Artificial Intelligence
Unit 3-1 Knowledge Representation and Reasoning

2020-2021 Odd BE CSE VII semester

Engels. R

Unit 3 – KRR

- **KNOWLEDGE AND REASONING:**
  - Representation
  - First Order Predicate Logic
  - Inference
  - Unification
  - Forward and Backward Chaining
  - Resolution
  - Statistical Reasoning
  - Probability and Bayes Theorem
  - Dempster-Shafer Theory
- Blended/Self learning/Self Study
  - Reasoning with Default Information (12.6 AIMA)
  - Truth Maintenance Systems (12.6 AIMA)
  - Acting under Uncertainty (13.1 AIMA)
  - Certainty Factors and Rule Based Systems (8.2 ERKK)

REPRESENTATION

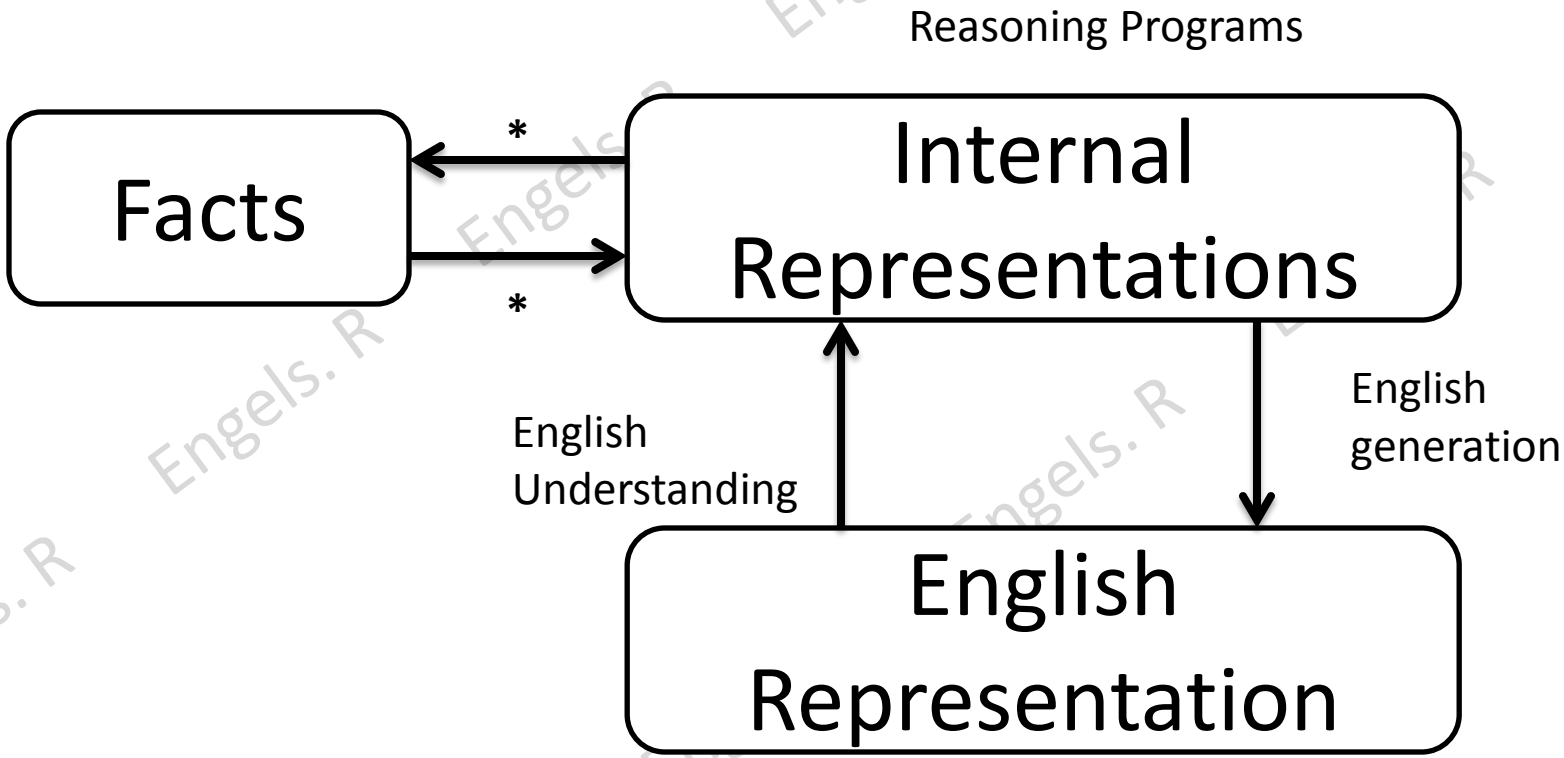
Knowledge Representation

- Problem solving requires knowledge
- Techniques for Knowledge Representation and manipulation are important
- Facts
  - Truths in some relevant world
  - Items that need to be represented
- Representation of facts
  - In some chosen formalism
  - Items that can actually be manipulated

Knowledge Levels

- Two levels of knowledge representation
- Knowledge level
  - Facts are described
  - Agent's behavior and current goals exist in this level
- Symbol level
  - Representations of objects at the knowledge level are defined in this level
  - These representations are defined in terms of symbols that can be manipulated by programs

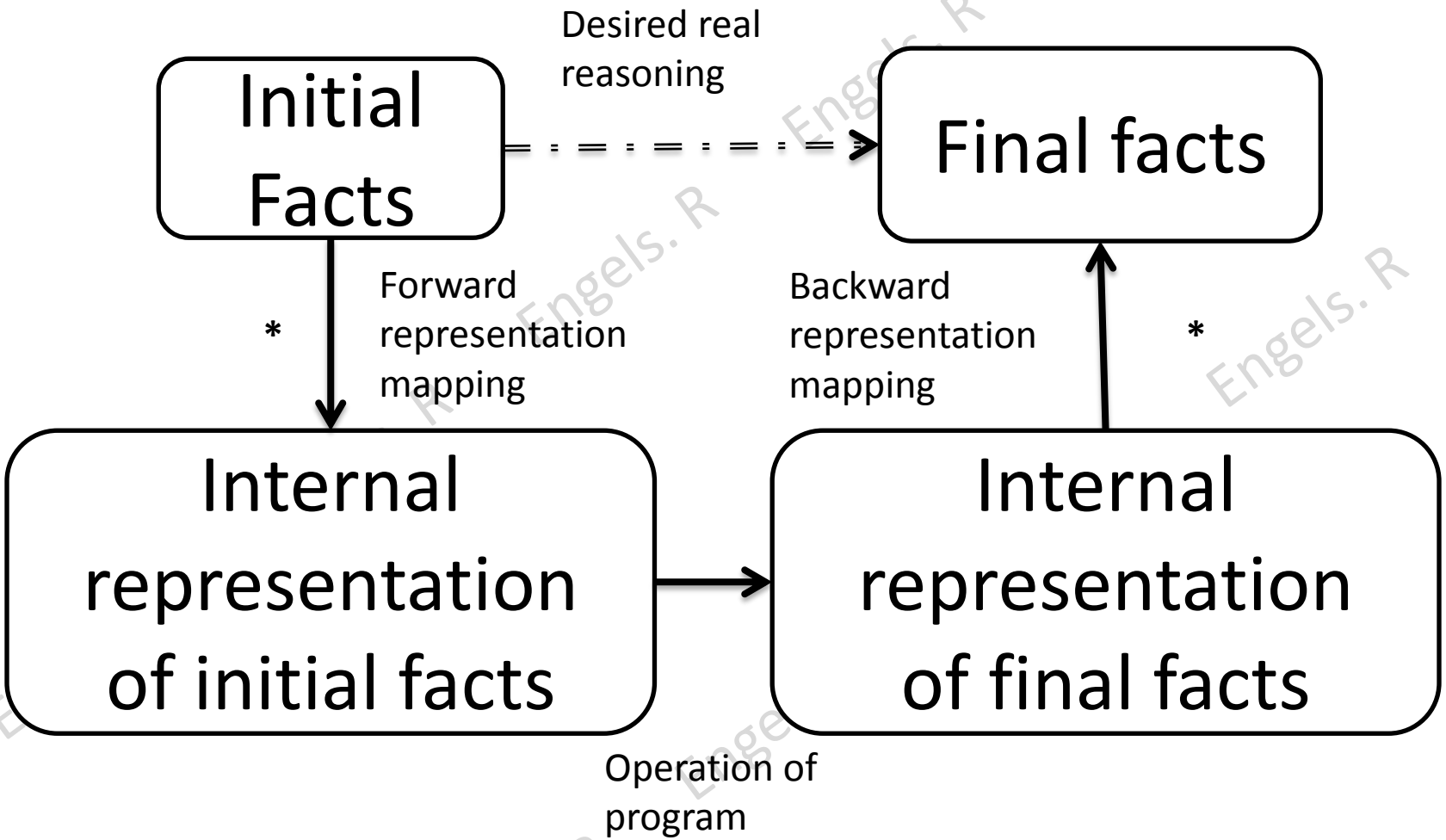
Mapping between facts & Representation



Logic as representational formalism

- English
  - Spot is a dog
- Logic
  - dog(spot)
- All dogs have tails
  - $\forall x$: dog(x) → hastail(x)
  - hastail(spot)
- English sentence would be *Spot has a tail*
- Usually mapping functions are not one to one!
  - Usually many-to-many relations
  - All dogs have tail
  - Every dog has a tail

Representation of facts



Mappings with every representation corresponds to only one fact OR with at least one representation for each fact, are not available for real world problems

Knowledge representation Approaches

- Representational adequacy
  - Ability to manipulate representational structures to derive new structures corresponding to new knowledge inferred from old knowledge
- Inferential adequacy
  - Ability to incorporate additional information into knowledge structure
- Inferential efficiency
  - Additional information can be used to focus attention of inference mechanisms in most promising direction
- Acquisitional efficiency
  - Ability to acquire new information easily
  - System itself controlling knowledge acquisition
- Currently, no existing systems has all the above

Simple relational knowledge

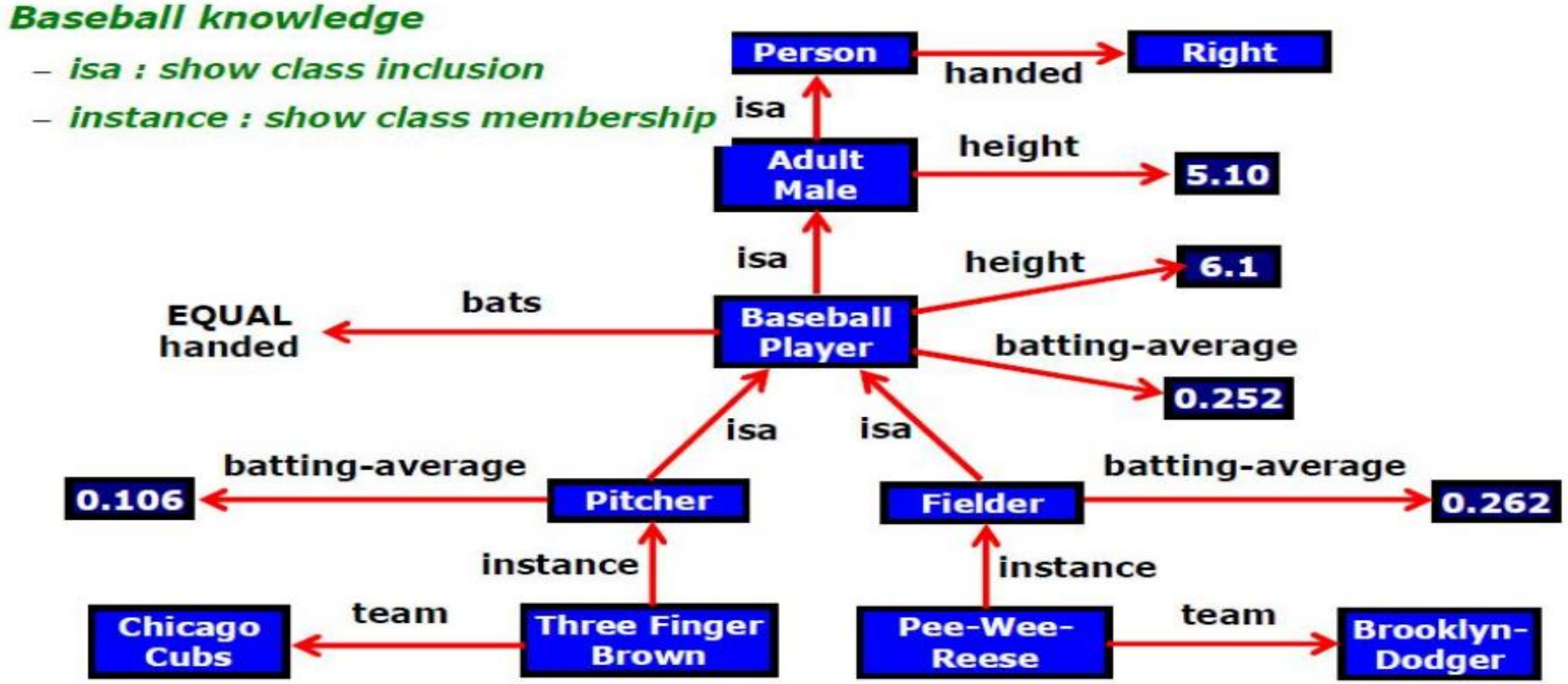
Player	Type	Run Rate	Wicket / Match
Kohli	Batsman	99.9	0.01
Stark	Bowler	4.5	3
Rohit	Batman	25.6	0.5

- Represents facts as a set of relations
- Can find the highest scoring batsman
- Can match best bowler against best batsman
- By itself, provides very weak inferential capabilities
- Knowledge can serve as an input (Database systems may do this) for other, powerful inference systems

Inheritable Knowledge

- Knowledge about objects, their attributes, and their values can be complex
  - Can be represented using property inheritance
- Property inheritance
  - Elements of a specific class inherit attributes and values from more general classes
  - Objects are organized into classes
  - Classes are arranged in a generalization hierarchy
- Such a structure is known as *slot-and-filler*
  - AKA *Semantic net* or a collection of *Frames*

Inheritable Knowledge - example



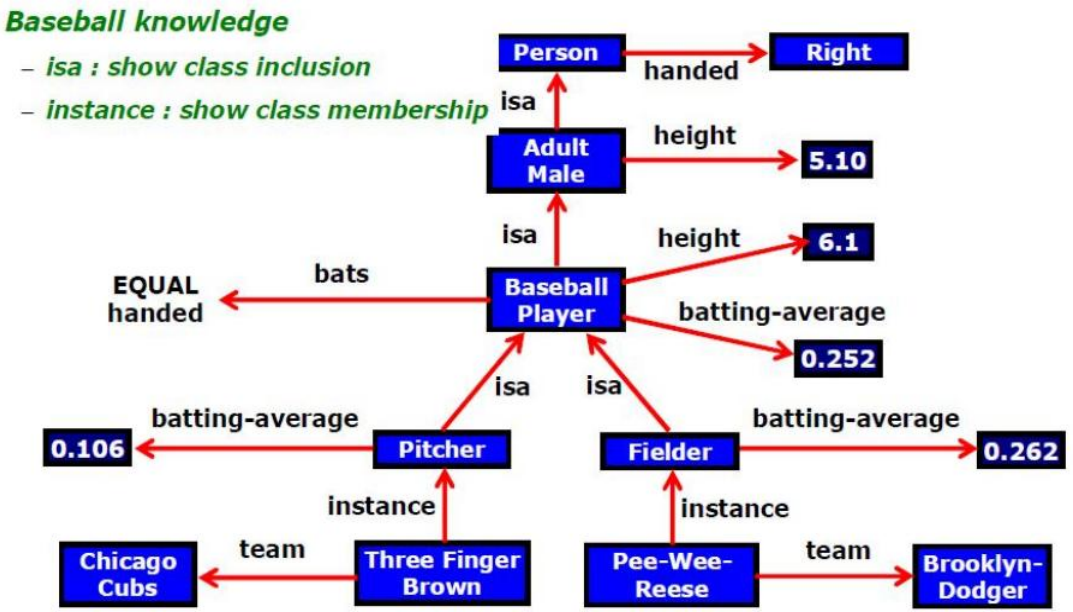
- Lines: attributes
- Boxed nodes: objects, values of attributes
- Arrows point from object to its value (along corresponding attribute line)

