ESCOLA SUPERIOR DE TECNOLOGIA IPCA

Logical Agents

Fundamentals of Artificial Intelligence

MSc in Applied Artificial Intelligence, 2023-24

Contents

- Topics included
  - Logical Agents
  - Propositional Logic
  - First-Order Logic

- These slides were based essentially on the following bibliography:
  - Norvig, P, Russell, S. (2021). Artificial Intelligence: A Modern Approach, 4th Edition. Pearson, ISBN-13: 978-1292401133

Logical Agents

Intelligent Agents

- Agent types from previous sections
  - They have just the knowledge that is useful for finding a path from the start to a goal
  - This knowledge is not useful for anything else
  - The atomic representations used by problem-solving agents are also very limiting

- The Constraint Satisfaction Problems (CSP) use a factored representation
  - States are represented as assignments of values to variables
  - This is a step in the right direction, enabling some parts of the agent to work in a domain-independent way and allowing for more efficient algorithms

- Knowledge-based agents use a **process of reasoning** over an **internal representation of knowledge** to decide what actions to take.

Knowledge-Based Agents

- For Logical Agents, we go further
  - we develop logic as a general class of representations to support knowledge-based agents

- A generic knowledge-based agent
  - Given a percept, the agent adds the percept to its knowledge base, asks the knowledge base for the best action, and tells the knowledge base that it has in fact taken that action.

function KB-AGENT(*percept*) **returns** an *action*
  persistent: *KB*, a knowledge base
    t, a counter, initially 0, indicating time

  TELL(*KB*, MAKE-PERCEPT-SENTENCE(*percept*, *t*))
  action ← ASK(*KB*, MAKE-ACTION-QUERY(*t*))
  TELL(*KB*, MAKE-ACTION-SENTENCE(*action*, *t*))
  t ← *t* + 1
  return *action*

Knowledge base

- A knowledge base (KB) is a set of sentences
  - Here “sentence” is related but not identical to the sentences of English and other natural languages
  - Each sentence is expressed in a language called a knowledge representation language and represents some assertion about the world.
- An **Axiom** is a sentence not derived from other sentences
- There are operations to add new sentences to the knowledge base and to query what is known
  - The standard names for these operations are TELL and ASK
  - Both operations may involve **inference** — that is, deriving new sentences from old.
- Inference must obey the requirement that when one ASKs a question of the knowledge base, the answer should follow from what has been told to the knowledge base previously
- Each time the agent program is called, it does three things
  - First, it TELLs the knowledge base what it perceives
  - Second, it ASKs the knowledge base what action it should perform. In the process of answering this query, extensive reasoning may be done about the current state, the outcomes of possible action sequences, etc.
  - Third, the agent program TELLs the knowledge base which action was chosen and returns the action so that it can be executed.

KB agent implementation

- The details of the representation language are hidden inside three functions that implement the interface between the sensors and actuators on one side and the core representation and reasoning system on the other
  - MAKE-PERCEPT-SENTENCE constructs a sentence asserting that the agent perceived the given percept at the given time.
  - MAKE-ACTION-QUERY constructs a sentence that asks what action should be done at the current time.
  - Finally, MAKE-ACTION-SENTENCE constructs a sentence asserting that the chosen action was executed.
- The KB agent is not an arbitrary program for calculating actions
  - It is amenable to a description at the **knowledge level**, where we need specify only what the agent knows and what its goals are in order to determine its behavior
- A knowledge-based agent can be built simply by TELLing it what it needs to know
  - In the **declarative approach** the agent is built starting with an empty knowledge base, the agent designer can TELL sentences one by one until the agent knows how to operate in its environment
  - In the **procedural approach** desired behaviors are directly inserted in the program code.
- A successful agent often combines both declarative and procedural elements in its design
  - Declarative knowledge can often be compiled into more efficient procedural code.

