In [5]:

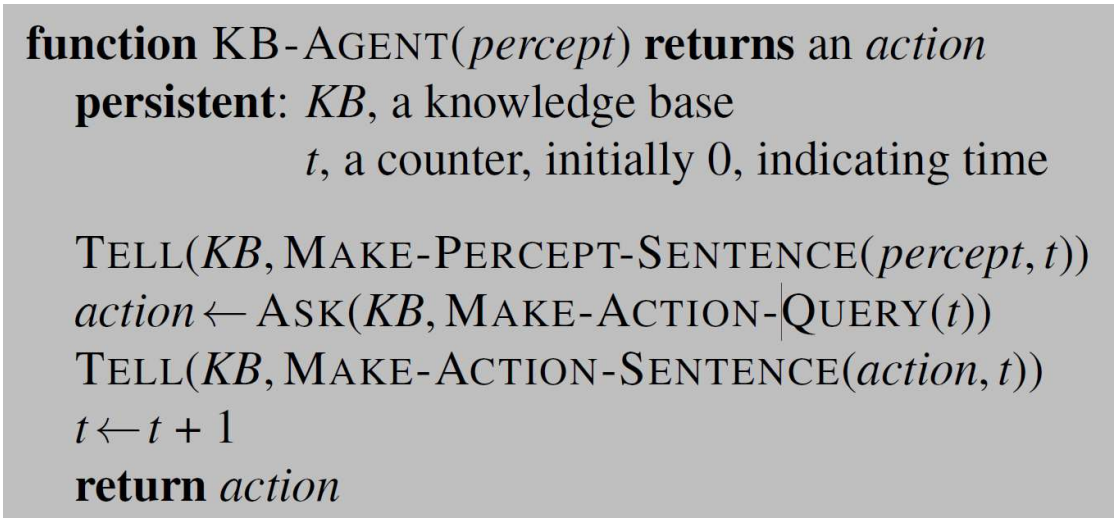
Logical Agents

1. Knowledge based agents

In AI, knowledge based use a process of reasoning over an internal representation of knowledge to agents to decide what actions to take.

A Knowledge base is a set of sentences. Each sentence is expressed in a language called a knowledge representation language and represents some assertion about the world. When the sentence is taken as being given without being derived from other sentences, we call it an **axiom**.

There must be a way to add new sentences to the knowledge base and a way to query what is known. The standard names for these operations are TELL and ASK, respectively. Both operations may involve inference—that is, deriving new sentences from old.



The sentences in the knowledge bases are expressed according to the syntax of the representation language, which specifies all the sentences that are well formed.

The notion of syntax is clear enough in ordinary arithmetic: “x+y = 4” is a **well-formed sentence**, whereas “x4y+ =” is not.

A logic must also define the semantics, or meaning, of sentences. The semantics defines the truth of each sentence with respect to each possible world.

For example, the semantics world for arithmetic specifies that the sentence “x+y=4” is true in a world where x is 2 and y is 2, but false in a world where x is 1 and y is 1.

In standard logics, every sentence must be either true or false in each possible world—there is no “in between.”

If a sentence α is true in model m, we say that m satisfies α or sometimes m is a model of α. We use the notation M(α) to mean the set of all models of α.

2. Logical Reasoning

This involves the relation of logical entailment between sentences—the idea that a sentence follows logically from another sentence.

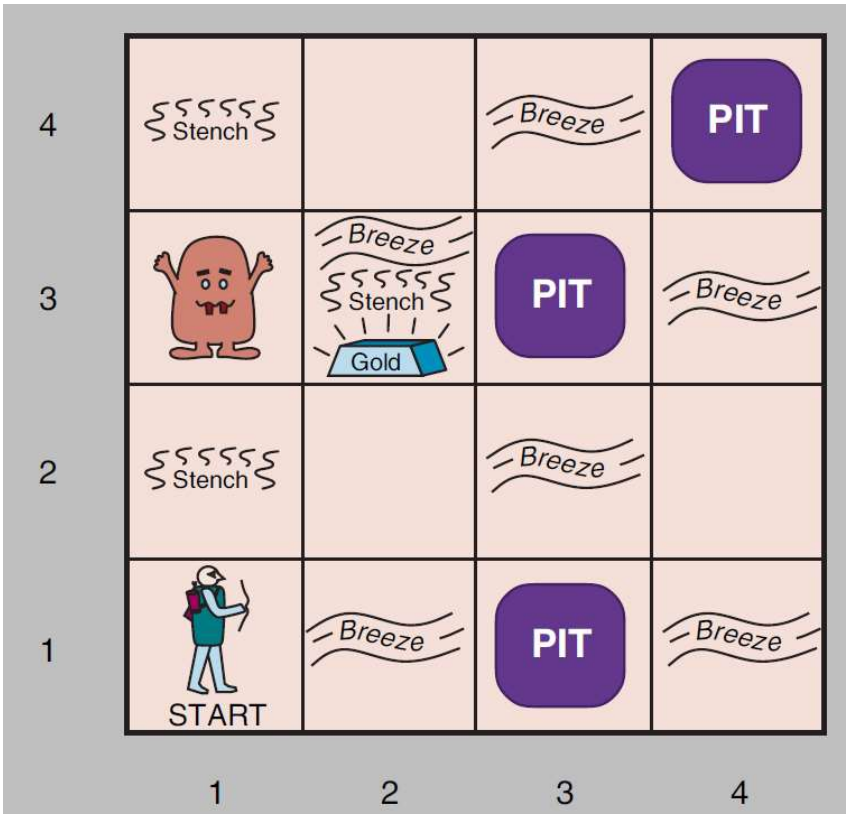
In mathematical notation, we write $\alpha \models \beta$ to mean that the sentence α entails the sentence β.

The formal definition of entailment is this: $\alpha \models \beta$ if and only if, in every model in which α is true, β is also true.

Using the notation just introduced, we can write $\alpha \models \beta$ if and only if M(α) $\subseteq$ M(β).

3. Wumpus World

A sample wumpus world is shown in below.



The precise definition of the task environment is given by the PEAS description:

Performance measure: +1000 for climbing out of the cave with the gold, –1000 for falling into a pit or being eaten by the wumpus, –1 for each action taken, and –10 for using up the arrow. The game ends either when the agent dies or when the agent climbs out of the cave.

Environment: A 4X4 grid of rooms, with walls surrounding the grid. The agent always starts in the square labeled [1, 1], facing to the east. The locations of the gold and the wumpus are chosen randomly, with a uniform distribution, from the squares other than the start square. In addition, each square other than the start can be a pit, with probability 0.2.

Actuators: The agent can move Forward, TurnLeft by 90°, or TurnRight by 90°. The agent dies a miserable death if it enters a square containing a pit or a live wumpus. (It is safe, albeit smelly, to enter a square with a dead wumpus.) If an agent tries to move forward and bumps into a wall, then the agent does not move. The action Grab can be used to pick up the gold if it is in the same square as the agent. The action Shoot can be used to fire an arrow in a straight line in the direction the agent is facing. The arrow continues until it either hits (and hence kills) the wumpus or hits a wall. The agent has only one arrow, so only the first Shoot action has any effect. Finally, the action Climb can be used to climb out of the cave, but only from square [1,1].