Property inheritance algorithm

- To retrieve a value V for attribute A of an instance object O:
  1. Find O in the knowledge base.
  2. If there is a value for attribute A, return value.
  3. Otherwise, check if there is a value for attribute ***instance***. If not, then fail.
  4. Otherwise move to node corresponding to that value, look for a value for the attribute A. If found, return value.
  5. Otherwise, do until there is no value for a ***isa*** attribute or until an answer is found
     - Get the value of a ***isa*** attribute and move to that node
     - Check if there is a value for attribute A. If found, return value

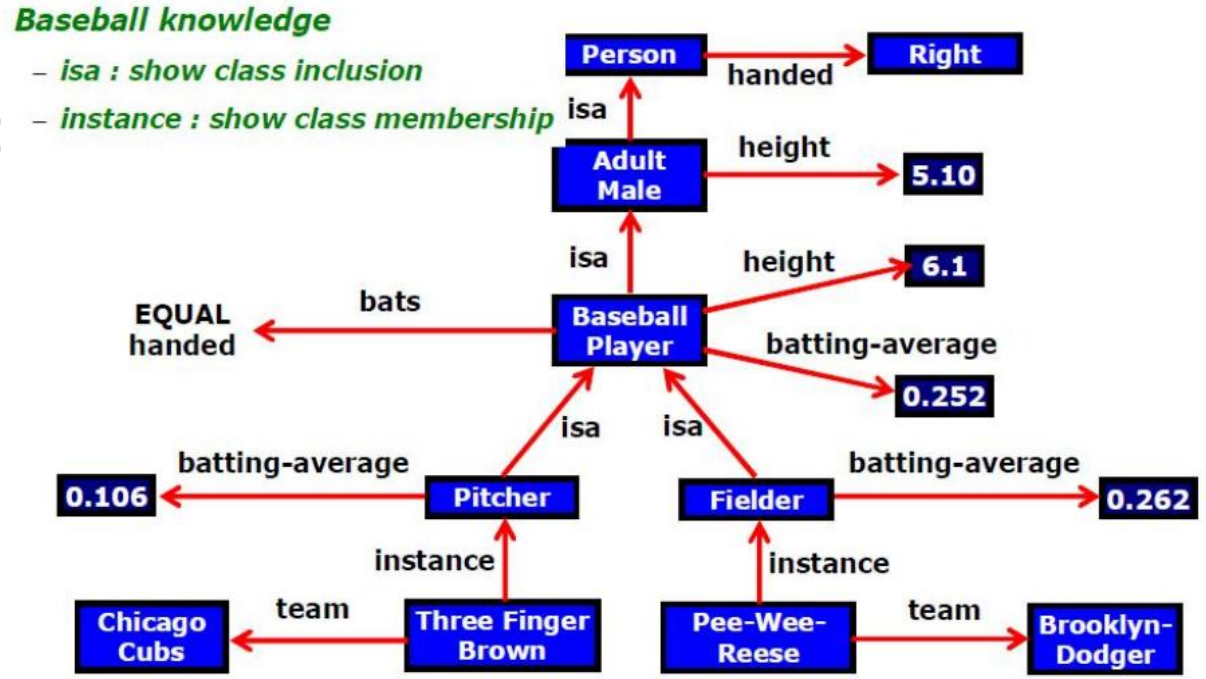
Property inheritance algorithm

- Query 1: team(Pee-Wee-Reese) = Brooklyn-Dodgers
  - Direct value
- Query 2: batting-average(Three-Finger Brown) = 0.106
  - No explicit batting average value → Follow *instance* attribute to Pitcher → Extract value
  - Property inheritance may produce default values that are not guaranteed to be correct, but represent ***best guesses***



Property inheritance algorithm

- Query 3: height(Pee-Wee-Reese): 6-1
  - Another default reference
  - More specific fact about height of player overrides general fact about height of adult males
- Query 4: bats(Three-Finger-Brown) = Right.
  - To get value for attribute *bats*
  - Go up *isa* hierarchy to the class *Baseball-player*
  - No value
  - Rule for computing a value is found
    - Need another value (handed) as input
  - Start entire process recursively to find a value for *handed*
  - Go all the way up to *Person*
  - Discover that default value for handedness for people is *Right*
  - Apply the rule for *bats* produce the result *Right*



Inferential Knowledge

- Property inheritance is powerful
- But, knowledge without reasoning / inference is not useful
- Traditional logic can help in reasoning
  - Provides powerful structure to describe relationships among values
- Reasoning procedures can
  - Reason forward from given facts (if-added)
  - Reason backward from desired conclusions to given facts (if-needed)
- Different techniques can be used as building blocks
  - Eg: Description language with an isa hierarchy combined to build efficient systems

Procedural Knowledge

- Static, declarative facts may not be sufficient
- Operational / procedural knowledge is needed
  - Specifies what to do when
- Procedural knowledge can be built using programming language code, eg: LISP
  - Low inferential adequacy: Writing a program that reasons about another programs behavior is difficult
  - Low acquisitional efficiency: Process of updating and debugging large code is difficult

Procedural Knowledge

- Production rules can be used to efficiently represent operational knowledge
- Can be augmented with information on how they should be used
- Example:

If: ***last batsman****, and*
very good bowler*, and*
4 runs to win*,*
Then: *ask batsman to make a wild swing*

- Not very different from code
- How this knowledge would be used by a procedure is important.

Issues in knowledge representation

- Important attributes
  - Are any attributes are so basic that they occur in every problem domain?
  - How should they be handled?
- Relationships among attributes
  - What are the most important relationships among attributes of objects?
- Granularity of representation
  - At what level should knowledge be represented?
  - Is there a good set of *primitives* to which knowledge can be broken down?
- Representing sets of objects
- Finding the correct structures as needed
  - When there is a large knowledge base, how to access relevant parts as needed?

Attributes

- Important attributes: Two attributes with generic significance
  - Class membership: ***instance***
  - Class inclusion (transitive property): ***isa***
  - Explicitly defined in slot-and-filler systems
  - Explicitly / Implicitly defined in logic-based systems
  - Support property inheritance
- Relationships among attributes
  - Inverse
  - Existence in an ***isa*** hierarchy
  - Techniques for reasoning about values
  - Single-valued attributes

Attribute Relationships: Inverses

- Entities are related in different ways in real world
- An object has binary relationships with other objects
  - Attributes are these binary relationships
  - Directions of attributes can be reversed with appropriate change in attribute as well
- Team(PWReese, Dodgers) is equivalent to Dodgers.member(PWReese)
- This approach is used in semantic nets and frame-based systems

Attribute Relationships: isa hierarchy

- Similar to classes of objects and specialized subsets of classes, attributes also have specialization of attributes
- Example: height

  specialization of (physical-size)

  specialization of (physical-attribute)
- These specializations support inheritance
- Inherited information
  - Constraints on values that attributes can have
  - Value computing mechanisms

Attribute Relationships: Reasoning

- Reasoning
  - Information that play a role in reasoning are
    - Type of the value
    - Constraints on the value, typically in terms of related entities
      - (ageofparent > ageofchild)
    - Rules for computing the value (aka if-needed, backward)
    - Rules that describe actions that should be taken if a value becomes known (aka if-added, forward)

Attribute Relationships: Single-valued

- Single-valued attributes
  - Guaranteed to take a unique value
  - Can be implemented by three methods
    - Explicit notion for temporal interval. If two values are same in a time interval, signal a contradiction
    - Assume that only time interval of interest is now. i.e., discard the old value when a new value is available
    - No explicit support (logic-based systems). Knowledge base should have conditions for supporting single-valued attributes

Attribute Relationships: Granularity

- At what level of details should the world be represented?
  - What should be the primitives?
- Example: John spotted Joe
- Stored as

  spotted(agent(John))

  Object(sue)
- Who spotted Joe can be answered
- Did John see Joe?
- Saw(agent(john), object(Joe), timespan(brief))
  - Spot is represented by seeing (past tense) and timespan
- Converting all statements into a representation in terms of small set of primitives
  - Rules that derive inferences need to be written only in terms of primitives

Granularity – Contd.

- Disadvantages of low-level granularity
  - Simple high-level facts require a lot of storage when broken down
  - Usually knowledge is initially presented in a high-level form (English language)
    - Need lot of effort to convert to low-level facts
  - In many domains, initially, it is not clear what the primitives should be
    - Even if facts are available, information required to convert them to low-level facts may not be available

Representing sets of objects

- There are English speaking people all over the world
  - Makes sense only if we talk about the set of people who speak English
  - No one English speaking person can be present all over the world!
- There are more sheep than people in Australia
  - ?
- If we can attach properties to Sets and not to individual objects, a lot of time and effort can be saved
  - Hence its beneficial to construct sets of objects

Representing sets of objects – Contd.

- Need to be careful when we attach properties to the Set
- Is the property being attached to the Set itself?
  - Ex: Large(Elephant). The set of elements is large in size (has many member elements)
- Or is it being attached to the elements of the set?
  - Ex: Large(Elephant). Every elephant is large

LOGIC (PROPOSITIONAL / FIRST ORDER)

Logic - Basics

- Standard logic symbols
- →: Material implication
- ¬: Not
- ∧: And
- ∨: Or
- ∀: For all / For every
- ∃: There exists
- $x \models y$ means x semantically **entails** y
  - Necessary or inevitable part/consequence

Logical Entailment
A set of sentences (called premises) logically **entails** a sentence (called a conclusion) if and only if every truth assignment that satisfies the premises also satisfies the conclusion

Representing simple facts using logic

- **Using propositional logic**
  – Represent real-world facts as logical propositions
    – written as well-formed formulas (wff's)
- Example 1:

It is raining	RAINING
It is sunny	SUNNY
It is Windy	WINDY
If it is raining, then it is not sunny	RAINING → ¬SUNNY

Representing simple facts using logic

- Example 2:

Socrates is a man	SOCRATESMAN
Plato is a man	PLATOMAN
All men are mortal	MORTALMAN

- Disadvantages:
  - Can't draw any conclusion about similarities between Socrates and Plato
  - Can't capture the relationship between
    - Any individual being a man
    - And that individual being a mortal
- Advantages:
  - Simple
  - A decision procedure exists

Propositional logic vs. predicate logic

- Decidability
  - In logic, a **true/false decision problem is decidable if there exists an effective method for deriving** the correct answer.
- Propositional logic
  - Theorem proving is decidable
  - Cannot represent objects and quantification
- Using predicate logic
  - Can represent objects and quantification
  - Theorem proving is semi-decidable
  - Can represent real world facts as **statements** written in wff's

Sentences to be represented in Predicate logic

- Marcus was a man.
- Marcus was a Pompeian.
- All Pompeians were Romans.
- Caesar was a ruler.
- All Romans were either loyal to Caesar or hated him.
- Every one is loyal to someone.
- People only try to assassinate rulers they are not loyal to.
- Marcus tried to assassinate Caesar

Facts as set of wff's (in predicate logic)

Describing the facts in sentences using set of wff's (in predicate logic)

- Marcus was a man.
  man(Marcus)
- Marcus was a Pompeian.
  Pompeian(Marcus)
- All Pompeians were Romans.
  ∀x: Pompeian(x) → Roman(x)
- Caesar was a ruler.
  ruler(Caesar)

Facts as set of wff's (in predicate logic)…2

- All Romans were either loyal to Caesar or hated him.
- **inclusive-or** representation

$\forall x$: Roman(x) $\rightarrow$

loyalto(x, Caesar) $\vee$ hate(x, Caesar)

- **exclusive-or (XOR)** representation

$\forall x$: Roman(x) $\rightarrow$

(loyalto(x, Caesar) $\wedge \neg$ hate(x, Caesar))

$\vee$ ($\neg$ loyalto(x, Caesar) $\wedge$ hate(x, Caesar))

Inclusive OR **allows both possibilities as well as either of them**. So, if either A or B is True, or if both are True, then the statement value is True

Whereas Exclusive OR only allows one possibility. So if either A or B is true, then and only then is the value True.14

Facts as set of wff's (in predicate logic)…2

- Every one is loyal to someone (need to decide which of the following is appopriate).

  $\forall x: \exists y:$ loyalto(x, y)

  - for every x there exists a y such that x is loyal to y, y may be different

  $\exists y: \forall x:$ loyalto(x, y)

  - there exists a y such that every x is loyal to y

- People only try to assassinate rulers they are not loyal to.

$\forall x: \forall y:$ *person(x)* $\wedge$ ruler(y) $\wedge$ tryassassinate(x, y)
$\rightarrow \neg$loyalto(x, y)

- Marcus tried to assassinate Caesar.
  tryassassinate(Marcus, Caesar)

Query answering using Predicate logic

- **Was Marcus loyal to Caesar?**

man(Marcus)

ruler(Caesar)

tryassassinate(Marcus, Caesar)

⇓ $\forall x: man(x) \rightarrow person(x)$ *(all men are people - required)*

¬loyalto(Marcus, Caesar)

- Many English sentences are ambiguous.
  - There is often a choice of how to represent knowledge
  - Obvious information may be necessary for reasoning
  - May not know in advance which statements to deduce (P or ¬P)

Formalizing sentences:

- Some politician is crooked
  - $\exists x\,(p(x) \wedge q(x))$ or $\exists x\,(p(x) \rightarrow q(x))$?
  - $\exists x\,(p(x) \rightarrow q(x)) \equiv \forall x\, p(x) \rightarrow \exists x\, q(x)$
  - “if everyone is a politician, then someone is crooked”
- Each formalization satisfies one of the following two properties:
  - Universal quantifier $\forall x$ quantifies a conditional
  - Existential quantifier $\exists x$ quantifies a conjunction

Formalizing Sentences

- Some politician is crooked $\exists x\ (p(x) \wedge q(x))$
- No politician is crooked $\forall x\ (p(x) \rightarrow \neg\ q(x))$
- All politicians are crooked $\forall x\ (p(x) \rightarrow q(x))$
- Not all politicians are crooked $\exists x\ (p(x) \wedge \neg\ q(x))$
- Every politician is crooked $\forall x\ (p(x) \rightarrow q(x))$
- There is an honest politician $\exists x\ (p(x) \wedge \neg\ q(x))$
- No politician is honest $\forall x\ (p(x) \rightarrow q(x))$
- All politicians are honest $\forall x\ (p(x) \rightarrow \neg\ q(x))$

Representing Instance & isa relations

- Pompeian(Marcus)

∀x: Pompeian(x) → Roman(x)

- **instance**(Marcus, Pompeian)

∀x: instance(x, Pompeian) → instance(x, Roman)

- **instance**(Marcus, Pompeian)
  isa(Pompeian, Roman)

∀x: ∀y: ∀z: instance(x, y) ∧ isa(y, z) → instance(x, z)

Making an exception

- Example
  - "Paulus was a Pompeian. Paulus neither hated Caesar nor was loyal to him"
  - Pompeian(Paulus) ∧ ¬[loyalto(Paulus, Caesar) v hate(Pualus, Caesar)]
  - Makes the knowledge base inconsistent
- Solution:
  - Modify original assertion to which an exception is being made
  - ∀X: Roman(X) ∧ ¬eq(X, Paulus) → loyalto(X, Caesar) v hate(X, Caesar)

Computable functions and predicates

- Computable predicates

greater-than(1, 0)	less-than(0, 1)
greater-than(2, 1)	less-than(1, 2)
greater-than(3, 2)	less-than(2, 3)
….	….

