Artificial Intelligence

academic year 2025/2026

Giorgio Fumera

Pattern Recognition and Applications Lab
Department of Electrical and Electronic Engineering
University of Cagliari (Italy)

Knowledge Representation and Inference: Logical Languages

Suggested textbook

S. Russell, P. Norvig, *Artificial Intelligence – A Modern Approach*, 4th Ed., Pearson, 2021 (or a previous edition)

Introduction

Example problem: automatic theorem proving

Assume your goal is to design **rational agents**, in the form of a computer program, capable of **autonomously** solving the following problems.

Prove or to **refute** the following statement:

Goldbach's conjecture (1742)

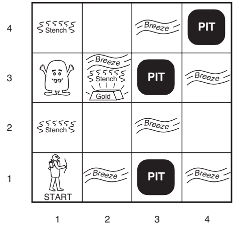
For any even number $p \geq 2$, there exists at least one pair of prime numbers q and r (identical or not) such that

$$q + r = p$$

Example problem: the wumpus game

A text-based computer game (G. Yob, c. 1972) used in a modified version as an AI's toy-problem. A basic version:

- **the wumpus world**: a cave made up of connected rooms, bottomless pits, a heap of gold, and the **wumpus**, a beast that eats anyone who enter its room



- player's **goal**: starting from room (1,1), finding the gold and going back to (1,1), without falling into a pit or hitting the wumpus
- main **rules of the game**:
  - the content of any room is known only **after** entering it
  - in rooms neighboring the wumpus and pits a **stench** and a **breeze** is perceived, respectively

Knowledge-based systems

Humans usually solve problems like the ones above by combining two **high-level** capabilities

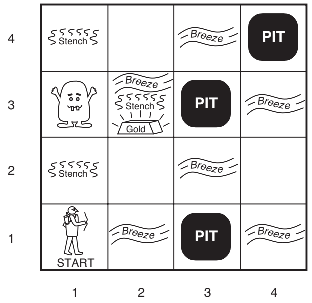
- abstract **knowledge representation**
- **reasoning**

Knowledge-based systems (KBSs) aim at **mechanising** the above human capabilities:

- representing knowledge about the world
- reasoning to derive new knowledge and to **guide action**

An example: playing the wumpus game

Consider the following initial configuration, and remember that the player knows the content of any room only after entering it:

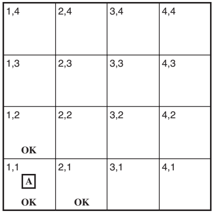


If **you** were the player, how would **you** reason to **decide** the next move to do at each game step?

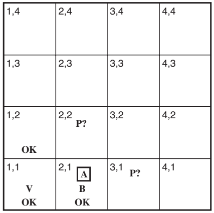
An example: playing the wumpus game

Sketch of a possible **reasoning process** for deciding the next move, starting from the configuration shown above (some moves are omitted).

a)

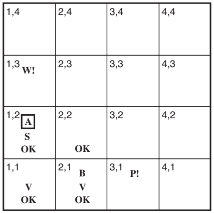


b)

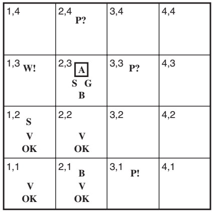


$\boxed{\textbf{A}}$	= *Agent*
B	= *Breeze*
G	= *Glitter, Gold*
OK	= *Safe square*
P	= *Pit*
S	= *Stench*
V	= *Visited*
W	= *Wumpus*

c)



d)



Main approaches to AI system design

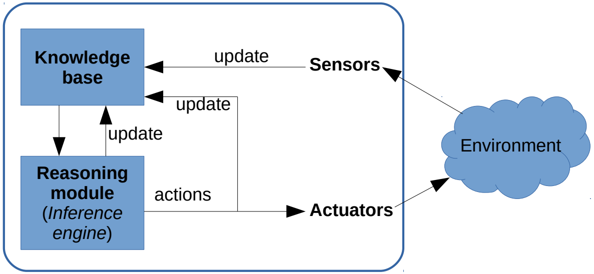
Procedural: the desired behavior (actions) are encoded directly as program code (no explicit knowledge representation and reasoning).

Declarative: explicit representation, in a **knowledge base**, of

- **background knowledge** (e.g., the rules of the wumpus game)
- knowledge about a specific **problem instance** (e.g., what the agent knows about a specific wumpus cave it is exploring)
- the agent's **goal** (e.g., finding the gold and going back to the starting square, without falling into a pit or hitting the wumpus)

Actions are then decided by **reasoning**.

Architecture of knowledge-based systems



Main feature: **separation** between **knowledge representation** and **reasoning**

- the **knowledge base** contains all the agent's knowledge about its environment, in **declarative** form
- the **inference engine** implements a reasoning process (algorithm) to derive new knowledge and to make decisions

Main applications in computer science and AI

Computer science:

- logic programming languages (**Prolog**, etc.)
- databases (relational calculus, SQL)
- semantic Web
- satisfiability (SAT) problems:
  hardware/software design, planning (travel, logistics, ...), etc.

Artificial Intelligence:

- expert systems (medicine, engineering, finance, etc.)
- automatic theorem provers

A short introduction to logic: outline

- What is logic?
- Propositions and argumentations
- Logical languages
- Logical inference

Logic

A long-standing discipline – some milestones:

- Aristotle (4th cent. BC): the “laws of thought”
- G. Boole (1815–64): Boolean algebra (propositional logic)
- G. Frege (1848–1925): predicate logic
- K. Gödel (1906–78): investigation of the limitations of logic, incompleteness theorem

Logic has become one of the main tools used in AI for

- **knowledge representation**: logical languages, such as
  - propositional logic
  - predicate (first-order) logic
- **reasoning**: inference rules and algorithms

Logic

A possible definition of logic:

> ***Logic** is the study of conditions under which an **argumentation** (reasoning) is **correct**.*

This definition involves the following concepts:

- **argumentation**: a set of statements consisting of some **premises** and one **conclusion**, e.g.:
  All men are mortal; Socrates is a man;
  then, Socrates is mortal
- **correctness**: an argumentation is said to be correct when its conclusion cannot be false when all its premises are true
- **proof**: a procedure to assess correctness

Propositions

Natural language is very complex and vague, and therefore difficult to formalize. Logic considers argumentations made up of only a subset of statements: **propositions** (**declarative statements**).

> *A **proposition** is a statement expressing a concept that can be either **true** or **false**.*

Example

- *Socrates is a man*
- *Two and two makes four*
- *If the Earth had been flat, then Columbus would have not reached America*

A counterexample: *Read that book!*

Simple and complex propositions

A proposition is:

- **simple**, if it does not contain simpler propositions
- **complex**, if it is made up of simpler propositions connected by **logical connectives**

Example

Simple propositions:

- *Socrates is a man*
- *Two and two makes four*

Complex propositions:

- *A tennis match can be won* ***or*** *lost*
- ***If*** *the Earth had been flat,* ***then*** *Columbus would have not reached America*

Argumentations

How to determine whether a proposition is true or false?
This is a **philosophical** question.
Logic does not address this question: it only analyses the **structure** of an argumentation.

Example

All men are mortal; Socrates is a man;
then, Socrates is mortal.

Informally, the **structure** of this argumentation is:

all P are Q; x is P; then x is Q.

Its correctness depends only on its structure, **whatever** P, Q and x mean, that is, **regardless** of whether the corresponding propositions "all P are Q", "x is P" and "x is Q" are true or false.

Formal languages

Logic provides **formal languages** for representing (the structure of) propositions, in the form of **sentences**.
A formal language is defined by a **syntax** and a **semantics**:

- **syntax** (grammar): rules that define what sentences are "well-formed", i.e., sentences which a meaning can be attributed to
- **semantics**: rules that define the meaning of well-formed sentences

Examples of formal languages:

- **arithmetic** encodes **propositions** about numbers
- **programming languages** encode **instructions** to be executed by a computer (for imperative languages like C)

Natural vs formal languages

In **natural languages**:

- syntax is not rigorously defined
- semantics defines the "content" of a statement, i.e., "what it refers to in the real world"

Example (syntax)

- *The book is on the table*: syntactically correct statement, with a clear semantics
- *Book the on is table the*: syntactically incorrect statement, no meaning can be attributed to it
- *Colorless green ideas sleep furiously*:[1] syntactically correct, but what does it mean?

[1] N. Chomsky, *Syntactic Structures*, 1957

Natural vs formal languages

Logical languages:

- syntax: formally defined
- semantics: rules that define the **truth value** of each well-formed sentence with respect to each possible **model**, i.e., a possible "world" represented by that sentence

Example (arithmetic)

- **Syntax**: $x + y = 4$ is a well-formed sentence, $x4y+ =$ is not
- **Model**: the symbol '4' represents the natural number four, 'x' and 'y' any pair of natural numbers, '$+$' the sum operator, etc.
- **Semantics**: $x + y = 4$ is true for $x = 1$ and $y = 3$, for $x = 2$ and $y = 2$, etc.

Logical entailment

Logical reasoning is based on the relation of **logical entailment** between sentences, that defines when a sentence **logically follows** from one or more other sentences:

> *a sentence α **entails** a sentence β, if and only if, in every model in which α is true, also β is true. In symbols:*
>
> $$\alpha \models \beta$$

Example (from arithmetic)

$$x + y = 4 \quad \models \quad x = 4 - y \ ,$$

because in every model (i.e., for any assignment of numbers to x and y) in which $x + y = 4$ is true, also $x = 4 - y$ is true.

Logical inference

Logical inference*: the process of deriving conclusions from premises*

Inference algorithm*: a procedure that derives sentences (conclusions) from other sentences (premises), in a given formal language*

Formally, the fact that an inference algorithm A derives a sentence α from a set of sentences ("knowledge base") KB is written as:

$$KB \vdash_A \alpha$$

Intuitively, one would like that an inference algorithms derives only (and possibly, all) entailed sentences. This is formalised as follows.

