge(b, d).
edge(c, e).
edge(e, f).

% DFS: Check if there is a path from Start to Goal

dfs(Start, Goal) :- edge(Start, Goal).        % Base case: Direct connection
dfs(Goal, Goal).                              % Base case: Found the goal.
dfs(Start, Goal) :-

        edge(Start, NextNode),    % Recursive case: Find a connected node
        dfs(NextNode, Goal).      % Continue searching from the next node
```

?- dfs(a, f). will return true

Prolog examples

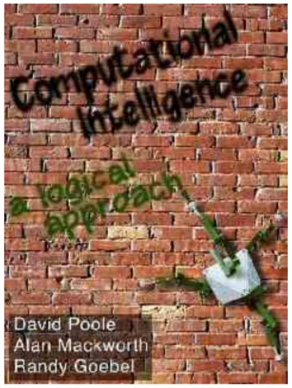
Appending two lists to produce a third:

```
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=[]
```

Can you implement a Neural Network in Prolog?



Computational Intelligence

A Logical Approach

David Poole
Alan Mackworth
Randy Goebel

Published by Oxford University Press, New York.

Computational Intelligence: A Logical Approach is a textbook on artificial intelligence. It was published in January 1998.

- Table of Contents (or front matter in PDF format).
- Preface (or PDF format).
- Chapter 1 (in PDF format)
- CIspace: tools for learning Computational Intelligence. We have applets for learning about graph searching, constraint satisfaction problem solving, stochastic local search, neural network learning, and robot control.
- Online Code for the book
- Solved exam-style problems (*not* exercises from the book).
- Overhead Transparencies
- Errata
- Sample 12 week course based on the book.
- `CILog` (or PDF format), a representation and reasoning system with declarative debugging and explanation tools.
- Order a copy of the book
- Price Compare (put in your own country or state and currency then redisplay the result). We make no guarantees about this service, but it seems to be a good idea.

---

accesses since 6 November 1997.

Computational Intelligence: A Logical Approach (ubc.ca)

Resolution: brief summary

Full first-order version:

$$\frac{p_1 \vee \cdots \vee p_k, \quad m_1 \vee \cdots \vee m_n}{(p_1 \vee \cdots \vee p_{i-1} \vee p_{i+1} \vee \cdots \vee p_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n)\theta}$$

where $\text{Unify}(p_i, \neg m_j) = \theta$. 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: $\neg\forall x, p\ \equiv \exists x\ \neg p$, $\neg\exists x, p\ \equiv \forall x\ \neg p$:

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

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

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

Conversion to CNF contd.