- Computable functions
  - Greater than (2+3, 1)
  - First compute the value of plus function given the arguments 2, 3 and then send the arguments 5, 1 to gt

Computable functions and predicates

- Marcus was a Pompeian
- All Pompeians died when the volcano erupted in 79 A.D
- Question: Is Marcus alive now?

Pompeian(Marcus)

erupted(volcano, 79) ∧ ∀x: (Pompeian(x) → died(x, 79))?

∀x: ∀t1: ∀t2: died(x, t1) ∧ greater-than(t2, t1) → ¬ alive(x, t2)

¬ alive(Marcus, 2020)

Conversion to Clause form

- All Romans who know Marcus either hate Caeser or think that anyone who hates anyone is crazy
  - Can be represented as

$\forall x : [Roman(x) \wedge \text{know}(\text{x, Marcus})]$
$\rightarrow [\text{hate (x, Caeser)} \vee (\forall y: \exists z: hate(y, z) \rightarrow thinkCrazy(x, y)]$

- Formula would be easier to work with if
  1. It were flatter, i.e., less embedding of components
  2. Quantifiers are separated from the formula and need not be considered
- Conjunctive normal Form (CNF) has both!

Conversion to Clause form

- Using CNF, the formula can be represented as

$$\neg Roman(x) \wedge \neg \text{know}(\text{x, Marcus}) \vee$$
$$\text{hate (x, Caeser)} \vee \neg hate(y, z) \vee thinkCrazy(x, y)$$

well formed formulas

- WFF to clause form does not lose information
- To use resolution, need to convert the set of WFFs to set of CNF clauses
  - No instances of connector A
- Can be achieved by
  - Convert WFFs to CNF expressions
  - Break apart each such expression into clauses , one for each conjunct

Conversion to Clause form

- Step 1: ***Eliminate all*** →, using (a → b is equivalent to ¬a ∨ b.

$\forall x : [Roman(x) \wedge \text{know}(x, \text{Marcus})]$
$\rightarrow$ [hate (x, Caeser) ∨ $(\forall y: \exists z: hate(y, z) \rightarrow thinkCrazy(x, y)]$

- Becomes

$\forall x: \neg[Roman(x) \wedge \text{know}(x, \text{Marcus})]$ ∨ [hate (x, Caeser)
∨ $(\forall y: \exists z: hate(y, z) \rightarrow thinkCrazy(x, y)]$

Conversion to Clause form

- Step 2: ***Reduce the scope of each ¬ to a single term*** using

$$\neg(\neg p) = p$$

$$\neg(a \vee b) = \neg a \wedge \neg b)$$

$$\neg(a \wedge b) = \neg a \vee \neg b)$$

[deMorgan Laws]

$$\neg\forall x: P(x) = \exists x: \neg P(x)$$

$$\neg\exists x: P(x) = \forall x: \neg P(x)$$

$$\forall x: \neg[Roman(x) \wedge know(x, Marcus)] \vee [hate(x, Caeser)$$
$$\vee (\forall y: \exists z: hate(y, z) \rightarrow thinkCrazy(x, y)]$$

- Becomes

$$\forall x: [\neg Roman(x) \vee \neg know(x, Marcus)] \vee [hate(x, Caeser)$$
$$\vee (\forall y: \forall z: \neg hate(y, z) \vee thinkCrazy(x, y)]$$

Conversion to Clause form

- Step 3: ***Standardize variables*** so that each quantifier binds to a unique variable (needed for 4)

$$\forall x: P(x) \vee \forall x: Q(x)$$

Would be converted to

$$\forall x: P(x) \vee \forall y: Q(y)$$

- Step 4: ***Move all the quantifiers to the left*** of the formula without changing their relative order
- Hence the formula becomes

$$\forall x: \forall y: \forall z: [\neg Roman(x) \vee \neg \text{know}(\text{x, Marcus})] \vee [\text{hate (x, Caeser)} \vee (\neg hate(y, z) \vee thinkCrazy(x, y)]$$

- The above is in Prenex normal form: Consists of prefix of quantifiers, followed by a matrix (quantifier-free!)

Conversion to Clause form

- Step 5: ***Eliminate existential quantifiers***
  - Our formula does not have one
  - ∃ : There is a value for a variable that makes the formula true
  - Replace the variable with a function

$$\exists y: President(y)$$

can be transformed to

$$President(S1)$$

SI: function that produces a value that satisfies

- Formulas like

$$\forall x : \exists y: \text{father}-\text{of}\ (\text{y}, \text{x})$$

can be changed to

$$\forall x : \text{father}-\text{of}\ (S2(\text{x}), \text{x})$$

for some function S2(x) that produces y

- S1, S2 are called **Skolem functions**
  - If no argument, called **Skolem constants**

Conversion to Clause form

- Step 6: **Drop the prefix**
  - Now, all variables are universally quantified
  - Drop the prefix , all variables are universally quantified
- Formula from 4, is changed to

$$[\neg Roman(x) \vee \neg \text{know}(\text{x, Marcus})] \vee [\text{hate (x, Caeser)} \vee (\neg hate(y,z) \vee thinkCrazy(x,y))]$$

Conversion to Clause form

- Step 7: **Convert matrix into conjunction of disjoints**
  - There are no ANDs, in our formula
  - Using associative property, of OR, we can simply remove the parentheses

$$(a \wedge b) \vee c = (a \vee c) \wedge (b \wedge c)$$

  - Many a times, even the distributive law will have to be used

$$(a \wedge b) \vee c = (a \vee c) \wedge (b \vee c)$$

Conversion to Clause form

- Step 8: **Create a separate clause corresponding to each conjunct**
  - For a wff to be true, all the clauses that are generated from it must be true
  - There are no ANDs, in our formula
- Step 9: **Standardize apart** variables in set of clauses generated in 8
  - i.e., rename variables that no two clauses make reference to same variable!

$$\forall x: P(x) \wedge \mathrm{Q}(\mathrm{x})) \; = \forall x: P(x) \wedge \forall x: \mathrm{Q}(\mathrm{x})$$

- There is no relationship between two variables as
  - Each clause is a separate conjunct
  - All variables are universally quantified

INFERENCE & UNIFICATION

Why resolution in predicate logic is difficult

- In propositional logic L and ¬ L are clear contradictions
- In predicate logic we have to consider the arguments
- man(John) and ¬ man(John) are contradictions
- man(John) and ¬ man(Spot) are not contradictions
- We need an procedure to
  - match arguments
  - Compare argument literals
  - Decide if there exists a set of substitutions making them identical
- Unification Algorithm

Resolution in predicate logic

- Substitution is a non---trivial matter - need an algorithm for Unify
  - Unification Algorithm
- UNIFY(p, q) = **unifier θ:** where **SUBST(θ, p) = SUBST(θ, q)**
  (common literal)
- Unification replaces variables:
  a. ∃x Loves (John, x)
  b. ∃x Hates (John, x)
  Are these x's in (a) and (b) the same?

- **Another example:** (Note: All unquantified variables are assumed universal)
- Consider the following

  ∀x: knows(John, x) → hates(John, x)
  knows(John, Jane)
  ∀y: knows(y, Leonid)
  ∀y: knows(y, mother(y))
  ∀x: knows(x, Elizabeth)

Resolution in predicate logic

- Unification

UNIFY(knows(John, x), knows(John, Jane)) = {Jane/x}

UNIFY(knows(John, x), knows(y, Leonid)) = {Leonid/x, John/y}

UNIFY(knows(John, x), knows(y, mother(y))) =
{John/y, mother(John)/x}

UNIFY(knows(John, x), knows(x, Elizabeth)) = **FAIL**

Standardizing apart:

UNIFY(knows(John, x), knows(y, Elizabeth)) = {John/y, Elizabeth/x}

Resolution Theorem Proving (FOL)

- Convert everything to CNF
- Resolve, with unification
  - Save bindings as you go!
- If resolution is successful, proof succeeds
- If there was a variable in the item to prove, return variable's value from unification bindings

Converting to CNF

1. Replace implication ($A \Rightarrow B$) by $\neg A \vee B$
2. Move $\neg$ "inwards"
   - $\neg \forall x\ P(x)$ is equivalent to $\exists x\ \neg P(x)$ & vice versa
3. Standardize variables
   - $\forall x\ P(x) \vee \forall x\ Q(x)$ becomes $\forall x\ P(x) \vee \forall y\ Q(y)$
4. Skolemize
   - $\exists x\ P(x)$ becomes $P(A)$
5. Drop universal quantifiers
   - Since all quantifiers are now $\forall$, we don't need them
6. Distributive Law

Convert to FOPL, then CNF

1. John likes all kinds of food
2. Apples are food.
3. Chicken is food.
4. Anything that anyone eats and isn’t killed by is food.
5. Bill eats peanuts and is still alive.
6. Sue eats everything Bill eats.

Prove Using Resolution

1. John likes peanuts.
2. Sue eats peanuts.
3. Sue eats apples.
4. What does Sue eat?
   - Translate to Sue eats X
   - Result is a valid binding for X in the proof

PROPOSITIONAL RESOLUTION

Inference Methods (Review)

- Unification (prerequisite)
- Forward Chaining
  - Production Systems
  - RETE Method (OPS)
- Backward Chaining
  - Logic Programming (Prolog)
- Resolution
  - Transform to CNF
  - Generalization of Prop. Logic resolution

Resolution (review)

- Resolution allows a complete inference mechanism (search-based) using only one rule of inference
- Resolution rule:
  - Given: $P_1 \vee P_2 \vee P_3 \ldots \vee P_n$, and $\neg P_1 \vee Q_1 \ldots \vee Q_m$
  - Conclude: $P_2 \vee P_3 \ldots \vee P_n \vee Q_1 \ldots \vee Q_{mA}$

    Complementary literals P_1 and $\neg P_1$ "cancel out"
- To prove a proposition F by resolution,
  - Start with $\neg F$
  - Resolve with a rule from the knowledge base (that contains F)
  - Repeat until all propositions have been eliminated
  - If this can be done, a contradiction has been derived and the original proposition F must be true.

Propositional Logic Resolution

- Procedure for producing proof by resolution of proposition P with respect to a set of axioms F
- **Algorithm: Propositional Resolution**

1. Convert all the propositions of F to clause form
2. Negate P and convert the result to clause form. Add it to the set of clauses obtained in step 1.
3. Repeat until either a contradiction is found or no progress can be made:

   a) Select two clauses. Call these the parent clauses.

   b) Resolve them together.

      The resulting clause, called the **resolvent**

      It will be **the disjunction of all of the literals of both of the parent clauses** with the following exception:

      If there are any pairs of literals L and ¬L such that one of the parent clauses contains L and the other contains ¬L, then select one such pair and eliminate both L and ¬L from the resolvent

   c) If the resolvent is the empty clause, then a contradiction has been found. If it is not then add it to the set of clauses available to the procedure.

Resolution

- We need a proof procedure that carries out
  - Variety of processes involved in reasoning with statements in predicate logic
  - In a single operation
- Resolution!
- $KB \models \alpha$ (α is a logical consequence of KB)
  How to prove it automatically?
- The basic ideas
- $KB \models \alpha \Leftrightarrow KB \wedge \neg\alpha \models false$
- **Refutation proof procedure**
  - Resolution inference rule

$$\frac{(\alpha \vee P) \wedge (\gamma \vee \neg P) \text{ premise}}{(\alpha \vee \gamma) \text{ conclusion}}$$

=> Value of P has no impact

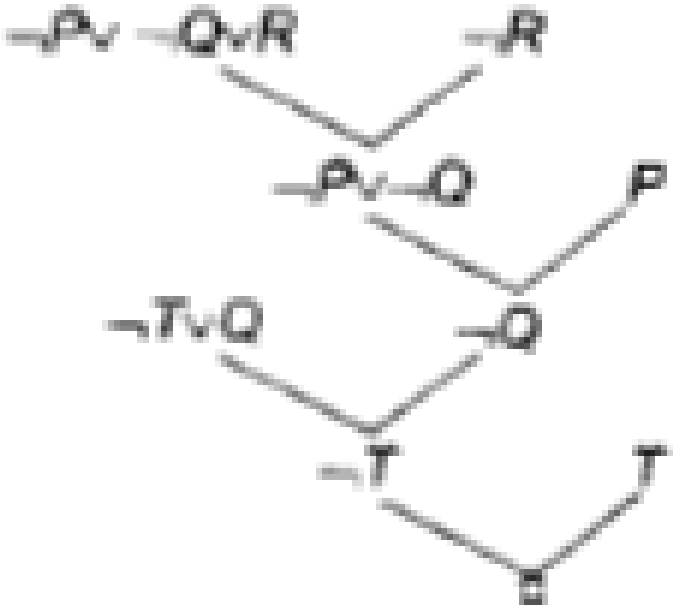
Resolution in propositional logic

1. Convert all propositions of KB to clause form (S)
2. Negate α and convert it to clause form
3. Add it to S
4. Repeat until either a contradiction is found or no progress can be made
   - Select two clauses (α ∨ P) and (γ ∨ ¬P)
   - Add the resolvent (α ∨ γ) to S

- Clause form = Conjunctive normal form – CNF

Propositional Logic Resolution

Given axioms (propositions)	Clause form	No.
P	P	(1)
$(P \wedge Q) \rightarrow R$	$\neg P \vee \neg Q \vee R$	(2)
$(S \vee T) \rightarrow Q$	$\neg S \vee Q$	(3)
Separate (3) in CF	$\neg T \vee Q$	(4)
T	T	(5)



- □ means that a contradiction is reached
- We want to prove R
- We started with ¬ R and combined all true propositions , thus our initial assumption was wrong and R is true

Propositional Resolution Example -2

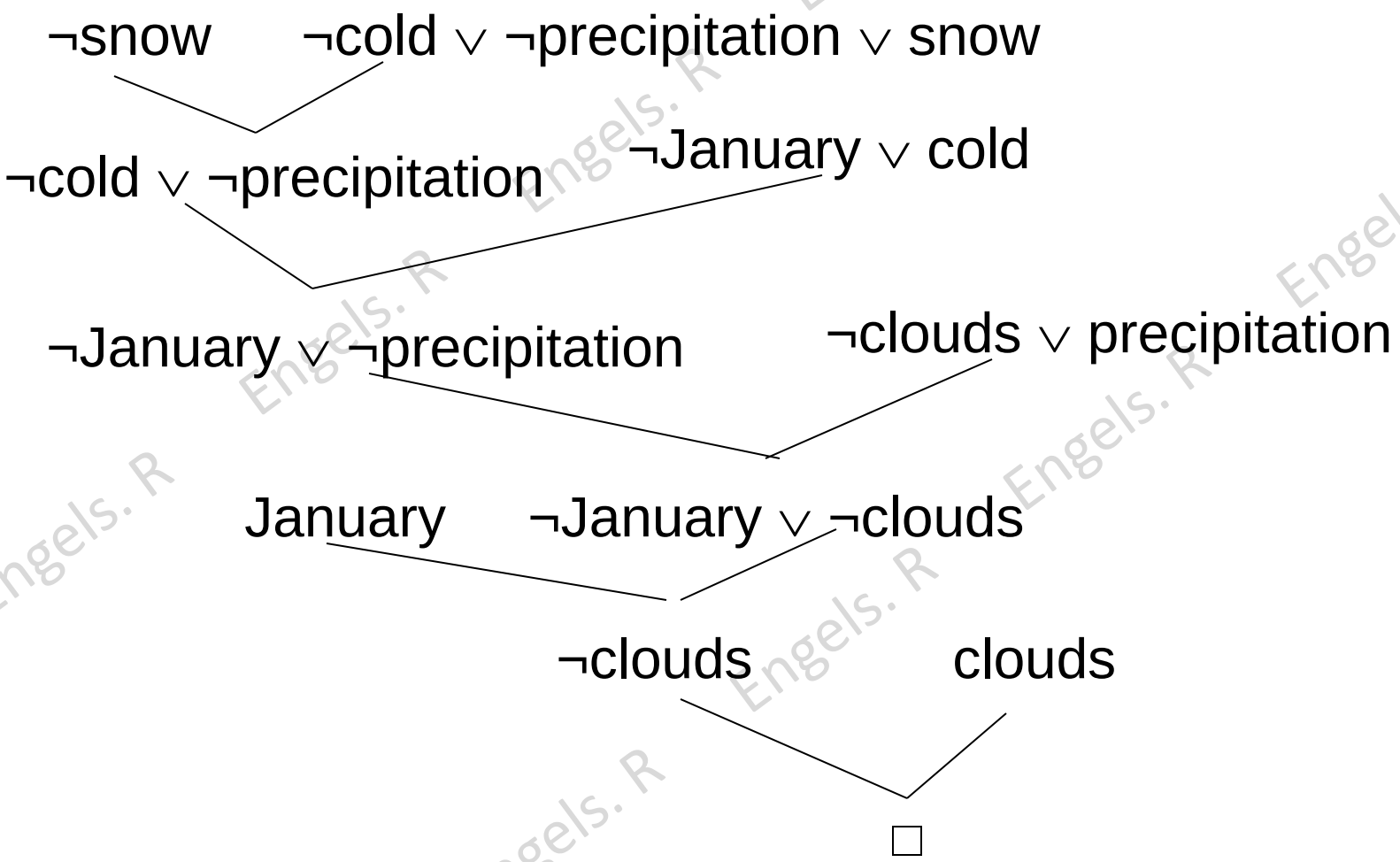
- Rules
  - Cold and precipitation -> snow