Properties of inference algorithms

Soundness *(truth-preservation): if an inference algorithm derives* ***only*** *sentences entailed by the premises, i.e.:*

$$\text{if } KB \vdash_A \alpha, \text{then } KB \models \alpha$$

Completeness*: if an inference algorithm derives* ***all*** *the sentences entailed by the premises, i.e.:*

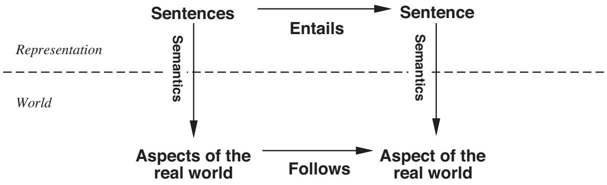
$$\text{if } KB \models \alpha, \text{then } KB \vdash_A \alpha$$

A **sound** algorithm derives conclusions that are guaranteed to be true in any world in which the premises are true.

Properties of inference algorithms

Inference algorithms operate only at the **syntactic** level:

- sentences are sequences of symbols, encoded as physical configurations of an agent (e.g., bits in registers)
- inference algorithms construct new physical configurations from previous ones
- logical reasoning should ensure that new configurations constructed by inference algorithms represent aspects of the world that actually follow from the ones represented by starting configurations

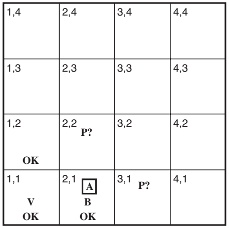


Applications of inference algorithms

In AI inference is used to answer two main kinds of questions:

- does **a given** conclusion α logically follows from the agent's knowledge *KB*? (i.e., $KB \models \alpha$?)
- what are **all** the conclusions that logically follow from the agent's knowledge? (i.e., find all α's such that $KB \models \alpha$)

Example (the wumpus world)



- does a breeze in room (2,1) entail the presence of a pit in room (2,2)?
- what conclusions can be derived about the presence of pits and of the wumpus in each room, from the current knowledge?

Inference algorithms: model checking

The definition of **entailment** can be directly applied to construct a simple inference algorithm, named **Model checking**:
Given a set of premises, KB, and a sentence α, **enumerate** all possible models and **check** whether α is true in **every** model in which KB is true.

Example (arithmetic)

- $KB : \{x + y = 4\}$
- $\alpha : y = 4 - x$

Is the following inference correct?

$$\{x + y = 4\} \vdash y = 4 - x$$

Model checking: enumerate all possible pairs of numbers x, y, and check whether $y = 4 - x$ is true whenever $x + y = 4$ is.

The issue of *grounding*

A knowledge base *KB* (i.e., a set of sentences that the agents considers true) is just "syntax" (a physical configuration of the agent):

- what is the connection between a KB and the real world?
- how does one know that sentences in the KB are true in the real world?

This is the same **philosophical** question met before. For humans:

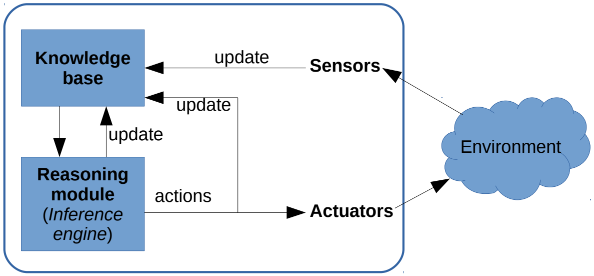
- a set of beliefs (set of statements considered true) is a physical configuration of our brain
- how do we know that our beliefs are true in the real world?

A simple answer can be given for agents (e.g., computer programs or robots): the connection is created by

- **sensors**, e.g.: perceiving a breeze in the wumpus world
- **learning**, e.g., when a breeze is perceived, there is a pit in some adjacent room

Of course, both perception and learning are **fallible**.

Architecture of knowledge-based systems revisited



If **logical languages** are used:

- **knowledge base**: a set of sentences in a given logical language
- **inference engine**: an inference algorithm for the same language

Focus of this course: propositional and predicate logic for knowledge representation, inference algorithms for propositional logic.

Logical languages considered in this course

Propositional logic

- the simplest logical language
- an extension of Boolean algebra (G. Boole, 1815–64)

Predicate (or **first-order**) **logic**

- a more expressive and concise extension of propositional logic
- close to the formal language of arithmetic
- seminal work: G. Frege (1848–1925)

Propositional Logic

Syntax

- **Atomic sentences**
  - either a **propositional symbol** that denotes a given proposition (usually written in capitals), e.g.: P, Q, ...
  - or a propositional symbol with fixed meaning: *True* and *False*
- **Complex sentences** consist of atomic or (recursively) complex sentences connected by **logical connectives** (corresponding to natural language connectives like *and*, *or*, *not*, etc.)
- **Logical connectives** (only the commonly used ones are shown – different notations exist):

$\wedge$	(and)
$\vee$	(or)
$\neg$	(not)
$\Rightarrow$	(implication / if...then...)
$\Leftrightarrow$	(biconditional / logical equivalence – if and only if)

Syntax

A formal grammar in Backus-Naur Form (BNF):

$$\begin{array}{rcl}
\mathit{Sentence} & \rightarrow & \mathit{AtomicSentence} \mid \mathit{ComplexSentence} \\
\mathit{AtomicSentence} & \rightarrow & \mathit{True} \mid \mathit{False} \mid \mathit{Symbol} \\
\mathit{Symbol} & \rightarrow & P \mid Q \mid R \mid \ldots \\
\mathit{ComplexSentence} & \rightarrow & \neg \mathit{Sentence} \\
 & \mid & (\ \mathit{Sentence} \land \mathit{Sentence}\ ) \\
 & \mid & (\ \mathit{Sentence} \lor \mathit{Sentence}\ ) \\
 & \mid & (\ \mathit{Sentence} \Rightarrow \mathit{Sentence}\ ) \\
 & \mid & (\ \mathit{Sentence} \Leftrightarrow \mathit{Sentence}\ )
\end{array}$$

Semantics

Semantics of propositional logic:

- **model** of a sentence: a possible assignment of truth values to all propositional symbols that appear in it
- **meaning** of a sentence: its **truth value** with respect to a particular **model**

Example

The sentence $P \wedge Q \Rightarrow R$ has $2^3 = 8$ possible models.
For instance, one model is $\{P = \mathit{True}, Q = \mathit{False}, R = \mathit{True}\}$.

Note: models are abstract mathematical objects with no unique connection to the real world (e.g., P may stand for **any** proposition in natural language).

Semantics

- **Atomic sentences**:
  - *True* is true in every model
  - *False* is false in every model
  - the truth value of every propositional symbol (atomic sentence) must be specified in the model
- **Complex sentences**:
  their truth value is **recursively** determined as a function of
  - the truth value of the simpler sentences they are made up of
  - the **truth tables** of the logical connectives they contain

Truth tables of commonly used connectives

The semantics of logical connectives is **defined** by their truth tables:

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

Note: $P \Rightarrow Q$ reads as "P implies Q, or "if P then Q".

Example

Determining the truth value of $\neg P \wedge (Q \vee R)$ in all possible models, i.e., for all possible assignment of truth values to P, Q and R:

P	Q	R	$(Q \vee R)$	$\neg P \wedge (Q \vee R)$
false	false	false	false	false
false	false	true	true	true
false	true	false	true	true
false	true	true	true	true
true	false	false	false	false
true	false	true	true	false
true	true	false	true	false
true	true	true	true	false

Logical connectives and natural language

The truth tables of the connectives $\wedge$, $\vee$ and $\neg$ agree with our intuition about the terms "and", "or" and "not".
However, they do not capture all the nuances of such terms.

Example

- *He felt down **and** broke his leg*
  Here "and" includes **temporal** and **causal** relations, that are not represented by $\wedge$: whereas $P \wedge Q \equiv Q \wedge P$, the proposition *He broke his leg and felt down* has a different meaning than the one above
- *A tennis match can be won **or** lost*
  This proposition cannot be represented as $P \vee Q$ since in this case "or" has a **disjunctive** meaning (a tennis match cannot be both won and lost), corresponding to the exclusive OR operator $\oplus$ of Boolean algebra, whereas $\vee$ has a **conjunctive** meaning

Logical connectives and natural language

The truth table of the implication may not seem in agreement with the intuitive understanding of "P implies Q", or "if P then Q".

P	Q	$P \Rightarrow Q$
false	false	true
false	true	true
true	false	false
true	true	true

It can be understood as representing the concept of "**sufficient** but **not necessary** condition", i.e.:

*P is a **sufficient** but **not necessary** condition for Q to be true*

In other words, the sentence $P \Rightarrow Q$ represents a proposition of the form:

if P is true, then I am claiming that also Q is true;
*otherwise, **I am making no claim***

In both cases, the only way for $P \Rightarrow Q$ to be false is when P is true and Q is false.

Logical connectives and natural language

Also the implication connective does not represent all the nuances of the term “implies”.