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

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

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

$$\forall x \; [Animal(F(x)) \land \neg Loves(x, F(x))] \lor Loves(G(x), x)$$

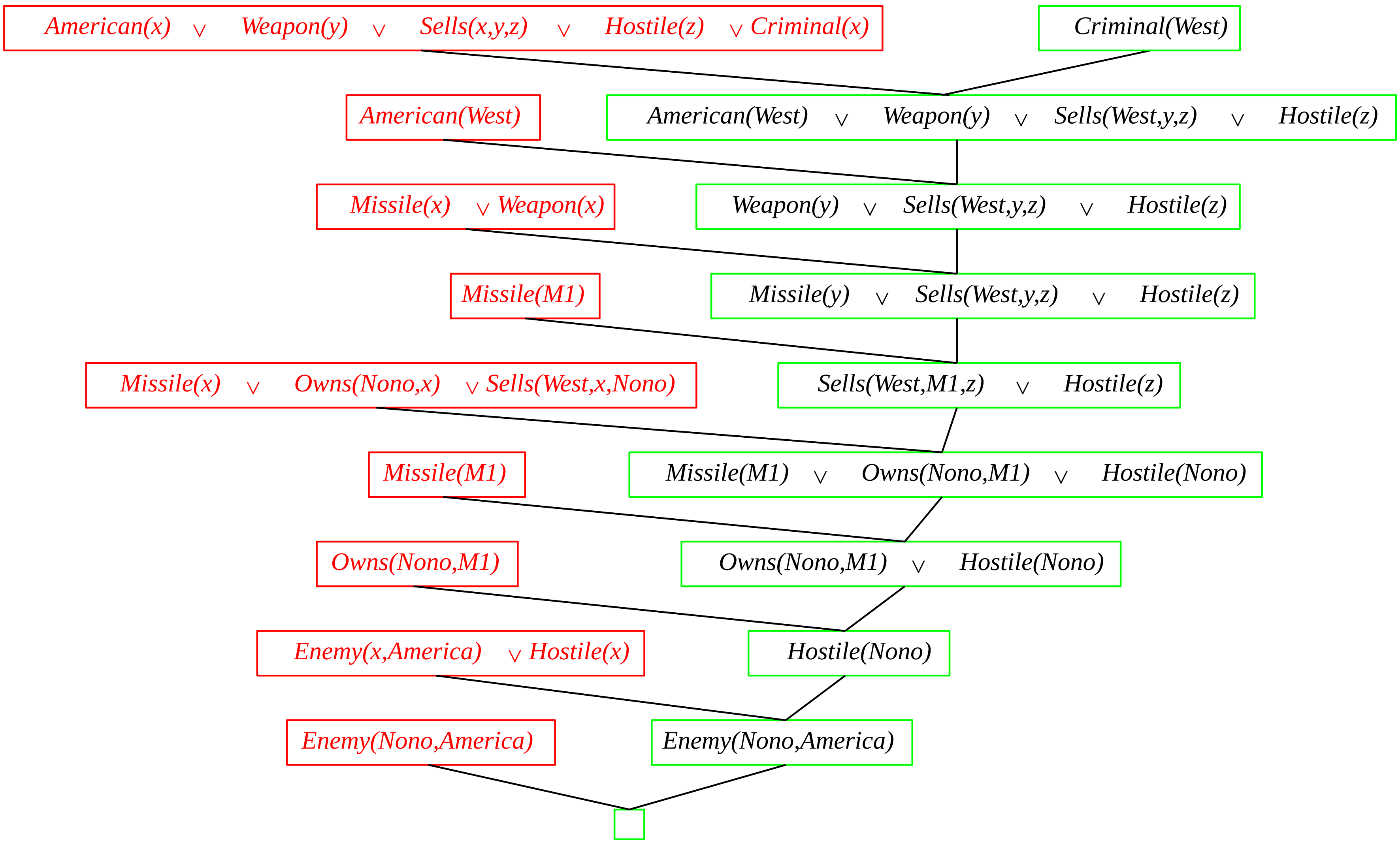
5. Drop universal quantifiers:

$$[Animal(F(x)) \land \neg Loves(x, F(x))] \lor Loves(G(x), x)$$

6. Distribute $\land$ over $\lor$:

$$[Animal(F(x)) \lor Loves(G(x), x)] \land [\neg Loves(x, F(x)) \lor Loves(G(x), x)]$$

Resolution proof: definite clauses



Real-world KBs: The CYC project

Cyc is a long-term AI project that aims to assemble a **comprehensive ontology** and **knowledge base** that spans the basic concepts and rules about how the world works.

The project began in July 1984 as the flagship project of the 400-person Microelectronics and Computer Technology Corporation (MCC), a research consortium started by two dozen large United States based corporations *"to counter a then ominous Japanese effort in AI, the so-called "fifth-generation" project."*

Hoping to capture *common sense knowledge*, Cyc focuses on implicit knowledge that other AI platforms may take for granted.

Cyc enables semantic reasoners to perform human-like reasoning and be **less** "***brittle***" when confronted with novel situations.

Douglas Lenat began the project in July 1984 at MCC, where he was Principal Scientist 1984–1994, and then, since January 1995, has been under active development by the **Cycorp company**, where he was the CEO and recently died (2023).

The CYC project

[Douglas Lenat: Cyc and the Quest to Solve Common Sense Reasoning in AI | Lex Fridman Podcast #221 - YouTube]

Pearson

The CYC project (Some details)

Within a few years of the launch of the Cyc project it became clear that even representing simple knowledge (e.g. a typical news story or advertisement, etc.) would require more than the expressive power of full first-order logic

By 1989,[6] CycL (the custom representation language of the CYC project) had expanded in expressive power to **higher-order logic** (HOL).

The Cyc **inference engine** design separates the epistemological problem (what content should be in the Cyc KB) from the heuristic problem (how Cyc could efficiently infer arguments hundreds of steps deep, in a sea of tens of millions of axioms).

To do the former, the **CycL language** and well-understood logical inference might suffice.