    $\neg cold \vee \neg precipitation \vee snow$
  - January -> cold

    $\neg January \vee cold$
  - Clouds -> precipitation

    $\neg clouds \vee precipitation$
- Facts
  - January, clouds
- Prove
  - snow

Proving “snow”



RESOLUTION IN PREDICATE LOGIC

Resolution

- Resolution and CNF
- Resolution is a single rule of inference that can operate efficiently on a special form of sentences
- The special form is called conjunctive normal form (CNF) or clausal form, and has the properties:
  - Every sentence is a disjunction (OR) of literals (clauses)
  - All sentences are implicitly conjuncted (ANDed)
- Resolve arguments to predicates, to know when two literals match and can be used by resolution
- For example, does the literal **Father(Bill, Chelsea)** match **Father(x, y)** ?
  - The answer depends on how we substitute values for variables

Proof procedure for predicate logic

- Same idea as propositional logic resolution, but a few added complexities:
- Conversion to CNF is much more complex
- Matching of literals requires providing a matching of variables, constants and/or functions
- ¬ Skates(x) ∨ LikesHockey(x)
- ¬ LikesHockey(y)
- We can resolve these only if we assume x and y refer to the same object.

Predicate Logic and CNF

We can now state the resolution algorithm for predicate logic as follows, assuming a set of given statements F and a statement to be proved P:

Algorithm: Resolution

1. Convert all the statements of F to clause form.
2. Negate P and convert the result to clause form. Add it to the set of clauses obttfHied in 1.
3. Repeat until either a contradiction is found, no progress can be made, or a prede- termined amount of effort has been expended.
   (a) Select two clauses. Call these the parent clauses.
   (b) Resolve them together. The resolvent will be the disjunction of all the literals of both parent clauses with appropriate substitutions performed and with the following exception: If there is one pair of literals $T1$ and $\neg T2$ such that one of the parent clauses contains $T2$ and the other contains $T1$ and if $T1$ and $T2$ are unifiable, then neither $T1$ nor $T2$ should appear in the resolvent. We call $T1$ and $T2$ *Complementary literals*. Use the substitution produced by the unification to create the resolvent. If there is more than one pair of complementary literals, only one pair should be omitted from the resolvent.
   (c) If the resolvent is the empty clause, then a contradiction has been found. If it is not, then add it to the set of clauses available to the procedure.

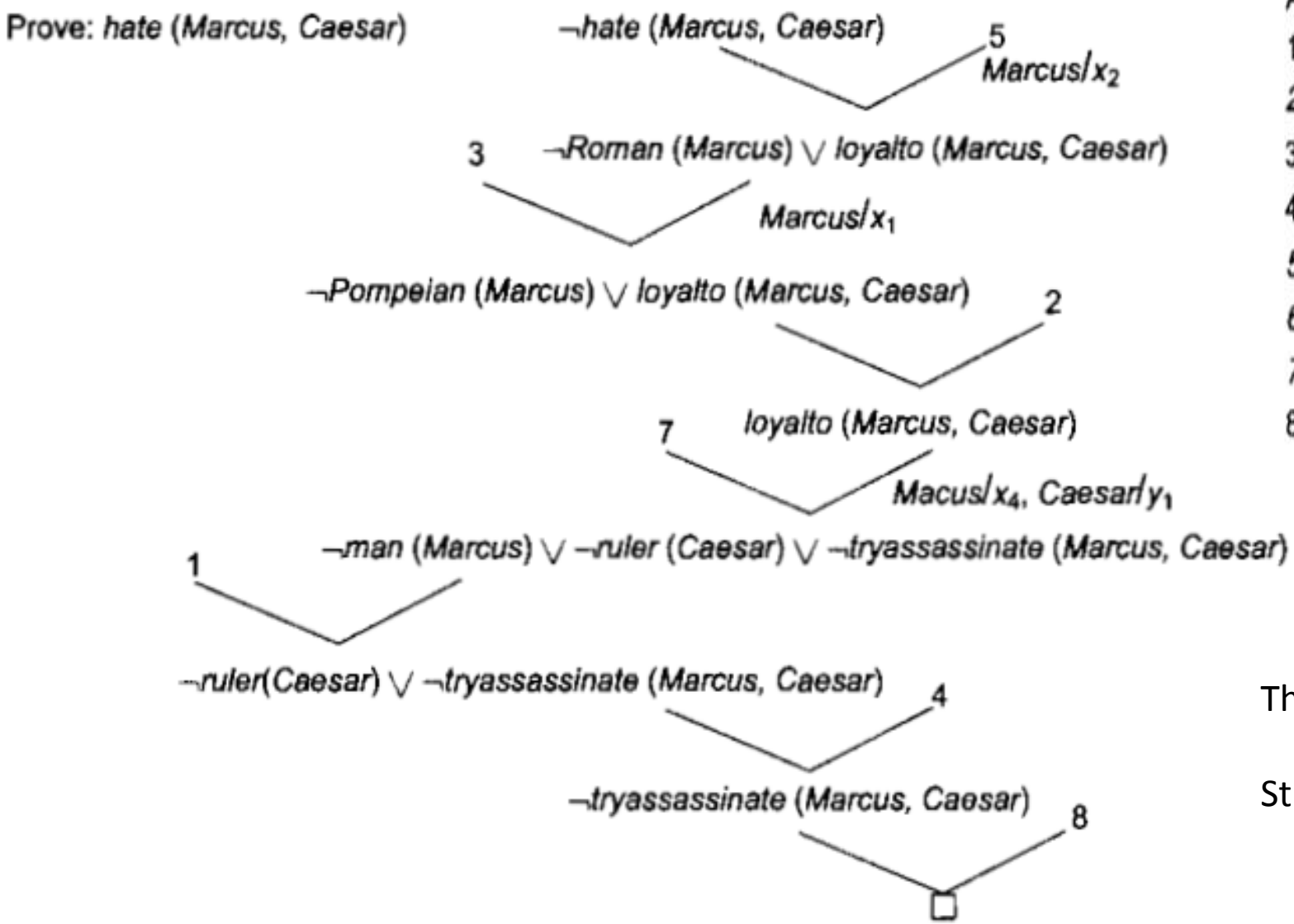
Proof procedure for predicate logic

Axioms in clause form:

1. $man(Marcus)$
2. $Pompeian(Marcus)$
3. $\neg Pompeian(x_1) \vee Roman(x_1)$
4. $ruler(Caesar)$
5. $\neg Roman(x_2) \vee loyalto(x_2, Caesar) \vee hate(x_2, Caesar)$
6. $loyalto(x_3, f(x_3))$
7. $\neg man(x_4) \vee \neg ruler(y_1) \vee \neg tryassassinate(x_4, y_1) \vee loyalto(x_4, y_1)$
8. $tryassassinate(Marcus, Caesar)$

Prove: *hate* (*Marcus*, *Caesar*)

Proof procedure for predicate logic



Axioms in clause form:

1. $man(Marcus)$
2. $Pompeian(Marcus)$
3. $\neg Pompeian(x_1) \vee Roman(x_1)$
4. $ruler(Caesar)$
5. $\neg Roman(x_2) \vee loyalto(x_2, Caesar) \vee hate(x_2, Caesar)$
6. $loyalto(x_3, f(x_3))$
7. $\neg man(x_4) \vee \neg ruler(y_1) \vee \neg tryassassinate(x_4, y_1) \vee loyalto(x_4, y_1)$
8. $tryassassinate(Marcus, Caesar)$

This is one of the many possible proof steps

Strategies exist to speed up this process

2nd Example

Example: Resolution for predicate logic

Anyone passing his history exams and winning the lottery is happy.

$\forall$ **X (pass (X,history)** $\wedge$ **win (X,lottery)** $\rightarrow$ **happy (X))**

Anyone who studies or is lucky can pass all his exams.

$\forall$ **X** $\forall$ **Y (study (X)** $\vee$ **lucky (X)** $\rightarrow$ **pass (X,Y))**

John did not study but he is lucky.

$\neg$ **study (john)** $\wedge$ **lucky (john)**

Anyone who is lucky wins the lottery.

$\forall$ **X (lucky (X)** $\rightarrow$ **win (X,lottery))**

These four predicate statements are now changed to clause form (Section 12.2.2):

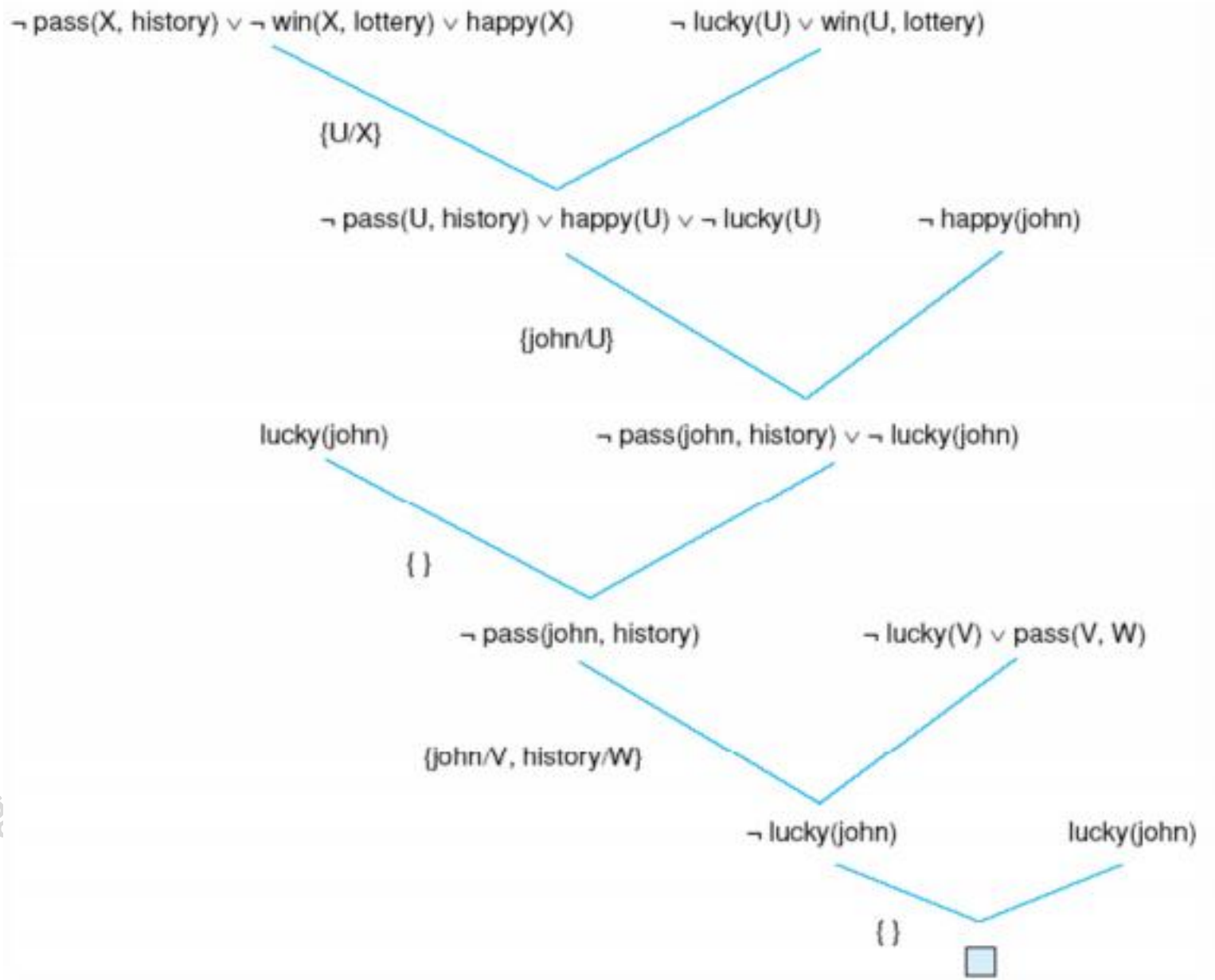
1. $\neg$ **pass (X, history)** $\vee$ $\neg$ **win (X, lottery)** $\vee$ **happy (X)**
2. $\neg$ **study (Y)** $\vee$ **pass (Y, Z)**
3. $\neg$ **lucky (W)** $\vee$ **pass (W, V)**
4. $\neg$ **study (john)**
5. **lucky (john)**
6. $\neg$ **lucky (U)** $\vee$ **win (U, lottery)**

Into these clauses is entered, in clause form, the negation of the conclusion:

7. $\neg$ **happy (john)**

From https://www.csd.uwo.ca/~mmorenom/cs2209_moreno/slide/lec16-predicate-resolution.pdf

2nd Example



From https://www.csd.uwo.ca/~mmorenom/cs2209_moreno/slide/lec16-predicate-resolution.pdf

IMPLEMENTATION CHALLENGES

Representing sets of objects – Contd.

- Need to be careful when we attach properties to the Set
- Is the property being attached to the Set itself?
  - Ex: Large(Elephant). The set of elements is large in size (has many member elements)
- Or is it being attached to the elements of the set?
  - Ex: Large(Elephant). Every elephant is large

Implementation issues

- All discussed logics have their weaknesses
- They fail to deal with four real world problems
- How to derive exactly only the non-monotonic conclusions that are relevant to the problem?
- How to update KB incrementally as problem solving progresses?
  - Truth status of a proposition is based on current facts in KB
  - A change in KB may cause truth status to change!