Example

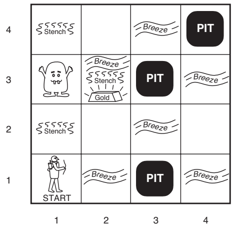
- *The number four is even implies Tokyo is the capital of Japan*
  One may consider this proposition as false, although both the antecedent and consequent are true, since in this case “implies” includes a relation of **causation** or **relevance**. However, if it is translated as $P \Rightarrow Q$, it is **true** according to the truth table of the implication connective, since this connective does not represent causation or relevance
- *The number five is even implies Sam is smart*
  Also this proposition may be considered as false since there is no relation of causation or relevance between antecedent and consequent, **regardless** of whether Sam is smart or not. However, if represented as $P \Rightarrow Q$, since P is false, also this proposition is **true**, **regardless** of the truth value of Q

Exercise

1. Define a set of propositional symbols to represent the wumpus world: the position of the agent, the wumpus, pits, etc.
2. Define the model corresponding to the configuration below
3. Define the part of the initial agent's KB corresponding to its knowledge about the cave configuration in the figure below
4. Write sentences for the propositions:

(a) *If no breeze is perceived in room (1,1), then there is no pit in room (1,2)*

(b) *If the wumpus is in room (3,1), there is a stench in rooms (2,1), (4,1) and (3,2)*



Solution of exercise 1

A possible choice of propositional symbols:

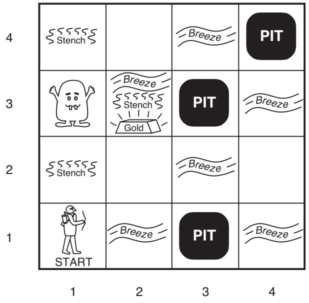
- $A_{1,1}$ ("the agent is in room (1,1)"), $A_{1,2}$, ..., $A_{4,4}$
- $W_{1,1}$ ("the wumpus is in room (1,1)"), $W_{1,2}$, ..., $W_{4,4}$
- $P_{1,1}$ ("there is a pit in room (1,1)"), $P_{1,2}$, ..., $P_{4,4}$
- $G_{1,1}$ ("the gold is in room (1,1)"), $G_{1,2}$, ..., $G_{4,4}$
- $B_{1,1}$ ("there is a breeze in room (1,1)"), $B_{1,2}$, ..., $B_{4,4}$
- $S_{1,1}$ ("there is stench in room (1,1)"), $S_{1,2}$, ..., $S_{4,4}$

Note that distinct propositional symbols are necessary for similar propositions related to different rooms, e.g.: "the agent is in room (1,1)" ($A_{1,1}$) and "the agent is in room (1,2)" ($A_{1,2}$).

Solution of exercise 2

Model corresponding to the configuration on the right:

- $A_{1,1}$ is true;
  $A_{1,2}, A_{1,3}, \ldots$ are false
- $W_{3,1}$ is true;
  $W_{1,1}, W_{1,2}, \ldots$ are false
- $P_{1,3}, P_{3,3}, P_{4,4}$ are true;
  $P_{1,1}, P_{1,2}, \ldots$ are false
- $G_{3,2}$ is true;
  $G_{1,1}, G_{1,2}, \ldots$ are false
- $B_{1,2}, B_{1,4}, \ldots$ are true;
  $B_{1,1}, B_{1,3}, \ldots$ are false
- $S_{2,1}, S_{3,2}, B_{4,1}$ are true;
  $S_{1,1}, S_{1,2}, \ldots$ are false



Solution of exercise 3

What the agent knows in the starting configuration:

- I am in room (1,1) (starting position of the game)
- there is no pit nor the wumpus in room (1,1)
- there is no gold in room (1,1)
- I do not perceive a breeze nor a stench in room (1,1)

The corresponding sentences in the agent's KB (i.e., the sentences the agent "believes" to be true):

- $A_{1,1}$, $\neg A_{1,2}$, $\neg A_{1,3}$, $\ldots$, $\neg A_{4,4}$ (16 sentences)
- $\neg W_{1,1}$, $\neg P_{1,1}$
- $\neg G_{1,1}$
- $\neg B_{1,1}$, $\neg S_{1,1}$

Solution of exercise 4(a)

If no breeze is perceived in room (1,1), then there is no pit in room (1,2)

It is easy to see that the absence of breeze in a room is a sufficient condition for the absence of a pit in any adjacent room. It is however not a necessary condition, since, even if a breeze is present, e.g., in room (1,1), it may be due to a pit in room (2,1), whereas room (1,2) can still contain no pit.

The above proposition can therefore be represented by an implication:

$$\neg B_{1,1} \Rightarrow \neg P_{1,2}$$

Solution of exercise 4(b)

If the wumpus is in room (3,1) then there is a stench in rooms (2,1), (4,1) and (3,2)

One may be tempted to use an implication:

$$W_{3,1} \Rightarrow (S_{2,1} \wedge S_{4,1} \wedge S_{3,2})$$

The above sentence is true, but **incomplete** in the wumpus world. Since there is only one wumpus, its presence in room (3,1) is also a **necessary** condition for a stench to be present in the neighboring rooms. In other words, the previous sentence does not rule out models in which $W_{3,1}$ is false and $S_{2,1} \wedge S_{4,1} \wedge S_{3,2}$ is true, which would violate the rules of the wumpus world. Indeed, the opposite is also true:

$$(S_{2,1} \wedge S_{4,1} \wedge S_{3,2}) \Rightarrow W_{3,1}$$

The correct representation of the above statement is therefore:

$$W_{3,1} \Leftrightarrow (S_{2,1} \wedge S_{4,1} \wedge S_{3,2})$$

Inference in Propositional Logic

The Model checking inference algorithm

Goal of logical inference: given a KB and a sentence α, deciding whether $KB \models \alpha$.

A simple inference algorithm: **model checking** (see above).

Application to propositional logic:

- enumerate all possible models for the sentences of KB and α
- check whether α is true in every model in which **all** sentences in KB are true

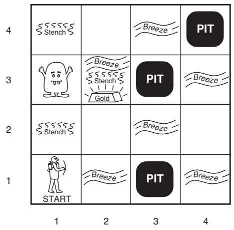
Straightforward implementation: **truth tables**.

Model checking: an example

In the initial configuration of the wumpus game shown below, the agent's KB includes:

(a) $\neg B_{1,1}$ (current percept)

(b) $\neg B_{1,1} \Rightarrow \neg P_{1,2} \wedge \neg P_{2,1}$
(a possible – partial – representation of the rule of the game related to pits and breezes)



The agent can be interested in knowing whether room $(1,2)$ does not contains any pit, i.e., whether $KB \models \neg P_{1,2}$, where:

$$KB = \{\ \neg B_{1,1} \Rightarrow \neg P_{1,2} \wedge \neg P_{2,1},\ \ \neg B_{1,1}, \ldots\ \}$$

(cont.)

Model checking: an example

1. Build the truth table of the sentence to prove and of all the sentences in the KB (for the sake of simplicity, only the two sentences shown above are considered).
   Three different propositional symbols appear in them: $B_{1,1}$, $P_{1,2}$ and $P_{2,1}$; therefore their truth table contains $2^3 = 8$ rows:

Prop. symbols			Premises		Conclusion
$B_{1,1}$	$P_{1,2}$	$P_{2,1}$	$\neg B_{1,1} \Rightarrow \neg P_{1,2} \wedge \neg P_{2,1}$	$\neg B_{1,1}$	$\neg P_{1,2}$
false	false	false	true	true	true
false	false	true	false	true	true
false	true	false	false	true	false
false	true	true	false	true	false
true	false	false	true	false	true
true	false	true	true	false	true
true	true	false	true	false	false
true	true	true	true	false	false

(cont.)

Model checking: an example

2. Focus only on the rows in which **all** the premises are **true** (highlighted in grey) – in this example, only the first row:

Prop. symbols			Premises		Conclusion
$B_{1,1}$	$P_{1,2}$	$P_{2,1}$	$\neg B_{1,1} \Rightarrow \neg P_{1,2} \wedge \neg P_{2,1}$	$\neg B_{1,1}$	$\neg P_{1,2}$
false	false	false	**true**	**true**	true
false	false	true	false	true	true
false	true	false	false	true	false
false	true	true	false	true	false
true	false	false	true	false	true
true	false	true	true	false	true
true	true	false	true	false	false
true	true	true	true	false	false

The conclusion turns out to be **true** in all the rows in which all the premises are true: therefore it logically follows from the premises.

Model checking: another example

Determine whether $\{P \vee Q, P \Rightarrow R, Q \Rightarrow R\} \models P \vee R$, using model checking:

Propositional symbols			Premises			Conclusion
P	Q	R	$P \vee Q$	$P \Rightarrow R$	$Q \Rightarrow R$	$P \vee R$
false	false	false	false	true	true	false
false	false	true	false	true	true	true
false	true	false	true	true	false	false
false	true	true	true	true	true	true
true	false	false	true	false	true	true
true	false	true	true	true	true	true
true	true	false	true	false	false	true
true	true	true	true	true	true	true

Also in this case the conclusion is true in every row in which the premises are true, thus it logically follows from the premises.

Properties of Model checking

Model checking

- is **sound**, since it directly implements the definition of entailment
- is **complete**, since it works for any (finite) KB and any conclusion, and the corresponding set of models is finite
- its **computational complexity** is $O(2^n)$, where n is the number of propositional symbols appearing in the KB and in α

The drawback of model checking is its **exponential** computational complexity, which makes it infeasible when the number of propositional symbols is high.

Example

For istance, in the previous exercise about the wumpus world 96 propositional symbols have been introduced: the corresponding truth table is made up of $2^{96} \approx 10^{28}$ rows.

Inference rules

To reduce computational complexity of logical inference, alternative inference algorithms based on **inference rules** have been devised.

An inference rule represents a **standard pattern of inference**:

- a **simple** but limited reasoning step
- its soundness can be easily proven
- it can be applied to a set of premises having a **specific** structure to derive a conclusion

Inference rules are represented as follows:

$$\frac{\text{premises}}{\text{conclusion}}$$

Proofs are carried out by a **sequence** of applications of inference rules.

Examples of inference rules

In the following, α and β denote any propositional **sentences**.

And Elimination	$\frac{\alpha_1 \wedge \alpha_2}{\alpha_i}$, $i = 1, 2$
And Introduction	$\frac{\alpha_1, \alpha_2}{\alpha_1 \wedge \alpha_2}$
Or Introduction	$\frac{\alpha_1}{\alpha_1 \vee \alpha_2}$ (α_2 can be **any** sentence)
First De Morgan's law	$\frac{\neg(\alpha_1 \wedge \alpha_2)}{\neg\alpha_1 \vee \neg\alpha_2}$
Second De Morgan's law	$\frac{\neg(\alpha_1 \vee \alpha_2)}{\neg\alpha_1 \ \wedge \ \neg\alpha_2}$
Double Negation	$\frac{\neg(\neg\alpha)}{\alpha}$
Modus Ponens	$\frac{\alpha \Rightarrow \beta, \ \alpha}{\beta}$

The first five rules above easily generalise to any set of sentences $\alpha_1, \ldots, \alpha_n$.

Soundness of inference rules

Since inference rules usually involve a few sentences, their soundness can be easily proven using model checking.