Sensors: The agent has five sensors, each of which gives a single bit of information:

- In the squares directly (not diagonally) adjacent to the wumpus, the agent will perceive a Stench.
- In the squares directly adjacent to a pit, the agent will perceive a Breeze.
- In the square where the gold is, the agent will perceive a Glitter.
- When an agent walks into a wall, it will perceive a Bump.
- When the wumpus is killed, it emits a woeful Scream that can be perceived anywhere in the cave.

The **percepts** will be given to the agent program in the form of a list of five symbols; for example, if there is a stench and a breeze, but no glitter, bump, or scream, the agent program will get [Stench, Breeze, None, None, None].

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 OK	2,2	3,2	4,2
1,1 A OK	2,1 OK	3,1	4,1

A = *Agent*
B = *Breeze*
G = *Glitter, Gold*
OK = *Safe square*
P = *Pit*
S = *Stench*
V = *Visited*
W = *Wumpus*

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 OK	2,2 P?	3,2	4,2
1,1 V OK	2,1 A B OK	3,1 P?	4,1

1,4	2,4	3,4	4,4
1,3 W!	2,3	3,3	4,3
1,2 A S OK	2,2 OK	3,2	4,2
1,1 V OK	2,1 B V OK	3,1 P!	4,1

A = *Agent*
B = *Breeze*
G = *Glitter, Gold*
OK = *Safe square*
P = *Pit*
S = *Stench*
V = *Visited*
W = *Wumpus*

1,4	2,4 P?	3,4	4,4
1,3 W!	2,3 A S G B	3,3 P?	4,3
1,2 S V OK	2,2 V OK	3,2	4,2
1,1 V OK	2,1 B V OK	3,1 P!	4,1

4. Propositional Logic

The Propositional Logic describes the syntax and symantics of the sentences used for knowledge representation.

The syntax of propositional logic defines the allowable sentences. The atomic sentences consist of a single proposition symbol. Each such symbol stands for a proposition that can be true or false.

There are two proposition symbols with fixed meanings: True is the always-true proposition and False is the always-false proposition. Complex sentences are constructed from simpler sentences, using parentheses and operators called logical connectives. There are five connectives in common use:

4.1 Logical connetives in Propositional Logic

- $\neg$ (not). A sentence such as $\neg W_{1,3}$ is called the **negation** of $W_{1,3}$. A **literal** is either an atomic sentence (a **positive literal**) or a negated atomic sentence (a **negative literal**).
- $\wedge$ (and). A sentence whose main connective is $\wedge$, such as $W_{1,3} \wedge P_{3,1}$, is called a **conjunction**; its parts are the **conjuncts**. (The $\wedge$ looks like an "A" for "And.")
- $\vee$ (or). A sentence whose main connective is $\vee$, such as $(W_{1,3} \wedge P_{3,1}) \vee W_{2,2}$, is a **disjunction**; its parts are **disjuncts**—in this example, $(W_{1,3} \wedge P_{3,1})$ and $W_{2,2}$.
- $\Rightarrow$ (implies). A sentence such as $(W_{1,3} \wedge P_{3,1}) \Rightarrow \neg W_{2,2}$ is called an **implication** (or conditional). Its **premise** or **antecedent** is $(W_{1,3} \wedge P_{3,1})$, and its **conclusion** or **consequent** is $\neg W_{2,2}$. Implications are also known as **rules** or **if–then** statements. The implication symbol is sometimes written in other books as $\supset$ or $\rightarrow$.
- $\Leftrightarrow$ (if and only if). The sentence $W_{1,3} \Leftrightarrow \neg W_{2,2}$ is a **biconditional**.

4.2 BNF (Backus Naur Form) grammar of sentences in propositional logic

$$
\begin{aligned}
\mathit{Sentence} &\rightarrow \mathit{AtomicSentence} \mid \mathit{ComplexSentence} \\
\mathit{AtomicSentence} &\rightarrow \mathit{True} \mid \mathit{False} \mid P \mid Q \mid R \mid \ldots \\
\mathit{ComplexSentence} &\rightarrow (\mathit{Sentence}) \\
&\mid \neg \mathit{Sentence} \\
&\mid \mathit{Sentence} \wedge \mathit{Sentence} \\
&\mid \mathit{Sentence} \vee \mathit{Sentence} \\
&\mid \mathit{Sentence} \Rightarrow \mathit{Sentence} \\
&\mid \mathit{Sentence} \Leftrightarrow \mathit{Sentence} \\
\text{OPERATOR PRECEDENCE} &: \neg, \wedge, \vee, \Rightarrow, \Leftrightarrow
\end{aligned}
$$

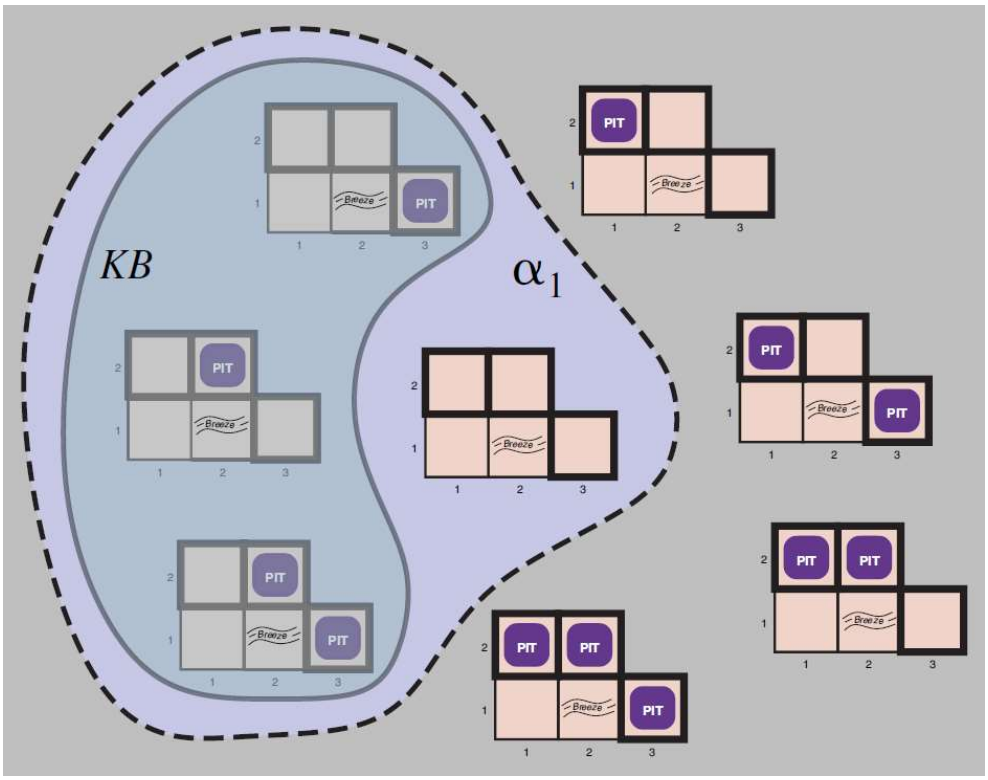
4.3 The semantics

The semantics defines the rules for determining the truth of a sentence with respect to a particular model.