Implementation issues

- Many extensions can become invalid when a new fact is identified
  - To find the extensions that change, we need a search process
- None of the logics are computationally effective
  - Most are not decidable, some are at best semi-decidable (only in proposition form)
  - None is efficient
- Reasoning component needs to be separated into two parts
  - A problem solver to draw conclusions
  - A Truth Maintenance System (TMS) to help update knowledge incrementally in KB without affecting problem solving

Augmenting a problem-solver

- Use forward or backward reasoning
- Forward reasoning
  - Reason forward from what is known (just like for monotonic system)
  - To support non-monotonicity, typically an unless clause is introduced
- Reasoning backward
  - To determine whether some expression P is true
  - Allow default clauses in backward rules
    - Use rule ordering to resolve conflicts
  - Construct a debate: for P and against P
    - Based on further evidence, decide on P

FORWARD AND BACKWARD CHAINING

Sample 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 Colonel West is a criminal

Forward chaining algorithm

function FOL-FC-ASK(KB, α) **returns** a substitution or *false*

repeat until *new* is empty
  new $\leftarrow \{\}$
  for each sentence r **in** KB **do**
    $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \leftarrow$ STANDARDIZE-APART(r)
    for each θ such that $(p_1 \wedge \ldots \wedge p_n)\theta = (p'_1 \wedge \ldots \wedge p'_n)\theta$
        for some $p'_1, \ldots, p'_n$ in KB
      $q' \leftarrow$ SUBST(θ, q)
      if q' is not a renaming of a sentence already in KB or *new* **then do**
        add q' to *new*
        $\phi \leftarrow$ UNIFY(q', α)
        if ϕ is not *fail* **then return** ϕ
  add *new* to KB
return *false*

Sample knowledge base

- 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., ∃x Owns(Nono,x) ∧ Missile(x):

$$Owns(Nono, M1) \wedge Missile(M1)$$

- All of its missiles were sold to it by Colonel West

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

Sample knowledge base

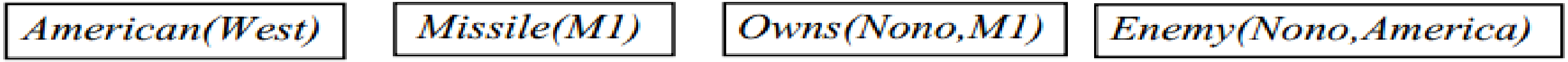
- All Romans who know Marcus either hate Caeser or think that anyone who hates anyone is crazy
  - Can be represented as

$$\forall x : [Roman(x) \wedge \text{know}(\text{x}, \text{Marcus})]$$
$$\rightarrow [\text{hate (x, Caeser)} \vee (\forall y: \exists z: hate(y, z) \rightarrow thinkCrazy(x, y)]$$

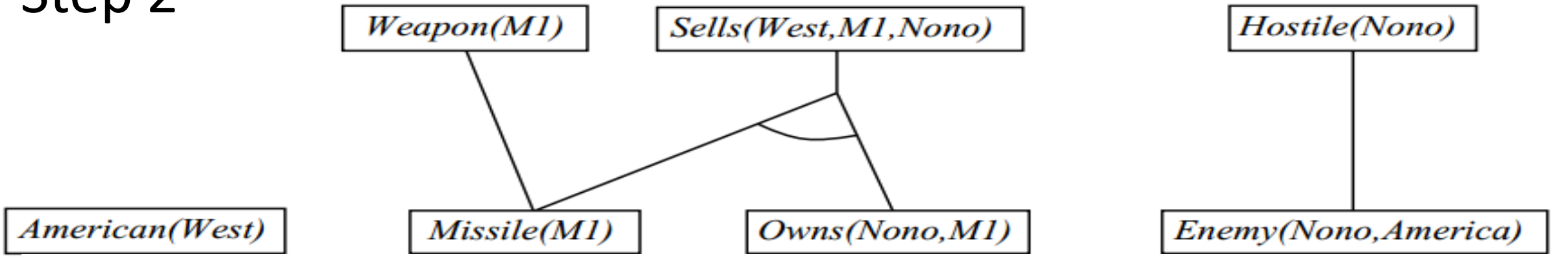
- Formula would be easier to work with if
  1. It were flatter, i.e., less embedding of components
  2. Quantifiers are separated from the formula and need not be considered
- Conjunctive normal Form (CNF) has both!

Forward chaining proof

Step 1



Step 2



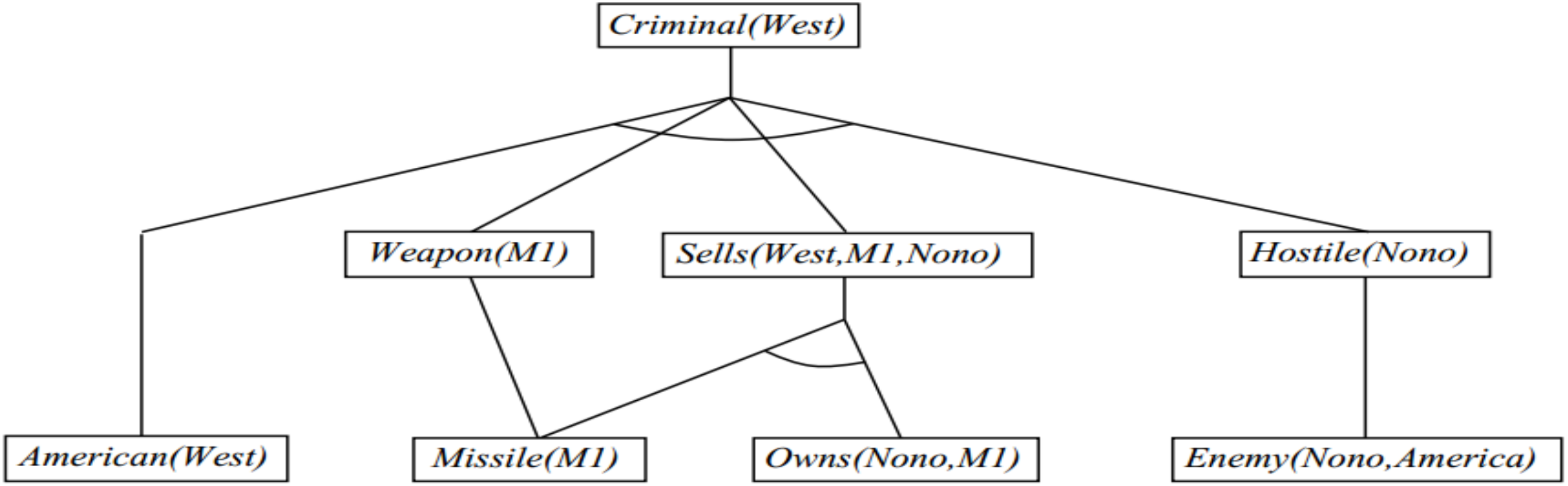
$Enemy(x, America) \Rightarrow Hostile(x)$

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

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

Forward chaining proof

Step 3



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

Backward chaining

Idea: work backwards from the query *q*:

to prove *q* by BC,

check if *q* is known already, or

prove by BC all premises of some rule concluding *q*

Avoid loops: check if new subgoal is already on the goal stack

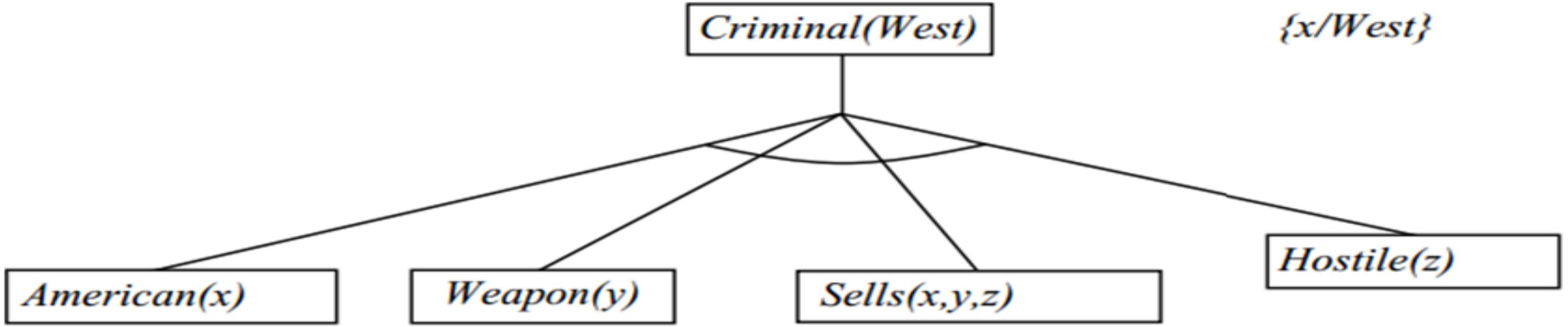
Avoid repeated work: check if new subgoal