An example: **Modus Ponens**

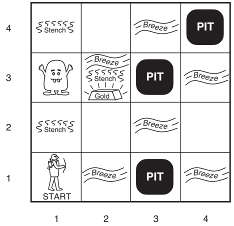
premise α	conclusion β	premise $\alpha \Rightarrow \beta$
false	false	true
false	true	true
true	false	false
true	true	true

Using inference rules: an example

Consider the same initial configuration of the wumpus game of the previous example, where the agent is interested in proving $KB \models \neg P_{1,2}$, and the KB includes:

(a) $\neg B_{1,1}$

(b) $\neg B_{1,1} \Rightarrow \neg P_{1,2} \wedge \neg P_{2,1}$



The proof can be carried out by suitably applying a sequence of inference rules to the sentences in *KB* and to the newly derived sentences:

- applying *Modus Ponens* to (a) and (b): $\neg P_{1,2} \wedge \neg P_{2,1}$
- applying *And elimination* to the new sentence above: $\neg P_{1,2}$

The agent can **conclude** that room $(1, 2)$ does not contain any pit.

Inference algorithms based on inference rules

Given a set of premises *KB* and a conclusion of interest α, the goal of an inference algorithm A based on inference rules is to find a **proof** $KB \vdash_A \alpha$ (if any), i.e., a **sequence** of applications of inference rules that leads from *KB* to α.

A basic inference algorithm based on a set of inference rules $\mathcal{I}$:

```
repeat
    apply in all possible ways the rules in $\mathcal{I}$ to the sentences of $KB$,
    and add to $KB$ each derived sentence, if not already present in it
until new sentences have been derived, and $\alpha$ is not present in $KB$
```

When this procedure ends

- if α is in the final *KB*, then $KB \vdash_A \alpha$, and (provided A is sound) one can conclude that $KB \models \alpha$
- if α is **not** in the final *KB*, only if A is **complete** one can conclude that $KB \not\models \alpha$

Properties of inference algorithms

Three main issues:

- is a given inference algorithm **sound** (correct)?
- is it **complete**?
- what is its **computational complexity**?

It is not difficult to see that, if the considered inference rules are **sound**, so is an inference algorithm based on them.

Completeness is more difficult to prove: it depends on

- the considered set of inference rules
- the structure of the sentences at hand

Properties of inference algorithms

What about **computational complexity**?

There can be a huge number of ways to apply the inference rules at hand to sentences of the current *KB* (remember that at each step newly derived sentences are added to *KB*), leading to a **very high** computational complexity.

Efficiency of inference algorithms can be improved in several ways:

- ignoring sentences of *KB* **irrelevant** to the conclusion α
- using a few inference rules (even only one), possibly without compromising completeness

A complete inference algorithm for propositional logic

Resolution is a family of complete inference algorithms for propositional logic, based on a **single** inference rule, it itself named **resolution**.

Resolution algorithms require that all the sentences (premises and conclusions) are written in **conjunctive normal form** (CNF), which is possible for **any** propositional sentence.

Conjunctive normal form

A **literal** is a propositional symbol, possibly negated

e.g., P and $\neg Q$ are literals

A **clause** is a sentence made up of

- either a single literal, e.g.: $\neg Q$
- or a **disjunction** of two or more literals, e.g.: $P \vee Q \vee \neg R$

A sentence is in **conjunctive normal form**, if it is made up of

- either a single clause, e.g.: $\neg Q$, and $P \vee Q \vee \neg R$
- or a **conjunction** of two or more clauses, e.g.: $(\neg R \vee S) \wedge P \wedge (Q \vee T \vee \neg Z)$

Conversion into conjunctive normal form

Any propositional sentence can be rewritten in CNF, as follows:

1. eliminating biconditionals, through the equivalence:
   $(P \Leftrightarrow Q) \equiv ((P \Rightarrow Q) \wedge (Q \Rightarrow P))$
2. eliminating implications, through the equivalence:
   $(P \Rightarrow Q) \equiv (\neg P \vee Q)$
3. "moving negations inwards" to individual propositional symbols, using the following equivalences:
   - De Morgan's laws:
     $\neg(P \vee Q) \equiv (\neg P \wedge \neg Q)$
     $\neg(P \wedge Q) \equiv (\neg P \vee \neg Q)$
   - double-negation elimination: $(\neg(\neg P)) \equiv P$
4. distributing $\vee$ over $\wedge$, using the equivalence:
   $(P \wedge Q) \vee R \equiv (P \vee R) \wedge (Q \vee R)$

Correctness of the above equivalences can be proven using truth tables.

Conversion into conjunctive normal form: an example

The following sentence encodes one of the rules of the Wumpus game:

$$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$$

It can be rewritten in CNF as follows:

1. eliminating the biconditional:
   $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$
2. eliminating the implication:
   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$
3. moving negations inwards (using one of De Morgan's laws):
   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$
4. distributing $\vee$ over $\wedge$:
   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$

The resolution inference rule

Consider two clauses (α_i's and α_j's denote literals):

$$\alpha_1 \vee \alpha_2 \vee \ldots \vee \alpha_m$$
$$\beta_1 \vee \beta_2 \vee \ldots \vee \beta_n$$

with $n, m \geq 1$, and assume they contain a pair of **complementary** literals α_p and α_q, say, $\alpha_p = P$ and $\beta_q = \neg P$.

The **resolution** inference rule derives a new **clause** made up of the disjunction of all the literals of the premises, **except** for α_p and α_q:

$$\frac{\alpha_1 \vee \alpha_2 \vee \ldots \vee \alpha_m, \qquad \beta_1 \vee \beta_2 \vee \ldots \vee \beta_n}{\alpha_1 \vee \ldots \vee \alpha_{p-1} \vee \alpha_{p+1} \vee \ldots \vee \alpha_m \vee \beta_1 \vee \ldots \vee \beta_{q-1} \vee \beta_{q+1} \vee \ldots \vee \beta_n}$$

Its soundness can be formally proven using truth tables.
Intuitively, since **one** among α_p and α_q must be **false**, the disjunction of the **remaining** literals of the corresponding clause must be true.

The resolution inference rule

An example, where $\neg R$ and R are two complementary literals:

$$\frac{P \vee \neg Q \vee \neg R, \quad\quad R \vee \neg S}{P \vee \neg Q \vee \neg S}$$

A particular case: two clauses made up of **single**, **complementary** literals (e.g., P and $\neg P$) are **contradictory**, i.e., cannot be simultaneously true. Applying the resolution rule to such clauses would lead to an "empty" clause, denoted by [ ]:

$$\frac{P, \quad \neg P}{[\ ]}$$

Note that contradictory sentences cannot be part of the **premises** of any **correct** argumentation (or of a KB), since by definition **all** the premises (or all sentences in a KB) are considered to be true.

Proof by refutation

A sentence is **satisfiable** if it is true in at least one model, e.g.

- P and $Q \lor \neg R$ are satisfiable (e.g., when P is *True* and R is *False*, regardless of Q)
- $P \land \neg P$ and $Q \land R \land \neg Q$ are unsatisfiable

Unsatisfiable sentences are said to be **contradictory**.

Remember that a sentence α logically follows from a set of premises KB ($KB \models \alpha$), if it cannot be false when all sentences in KB are true. Since any KB is equivalent to a single sentence made up of the **conjunction** of all the original sentences, entailment can be restated as:

> $KB \models \alpha$ *if and only if* $KB \land \neg\alpha$ *is unsatisfiable (i.e., contradictory).*

Proof by refutation or **by contradiction** (*reductio ad absurdum*) is a technique (widely used also in mathematics) that proves $KB \models \alpha$ by showing that $KB \land \neg\alpha$ is unsatisfiable.

Inference algorithms based on the resolution rule

The resolution rule, coupled with the proof by refutation technique, leads to **refutation-complete** algorithms for propositional logic:

- for any **given** KB and α, they can determine in a **finite** number of steps whether $KB \models \alpha$ or $KB \not\models \alpha$
- however, they **cannot** derive **all** the sentences that logically follow from a given set of premises

A resolution inference algorithm

```
**function** RESOLUTION (*KB*, $\alpha$) **returns** *true* or *false*
    *clauses* $\leftarrow$ the set of clauses of $KB \cup \{\neg\alpha\}$ in CNF form
    *new* $\leftarrow \emptyset$
    **loop do**
        **for each** distinct $\alpha_i, \alpha_j$ **in** *clauses* **do**
            *resolvents* $\leftarrow$ resolve($\alpha_i, \alpha_j$)
            **if** *resolvents* contains [ ] **then return** *true*
            *new* $\leftarrow$ *new* $\cup$ *resolvents*
        **if** *new* $\subseteq$ *clauses* **then return** *false*
        *clauses* $\leftarrow$ *clauses* $\cup$ *new*
```

`resolve` is a function that applies the resolution rule to **all** pairs of complementary literals of **all** pairs of clauses in the current KB.

Computational complexity of Resolution

Inference algorithms based on the resolution rule and on the proof by refutation technique check whether a set of sentences is **satisfiable**.

However, checking satisfiability is a **NP-complete** problem – informally, its computational complexity is very high: it requires up to an **exponential** number of steps in the size of the KB.

Efficiency can be improved through several strategies, such as discarding each newly derived clause containing complementary literals, which is true by definition, e.g.: $P \vee \neg R \vee \mathbf{Q} \vee \neg \mathbf{Q}$.

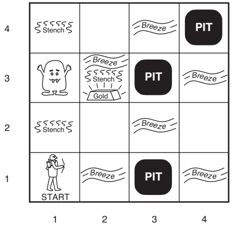
Resolution inference algorithms: an example

A slight variant of a previous example of the Wumpus game: the agent is interested in proving $KB \models \neg P_{1,2}$, and the KB includes:

(a) $\neg B_{1,1}$

(b) $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$

Note that only sentence (b) is not in CNF. It can be converted into CNF as shown above, leading to:



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

Separating the above conjuncts and adding the negation of the sentence to be proven leads to the following initial KB:

$$KB' = \{\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}, \quad \neg P_{1,2} \vee B_{1,1}, \quad \neg P_{2,1} \vee B_{1,1}, \quad \neg B_{1,1}, \quad P_{1,2}\}$$

Resolution inference algorithms: an example

Clauses 1–5 correspond to KB'. Clauses 6–12 are derived by resolving the indicated pairs of clauses of the current KB, and are then added to it.