In propositional logic, a model simply sets the truth value—true or false—for every proposition symbol. For example, if the sentences in the knowledge base make use of the proposition symbols $P_{1,2}$, $P_{2,2}$, and $P_{3,1}$, then one possible model is

$$m_1 = \{P_{1,2} = false, P_{2,2} = false, P_{3,1} = true\}$$

With three proposition symbols, there are $2^3 = 8$ possible models—exactly those depicted below



The semantics for propositional logic must specify how to compute the truth value of any sentence, given a model.

This is done recursively.

All sentences are constructed from atomic sentences and the five connectives; therefore, we need to specify how to compute the truth of atomic sentences and how to compute the truth of sentences formed with each of the five connectives.

Finding truth value of atomic sentences is easy:

- True is true in every model and False is false in every model.
- The truth value of every other proposition symbol must be specified directly in the model. For example, in the model m_1 given earlier, $P_{1,2}$ is false.

For complex sentences, we have five rules, which hold for any subsentences P and Q (atomic or complex) in any model m (here "iff" means "if and only if"):

- $\neg P$ is true iff P is false in m.
- $P \wedge Q$ is true iff both P and Q are true in m.
- $P \vee Q$ is true iff either P or Q is true in m.
- $P \Rightarrow Q$ is true unless P is true and Q is false in m.
- $P \Leftrightarrow Q$ is true iff P and Q are both true or both false in m.

The rules can also be expressed with **truth tables** that specify the truth value of a complex sentence for each possible assignment of truth values to its components. Truth tables for the five connectives are given below

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	*false*	*true*	*false*	*false*	*true*	*true*
false	*true*	*true*	*false*	*true*	*true*	*false*
true	*false*	*false*	*false*	*true*	*false*	*false*
true	*true*	*false*	*true*	*true*	*true*	*true*

Knowledge Base of Wumpus World using Propositional Logic

We need the following symbols for each [x, y] location:
$P_{x,y}$ is true if there is a pit in [x, y].
$W_{x,y}$ is true if there is a wumpus in [x, y], dead or alive.
$B_{x,y}$ is true if there is a breeze in [x, y].
$S_{x,y}$ is true if there is a stench in [x, y].
$L_{x,y}$ is true if the agent is in location [x, y].

We label each sentence R_i so that we can refer to them:

$R_1 : \neg P_{1,1}$
$R_2 : B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
$R_3 : B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$
$R_4 : \neg B_{1,1}$
$R_5 : B_{2,1}$

A simple inference procedure

Our goal now is to decide whether KB $\models \alpha$ for some sentence α.

- For example, is $\neg P_{1,2}$ entailed by our KB?
  - Let us use model-checking algorithm for inference.
  - Enumerate the models, and check that α is true in every model in which KB is true.
  - Models are assignments of true or false to every proposition symbol. Returning to our wumpus-world example, the relevant proposition symbols are $B_{1,1}$, $B_{2,1}$, $P_{1,1}$, $P_{1,2}$, $P_{2,1}$, $P_{2,2}$, $P_{3,1}$. With seven symbols, there are 2^7=128 possible models; in three of these, KB is true.
  - In those three models, $\neg P_{1,2}$ is true, hence there is no pit in [1, 2].

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	R_1	R_2	R_3	R_4	R_5	KB
false	*false*	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*false*	*false*
false	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*false*	*true*	*false*	*false*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
false	*true*	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*false*	*true*	*true*	*false*
false	*true*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*false*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*true*	*false*	*false*	*true*	*false*	*false*	*true*	*true*	*false*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
true	*true*	*true*	*true*	*true*	*true*	*true*	*false*	*true*	*true*	*false*	*true*	*false*

- For example, is $P_{2,2}$ entailed by our KB?
  - $P_{2,2}$ is true in two of the three models and false in one, so we cannot yet tell whether there is a pit in [2, 2].

Basic Terminology related to PL

Logical equivalence: Two sentences α and β are logically equivalent if they are true in the same set of models. We write this as $\alpha \equiv \beta$. (Note that $\equiv$ is used to make claims about sentences, while $\Leftrightarrow$, is used as part of a sentence.)

- For example, we can easily show (using truth tables) that $P \wedge Q$ and $Q \wedge P$ are logically equivalent; other equivalences are shown below

$$
\begin{aligned}
(\alpha \wedge \beta) &\equiv (\beta \wedge \alpha) && \text{commutativity of } \wedge \\
(\alpha \vee \beta) &\equiv (\beta \vee \alpha) && \text{commutativity of } \vee \\
((\alpha \wedge \beta) \wedge \gamma) &\equiv (\alpha \wedge (\beta \wedge \gamma)) && \text{associativity of } \wedge \\
((\alpha \vee \beta) \vee \gamma) &\equiv (\alpha \vee (\beta \vee \gamma)) && \text{associativity of } \vee \\
\neg(\neg\alpha) &\equiv \alpha && \text{double-negation elimination} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\beta \Rightarrow \neg\alpha) && \text{contraposition} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\alpha \vee \beta) && \text{implication elimination} \\
(\alpha \Leftrightarrow \beta) &\equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) && \text{biconditional elimination} \\
\neg(\alpha \wedge \beta) &\equiv (\neg\alpha \vee \neg\beta) && \text{De Morgan} \\
\neg(\alpha \vee \beta) &\equiv (\neg\alpha \wedge \neg\beta) && \text{De Morgan} \\
(\alpha \wedge (\beta \vee \gamma)) &\equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) && \text{distributivity of } \wedge \text{ over } \vee \\
(\alpha \vee (\beta \wedge \gamma)) &\equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) && \text{distributivity of } \vee \text{ over } \wedge
\end{aligned}
$$

Validity: A sentence is valid if it is true in all models. For example, the sentence $P \vee \neg P$ is valid. Valid sentences are also known as **tautologies**—they are necessarily true.

Because the sentence True is true in all models, every valid sentence is logically equivalent to True.

Deduction theorem : For any sentences α and β, $\alpha \models \beta$ if and only if the sentence $(\alpha \Rightarrow \beta)$ is valid.

Satisfiability: A sentence is satisfiable if it is true in, or satisfied by, some model.

Satisfiability can be checked by enumerating the possible models until one is found that satisfies the sentence.

- For example, the knowledge base given earlier, $(R_1 \wedge R_2 \wedge R_3 \wedge R_4 \wedge R_5)$, is satisfiable because there are three models in which it is true.

Validity and satisfiability are of course connected: α is valid iff : $\neg\alpha$ is unsatisfiable; contrapositively, α is satisfiable iff : $\neg\alpha$ is not valid.

We also have the following useful result: $\alpha \models \beta$ if and only if the sentence $(\alpha \wedge \neg\beta)$ is unsatisfiable. This procedure of proving β from α by checking the unsatisfiability is called **proof by refutation or proof by contradiction**.