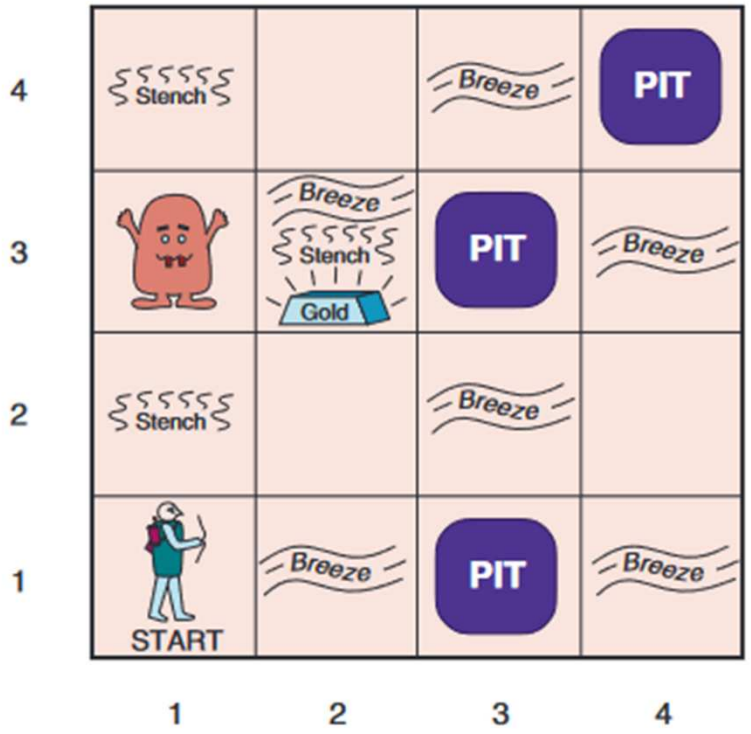
The Wumpus World

The Wumpus world is a cave consisting of rooms connected by passageways

- Hang about somewhere is the terrible wumpus, a beast that eats anyone who enters its room.
- The wumpus can be shot by an agent, but the agent has only one arrow.
- Some rooms contain bottomless pits that will trap anyone who wanders into these rooms (except for the Wumpus which is too big to fall in).
- The only saving feature of this unwelcoming environment is the possibility of finding a stack of gold.

Although the Wumpus world is rather tame by modern computer game standards, it illustrates some important points about intelligence.

A typical wumpus world. The agent is in the bottom left corner, facing east.



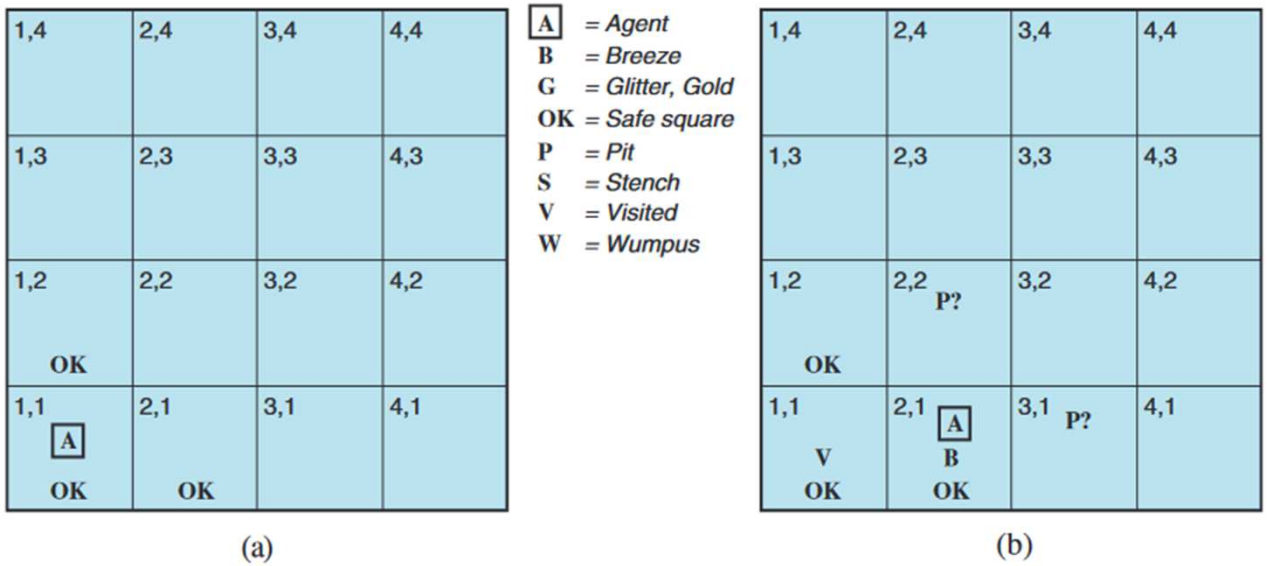
Wumpus example task environment

- The PEAS description gives a precise picture of the task environment of the sample Wumpus world:
- **Performance** measure — the game ends either when the agent dies or when the agent climbs out of the cave
  - +1000 for climbing out of the cave with the gold, –1000 for falling into a pit or being eaten,
  - –1 for each action taken, and –10 for using up the arrow
- **Environment** — A 4 × 4 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. Each square can be a pit with probability 0.2. The start square just has the agent.
- **Actuators** — The agent can move Forward, TurnLeft by 90 ◦ , or TurnRight by 90 ◦. The agent does not move against the walls. The agent dies if it enters a square containing a pit or a live wumpus.
  - The action Shoot can be used to fire an arrow in a straight line in the direction the agent is facing. The agent has only one arrow, so only the first Shoot action has any effect.
  - The action Grab can be used to pick up the gold.
  - The action Climb can be used to climb out of the cave from square [1,1].
- **Sensors** — The agent has five sensors, each of which gives a single bit of information