1. $\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}$
2. $\neg P_{1,2} \vee B_{1,1}$
3. $\neg P_{2,1} \vee B_{1,1}$
4. $\neg B_{1,1}$
5. $P_{1,2}$
6. $\neg B_{1,1} \vee P_{1,2} \vee B_{1,1}$ (1, 3)
7. $P_{1,2} \vee P_{2,1} \vee \neg P_{2,1}$ (1, 3)
8. $\neg B_{1,1} \vee P_{2,1} \vee B_{1,1}$ (1, 2)
9. $P_{1,2} \vee P_{2,1} \vee \neg P_{1,2}$ (1, 2)
10. $\neg P_{2,1}$ (3, 4)
11. $\neg P_{1,2}$ (2, 4)
12. [ ] (5, 11)

A contradiction appears in clause 12, which proves that room $(1, 2)$ does not contain any pit.

Resolution inference algorithms: an example

Not all possible resolution steps are useful to complete a proof.

For instance, in the above proof clauses 6–9 contain complementary literals, e.g.:

6. $\neg \mathbf{B_{1,1}} \vee P_{1,2} \vee \mathbf{B_{1,1}} \quad (1,\ 3)$

Since each of such clauses is trivially true, they can be discarded during a proof, which reduces computational complexity.

Horn clauses

In many domains of practical interest, the whole KB can be expressed in the form of "if... then..." propositions that can be encoded as **Horn clauses**, i.e., implications where:

- the **antecedent** is an atomic sentence (a propositional symbol) or a conjunction ($\wedge$) of atomic sentences
- the **consequent** is a single atomic sentence

$$P_1 \wedge \ldots \wedge P_n \Rightarrow Q$$

For instance, $S_{2,1} \wedge S_{4,1} \wedge S_{3,2} \Rightarrow W_{3,1}$ is a Horn clause.

As particular cases, also **atomic sentences** and their negation can be rewritten as Horn clauses. Indeed, since $(P \Rightarrow Q) \equiv (\neg P \vee Q)$:

$$\begin{array}{rcccc} P & \equiv & \neg True \vee P & \equiv & True \Rightarrow P \\ \neg P & \equiv & \neg P \vee False & \equiv & P \Rightarrow False \end{array}$$

Inference algorithms for Horn clauses

In the particular case when:

- the KB is made up of Horn clauses
- the conclusion is an atomic sentence (a particular case of Horn clause)

two inference algorithms, **Forward Chaining** (FC) and **Backward Chaining** (BC), can be used:

- they are based on a **single** inference rule: **Modus Ponens**
- they are **complete** with respect to **atomic sentences**, i.e., they can determine whether any atomic sentence is or is not entailed by any given KB
- their computational complexity is **linear** in the size of the KB

Forward Chaining

Given a KB made up of Horn clauses, FC derives **all** the entailed atomic sentences (note that this task cannot be carried out by the Resolution inference algorithm):

```
function FORWARD-CHAINING (KB)
    repeat
        apply MP in all possible ways to sentences in KB
        add to KB the derived sentences not already present in it
    until some sentences not yet present in KB have been derived
    return KB
```

Basically, FC applies Modus Ponens to each implication in the KB whose antecedents are in the KB as well, as atomic sentences.

The KB returned by FC contains all the atomic sentences entailed by the input KB (in addition to the original sentences).

Forward Chaining: an example

Consider the KB shown below, made up of Horn clauses:

1. $P \Rightarrow Q$
2. $L \wedge M \Rightarrow P$
3. $B \wedge L \Rightarrow M$
4. $A \wedge P \Rightarrow L$
5. $A \wedge B \Rightarrow L$
6. A
7. B

(cont.)

Forward Chaining: an example

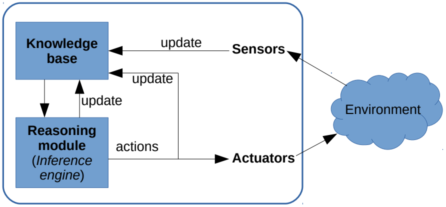
By applying FC one obtains:

8. the only implication whose antecedents (as atomic sentences) are in the KB is 5: MP derives **L** and adds it to the current KB
9. now all antecedents of 3 are in the KB: MP derives **M** and adds it to the KB
10. all antecedents of 2 are now in the KB: MP derives **P** and adds it to the KB
11. the antecedents of 1 and 4 are now all in the KB: MP derives **Q** form 1 and adds it to the KB, but disregards **L** derived from 4, since it is already present in the KB
12. no **new** sentences can be derived from 1–11 using MP: FC ends and returns the updated KB containing the original sentences 1–7 and the newly derived ones, entailed by the original KB: $\{L, M, P, Q\}$

Forward Chaining

FC is an example of **data-driven** reasoning: it starts from some premises (known data), and derives their consequences.

In KBSs this is useful to update the KB after new percepts are received from sensors. For instance, in the wumpus game FC can **update** the agent's knowledge about the environment (e.g., the location of pits), based on the new percepts after each move.



As a notable example, the inference engine of **expert systems** is inspired by the FC inference algorithm.

Backward Chaining

FC can also be used to prove whether or not a conclusion of interest α, in the form of atomic sentence, is entailed by a KB made up of Horn clauses. To this aim, one has to check whether α is present or not among the sentences derived by FC.

This is however not that efficient: why wasting time deriving first **all** entailed sentences, to prove **one** sentence of interest?

To this aim, the more direct and efficient BC inference algorithm can be used: it **recursively** applies MP "backwards", starting from α, exploiting the fact that $KB \models \alpha$, if and only if:

- either $\alpha \in KB$ (this terminates recursion)
- or KB contains some implication $\beta_1, \ldots, \beta_n \Rightarrow \alpha$ such that (recursively) $KB \models \beta_1$, ..., $KB \models \beta_n$

The sentence α to be proven is also called **query**.

Backward Chaining

```
function BACKWARD-CHAINING ($KB$, $\alpha$)
    if $\alpha \in KB$ then return True
    let $B$ be the set of sentences of $KB$ having $\alpha$ as the consequent
    for each $\beta \in B$
        let $\beta_1, \beta_2, \ldots$ be the propositional symbols in the antecedent of $\beta$
        if BACKWARD-CHAINING ($KB$, $\beta_i$) $=$ True for all $\beta_i$'s
        then return True
    return False
```

Backward Chaining: an example

Consider a KB representing the rules followed by a financial institution for deciding whether to grant a loan to an applicant. The following propositional symbols are used:

- OK: the loan should be approved
- $COLLAT$: the collateral for the loan is satisfactory
- $PYMT$: the applicant is able to repay the loan
- REP: the applicant has a good financial reputation
- APP: the appraisal on the collateral is sufficiently greater than the loan amount
- $RATING$: the applicant has a good credit rating
- INC: the applicant has a good, steady income
- BAL: the applicant has an excellent balance sheet

Backward Chaining: an example

The KB is made up of five rules shown below on the left, in the form of implications, and of the data about a specific applicant, encoded by the four sentences on the right; note that of them are Horn clauses.

1. $COLLAT \land PYMT \land REP \Rightarrow OK$
2. $APP \Rightarrow COLLAT$
3. $RATING \Rightarrow REP$
4. $INC \Rightarrow PYMT$
5. $BAL \land REP \Rightarrow OK$
6. APP
7. $RATING$
8. INC
9. $\neg BAL$

Should the loan be approved for this specific applicant?

This amounts to prove whether OK is entailed by the KB, i.e., whether:

$$KB \models OK$$

Backward Chaining: an example

The BC **recursive** proof $KB \vdash_{\mathrm{BC}} OK$ can be conveniently represented as an **AND-OR graph**, a tree-like graph in which:

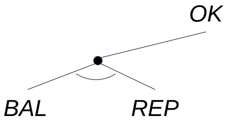
- multiple links joined by an arc indicate a **conjunction**: **every** linked proposition must be proven to prove the proposition in the parent node
- multiple links without an arc indicate a **disjunction**: **any** linked proposition can be proven to prove the proposition in the parent node

Backward Chaining: an example

The first call BACKWARD-CHAINING(KB, OK) is represented by the tree root, corresponding to the sentence to be proven:

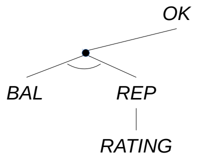
$$OK$$

Since $OK \notin KB$, implications having OK as the consequent are searched for. There are two such sentences: 1 and 5. The BC procedure tries to prove the antecedent of **at least one** of them. Considering first 5, a recursive call to BACKWARD-CHAINING is made for each atomic sentence in its antecedent, represented by an AND-link:

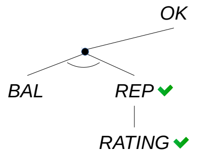


Backward Chaining: an example

Consider the call BACKWARD-CHAINING(*KB*, *REP*): since $REP \notin KB$, and the only implication having *REP* as the consequent is 3, another recursive call is made for the antecedent of 3:

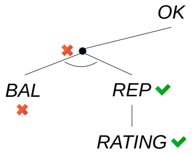


The call BACKWARD-CHAINING(*KB*, *RATING*) returns *True*, since $RATING \in KB$, and thus also the call BACKWARD-CHAINING(*KB*, *REP*) returns *True*:

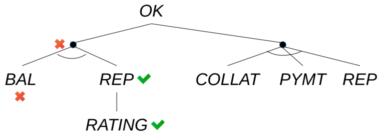


Backward Chaining: an example

However, the call BACKWARD-CHAINING(*KB*, *BAL*) returns *False*, since *BAL* $\notin$ *KB* and there are no implications having *BAL* as the consequent. Therefore, the first call BACKWARD-CHAINING(*KB*, *OK*) is not able to prove *OK* through this AND-link:

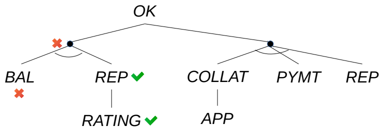


The other sentence in the KB having *OK* as the consequent is 1; another AND-link is generated with three recursive calls for each atomic sentence in the antecedent of 1:

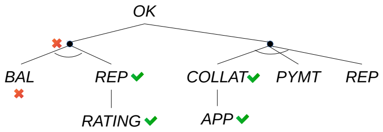


Backward Chaining: an example

The call BACKWARD-CHAINING(*KB*, *COLLAT*) generates in turn another recursive call to prove the antecedent of the only implication having *COLLAT* as the consequent, 2:



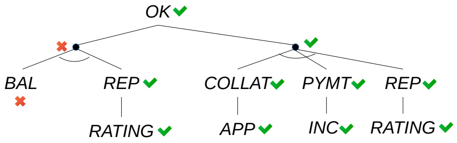
The call BACKWARD-CHAINING(*KB*, *APP*) returns *True*, since $APP \in KB$, and thus also BACKWARD-CHAINING(*KB*, *COLLAT*) returns *True*



Backward Chaining: an example

Similarly, the calls BACKWARD-CHAINING(*KB*, *PYMT*) and BACKWARD-CHAINING(*KB*, *REP*) return *True*.