Monotonicity, which says that the set of entailed sentences can only increase as information is added to the knowledge base.

- For any sentences α and β, if $KB \models \alpha$ then $KB \wedge \beta \models \alpha$

This knowledge might help the agent draw additional conclusions, but it cannot invalidate any conclusion α already inferred—such as the conclusion that there is no pit in [1, 2].

Inference rules

$$\frac{\alpha \Rightarrow \beta, \quad \alpha}{\beta} \quad \text{Modus Ponens}$$

And-Elimination, which says that, from a conjunction, any of the conjuncts can be inferred:

$$\frac{\alpha \wedge \beta}{\alpha} \quad \text{And-Elimination}$$

By considering the possible truth values of α and β, one can easily show once and for all that Modus Ponens and And-Elimination are sound.

These rules can then be used in any particular instances where they apply, generating sound inferences without the need for enumerating models.

All of the logical equivalences can be used as inference rules. For example, the equivalence for biconditional elimination yields the two inference rules

$$\frac{\alpha \Leftrightarrow \beta}{(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)} \qquad \frac{(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)}{\alpha \Leftrightarrow \beta}$$

Inference Rule — Biconditional Elimination

Not all inference rules work in both directions like this. For example, we cannot run Modus Ponens in the opposite direction to obtain $\alpha \Rightarrow \beta$ and α from β.

How these inference rules and equivalences can be used in the wumpus world.

We start with the knowledge base containing R_1 through R_5 and show how to prove $\neg P_{1,2}$ that is, there is no pit in [1,2]:

$R_1 : \neg P_{1,1}$
$R_2 : B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
$R_3 : B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$
$R_4 : \neg B_{1,1}$
$R_5 : B_{2,1}$

1. Apply biconditional elimination to R_2 to obtain

$$R_6 : \quad (B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1}) .$$

2. Apply And-Elimination to R_6 to obtain

$$R_7 : \quad ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1}) .$$

3. Logical equivalence for contrapositives gives

$$R_8 : \quad (\neg B_{1,1} \Rightarrow \neg(P_{1,2} \vee P_{2,1})) .$$

4. Apply Modus Ponens with R_8 and the percept R_4 (i.e., $\neg B_{1,1}$), to obtain

$$R_9 : \quad \neg(P_{1,2} \vee P_{2,1}) .$$

5. Apply De Morgan's rule, giving the conclusion

$$R_{10} : \quad \neg P_{1,2} \wedge \neg P_{2,1} .$$

That is, neither [1,2] nor [2,1] contains a pit.

Any of the search algorithms can be used to find a sequence of steps that constitutes a proof like this. We just need to define a proof problem as follows:

- INITIAL STATE: the initial knowledge base.
- ACTIONS: the set of actions consists of all the inference rules applied to all the sentences that match the top half of the inference rule.
- RESULT: the result of an action is to add the sentence in the bottom half of the inference rule.
- GOAL: the goal is a state that contains the sentence we are trying to prove.

Thus, searching for proofs is an alternative to enumerating models. In many practical cases finding a proof can be more efficient because the proof can ignore irrelevant propositions, no matter how many of them there are. The simple truth-table algorithm, on the other hand, would be overwhelmed by the exponential explosion of models.

Proof by resolution

We have argued that the inference rules covered so far are sound, but we have not discussed the question of completeness for the inference algorithms that use them.

The current section introduces a single inference rule, resolution, that yields a complete inference algorithm when coupled with any complete search algorithm.

Let us assume that the agent returns from [2,1] to [1,1] and then goes to [1,2], where it perceives a stench, but no breeze.

We add the following facts to the knowledge base:

$R_{11} : \neg B_{1,2}$.
$R_{12} : B_{1,2} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{1,3})$

Applying Biconditional eliination to R_{12} we get

$$R_{12a} : (B_{1,2} \Rightarrow (P_{1,1} \vee P_{2,2} \vee P_{1,3})) \wedge ((P_{1,1} \vee P_{2,2} \vee P_{1,3}) \Rightarrow B_{1,2})$$

Applying And-Eliination to R_{12a} we get

$$R_{12b} : (P_{1,1} \vee P_{2,2} \vee P_{1,3}) \Rightarrow B_{1,2}$$

Logical equivalence for contrapositives gives

$$R_{12c} : \neg B_{1,2} \Rightarrow \neg(P_{1,1} \vee P_{2,2} \vee P_{1,3})$$

Apply Modus Ponens with R_{12c} and the percept $R_{11} : \neg B_{1,2}$., to obtain

$$R_{12d} : \neg(P_{1,1} \vee P_{2,2} \vee P_{1,3})$$

Apply De Morgan's rule, giving the conclusion

$$R_{12e} : (\neg P_{1,1} \wedge \neg P_{2,2} \wedge \neg P_{1,3})$$

Apply AND elimination to R_{12e} gives us

$$R13 : \neg P_{2,2}$$
$$R14 : \neg P_{1,3}$$

Apply biconditional elimination to R_3

$$R_3 : B_{2,1} \Leftrightarrow \left(P_{1,1} \vee P_{2,2} \vee P_{3,1}\right)$$

$$R_3 a : \left(B_{2,1} \Rightarrow \left(P_{1,1} \vee P_{2,2} \vee P_{3,1}\right)\right) \wedge \left(\left(P_{1,1} \vee P_{2,2} \vee P_{3,1}\right) \Rightarrow B_{2,1}\right)$$

Apply AND elimination

$$R_{3b} : B_{2,1} \Rightarrow \left(P_{1,1} \vee P_{2,2} \vee P_{3,1}\right)$$

Apply Modus Ponens with R3b and R5, to obtain the fact that there is a pit in [1,1], [2,2], or [3,1]

$$R_{15} : \left(P_{1,1} \vee P_{2,2} \vee P_{3,1}\right)$$

Apply the resolution rule: the literal $\neg P_{2,2}$ in R13 resolves with the literal $P_{2,2}$ in R_{15} to give the resolvent

$$R_{16} : (P_{1,1} \vee P_{3,1})$$

Similarly, the literal $\neg P_{1,1}$ in R1 resolves with the literal $\neg P_{1,1}$ in R16 to give

$$R_{17} : P_{3,1}$$

Unit and Generalized inference rule

These last two inference steps are examples of the **unit resolution inference rule**

$$\frac{l_1 \vee l_2 \vee \cdots \vee l_k, \quad m}{l_1 \vee l_2 \cdots l_{i-1} \vee l_{i+1} \vee \cdots \vee l_k}$$

where each l is a literal and l_i and m are complementary literals (i.e., one is the negation of the other). Thus, the unit resolution rule takes a clause—a disjunction of literals—and a literal and produces a new clause. Note that a single literal can be viewed as a disjunction of Unit clause one literal, also known as a unit clause.