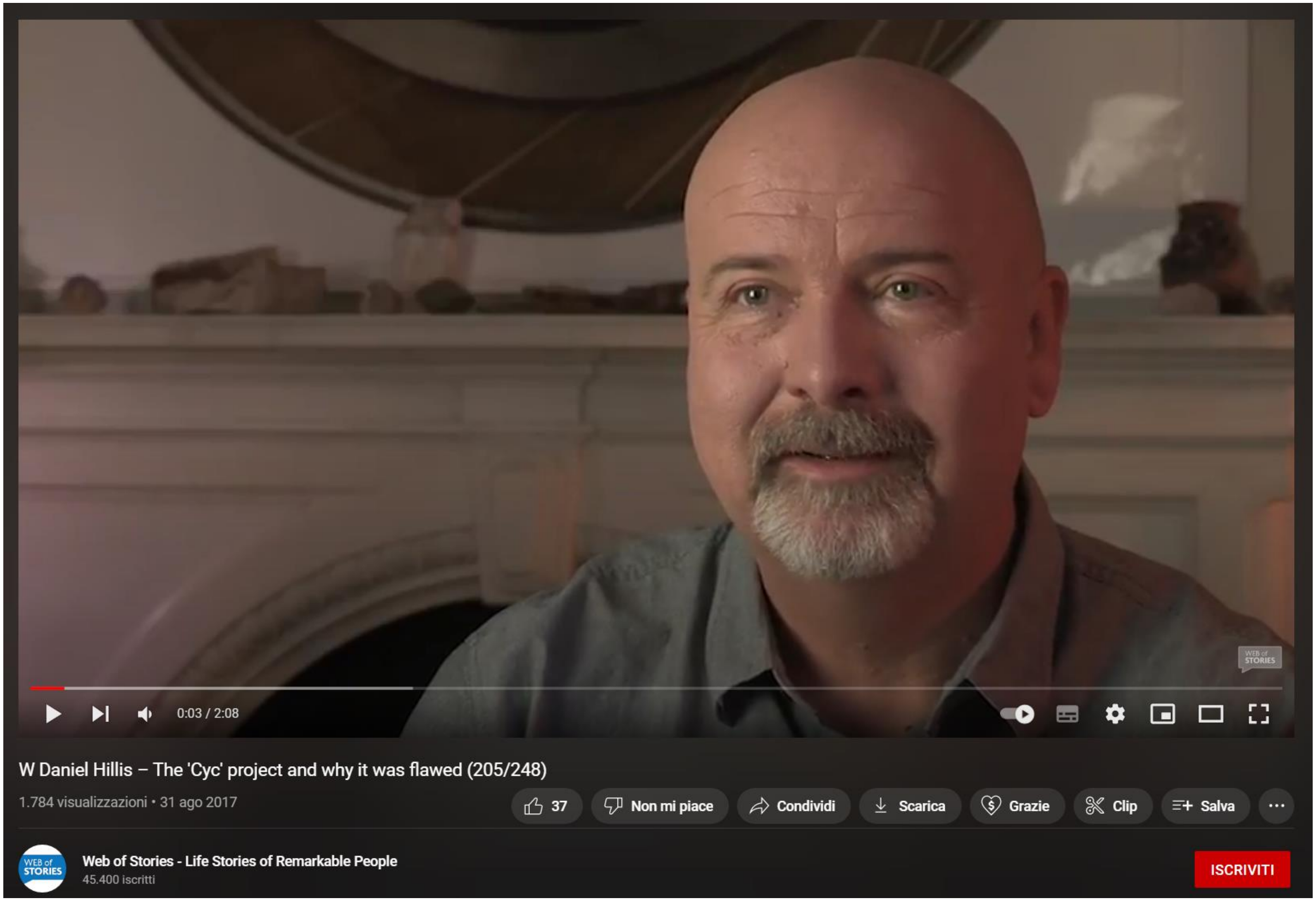
For the latter, Cyc used a **community-of-agents architecture**, where **specialized reasoning modules** can be used to attack different sub-problems.

By 1994 there were 20 such **heuristic level (HL)** modules;[8] as of 2017 there are over **1,050 HL modules**. **Common-sense rules and assertions** grew to about 1 million in 1994, and as of 2017 is about 24.5 million.

The CYC KB Visualization

[Demo di visualizzazione della Knowledge Base di Cyc - YouTube]

The CYC Criticism



W Daniel Hillis – The 'Cyc' project and why it was flawed (205/248) - YouTube

The CYC Criticism

The Cyc project has been described as *"one of the most controversial endeavors of the artificial intelligence history"*.

Catherine Havasi, CEO of Luminoso, says that Cyc is the predecessor project to IBM's Watson.

Machine-learning scientist **Pedro Domingos** refers to the project as a "*catastrophic failure*" for several reasons, including the unending amount of data required to produce any viable results and the **inability for Cyc to evolve on its own**.

Further readings: Domingos, Pedro (2015). *The Master Algorithm: How the Quest for the Ultimate Learning Machine Will Remake Our World*. ISBN 978-0465065707.