The corresponding AND-link is then proven, which finally allows the first call BACKWARD-CHAINING(*KB*, *OK*) to return *True*:

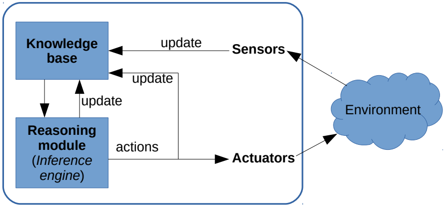


The proof $KB \vdash_{BC} OK$ is then **successfully completed**.
This proves that $KB \models OK$.

Backward Chaining

BC is a form of **goal-directed** reasoning.

In KBSs it is useful to decide which action to perform, based on the current agent's knowledge (KB). For instance, in the wumpus game BC can be used to answer queries like: given the current agent's knowledge, is *moving upward* the best action?



The computational complexity of BC is even **lower** than that of FC, since BC focuses only on sentences **relevant** to the conclusion of interest.

As a notable example, the **Prolog** inductive logic programming language is based on the BC inference algorithm for predicate logic.

Exercise 1

Try to define the agent's initial KB for the wumpus game.

The KB should contain:

- the rules of the game: the agent starts in room (1,1); there is a breeze in rooms adjacent to pits; etc.
- rules to decide the agent's move at each step of the game

Note that the KB must be **updated** at each step of the game:

1. adding percepts in the current room (from **sensors**)
2. **reasoning** to derive new knowledge about the position of pits and wumpus
3. **reasoning** to decide the next move
4. updating the agent's position after a move

Exercise 1

Rules of the wumpus game:

- *the agent starts in room (1,1)*:
  $A_{1,1} \land \neg A_{1,2} \land \ldots \land \neg A_{4,4}$
- *there is a breeze in rooms adjacent to pits*:
  $P_{1,1} \Rightarrow (B_{2,1} \land B_{1,2})$,
  $P_{1,2} \Rightarrow (B_{1,1} \land B_{2,2} \land B_{1,3})$, ...
  (**one** proposition in natural language, **sixteen** sentences in propositional logic – one for each room)
- *there is only one wumpus*:
  $(W_{1,1} \land \neg W_{1,2} \land \neg W_{1,3} \land \ldots \land \neg W_{4,4}) \lor$
  $(\neg W_{1,1} \land W_{1,2} \land \neg W_{1,3} \land \ldots \land \neg W_{4,4}) \lor \ldots$
  (**one** proposition in natural language, **sixteen** sentences in propositional logic – one for each room)
- ...

Often, **one** concise proposition in natural language needs to be represented by **many** complex sentences in propositional logic.

Exercise 1

How to **update** the KB to account for the **change** of the agent's position after each move? E.g., $A_{1,1}$ is true in the starting position, and becomes false after the first move:

- just adding $\neg A_{1,1}$ makes the KB contradictory, since $A_{1,1}$ is still present in it...
- ...on the other hand, inference rules do not allow to **remove** a sentence from the KB...

Solution: using a **different** propositional symbol for each time step, e.g., $A^t_{i,j}$, $t = 1, 2, \ldots$, and **adding** to the initial KB suitable rules, e.g.:

- agent's position in the initial KB: $A^1_{1,1}$, $\neg A^1_{1,2}$, $\ldots \neg A^1_{4,4}$
- rules that update the agents' position after each move:
  $A^1_{1,1} \wedge MoveUp^1 \rightarrow \neg A^2_{1,1}$
  $A^1_{1,1} \wedge MoveUp^1 \rightarrow A^2_{1,2}$
  ...

Things get complicated ...

Exercise 2

The following argumentation (an example of **syllogism**) is intuitively correct; prove its correctness using propositional logic:
All men are mortal; Socrates is a man; then, Socrates is mortal.

Three **distinct** propositional symbols must be used:
P (*All men are mortal*), Q (*Socrates is a man*), R (*Socrates is mortal*)

Therefore:

- premises: $\{P, Q\}$
- conclusion: R

Do the premises **entail** the conclusion, i.e., $\{P, Q\} \models R$?

Exercise 2

Model checking can be easily applied, since there are only three propositional symbols:

Propositional symbols			Premises		Conclusion
P	Q	R	P	Q	R
false	false	false	false	false	false
false	false	true	false	false	true
false	true	false	false	true	false
false	true	true	false	true	true
true	false	false	true	false	false
true	false	true	true	false	true
true	true	false	true	true	**false**
true	true	true	true	true	true

In one of the rows when both premises are true, the conclusion is false; therefore one cannot conclude that Socrates is mortal. . . **what's wrong?**

Limitations of propositional logic

Main problems:

- too limited **expressive power**
- lack of **conciseness**

Example (wumpus world)

Even small knowledge bases (in natural language) require a large number of propositional symbols and sentences.

Example (syllogisms)

Inferences involving the **structure** of **atomic** sentences (e.g., *All men are mortal, . . .*) are not feasible.

From propositional to predicate logic

The description of many real-world domains of interest (e.g., mathematics, philosophy, AI) involve the following elements of natural language:

- **nouns** denoting **individuals** (a person, an animal, a physical object, an entity), e.g.: Socrates, a dog, the sun, the number four
- **predicates** denoting **properties** of individuals or **relations** between them, e.g.: Socrates **is a man**, five **is prime**, four **is lower than** five; the sum of two and two **equals** four
- **functions** that indirectly denote an individual in terms of other ones, e.g.: the **father of Mary**, **one plus three**
- facts involving **some** or **all** individuals of a set, e.g.: **all** squares neighboring the wumpus are smelly; **some** numbers are prime

These elements cannot be represented in propositional logic, and require the more expressive **predicate logic**.

Predicate Logic

Models

In predicate logic a **model** consists of:

- a **domain of discourse**, i.e., the set of all individuals considered in the propositions, e.g., the set of natural numbers, a set of people, etc. ("mixed" domains can be considered, e.g., a set of numbers and people)
- **relations** between domain elements, defined **extensionally**, i.e., enumerating the set of tuples among which a relation holds; e.g. if the domain includes natural numbers:
  - being a prime number (unary relation): $\{1, 2, 3, 5, 7, 11, \ldots\}$
  - being greater than (binary relation): $\{(2,1), (3,1), \ldots\}$
  - being equal to (binary relation): $\{(1,1), (2,2), \ldots\}$

  unary relations are also called **properties**
- **functions** mapping tuples of domain elements to a single one, defined extensionally, e.g.:
  - arithmetic sum: $(1,1) \rightarrow 2$, $(1,2) \rightarrow 3$, ...
  - father of: John $\rightarrow$ Mary, ...

Syntax

The basic elements are symbols that denote domain elements, relations and functions, with upper-case initial by convention:

- **constant symbols**: domain elements (individuals), e.g.: *One*, *Two*, *Three*, *John*, *Mary*
- **predicate symbols**: relations, e.g.: *GreaterThan*, *Prime*, *Sum*, *Father*
- **function symbols** denote functions, e.g.: *Plus*, *FatherOf*

Note that there are no distinctions between constant symbols, predicates and functions names: the category of each name should be specified beforehand.

Syntax

A formal grammar in Backus-Naur Form (BNF):

$$
\begin{array}{rcl}
\mathit{Sentence} & \rightarrow & \mathit{AtomicSentence} \\
 & | & (\mathit{Sentence}\ \mathit{Connective}\ \mathit{Sentence}) \\
 & | & \mathit{Quantifier}\ \mathit{Variable}, \ldots \mathit{Sentence} \\
 & | & \neg\ \mathit{Sentence} \\
\mathit{AtomicSentence} & \rightarrow & \mathit{Predicate}(\mathit{Term}, \ldots) \\
\mathit{Term} & \rightarrow & \mathit{Function}(\mathit{Term}, \ldots) \mid \mathit{Constant} \mid \mathit{Variable} \\
\mathit{Connective} & \rightarrow & \Rightarrow \mid \wedge \mid \vee \mid \Leftrightarrow \\
\mathit{Quantifier} & \rightarrow & \forall \mid \exists \\
\mathit{Constant} & \rightarrow & \mathit{John} \mid \mathit{Mary} \mid \mathit{One} \mid \mathit{Two} \mid \ldots \\
\mathit{Variable} & \rightarrow & a \mid x \mid s \mid \ldots \\
\mathit{Predicate} & \rightarrow & \mathit{GreaterThan} \mid \mathit{Father} \mid \ldots \\
\mathit{Function} & \rightarrow & \mathit{Plus} \mid \mathit{FatherOf} \mid \ldots
\end{array}
$$

Semantics: interpretations

Remember that semantics defines the truth of well-formed sentences, related to a particular model.
In predicate logic this requires an **interpretation**: defining which domain elements, relations and functions are referred to by symbols.