The unit resolution rule can be generalized to the full resolution rule

$$\frac{l_1 \vee l_2 \cdots l_k, \quad m_1 \vee m_2 \cdots m_n}{l_1 \vee l_2 \cdots l_{i-1} \vee l_{i+1} \vee \cdots \vee l_k \vee m_1 \vee m_2 \cdots m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n}$$

Conjunctive normal form or CNF

A sentence expressed as a conjunction of clauses is said to be in conjunctive normal

$$\begin{aligned}
CNFSentence &\rightarrow Clause_1 \wedge \cdots \wedge Clause_n \\
Clause &\rightarrow Literal_1 \vee \cdots \vee Literal_m \\
Fact &\rightarrow Symbol \\
Literal &\rightarrow Symbol \mid \neg Symbol \\
Symbol &\rightarrow P \mid Q \mid R \mid \ldots \\
HornClauseForm &\rightarrow DefiniteClauseForm \mid GoalClauseForm \\
DefiniteClauseForm &\rightarrow Fact \mid (Symbol_1 \wedge \cdots \wedge Symbol_l) \Rightarrow Symbol \\
GoalClauseForm &\rightarrow (Symbol_1 \wedge \cdots \wedge Symbol_l) \Rightarrow False
\end{aligned}$$

Convert the sentence $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$ into CNF

1. Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$.

$$(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1}) .$$

2. Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$:

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1}) .$$

3. CNF requires $\neg$ to appear only in literals, so we "move $\neg$ inwards" by repeated application of the following equivalences from Figure 7.11:

$$\neg(\neg\alpha) \equiv \alpha \quad \text{(double-negation elimination)}$$
$$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta) \quad \text{(De Morgan)}$$
$$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta) \quad \text{(De Morgan)}$$

In the example, we require just one application of the last rule:

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1}) .$$

4. Now we have a sentence containing nested $\wedge$ and $\vee$ operators applied to literals. We apply the distributivity law from Figure 7.11, distributing $\vee$ over $\wedge$ wherever possible.

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1}) .$$

Horn clauses and definite clauses

Definite clause, is a disjunction of literals of which exactly one is positive.

> For example, the clause $(\neg L_{1,1} \vee \neg Breeze \vee B_{1,1})$ is a definite clause, whereas $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1})$ is not, because it has two positive clauses.

Horn clause, is a disjunction of literals of which at most one is positive. So all definite clauses are Horn clauses, as are clauses with no positive literals; these are called goal clauses.

Horn clauses are closed under resolution: if you resolve two Horn clauses, you get back a Horn clause.

Knowledge bases containing only definite clauses are interesting for three reasons:

1. Every definite clause can be written as an implication whose premise is a conjunction of positive literals and whose conclusion is a single positive literal.

> For example $(\neg L_{1,1} \vee \neg Breeze \vee B_{1,1})$ can be written as the implication $(L_{1,1} \wedge Breeze) \Rightarrow B_{1,1}$.

The premise is called the **body** and the conclusion is called the **head**. A sentence consisting of a single positive literal, such as $L_{1,1}$, is called a **fact**.

Inference with Horn clauses can be done through the **forward-chaining** and **backward chaining algorithms**.

5. First Order Predicate Logic

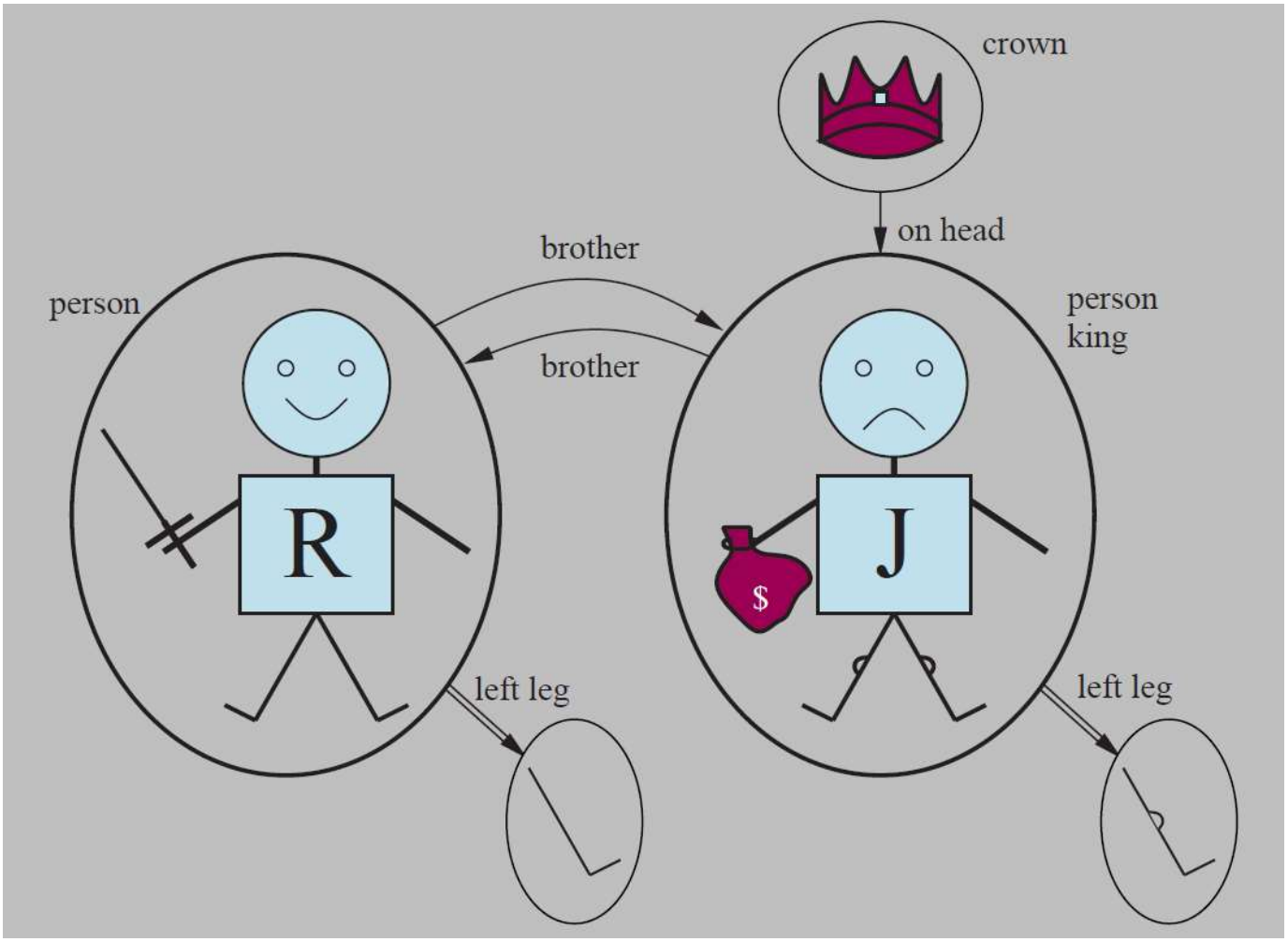
Models of a logical language are the formal structures that constitute the possible worlds under consideration.

Each model links the vocabulary of the logical sentences to elements of the possible world, so that the truth of any sentence can be determined.

Thus, models for propositional logic link proposition symbols to predefined truth values.