1. has already been proved true, or
2. has already failed

Backward chaining proof

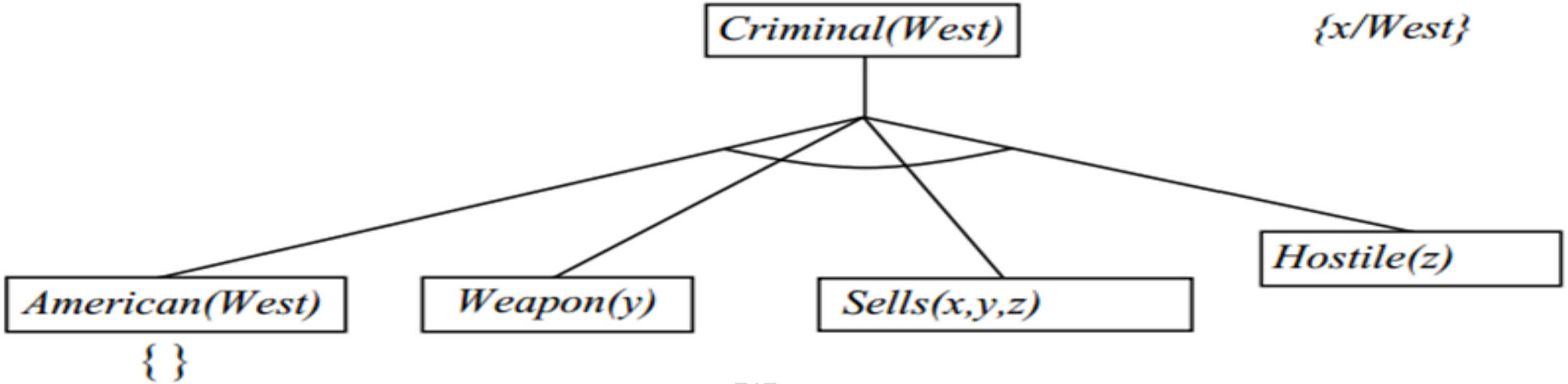
Step 1

Step 2



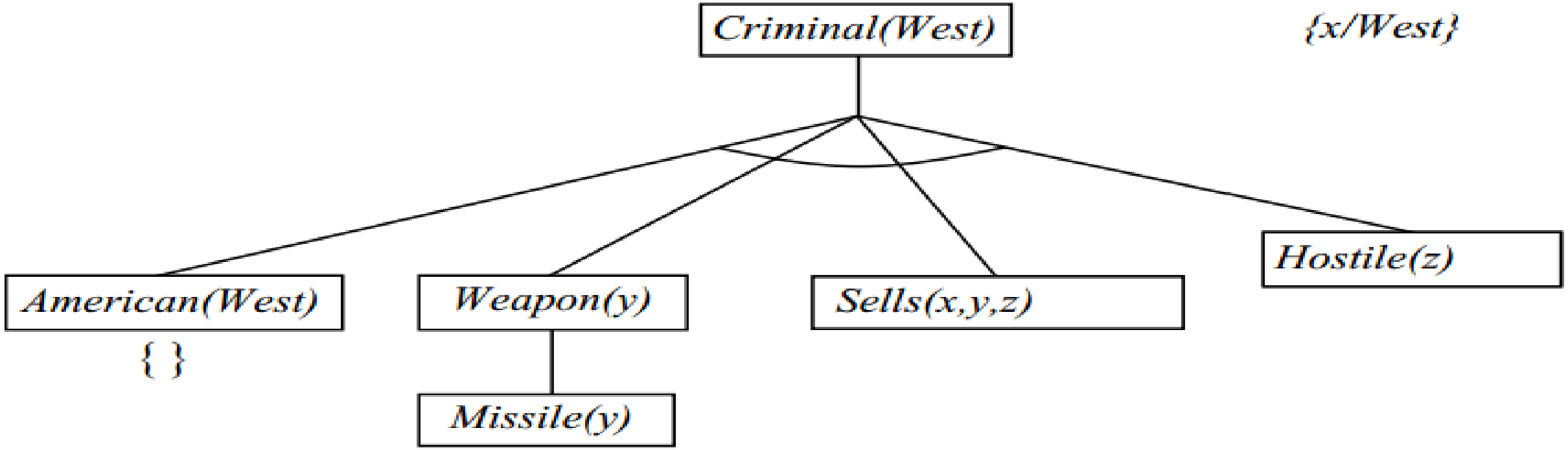
Backward chaining proof

Step 3



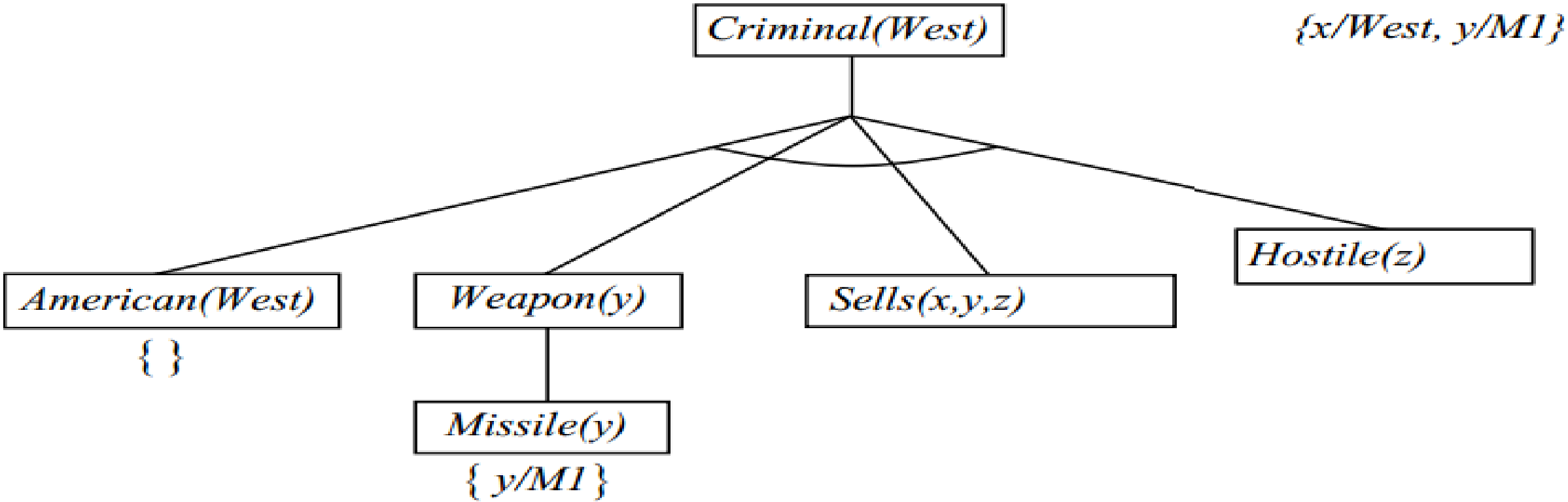
Backward chaining proof

Step 4



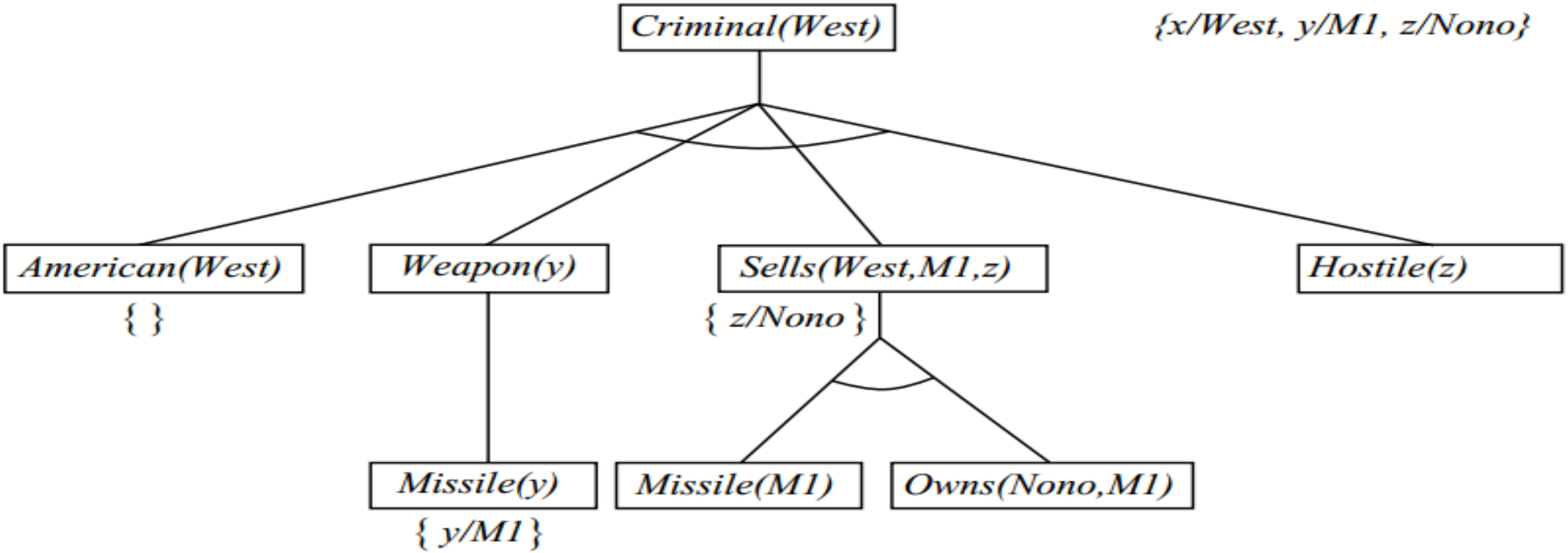
Backward chaining proof

Step 5



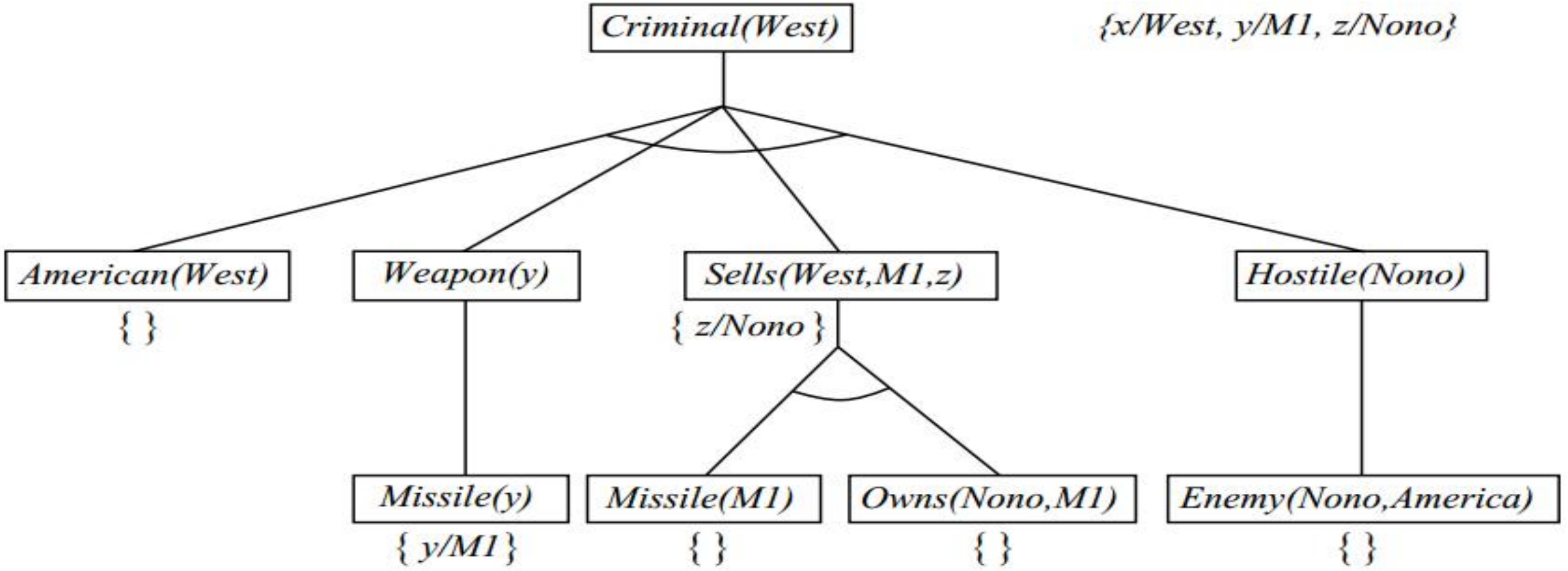
Backward chaining proof

Step 6



Backward chaining proof

Step 7



STATISTICAL REASONING, PROBABILITY AND BAYES THEOREM

Probability and Bayes' Theorem

- 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., ∃x Owns(Nono,x) ∧ Missile(x):

$$Owns(Nono, M1) \wedge Missile(M1)$$

- All of its missiles were sold to it by Colonel West

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

Probability and Bayes' Theorem

- All Romans who know Marcus either hate Caeser or think that anyone who hates anyone is crazy
  - Can be represented as

$$\forall x : [Roman(x) \wedge \text{know}(\text{x}, \text{Marcus})]$$
$$\rightarrow [\text{hate (x, Caeser)} \vee (\forall y: \exists z: hate(y, z) \rightarrow thinkCrazy(x, y)]$$

- Formula would be easier to work with if
  1. It were flatter, i.e., less embedding of components
  2. Quantifiers are separated from the formula and need not be considered
- Conjunctive normal Form (CNF) has both!

Probability and Bayes' Theorem

- Using CNF, the formula can be represented as

$$\neg Roman(x) \wedge \neg \text{know}(\text{x}, \text{Marcus}) \vee$$
$$\text{hate (x, Caeser)} \vee \neg hate(y, z) \vee thinkCrazy(x, y)$$

- WFF to clause form does not lose information
- To use resolution, need to convert the set of WFFs to set of CNF clauses
  - No instances of connector A
- Can be achieved by
  - Convert WFFs to CNF expressions
  - Break apart each such expression into clauses , one for each conjunct

Probability and Bayes' Theorem

- Step 1: ***Eliminate all*** →, using (a → b is equivalent to ¬a ∨ b.

$\forall x : [Roman(x) \wedge \text{know}(x, \text{Marcus})]$
$\rightarrow [\text{hate}(x, \text{Caeser}) \vee (\forall y: \exists z: hate(y, z) \rightarrow thinkCrazy(x, y)]$

- Becomes

$\forall x: \neg[Roman(x) \wedge \text{know}(x, \text{Marcus})] \vee [\text{hate}(x, \text{Caeser})$
$\vee (\forall y: \exists z: hate(y, z) \rightarrow thinkCrazy(x, y)]$

Probability and Bayes' Theorem

- Step 2: ***Reduce the scope of each ¬ to a single term*** using

$$\neg(\neg p) = p$$

$$\neg(a \vee b) = \neg a \wedge \neg b)$$

$$\neg(a \wedge b) = \neg a \vee \neg b)$$

$$\neg \forall x: P(x) = \exists x: \neg P(x)$$

$$\neg \exists x: P(x) = \forall x: \neg P(x)$$

$$\forall x: \neg[Roman(x) \wedge know(x, Marcus)] \vee [hate(x, Caeser) \vee (\forall y: \exists z: hate(y, z) \rightarrow thinkCrazy(x, y))]$$

- Becomes

$$\forall x: [\neg Roman(x) \vee \neg know(x, Marcus)] \vee [hate(x, Caeser) \vee (\forall y: \forall z: \neg hate(y, z) \vee thinkCrazy(x, y))]$$

Probabilistic Graphical Models

- Algebraic probability analysis can be augmented using diagrammatic representations of probability distributions
  - Called probabilistic graphical models
- Advantages
  - They provide a simple way to visualize the structure of a probabilistic model
    - Can be used to design and motivate new models
  - Insights into the properties of the model can be obtained by inspection of the graph
    - Example: Conditional independence properties
  - Complex computations, required to perform inference and learning in sophisticated models, can be expressed in terms of graphical manipulations
    - Underlying mathematical expressions are carried along implicitly

Probabilistic Graphical Models

- A graph comprises nodes (vertices) connected by links (edges, arcs)
  - Each node represents a random variable (or group of random variables)
  - Links express probabilistic relationships between these variables
- Graph captures the way in which the joint distribution over all of the random variables can be decomposed into a product of factors
  - These product of factors depend only on a subset of the random variables in the joint distribution
- A specific graph can make probabilistic statements for a broad class of distributions

Probabilistic Graphical Models

- PGM has two broad classes
- Directed GM or Bayesian networks
  - Directed links with arrows
  - Useful for expressing causal relationships between random variables
- Undirected GM or Markov random fields
  - Undirected links, no directional significance
  - Suited to expressing soft constraints between random variables
- To solve inference problems, both types may be converted into another representation known as Factor graph

PGM – Bayes Network

- Consider an arbitrary joint distribution $p(a, b, c)$ over three variables a, b, and c
- By application of the product rule of probability, can write the joint distribution as

$$p(a, b, c) = p(c|a, b)p(a, b)$$

- A second application of the product rule, this time to the second term on the right hand side, gives

$$p(a, b, c) = p(c|a, b)p(b|a)p(a)$$

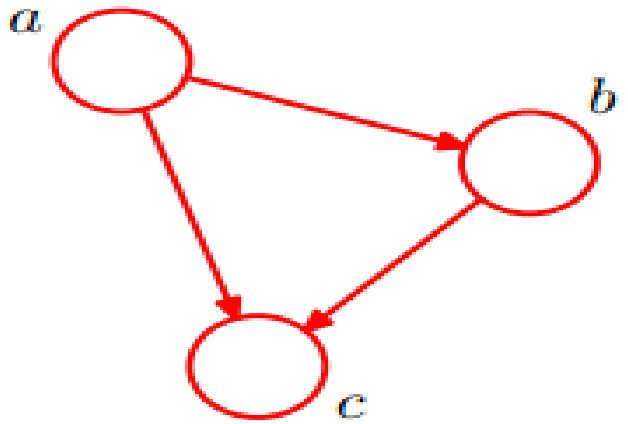
  - Note that this decomposition holds for any choice of the joint distribution

PGM – Bayes Network

- Represent $p(a,b,c) = p(c|a,b)p(b|a)p(a)$ using a simple graphical model
  - Introduce a node for each of the random variables *a*, *b*, and *c*
  - Associate each node with corresponding conditional distribution on the right-hand side of the equation
  - For each conditional distribution, add directed links (arrows) to graph from nodes corresponding to variables on which the distribution is conditioned
- Thus for the factor $p(c|a,b)$, there will be links from nodes *a* and *b* to node *c*, whereas for the factor $p(a)$ there will be no incoming links.

PGM – Bayes Network

- Represent $p(a, b, c) = p(c|a, b)p(b|a)p(a)$ using a simple graphical model



- If there is a link from node *a* to a node *b* then
  - Node *a* is the parent of node *b*
  - Node *b* is the child of node *a*
- Note
  - No distinction between a node and the variable
  - Directed Acyclic Graph (DAG)

PGM – Bayes Network

$$p(a,b,c) = p(c|a,b)p(b|a)p(a)$$

- Note: LHS is symmetrical with respect to the three variables *a*, *b*, and *c* and RHS is not!
  - In making the decomposition, implicitly chosen a particular ordering, namely *a, b, c*
  - Had a different ordering was chosen, a different decomposition would have been obtained
    - And hence a different graphical representation!

PGM – Bayes Network

- Conside the joint distribution over *K* variables given by $p(x_1, \ldots, x_K)$
- By repeated application of the product rule of probability, this joint distribution can be written as a product of conditional distributions, one for each of the variables

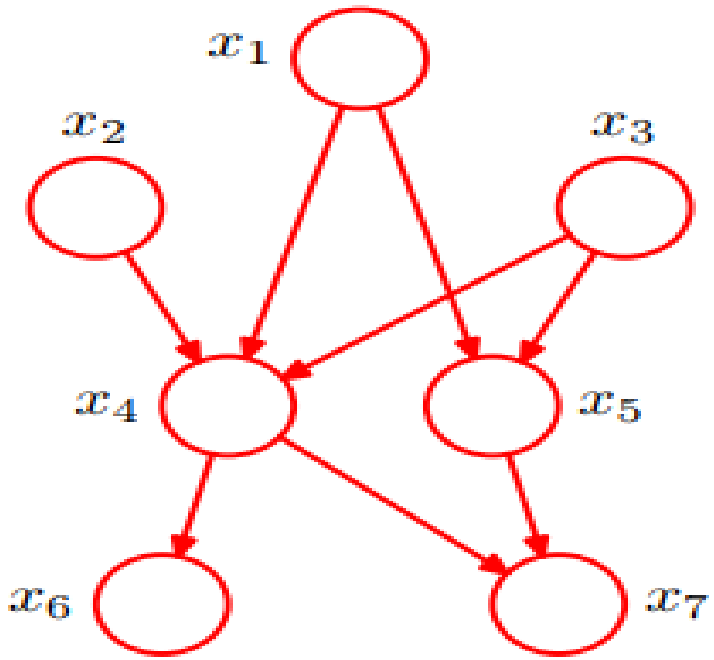
$$p(x_1, \ldots, x_K) = p(x_K|x_1, \ldots, x_{K\_1}) \ldots p(x_2|x_1)p(x_1)$$

- can again represent this as a directed graph having *K* nodes, one for each conditional distribution on RHS
  - Each node has incoming links from all lower numbered nodes (fully connected)

PGM – Bayes Network

- Example of a <u>non</u> fully connected directed acyclic graph describing the joint distribution over variables $x_1, \ldots, x_7$

$$p(x_1)\,p(x_2)\,p(x_3)\,p(x_4|x_1,x_2,x_3)$$
$$p(x_5|x_1,x_3)p(x_6|x_4)p(x_7|x_4,x_5)$$



PGM – Bayes Network

- The absence of links in the graph that conveys interesting information about the properties of the class of distributions that the graph represents
- Each conditional distribution is conditioned only on the parents of the corresponding node in the graph
  - For instance, x_5 will be conditioned only on x_1 and x_3

PGM – Bayes Network

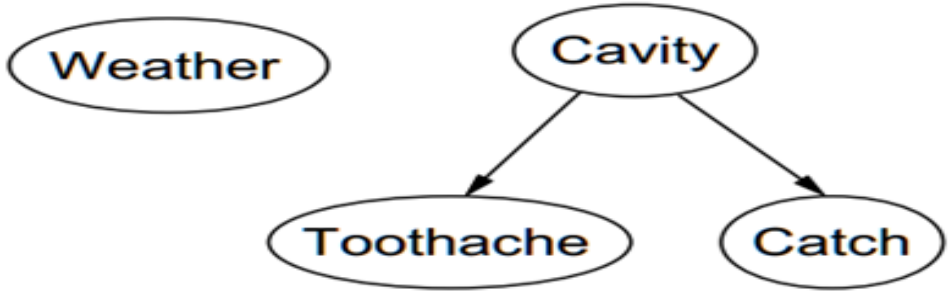
- The joint distribution defined by a graph is given by the product, over all of the nodes of the graph, of a conditional distribution for each node conditioned on the variables corresponding to the parents of that node in the graph
- For a graph with *K* nodes, the joint distribution is

$$p(\mathbf{x}) = \prod_{k=1}^{K} p(x_k|\mathrm{pa}_k)$$

  - where pa_k denot
  - and $x = \{x_1, \ldots, x_K\}$
  - This equation expresses the factorization properties of the joint distribution for a directed graphical model

Bayesian networks

- A data structure to
  - Represent the dependencies among conditional and independent variables
  - Give a concise specification of any full joint probability distribution
- Examples
  - Weather is independent of other variables
  - Toothache and catch and conditionally dependent on cavity