Examples:

- *One*, *Two* and *Three*: the natural numbers 1, 2, 3;
  John and *Mary*: two specific persons named John and Mary
- *GreaterThan*: binary relation "to be greater than" between numbers;
  Sum: ternary relation "to be the sum of two numbers";
  Father: the fatherhood relation between pairs of individuals
- *Plus*: function mapping a pair of numbers to the number corresponding to their sum;
- *FatherOf*: function mapping a person to its father

Semantics: terms

Terms are logical expressions denoting domain elements.
A term can be:

- **simple**: a constant symbol, e.g.: *One*, *Two*, *John*
- **complex**: a function symbol applied (possibly, **recursively**) to other terms, e.g.:
  FatherOf(*Mary*)
  Plus(*One*, *Two*)
  Plus(*One*, *Plus*(*One*, *One*))

Note:

- assigning a constant symbol to every domain element is not required (domains can be even infinite): only elements **explicitly** mentioned in propositions (e.g., Socrates, the number zero) should be assigned a constant symbol
- a domain element can be denoted by more than one symbol

Semantics: atomic sentences

Atomic sentences are the simplest kind of proposition: a predicate symbol applied to a list of terms.

Examples (with a possible translation in natural language):

- $\mathit{GreaterThan}(\mathit{Two}, \mathit{One})$
  two is greater than one
- $\mathit{Prime}(\mathit{Two})$
  two is prime
- $\mathit{Prime}(\mathit{Plus}(\mathit{Two}, \mathit{Two}))$
  two and two is prime
- $\mathit{Sum}(\mathit{One}, \mathit{One}, \mathit{Two})$
  the sum of one and one is two
- $\mathit{Father}(\mathit{John}, \mathit{Mary})$
  John is the father of Mary
- $\mathit{Father}(\mathit{FatherOf}(\mathit{John}), \mathit{FatherOf}(\mathit{Mary}))$
  John's father is the father of Mary's father

Semantics: atomic sentences

An atomic sentence is true, in a given model and under a given interpretation, if the relation referred to by its predicate symbol holds between the objects referred to by its arguments (terms)

Example

According to the above model and interpretation:

- $GreaterThan(One, Two)$ is false
- $Prime(Two)$ is true
- $Prime(Plus(One, One))$ is true
- $Sum(One, One, Two)$ is true
- $Father(John, Mary)$ is true

Semantics: complex sentences

Complex sentences are obtained as in propositional logic, using logical connectives.

Examples (with a possible translation in natural language):

- $Prime(Two) \wedge Prime(Three)$
  two and three are prime numbers
- $\neg Sum(One, One, Two)$
  the sum of one and one is not two
- $GreaterThan(One, Two) \Rightarrow (\neg GreaterThan(Two, One))$
  if one is greater than two then two is grater than one
- $Father(John, Mary) \vee Father(Mary, John)$
  John is the father of Mary, or Mary is the father of Jonh

Semantics (truth value) is determined as in propositional logic.
Examples: the second sentence above is false, the others are true.

A note on predicates and functions

Predicate and function symbols have identical syntax, but different roles

- functions denote **terms** (i.e., domain elements, like constant symbols), and therefore can **only** appear as **arguments** of predicates and (recursively) functions
- predicates denote **propositions**, and **cannot** appear as arguments of predicates or functions

For instance, consider representing in predicate logic the proposition "two is the sum of one and one and is a prime number"

- $Prime(Sum(One, One, Two))$ is not a syntactically correct sentence, thus it has no meaning: Sum is a predicate, not a term
- $Sum(One, One, Two) \wedge Prime(Two)$ is syntactically correct
- $Prime(Plus(One, One))$ is syntactically correct too, since $Plus$ is a function

Semantics: quantifiers

Quantifiers allow one to express propositions involving **collections** of domain elements, **without** enumerating them **explicitly**.

For instance, in natural language we can state

- "**all** rooms neighbouring a pit are breezy", instead of "if room (1,1) neighbours a pit it is breezy, if room (1,2) neighbours a pit it is breezy, ..."
- "**some** natural number is prime", instead of "one is prime, or two is prime, or three is prime, ..."

We have already seen that propositional logic does not exhibit this expressive capability.

Terms like *all*, *every*, *some*, *any*, etc. are called **quantifiers**.

Semantics: quantifiers

Two quantifiers are used in predicate logic:

- **universal** quantifier, e.g.:
  all *men are mortal*
  all *rooms neighboring a pit are breezy*
  all *even numbers that are greater than two are not prime*
- **existential** quantifier, e.g.:
  some *numbers are prime*
  some *rooms contain pits*
  some *men are philosophers*

Quantifiers require a new kind of term: **variable symbols**, usually denoted with lower-case letters.

Semantics: universal quantifier

Example

Assume that the **domain** is the set of natural numbers.
The proposition "All natural numbers are greater than or equal to one" can be represented by the sentence

$$\forall x \; \mathit{GreaterOrEqual}(x, \mathit{One})$$

where

- *GreaterOrEqual* is a binary predicate with intuitive meaning
- x is a variable symbol which can denote any domain element

Semantics: universal quantifier

The semantics of a sentence of the form $\forall x\ \alpha[x]$, where $\alpha[x]$ denotes a sentence containing the variable x, is:

$\alpha[x]$ is true for **each** domain element in place of x

Example

If the domain is the set of natural numbers,

$$\forall x\ \mathit{GreaterOrEqual}(x, \mathit{One})$$

states that the following (infinite) sentences are **all** true:
$\mathit{GreaterOrEqual}(\mathbf{One}, \mathit{One})$
$\mathit{GreaterOrEqual}(\mathbf{Two}, \mathit{One})$
...

Semantics: universal quantifier

Consider the proposition:

"all even numbers greater than two are not prime"

A common mistake is to represent it as:

$$\forall x \; \mathit{Even}(x) \land \mathit{GreaterThan}(x, \mathit{Two}) \land (\neg \mathit{Prime}(x))$$

The above sentence corresponds to "all numbers are even, greater than two, and are not prime", which is different from the intended meaning.

Actually, the property of being not prime is not attributed to all natural numbers, but only to all the elements of a **subset** of natural numbers satisfying some other property (being even and greater than two).

Semantics: universal quantifier

The correct sentence can be obtained by restating the original proposition as:

> "**for all** x, **if** x is even and greater than two,
> **then** x is not prime"

This proposition states a sufficient condition, and therefore can be represented using the **implication** connective:

$$\forall x \ (Even(x) \land GreaterThan(x, Two)) \Rightarrow (\neg Prime(x))$$

In general, propositions where "all" refers to all the elements of a **subset** of the domain that **satisfy some condition** should be represented using an **implication**.

Semantics: universal quantifier

Consider again this sentence:

$$\forall x \ (\mathit{Even}(x) \land \mathit{GreaterThan}(x, \mathit{Two})) \Rightarrow (\neg \mathit{Prime}(x))$$

Claiming that it is true corresponds to claim that also sentences like the following ones are true:

$$(\mathit{Even}(\mathit{One}) \land \mathit{GreaterThan}(\mathit{One}, \mathit{Two})) \Rightarrow (\neg \mathit{Prime}(\mathit{One}))$$

Note that the antecedent of the implication is false (the number one is not even, nor it is greater than the number two). This is not contradictory, since implications with false antecedents are true **by definition** (see again the truth table of $\Rightarrow$).

Semantics: existential quantifier

Example

Assume the domain is the set of natural numbers.

- "Some numbers are prime" can be represented as

  $$\exists x \; Prime(x)$$

  This is read as: "there exists some x such that x is prime"

- "Some numbers are not greater than three, and are even" can be represented as

  $$\exists x \; \neg GreaterThan(x, Three) \wedge Even(x)$$

Semantics: existential quantifier

Consider a proposition like the following:

“some odd numbers are prime”

A common mistake is to represent it using an implication:

$$\exists x \; \mathit{Odd}(x) \Rightarrow \mathit{Prime}(x)$$

The above sentence actually corresponds to “for some natural numbers, being odd is a necessary condition to be prime”, which does not correspond to the original proposition.

(cont.)

Semantics: existential quantifier

The sentence corresponding to the original proposition can be obtained by restating the latter as:

“**there exists** some x such that x is odd **and** x is prime”

The above proposition can be represented using a **conjunction**:

$$\exists x \; \mathit{Odd}(x) \wedge \mathit{Prime}(x)$$

In general, propositions stating several properties about some of the domain elements should be represented using a **conjunction**.

Semantics: nested quantifiers

A sentence can contain more than one quantified variable.
If the quantifier is the same for all variables, e.g.:

$$\forall x(\forall y(\forall z \ldots \alpha[x, y, z, \ldots] \ldots))$$

then the sentence can be rewritten more concisely as:

$$\forall x, y, z \ldots \alpha[x, y, z, \ldots]$$

For instance, in the domain of natural numbers, the proposition:

> "if a number x is greater than another number y,
> then also the successor of x is greater than y"

can be represented by the following sentence, where *Successor* denotes a function associating to each natural number its successor:

$$\forall x, y \; \mathit{GreaterThan}(x, y) \Rightarrow \mathit{GreaterThan}(\mathit{Successor}(x), y)$$

Semantics: nested quantifiers

If a sentence contains both universally and existentially quantified variables, its meaning depends on the **order** of quantification. In particular, $\forall x(\exists y\ \alpha[x, y])$ and $\exists y(\forall x\ \alpha[x, y])$ are **not** equivalent, i.e., they are not true under the **same** models.

For instance,

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

states that (i.e., is true under any model in which) "everybody loves somebody". Note that the domain element denoted by y can be **different** for different x's.

Instead,

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

states that "there is someone who is loved by everyone" (now the domain element denoted by y must be the **same** for **all** the x's). Therefore the meaning of the above sentences is different, i.e., they can be true under **different** sets of models.

Semantics: connections between quantifiers

The quantifiers $\forall$ and $\exists$ are connected with each other through **negation**, just like in natural language.

For instance, asserting that "every natural number is greater than or equal to zero" is the same as asserting that "there does not exist any natural number which is not greater than or equal to zero".