Models for first-order logic are much more interesting. First, they have objects in them. The domain of a model is the set of objects or domain elements it contains.

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



5.1 Syntax of First-order logic

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

The symbols, therefore, come in three kinds:

> constant symbols, which stand for objects;
> predicate symbols, which stand for relations;
> and function symbol, which stand for functions.

We adopt the convention that these symbols will begin with uppercase letters.

> For example, we might use the constant symbols Richard and John;
> the predicate symbols Brother, OnHead, Person, King, and Crown;
> and the function symbol LeftLeg.

In addition to its objects, relations, and functions, each model includes an interpretation that specifies exactly which objects, relations and functions are referred to by the constant, predicate, and function symbols.

One possible interpretation for our example — which a logician would call the intended interpretation—is as follows:

- Richard refers to Richard the Lionheart and John refers to the evil King John.
- Brother refers to the brotherhood relation;
- OnHead is a relation that holds between the crown and King John;
- Person, King, and Crown are unary relations that identify persons, kings, and crowns.
- LeftLeg refers to the “left leg” function.

A `term` is a logical expression that refers to an object. Constant symbols are terms, but it is not always convenient to have a distinct symbol to name every object. In English we might use the expression “King John’s left leg” rather than giving a name to his leg. This is what function symbols are for: instead of using a constant symbol, we use `LeftLeg(John)` .

We have **terms for referring to objects and predicate symbols for referring to relations**, we can combine them to make atomic sentences that state facts.

An atomic sentence (or atom for short) is formed from a predicate symbol optionally followed by a parenthesized list of terms, such as `Brother(Richard, John)`

Atomic sentences can have complex terms as arguments. Thus, `Married(Father(Richard), Mother(John))`

We can use logical connectives to construct more `complex sentences` , with the same syntax and semantics as in propositional calculus.

$\neg$ `Brother(LeftLeg(Richard), John)`

`Brother(Richard, John)` $\wedge$ `Brother(John, Richard)`

`King(Richard)` $\vee$ `King(John)`

$\neg$ `King(Richard)` $\Rightarrow$ `King(John)`

Once we have a logic that allows objects, it is only natural to want to express properties of entire collections of objects, instead of enumerating the objects by name. Quantifiers let us do this. First-order logic contains two standard quantifiers, called universal and existential.

Rules such as “Squares neighboring the wumpus are smelly” and “All kings are persons”.

$\forall x$ `King(x)` $\Rightarrow$ `Person(x)` .

We can make a statement about some object without naming it, by using an existential quantifier. To say, for example, that King John has a crown on his head, we write $\exists x$ `Crown(x)` $\wedge$ `OnHead(x, John)` .

The syntax of first-order logic with equality, specified in Backus–Naur form. Operator precedences are specified, from highest to lowest. The precedence of quantifiers is such that a quantifier holds over everything to the right of it.

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

5.2 Inference in First-order Logic

One way to do first-order inference is to convert the first-order knowledge base to propositional logic and use propositional inference.

> Eliminating universal quantifiers

All greedy kings are evil:

$\forall x$ `King(x)` $\wedge$ `Greedy(x)` $\Rightarrow$ `Evil(x)`

From that we can infer any of the following sentences:

`King(John)` $\wedge$ `Greedy(John)` $\Rightarrow$ `Evil(John)`

`King(Richard)` $\wedge$ `Greedy(Richard)` $\Rightarrow$ `Evil(Richard)`

$\texttt{King(Father(John))} \land \texttt{Greedy(Father(John))} \Rightarrow \texttt{Evil(Father(John))}$

Universal Instantiation (UI for short) rule says that we can infer any sentence obtained by substituting a ground term (a term without variables) for a universally quantified variable.

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

For example, the three sentences given earlier are obtained with the substitutions {x/John}, {x/Richard}, and {x/Father(John)}.

Similarly, the rule of Existential Instantiation replaces an existentially quantified variable with a single new constant symbol. The formal statement is as follows: for any sentence α, variable v, and constant symbol k that does not appear elsewhere in the knowledge base,

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

For example, from the sentence $\exists x$ $\texttt{Crown(x)} \land \texttt{OnHead(x, John)}$ we can infer the sentence $\texttt{Crown(C1)} \land \texttt{OnHead(C1, John)}$ as long as C1 does not appear elsewhere in the knowledge base. In logic, the new name is called a Skolem constant.

Whereas Universal Instantiation can be applied many times to the same axiom to produce many different consequences, Existential Instantiation need only be applied once, and then the existentially quantified sentence can be discarded.

Reduction to propositional inference

For example, suppose our knowledge base contains just the sentences

$\forall x$ $\texttt{King(x)} \land \texttt{Greedy(x)} \Rightarrow \texttt{Evil(x)}$

$\texttt{King(John)}$

$\texttt{Greedy(John)}$

$\texttt{Brother(Richard, John)}$

and that the only objects are $\texttt{John}$ and $\texttt{Richard}$. We apply UI to the first sentence using all possible substitutions, {x/John}, {x/Richard}. We obtain

$\texttt{King(John)} \land \texttt{Greedy(John)} \Rightarrow \texttt{Evil(John)}$

$\texttt{King(Richard)} \land \texttt{Greedy(Richard)} \Rightarrow \texttt{Evil(Richard)}$

Next replace ground atomic sentences, such as King(John), with proposition symbols, such as $\texttt{JohnIsKing}$. Finally, apply any of the complete propositional algorithms to obtain conclusions such as JohnIsEvil, which is equivalent to Evil(John). This technique of **Propositionalization**.

When the knowledge base includes a function symbol, the set of possible ground-term substitutions is infinite! For example, if the knowledge base mentions the $\texttt{Father}$ symbol, then infinitely many nested terms such as $\texttt{Father(Father(Father(John)))}$ can be constructed.

Unification and First-Order Inference

The UNIFY algorithm takes two sentences and returns a unifier for them (a substitution) if one exists:

UNIFY(p, q) = θ where SUBST(θ , p) = SUBST(θ , q)

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

UNIFY(Knows(John, x) , Knows(y, Bill)) = {x/Bill,y/John}

UNIFY(Knows(John,x) , Knows(y,Mother(y))) = {y/John, x/Mother(John)}

UNIFY(Knows(John, x) , Knows(x, Elizabeth)) = failure

The problem arises in the last unification only because the two sentences happen to use the same variable name, x. The problem can be avoided by Standardizing apart one of the two sentences being unified, which means renaming its variables to avoid name clashes.

UNIFY(Knows(John, x) , Knows(z, Elizabeth)) = {x/Elizabeth, z/John}

UNIFY should return a substitution that makes the two arguments look the same. But there could be more than one such unifier.

For example,

UNIFY(Knows(John, x) , Knows(y, z)) could return {y/John, x/z} or could return {y/John, x/John, z/John} .

The first unifier gives Knows(John, z) as the result of unification, whereas the second gives Knows(John, John) .

The second result could be obtained from the first by an additional substitution {z/John} , we say that the first unifier is more general than the second, because it places fewer restrictions on the values of the variables.

First-order definite clauses

First-order definite clauses are disjunctions of literals of which exactly one is positive.