Bayesian networks

- Step 6: **Drop the prefix**
  - Now, all variables are universally quantified
  - Drop the prefix , all variables are universally quantified
- Formula from 4, is changed to

$$[\neg Roman(x) \vee \neg \text{know}(\text{x}, \text{Marcus})] \vee [\text{hate}\ (\text{x}, \text{Caeser}) \vee\ (\neg hate(y,z) \vee thinkCrazy(x,y))]$$

Example

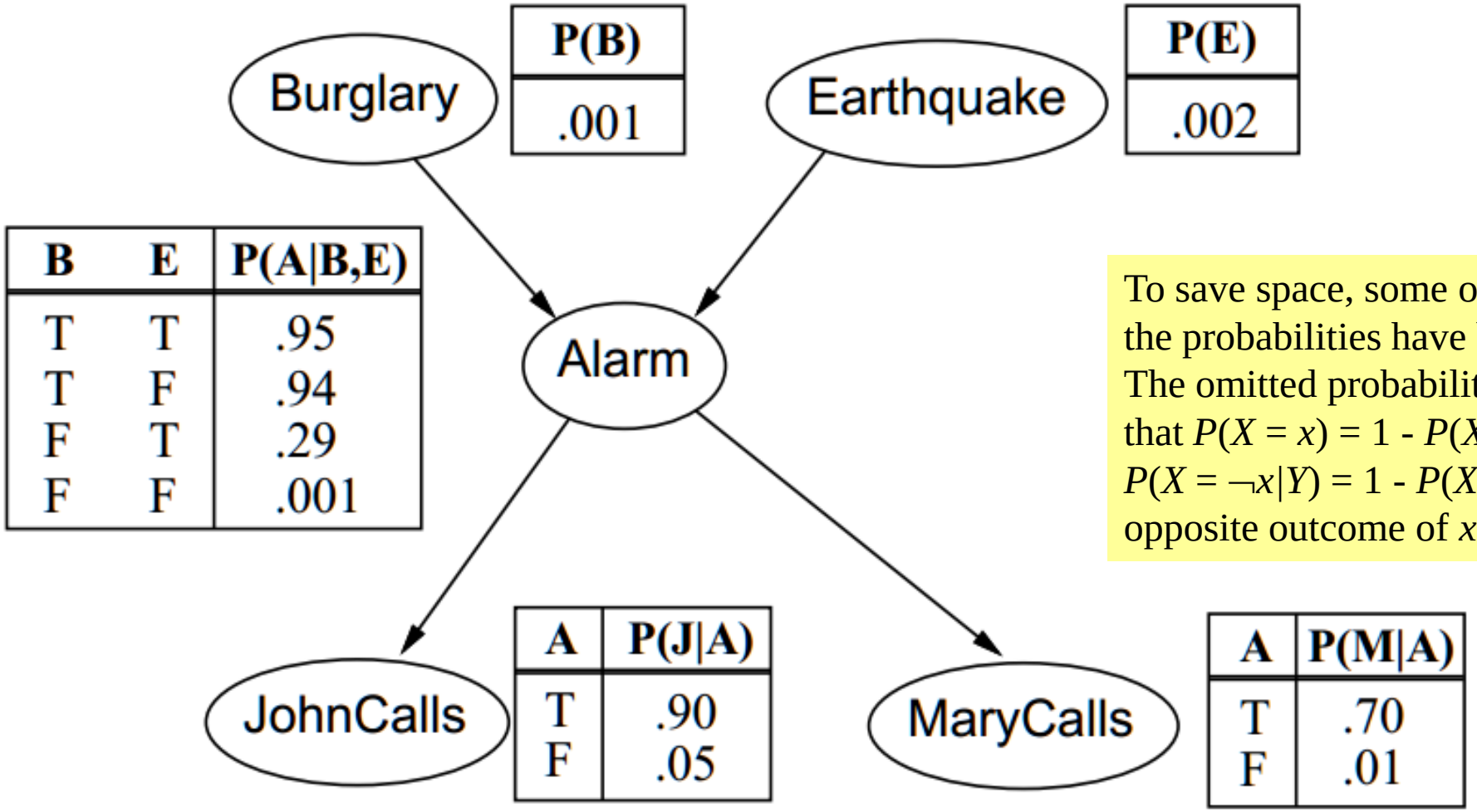
- Consider that you are at work.
- You have set up an alarm in your house for ringing when there is a burglar in the house.
- Your neighbour John calls to say your alarm is ringing, but your other neighbour Mary doesn't call.
- Sometimes the alarm is set off by minor earthquakes.
- John always calls when he hears the alarm, but sometimes confuses the telephone ringing with the alarm.
- Mary likes rather loud music and sometimes misses the alarm.

Example 2

- I'm at work, neighbor John calls to say my alarm is ringing, but neighbor Mary doesn't call. Sometimes it's set off by minor earthquakes. Is there a burglar?
- John always calls when he hears the alarm, but sometimes confuses the telephone ringing with the alarm.
- Mary likes rather loud music and sometimes misses the alarm.
- Variables: *Burglary*, *Earthquake*, *Alarm*, *JohnCalls*, *MaryCalls*
- Network topology reflects "causal" knowledge:
  - A burglar can set the alarm off
  - An earthquake can set the alarm off
  - The alarm can cause Mary to call
  - The alarm can cause John to call

Example 2

- P(Alarm|Burglary) A burglary alarm system is fairly reliable at detecting burglary
- P(Alarm|Earthquake) It may also respond to minor earthquakes
- P(JohnCalls|Alarm), P(MaryCalls|Alarm) Neighbors John and Mary will call when they hear the alarm
- John always calls when he hears the alarm
- P(JohnCalls|~Alarm) He sometimes confuses the telephone with the alarm and calls
- Mary sometimes misses the alarm
- Given the evidence of who has or has not called, we would like to estimate the probability of a burglary. P(Burglary|JohnCalls,MaryCalls)



To save space, some of the probabilities have been omitted from the diagram. The omitted probabilities can be recovered by noting that $P(X = x) = 1 - P(X = \neg x)$ and $P(X = \neg x|Y) = 1 - P(X = \neg x|Y)$, where $\neg x$ denotes the opposite outcome of x.

The topology shows that burglary and earthquakes directly affect the probability of alarm, but whether Mary or John call depends only on the alarm.

Thus our assumptions are that they don't perceive any burglaries directly, and they don't confer before calling.

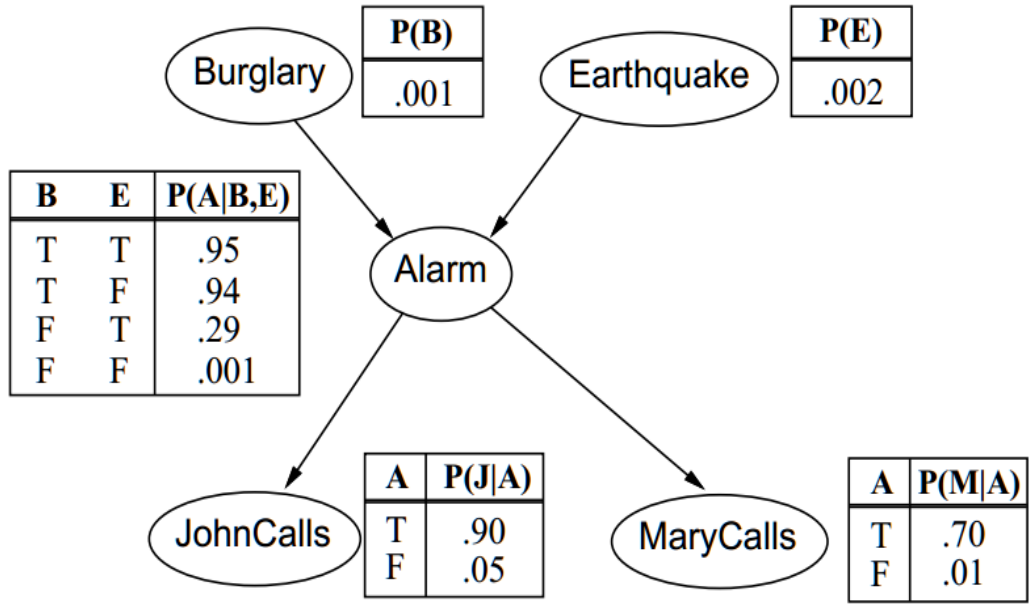
Example problem – BBN

"Global" semantics defines the full joint distribution as the product of the local conditional distributions:

$$P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i | parents(X_i))$$

- Earthquake Example (Global Semantics)
- We just proved

P(JC,MC,A,E,B) = P(JC|A) ·P(MC|A) ·P(A|B,E) ·P(E) ·P(B)



What is the probability that John calls, Mary calls, there was an alarm, but there was no burglary and no earthquake?

Example problem – BBN

What is the probability that John calls, Mary calls, there was an alarm, but there was no burglary and no earthquake?

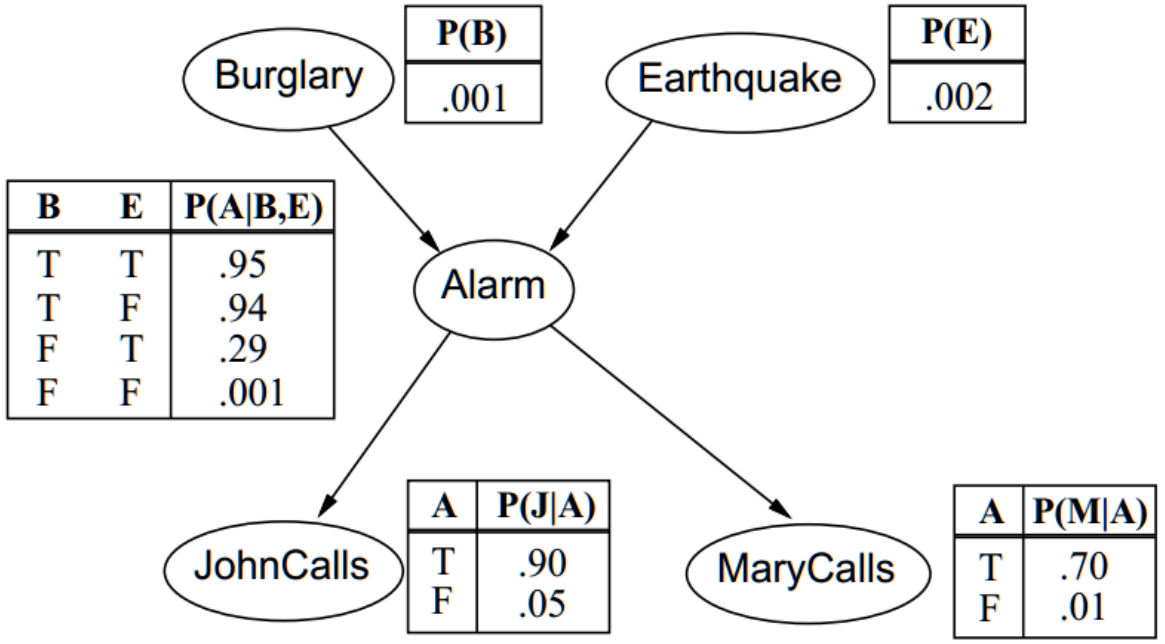
Can you rewrite the full joint distribution as product of local conditional distributions?

P(JC,MC,A,¬E,¬B)

= P(JC|A) ·P(MC|A) ·P(A|¬B, ¬E) ·P(¬E) ·P(¬B)

= 0.9 x 0.7 x 0.001 x 0.998 x 0.999

= 0.00063



Semantics

Suppose we have the variables $X_1,\ldots,X_n$.

The probability for them to have the values $x_1,\ldots,x_n$ respectively is $P(x_n,\ldots,x_1)$:

$$= P(x_n,\ldots,x_1)$$
$$= P(x_n \mid x_{n-1},\ldots,x_1)P(x_{n-1},\ldots,x_1)$$
$$= P(x_n \mid x_{n-1},\ldots,x_1)P(x_{n-1} \mid x_{n-2},\ldots,x_1)P(x_{n-2},\ldots,x_1)$$
$$= \ldots$$
$$= \prod_{i=1}^{n} P(x_i \mid x_{i-1},\ldots,x_1) = \prod_{i=1}^{n} P(x_i \mid parents(x_i))$$

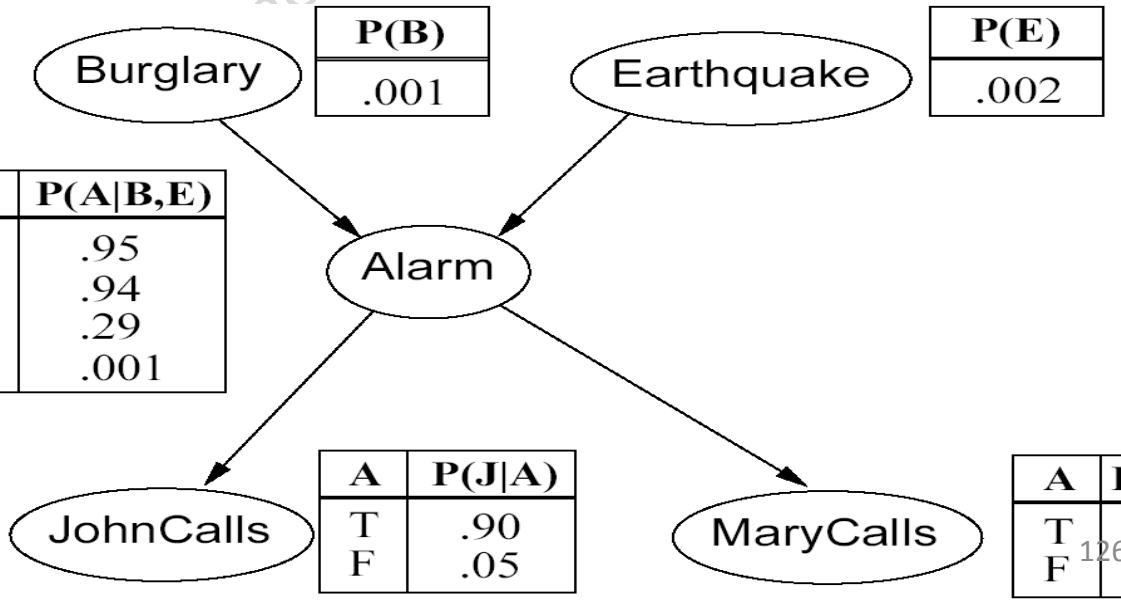
$P(x_n,\ldots,x_1)$:
is short for
$P(X_n=x_n,\ldots,X_n=x_1)$:

e.g.,

$P(j \wedge m \wedge a \wedge \neg b \wedge \neg e)$

$= P(j \mid a)\, P(m \mid a)\, P(a \mid \neg b, \neg e)\, P(\neg b)\, P(\neg e)$

$= \ldots$



Example problem – BBN

What is the probability that the alarm has sounded and there was a burglary but no earth-quake, and both Mary and John call?