Open-access KBs

Semantic Web 1 (2015) 1–5
IOS Press

A Comparative Survey of DBpedia, Freebase, OpenCyc, Wikidata, and YAGO

Michael Färber *,**, Basil Ell, Carsten Menne, and Achim Rettinger
Karlsruhe Institute of Technology (KIT), Institute AIFB,
76131 Karlsruhe, Germany

Abstract. In recent years, several noteworthy large, crossdomain and openly available knowledge graphs (KGs) have been created. These include DBpedia, Freebase, OpenCyc, Wikidata, and YAGO. Although extensively in use, these KGs have not been subject to an in-depth comparison so far. In this survey, we first define aspects according to which KGs can be analyzed. Next, we analyze and compare the above mentioned KGs along those aspects and finally propose a method for finding the most suitable KG for a given setting.

Keywords: Knowledge Graph, Comparison, DBpedia, Freebase, OpenCyc, Wikidata, YAGO

1. Introduction

The idea of the Semantic Web is that of publishing and querying knowledge on the Web in a semantically structured way. According to Guns [27], the term "Se-

between entities (e.g., *isSpouseOf*) can be represented on the Web.

When it comes to realizing the idea of the Semantic Web, knowledge graphs (KGs) are currently seen as one of the most essential components. We define

https://www.semantic-web-journal.net/system/files/swj1141.pdf

Pearson

Wikidata + SPARQL

https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial

Conceptnet 5

ConceptNet

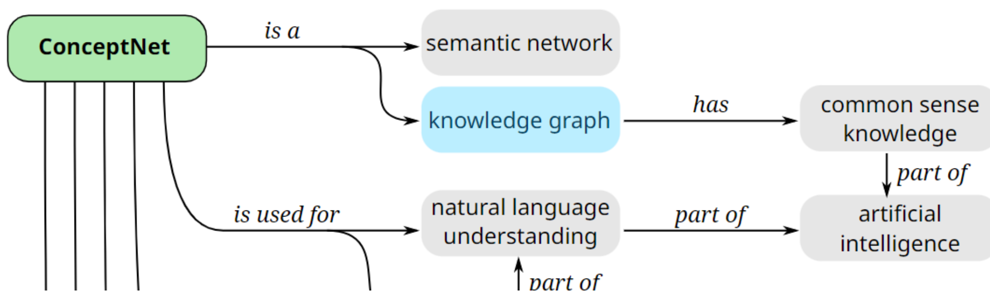
An open, multilingual knowledge graph

apple

What is ConceptNet?

ConceptNet is a freely-available semantic network, designed to help computers understand the meanings of words that people use.

ConceptNet originated from the crowdsourcing project Open Mind Common Sense, which was launched in 1999 at the MIT Media Lab. It has since grown to include knowledge from other crowdsourced resources, expert-created resources, and games with a purpose.



https://conceptnet.io

[API · commonsense/conceptnet5 Wiki · GitHub]()

Pearson

AIMA notebooks: «logic.ipynb»

Logic

This Jupyter notebook acts as supporting material for topics covered in **Chapter 6 Logical Agents**, **Chapter 7 First-Order Logic** and **Chapter 8 Inference in First-Order Logic** of the book *Artificial Intelligence: A Modern Approach*. We make use of the implementations in the logic.py module. See the intro notebook for instructions.

Let's first import everything from the `logic` module.

In [1]:
```
from utils import *
from logic import *
from notebook import psource
```

CONTENTS

- Logical sentences
  - Expr
  - PropKB
  - Knowledge-based agents
  - Inference in propositional knowledge base
    - Truth table enumeration
    - Proof by resolution
    - Forward and backward chaining
    - DPLL
    - WalkSAT
    - SATPlan
  - FolKB
  - Inference in first order knowledge base
    - Unification
    - Forward chaining algorithm
    - Backward chaining algorithm

https://github.com/aimacode/aima-python/blob/master/logic.ipynb

Summary

- **Unification** identify appropriate substitutions for variables eliminates the instantiation step in first-order proofs, making the process more efficient in many cases

- **Forward chaining** is used in deductive databases, where it can be combined with relational database operations. It is also used in *production systems*

- **Backward chaining** is used in logic programming systems, which employ sophisticated compiler technology to provide very fast inference

- **Prolog**, unlike first-order logic, uses a closed world with the unique names assumption and negation as failure.

- The generalized **resolution** inference rule provides a complete proof system for first order logic, using knowledge bases in conjunctive normal form.

- Some **real-world knowledge bases** are open-access to use!

In the next lecture...

- Definition of Classical Planning
- Algorithms for Classical Planning
- Heuristics for Planning
- Hierarchical Planning
- Planning and Acting in Nondeterministic Domains
- Time, Schedules, and Resources
- Analysis of Planning Approaches