The first step taken by the agent in the Wumpus world

The agent **five sensors**:

- 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.



(a) (b)

a) The initial situation, after percept [None, None, None, None, None].

b) After moving to [2,1] and perceiving [None, Breeze, None, None, None].

Logic

- A **Knowledge base (KB)** consist of sentences
- Sentences are expressed according to the **syntax** of the representation language,
  - The notion of syntax is clear enough in ordinary arithmetic
    "x + y = 4" is a well-formed sentence, whereas "x4y+ =" is not
- The **semantics** defines the truth of each sentence with respect to each **model (**possible world).
  - For example, the semantics 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.
  - Every sentence must be either true or false in each—there is no "in between."
- Formally, the possible models are just all possible assignments of nonnegative integers to the variables x and y.
  - Each assignment determines the truth of any sentence of arithmetic whose variables are x and y.
  - 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**.

Entailment

- Now that we have a notion of truth, we are ready to talk about logical reasoning
  - This involves the relation of logical entailment between sentences: a sentence follows logically from another sentence.
  - In mathematical notation, the sentence "α entails the sentence β" is $\alpha \models \beta$ (we will use "|==")
  - This is true if and only if (**iff**), in every model in which α is true, β is also true.

- Using the notation just introduced, we can write

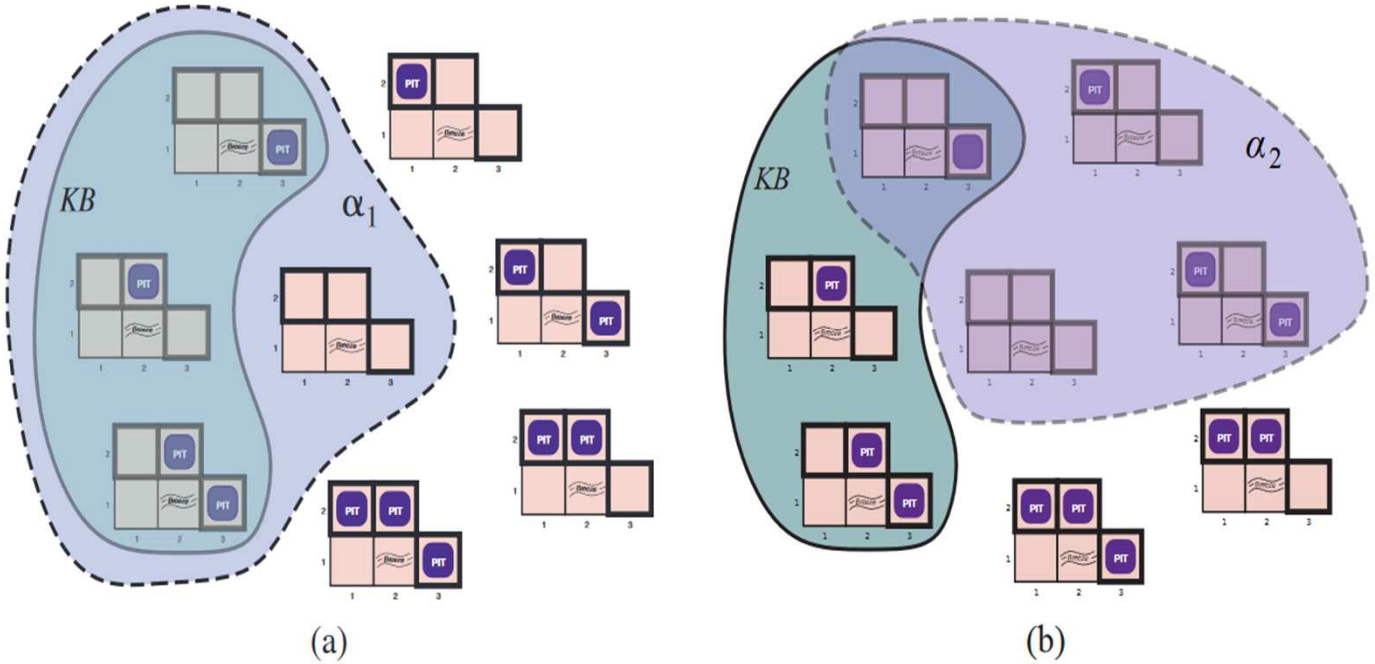
$$\alpha \models \beta \text{ if and only if } M(\alpha) \subseteq M(\beta)$$