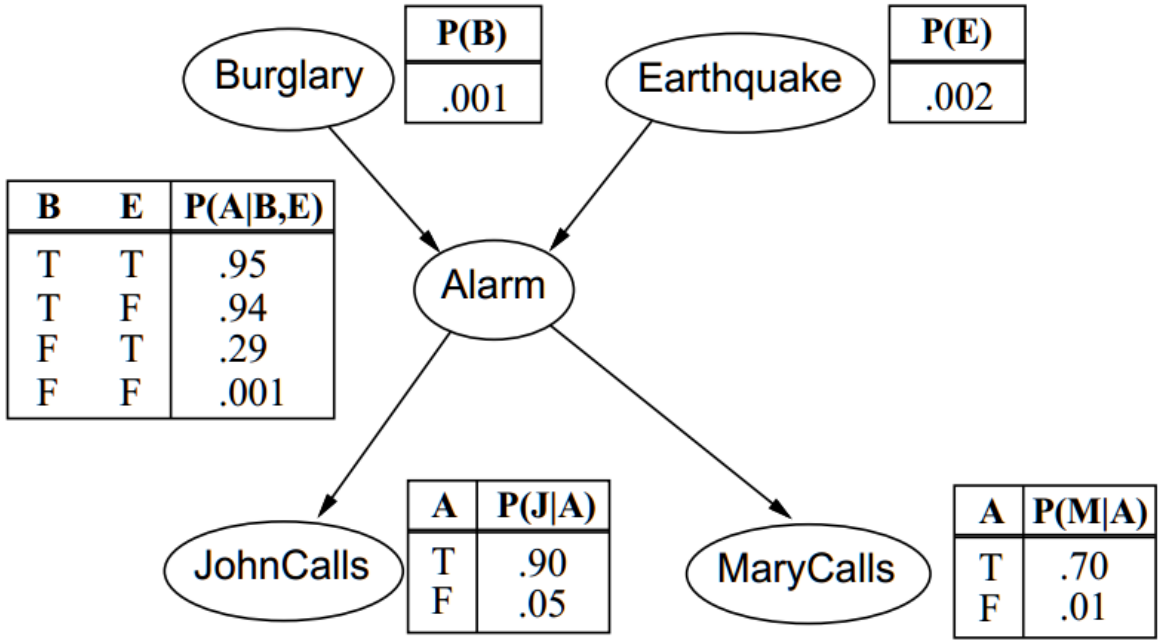
Can you rewrite the full joint distribution as product of local conditional distributions?

P(JC,MC,A,¬E,B)

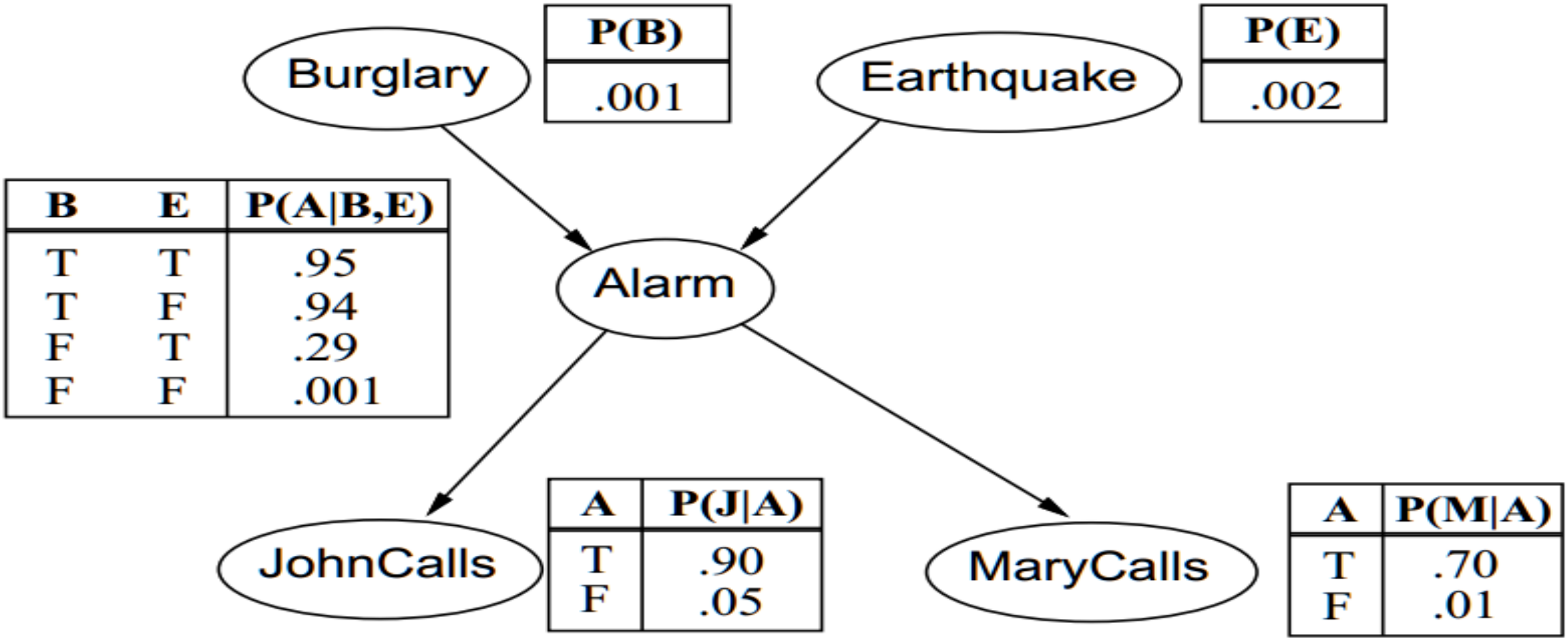
= P(JC|A) ·P(MC|A) ·P(A|B, ¬E) ·P(¬E) ·P(B)

= 0.9 x 0.7 x 0.94 x 0.998 x 0.001

= 0.000591



Bayesian networks



Bayesian networks

- Step 8: **Create a separate clause corresponding to each conjunct**
  - For a wff to be true, all the clauses that are generated from it must be true
  - There are no ANDs, in our formula
- Step 9: **Standardize apart** variables in set of clauses generated in 8
  - i.e., rename variables that no two clauses make reference to same variable!

$$\forall x: P(x) \wedge \mathrm{Q}(\mathrm{x})) = \forall x: P(x) \wedge \forall x: \mathrm{Q}(\mathrm{x})$$

- There is no relationship between two variables as
  - Each clause is a separate conjunct
  - All variables are universally quantified

DEMPSTER-SHAFER THEORY

Dempster-Shafer

- The **Dempster–Shafer theory** is designed to deal with the distinction between **uncertainty** and **ignorance**
- D-S theory treats ignorance differently compared to probability theory
- Probability theory distributes an equal amount of probability even in ignorance
  - For example, with no prior knowledge, assumes the probability P of each possibility is $P = \frac{1}{N}$, where N is the number of possibilities
  - E.g., The formula P(H) + P(H') = 1 must be enforced
- Rather than computing the probability of a proposition, DS computes the probability that the evidence supports the proposition
- This measure of belief is called a **belief function**, written Bel(X)

Dempster-Shafer

- D-S theory does not force belief to be assigned to ignorance or refutation of a hypothesis
  - Mathematical underpinnings of DS theory = similar to probability theory
  - Main difference = instead of assigning probabilities to possible worlds, the theory assigns **masses** to *sets* of possible world, that is, to events
    - Mass is assigned only to those subsets of the environment to which you wish to assign belief
    - The masses still must add to 1 over all possible events
  - Bel(A) is defined to be the sum of masses for all events that are subsets of (i.e., that entail) A, including A itself
    - Bel(A) and Bel(¬A) sum to *at most* 1
    - and the gap—the interval between Bel(A) and 1 - Bel(¬A) is often interpreted as bounding the probability of A

D-S Environment

- All Romans who know Marcus either hate Caeser or think that anyone who hates anyone is crazy
  - Can be represented as

$$\forall x : [Roman(x) \wedge \text{know}(\text{x}, \text{Marcus})]$$
$$\rightarrow [\text{hate (x, Caeser)} \vee (\forall y: \exists z: hate(y, z) \rightarrow thinkCrazy(x, y)]$$

- Formula would be easier to work with if
  1. It were flatter, i.e., less embedding of components
  2. Quantifiers are separated from the formula and need not be considered
- Conjunctive normal Form (CNF) has both!

Frame of discernment

- An environment whose elements may be interpreted as possible answers, and only one answer is correct
  - Discern: Possible to distinguish the one correct answer from all other possible answers to a question

- The power set of the environment (with 2^N subsets for a set of size N)
  - Has as its elements all answers to the possible questions of the frame of discernment

Frame of discernment

- The “frame of discernment” (or “Power set”) of Θ is the set of all possible subsets of Θ:
  - E.g., if $\Theta = \{ \theta_1, \theta_2, \theta_3 \}$
- Then the frame of discernment of Θ is:
  - $(\emptyset, \theta_1, \theta_2, \theta_3, \{\theta_1, \theta_2\}, \{\theta_1, \theta_3\}, \{\theta_2, \theta_3\}, \{ \theta_1, \theta_2, \theta_3\})$
- Ø, the empty set, has a probability of 0
  - Since at least one of the outcomes has to be true
- Each of the other elements in the power set has a probability between 0 and 1
- Probability of $\{ \theta_1, \theta_2, \theta_3\}$ is 1
  - Since at least one element has to be true

Frame of discernment - Example

- For example,
  - Θ = {airline, bomber, fighter}
  - Θ = {red, green, blue, orange, yellow}
- One way of thinking about Θ is in terms of questions and answers
- Suppose Θ = {airline, bomber, fighter}, and the questions is, "what are the military aircraft?"
- The answer is the subset of Θ
  - $\{\theta_2, \theta_3\}$ = {bomber, fighter}

Mass Function

- Using CNF, the formula can be represented as

$$\neg Roman(x) \wedge \neg \text{know}(x, \text{Marcus}) \vee$$
$$\text{hate}(x, \text{Caeser}) \vee \neg hate(y, z) \vee thinkCrazy(x, y)$$

- WFF to clause form does not lose information
- To use resolution, need to convert the set of WFFs to set of CNF clauses
  - No instances of connector A
- Can be achieved by
  - Convert WFFs to CNF expressions
  - Break apart each such expression into clauses , one for each conjunct

Mass function - properties

- Mass function m(A): (where A is a member of the power set) is the proportion of all evidence that supports this element of the power set.
- The mass m(A) of a given member of the power set, A, expresses the proportion of all relevant and available evidence that supports the claim that the actual state belongs to A but to no particular subset of A
- The value of m(A) pertains only to the set A and makes no additional claims about any subsets of A, each of which has, by definition, its own mass

Belief

- The belief in an element A of the Power set is the sum of the masses of elements which are subsets of A (including A itself)
- E.g., given A={q1, q2, q3}
- Bel(A) = m(q1)+m(q2)+m(q3)+ m({q1, q2})+m({q2, q3})+m({q1, q3}) +m({q1, q2, q3})

Disbelief

- Step 1: ***Eliminate all*** →, using (a → b is equivalent to ¬a ∨ b.

$$\forall x : [Roman(x) \wedge \text{know}(x, \text{Marcus})] \rightarrow [\text{hate}(x, \text{Caeser}) \vee (\forall y: \exists z: hate(y, z) \rightarrow thinkCrazy(x, y)]$$

- Becomes

$$\forall x: \neg[Roman(x) \wedge \text{know}(x, \text{Marcus})] \vee [\text{hate}(x, \text{Caeser}) \vee (\forall y: \exists z: hate(y, z) \rightarrow thinkCrazy(x, y)]$$

Probability Vs. D-S

- D-S theory has a major difference with probability theory which would assume that
  - P(hostile) = 0.7
  - P(non-hostile) = 1 - 0.7 = 0.3
- 0.3 in D-S theory is held as nonbelief in the environment by $m(\Theta)$
  - Means *neither belief, nor disbelief* in the evidence to a degree of 0.3.

D-S Example

- 4 people (B, J, S and K) are locked in a room when the lights go out
- When the lights come on, K is dead, stabbed with a knife.
  - Not suicide (stabbed in the back)
  - No-one entered the room.
  - Assume only one killer
- Θ = { B, J, S}
- P(Θ) = (Ø, {B}, {J}, {S}, {B,J}, {B,S}, {J,S}, {B,J,S})

D-S Example

- Detectives, after reviewing crime-scene, assign mass probabilities to various elements of power set:

Event	Mass
No one is guilty	0
B is guilty	0.1
J is guilty	0.2
S is guilty	0.1
B or J is guilty	0.1
B or S is guilty	0.1
S or J is guilty	0.3
One of the three is guilty	0.1

D-S Example: Belief

- Given the mass assignments as assigned by the detectives:

A	{B}	{J}	{S}	{B, J}	{B, S}	{J, S}	{B, J, S}
M(A)	0.1	0.2	0.1	0.1	0.1	0.3	0.1

- bel({B}) = m({B}) = 0.1
- bel({B,J}) = m({B})+m({J})+m({B,J}) = 0.1+0.2+0.1=0.4

A	{B}	{J}	{S}	{B, J}	{B, S}	{J, S}	{B, J, S}
m(A)	0.1 (0.1)	0.2	0.1	0.1	0.1	0.3	0.1
bel(A)	0.1	0.2	0.1	0.4	0.3	0.6	1.0

Plausibility

- Plausability of an element A, pl(A), is the ***sum of all masses of sets that intersect with set A***:

$$pl(\{B,J\}) = m(B) + m(J) + m(B,J) + m(B,S) + m(J,S) + m(B,J,S)$$

$$= 0.9$$

A	{B}	{J}	{S}	{B, J}	{B, S}	{J, S}	{B, J, S}
m(A)	**0.1**	**0.2**	0.1	**0.1**	**0.1**	**0.3**	**0.1**
bel(A)	0.1	0.2	0.1	0.4	0.3	0.6	1.0
pl(A)	0.4	0.7	0.6	**0.9**	0.8	0.9	1.0

D-S Example: Disbelief

- Disbelief (or Doubt) in A: dis(A) is, bel(¬A)
- Calculated by ***summing all masses of elements which <u>do not</u> intersect with A***
- Plausibility of A, $pl(A) = 1 - dis(A)$

A	{B}	{J}	{S}	{B, J}	{B, S}	{J, S}	{B, J, S}
m(A)	0.1	0.2	0.1	0.1	0.1	0.3	0.1
bel(A)	0.1	0.2	0.1	0.4	0.3	0.6	1.0
pl(A)	0.4	0.7	0.6	0.9	0.8	0.9	1.0
dis(A)	0.6	0.3	0.4	0.1	0.2	0.1	0

D-S Example: Belief interval

- The certainty associated with a given subset A is defined by the belief interval:[bel(A) pl(A)]
- E.g. the belief interval of {B,S} is: [0.3 0.8]

A	{B}	{J}	{S}	{B, J}	{B, S}	{J, S}	{B, J, S}
m(A)	0.1	0.2	0.1	0.1	0.1	0.3	0.1
bel(A)	0.1	0.2	0.1	0.4	**0.3**	0.6	1.0
pl(A)	0.4	0.7	0.6	0.9	**0.8**	0.9	1.0
dis(A)	0.6	0.3	0.4	0.1	0.2	0.1	0

Belief Intervals & Probability

- Probability in A falls somewhere between bel(A) and pl(A)
- bel(A) represents the evidence we have for A directly
  - So prob(A) cannot be less than this value
- pl(A) represents the maximum share of the evidence we could possibly have, if, for all sets that intersect with A, the part that intersects is actually valid
  - So pl(A) is the maximum possible value of prob(A).

Belief Intervals & Probability

- Belief intervals allow Demspter-Shafer theory to reason about the degree of certainty or certainty of our beliefs.
  - A small ***difference between belief and plausibility shows that we are certain*** about our belief
  - A ***large difference shows that we are uncertain*** about our belief.
- However, even with a 0 interval, this does not mean we know which conclusion is right
  - **Just how probable it is!**

Unit 3 – KRR - Summary

- **KNOWLEDGE AND REASONING:**
  - Representation
  - First Order Predicate Logic
  - Inference
  - Unification
  - Forward and Backward Chaining
  - Resolution
  - Statistical Reasoning
  - Probability and Bayes Theorem
  - Dempster-Shafer Theory
- Blended/Self learning/Self Study
  - Reasoning with Default Information (12.6 AIMA)
  - Truth Maintenance Systems (12.6 AIMA)
  - Acting under Uncertainty (13.1 AIMA)
  - Certainty Factors and Rule Based Systems (8.2 ERKK)

References

- AI
  - Artificial Intelligence – Elaine Rich, Kevin Knight, B. Nair 3rd Edition
- AIMA
  - Artificial Intelligence - A Modern Approach 3rd Edition - RUSSELL & NORVIG