That means a definite clause is either atomic, or is an implication whose antecedent is a conjunction of positive literals and whose consequent is a single positive literal.

Existential quantifiers are not allowed, and universal quantifiers are left implicit: if you see an x in a definite clause, that means there is an implicit $\forall x$ quantifier.

A typical first-order definite clause looks like this:

King(x) $\wedge$ Greedy(x) $\Rightarrow$ Evil(x)

King(John)

Greedy(y)

Let us use definite clauses to represent knowledge about the following problem:

> 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.

American(x) $\wedge$ Weapon(y) $\wedge$ Sells(x,y,z) $\wedge$ Hostile(z) $\Rightarrow$ Criminal(x) --------> 1

The sentence $\exists x$ `Owns(Nono, x) ∧ Missile(x)` is transformed into two definite clauses by Existential Instantiation, introducing a new constant M1:

`Owns(Nono, M1)` --------> 2

`Missile(M1)` ---------> 3

`Missile(x) ∧ Owns(Nono, x) ⇒ Sells(West, x, Nono)` ------> 4

`Missile(x) ⇒ Weapon(x)` --------> 5

`Enemy(x, America) ⇒ Hostile(x)` --------> 6

`American(West)` ---------> 7

`Enemy(Nono, America)` ----------> 8

Forward-chaining algorithm

Starting from the known facts, it triggers all the rules whose premises are satisfied, adding their conclusions to the known facts.

The process repeats until the query is answered (assuming that just one answer is required) or no new facts are added.

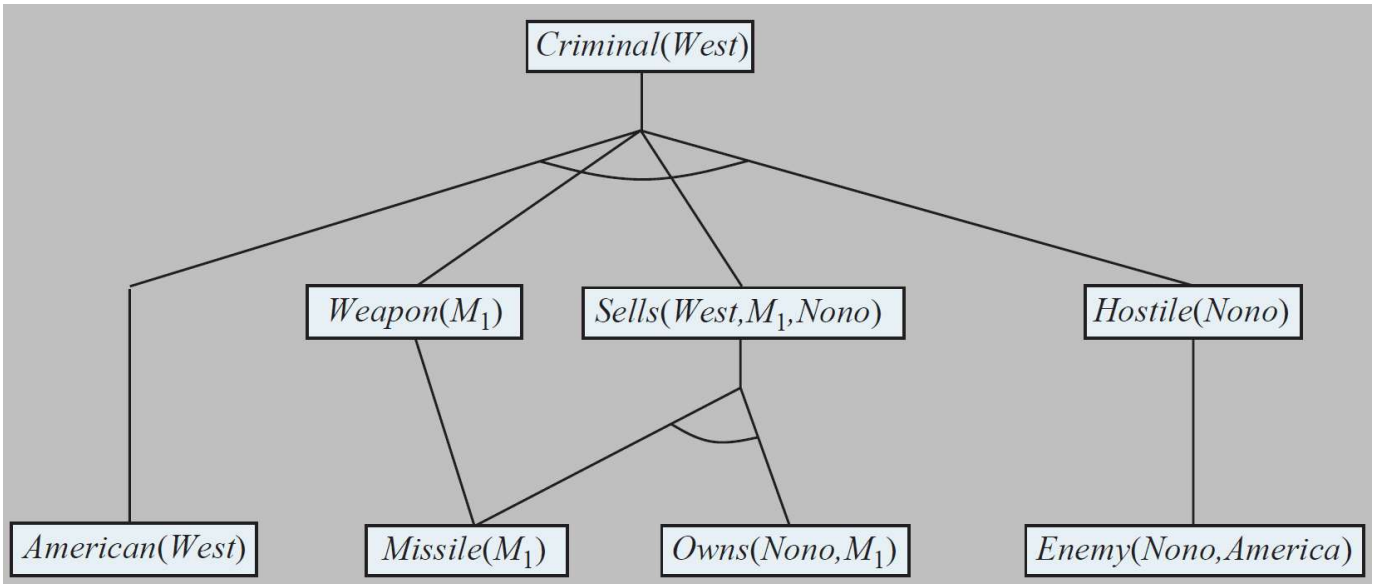
A fact is not “new” if it is just a renaming of a known fact

A sentence is a renaming of another if they are identical except for the names of the variables.

> `Likes(x, IceCream)` and `Likes(y, IceCream)` are renamings of each other. They both mean the same thing: “Everyone likes ice cream.”

- On the first iteration, rule (1) has unsatisfied premises.
  - Rule (4) is satisfied with {x/M1}, and `Sells(West,M1,Nono)` is added.
  - Rule (5) is satisfied with {x/M1}, and `Weapon(M1)` is added.
  - Rule (6) is satisfied with {x/Nono}, and `Hostile(Nono)` is added.
- On the second iteration, rule (1) is satisfied with {x/West,y/M1,z/Nono} and the inference `Criminal(West)` is added.

The proof tree generated by forward chaining on the crime example. The initial facts appear at the bottom level, facts inferred on the first iteration in the middle level, and facts inferred on the second iteration at the top level.



There are some logical sentences that cannot be stated as a definite clause, and thus cannot be handled by forward chaining algoritham.