Assuming the domain is the set of natural numbers, the two propositions above can be respectively represented by the following, **equivalent** sentences:

$$\forall x \; \mathit{GreaterOrEqual}(x, \mathit{Zero})$$
$$\neg \, (\exists x \; \neg \mathit{GreaterOrEqual}(x, \mathit{Zero}))$$

Semantics: connections between quantifiers

In general, since $\forall x\ \alpha[x]$ and $\exists x\ \alpha[x]$ are equivalent respectively to a conjunction ($\wedge$) and to a disjunction ($\vee$) of sentences $\alpha[x]$ over **all** domain elements, they obey De Morgan's rules (shown below on the left, in the usual form involving two propositional variables):

$$
\begin{array}{rcl|rcl}
\neg P \wedge \neg Q & \equiv & \neg(P \vee Q) & \forall x(\neg\alpha[x]) & \equiv & \neg(\exists x\ \alpha[x]) \\
\neg(P \wedge Q) & \equiv & (\neg P) \vee (\neg Q) & \neg(\forall x\ \alpha[x]) & \equiv & \exists x(\neg\alpha[x]) \\
P \wedge Q & \equiv & \neg(\neg P \vee \neg Q) & \forall x\ \alpha[x] & \equiv & \neg(\exists x(\neg\alpha[x])) \\
P \vee Q & \equiv & \neg(\neg P \vee \wedge Q) & \exists x\ \alpha[x] & \equiv & \neg(\forall x\ (\neg\alpha[x]))
\end{array}
$$

Exercises

Represent the following propositions using sentences in predicate logic, including the definition of the domain

1. All men are mortal; Socrates is a man; Socrates is mortal
2. All rooms neighboring a pit are breezy (wumpus game)
3. A subset of Peano-Russell's axioms of arithmetic, that define natural numbers (nonnegative integers):
   - P1 zero is a natural number
   - P2 the successor of any natural number is a natural number
   - P3 zero is not the successor of any natural number
   - P4 no two natural numbers have the same successor
4. The law says that it is a crime for an American to sell weapons to hostile countries. 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 (*assume the goal is to prove that West is a criminal*)

Solution of exercise 1

A **possible** choice of domain and symbols:

- domain: a set including all human beings
- constant symbols: *Socrates*
- predicate symbols: *Man* and *Mortal*, unary predicates
  e.g., $Man(Socrates)$ means “Socrates is a man”

Sentences in predicate logic:

$$\forall x \; Man(x) \Rightarrow Mortal(x)$$
$$Man(Socrates)$$
$$Mortal(Socrates)$$

Solution of exercise 2

A **possible** choice of domain and symbols:

- domain: row and column coordinates
- constant symbols: 1, 2, 3, 4
- predicate symbols:
  - *Pit*, binary predicate; e.g., $P(1,2)$ means that there is a pit in room (1,2)
  - *Adjacent*, predicate with four terms; e.g., $Adjacent(1,1,1,2)$ means that room (1,1) is adjacent to room (1,2)
  - *Breezy*, binary predicate; e.g., $Breezy(2,2)$ means that there is a breeze in room (2,2)

Solution of exercise 2

To represent in predicate logic the considered proposition, it can be restated as follows:

"if a room contains a pit, then all the adjacent rooms are breezy"

This represents a **sufficient** condition, not a necessary one, since all the rooms adjacent to a given room R can be breezy, even if R does not contain a pit (due to pits in other rooms than R).
The above proposition can therefore be represented using the **implication** connective:

$$\forall x, y\ Pit(x, y) \Rightarrow (\forall p, q\ Adjacent(x, y, p, q) \Rightarrow Breezy(p, q))$$

(cont.)

Solution of exercise 2

One could however argue that the original proposition does not **completely** represent the corresponding rule of the wumpus game, that can be stated as follows:

> "a room is breezy, if and only if at least one of the adjacent rooms contains a pit"

It is easy to see that the above proposition states a **necessary** and **sufficient** condition, which can be represented as an **equivalence**:

$$\forall x, y \; Breezy(x, y) \Leftrightarrow (\exists p, q \; Adjacent(x, y, p, q) \wedge Pit(p, q))$$

Solution of exercise 3

A **possible** choice of domain and symbols:

- domain: any set including all natural numbers (e.g., the set of real numbers)
- constant symbols: Z, denoting the number zero
- predicate symbols:
  - N, unary predicate denoting the fact of being a natural number; e.g., $N(Z)$ means that zero is a natural number
  - Eq, binary predicate denoting equality; e.g., $Eq(Z, Z)$ means that zero equals zero
- function symbols: S, mapping a natural number to its successor; e.g., $S(Z)$ denotes one, $S(S(Z))$ denotes two

Solution of exercise 3

Sentences in predicate logic:

P1 $N(Z)$

P2 $\forall x \; N(x) \Rightarrow N(S(x))$

P3 $\neg(\exists x \; Eq(Z, S(x)))$

P4 $\forall x, y \; Eq(S(x), S(y)) \Rightarrow Eq(x, y)$

Solution of exercise 4

A **possible** choice of domain and symbols:

- domain: a set including different individuals (among which Colonel West), nations (among which America and Nono), and missiles
- constant symbols: *West*, *America* and *Nono*
- predicate symbols:
  - $Country(\cdot)$, $American(\cdot)$, $Missile(\cdot)$, $Weapon(\cdot)$, $Hostile(\cdot)$: respectively, being: a country, an American citizen, a missile, a weapon, hostile to America
  - $Enemy(< who >, < to\ whom >)$: being enemies
  - $Owns(< who >, < what >)$: owning something
  - $Sells(< who >, < what >, < to\ whom >)$: selling something to someone
- no function symbols are necessary

Solution of exercise 4

The law says that it is a crime for an American to sell weapons to hostile nations:

$$\forall x, y, z\ (American(x) \land Country(y) \land Hostile(y) \land Weapon(z) \land Sells(x, y, z)) \Rightarrow Criminal(x)$$

The second proposition can be conveniently split into simpler ones:

Nono is a country...:

$Country(Nono)$

...Nono is an enemy of America (which is also a country)...:

$Enemy(Nono, America)$

$Country(America)$

...Nono has some missiles...:

$\exists x\ Missile(x) \land Owns(Nono, x)$

...all Nono's missiles were sold to it by Colonel West:

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

Solution of exercise 4

A human would intuitively say that the above propositions in natural language imply that West is a criminal.
However, it is not difficult to see that the above sentences in predicate logic are **not** sufficient to prove this.
The reason is that humans exploit **background knowledge** (or **common sense**) that is **not** represented **explicitly** in the above propositions. In particular, there are two "missing links":

- an enemy nation is hostile to America
- a missile is a weapon

To enable using such additional knowledge for inference, it must be **explicitly** represented in predicate logic:

- $\forall x, y \; (\mathit{Country}(x) \land \mathit{Enemy}(x, \mathit{America})) \Rightarrow \mathit{Hostile}(x)$
- $\forall x \; \mathit{Missile}(x) \Rightarrow \mathit{Weapon}(x)$

Knowledge engineering

Knowledge engineering is the process of constructing a KB.

It consists of investigating a specific domain, identifying the relevant concepts (**knowledge acquisition**), and formally representing them.

This requires the interaction between

- a **domain expert** (DE)
- a **knowledge engineer** (KE), who is expert in knowledge representation and inference, but usually **not** in the domain of interest

A possible approach, suitable for **special-purpose** KBs (in predicate logic), is the following.

Knowledge engineering

1. Identify the task:
   - what range of queries will the KB support?
   - what kind of facts will be available for each problem instance?
2. Knowledge acquisition: eliciting from the domain expert the **general** knowledge about the domain (e.g., the rules of chess)
3. Choice of a **vocabulary**: what concepts have to be represented as objects, predicates, functions?
   The result is the domain's **ontology**, which affects the complexity of the representation and the inferences that can be made.
   E.g., in the wumpus game pits can be represented either as objects, or as unary predicates on squares.

(cont.)

Knowledge engineering

4. Encoding the domain's general knowledge acquired in step 2 (this may require to revise the vocabulary of step 3)
5. Encoding a specific problem instance (e.g., a specific chess game)
6. Posing queries to the inference procedure and getting answers
7. Debugging the KB, based on the results of step 6

Applications of predicate logic

Logic programming languages, in particular **Prolog**, used for:

- rapid prototyping
- symbol processing (compilers, natural language parsers, etc.)
- developing expert systems

Example of a Prolog **clause**:

```
criminal(X) :- american(X), weapon(Y), sells(X,Y,Z),
               hostile(Z).
```

Running a program consists of proving a sentence (**query**) using a specific inference algorithm, e.g., in the Colonel West example:

- `?- criminal(west)`
  would produce `Yes`
- `?- criminal(A)`
  would produce `A=west, Yes`

Applications of predicate logic

Theorem provers, used for:

- assisting mathematicians in developing complex proofs
- proof checking
- verification and synthesis of hardware and software
  - hardware design, e.g., entire CPUs
  - programming languages: syntax checking
  - software engineering: verifying program specifications, e.g., RSA public key encryption algorithm

Other applications

Encoding condition-action rules to recommend actions, based on a data-driven approach: **expert systems** and **production systems** (production: condition-action rule).

Expert systems

- encoding human experts' problem-solving knowledge in the form of `IF...THEN...` rules, in **specific** application domains for which **no algorithmic solutions exist** (e.g., medical diagnosis)
- used as **decision support systems**, to support (not to replace) experts' decisions
- popular in the 1980s, used in niche/restricted domains since the 1990s (medical diagnosis, geology, finance, military strategies, software engineering, help desk)
- examples of free Expert System shells
  - CLIPS (C Language Integrated Production System) `https://www.clipsrules.net/`
  - Drools (Business Rules Management System) `http://www.drools.org`

Beyond classical logic

Classical logic is based on two principles:

- **bivalence**: there exist only two truth values, *true* and *false*
- **determinateness**: each proposition has only one truth value

But: how to deal with propositions like the following ones?

- *Tomorrow will be a sunny day*: is this true or false, **today**?
- *John is tall*: is this “completely” true (or false)?
  This kind of problem is addressed by **fuzzy logic**
- Goldbach’s conjecture: *Every even number is the sum of a pair of prime numbers*
  No proof has been found yet: can we still say this is either true or false?