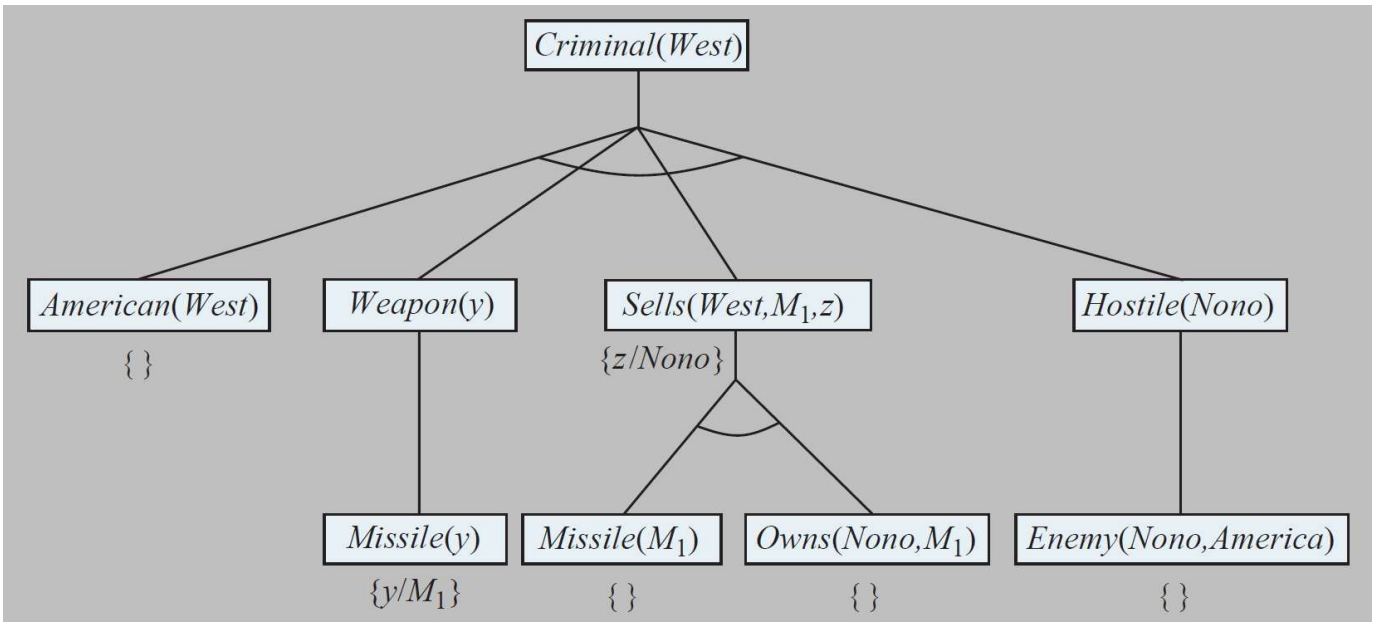
Backward Chaining

The second major family of logical inference algorithms uses backward chaining over definite clauses. These algorithms work backward from the goal, chaining through rules to find known facts that support the proof.

FOL-BC-ASK(KB, goal) will be proved if the knowledge base contains a rule of the form lhs ⇒ goal, where lhs(left-hand side) is a list of conjuncts. An atomic fact like American(West) is considered as a clause whose lhs is the empty list.

The proof tree for deriving `Criminal(West)` from sentences (1) through (8).

Proof tree constructed by backward chaining to prove that West is a criminal. The tree should be read depth first, left to right. To prove `Criminal(West)` , we have to prove the four conjuncts below it. Some of these are in the knowledge base, and others require further backward chaining. Bindings for each successful unification are shown next to the corresponding subgoal. Note that once one subgoal in a conjunction succeeds, its substitution is applied to subsequent subgoals. Thus, by the time FOL-BC-ASK gets to the last conjunct, originally `Hostile(z)` , z is already bound to Nono.



Resolution

The last of our three families of logical systems, and the only one that works for any knowledge base, not just definite clauses, is resolution.

Resolution proves that $KB \models \alpha$ by proving that $KB \wedge \neg\alpha$ unsatisfiable—that is, by deriving the empty clause.

The first step is to convert sentences to conjunctive normal form (CNF)—that is, a conjunction of clauses, where each clause is a disjunction of literals.

$$\begin{array}{lll} \neg\forall x\ p & \text{becomes} & \exists x\ \neg p \\ \neg\exists x\ p & \text{becomes} & \forall x\ \neg p\,. \end{array}$$

In CNF, literals can contain variables, which are assumed to be universally quantified.

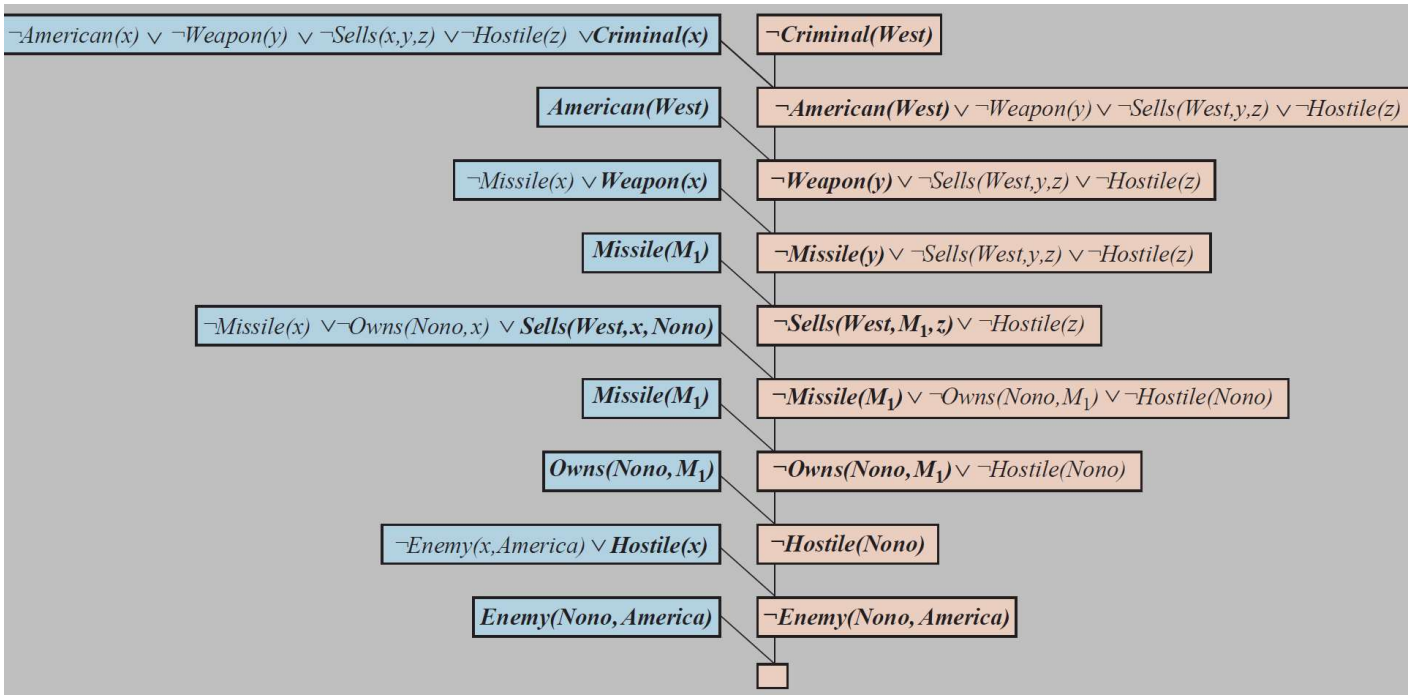
$$\begin{array}{ll} \neg American(x) \vee \neg Weapon(y) \vee \neg Sells(x,y,z) \vee \neg Hostile(z) \vee Criminal(x) & \\ \neg Missile(x) \vee \neg Owns(Nono,x) \vee Sells(West,x,Nono) & \\ \neg Enemy(x,America) \vee Hostile(x) & \\ \neg Missile(x) \vee Weapon(x) & \\ Owns(Nono,M_1) & Missile(M_1) \\ American(West) & Enemy(Nono,America)\,. \end{array}$$

We also include the negated goal $\neg$ `Criminal(West)` .

Binary resolution

We can resolve the two clauses `[Animal(F(x)) ∨ Loves(G(x),x)]` and `[ ¬ Loves(u,v) ∨ ¬ Kills(u,v)]` by eliminating the complementary literals `Loves(G(x),x)` and ¬ `Loves(u,v)` , with the unifier θ = {u/G(x),v/x}, to produce the resolvent clause `Animal(F(x)) ∨ ¬ Kills(G(x),x)` .

This rule is called the binary resolution rule because it resolves exactly two literals. The full resolution rule resolves subsets of literals in each clause that are unifiable.



Factoring

Propositional factoring reduces two literals to one if they are identical; first-order factoring reduces two literals to one if they are unifiable. The unifier must be applied to the entire clause.