- Let us suppose the agent decides to move forward to [2,1]
  - The agent has detected nothing in [1,1]
  - The agent detected a breeze in [2,1]

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

Possible models for pits in squares [1,2], [2,2], and [3,1]

- The KB consists of the percepts combined with the agent's knowledge of the rules
- The agent is interested in whether the adjacent squares [1,2], [2,2], and [3,1] contain pits.
- Each of the three squares might or might not contain a pit, so (ignoring other aspects of the world for now) there are 2^3 = 8 possible models.



- The KB corresponding to the observations in [1,1] and [2,1] is shown by the solid line.

  (a) Dotted line shows models of **α1 (no pit in [1,2])** (b) Dotted line shows models of **α2 (no pit in [2,2])**.

Entailment illustration

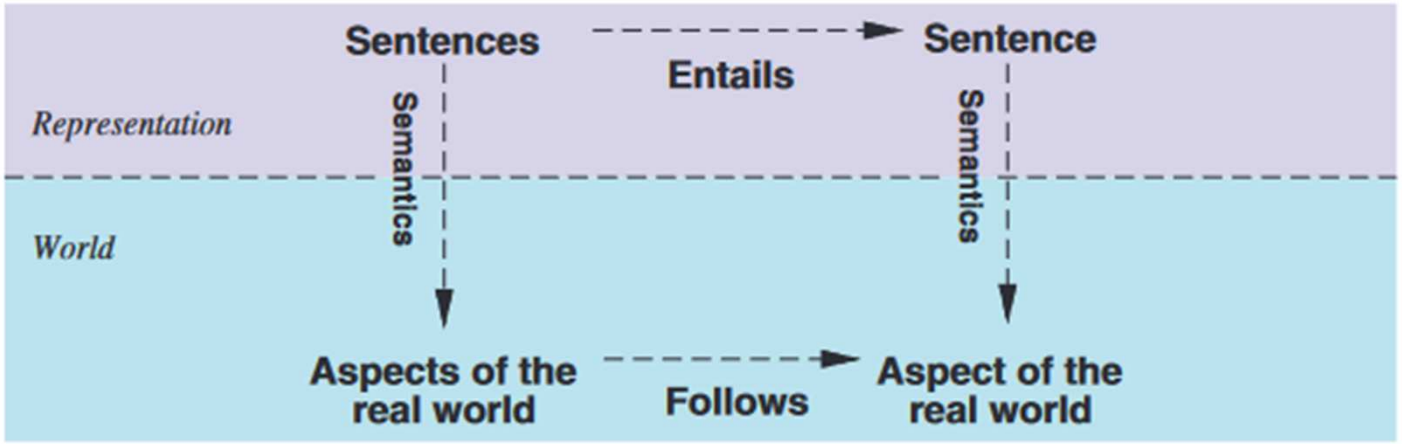
- The KB can be thought of as a set of sentences or as a single sentence that asserts all the individual sentences.
  - The KB is false in models that contradict what the agent knows — for example, the KB is false in any model in which [1,2] contains a pit, because there is no breeze in [1,1].
  - There are in fact just three models in which the KB is true, and these are shown surrounded by a solid line in previous Figure.
- Now let us consider two possible conclusions:
  - α1 = "There is no pit in [1,2]."
  - α2 = "There is no pit in [2,2]."
- We have surrounded the models of sentences α1 and α2 with dotted lines. We see that:
  - In every model in which KB is true, sentence α1 is also true.
- Hence, KB |== α1 : there is no pit in [1,2]. We can also see that:
  - In some models in which KB is true, sentence α2 is false.
- Hence, **KB does not entail α2**:
  - the agent cannot conclude that there is no pit in [2,2]. Nor can it conclude that there is a pit in [2,2].

Logical inference and model checking

- The preceding example also shows how the definition of entailment can be applied to derive conclusions, carrying out **logical inference**.
- The inference algorithm is called **model checking**, because it enumerates all possible models to check that α is true in all models in which KB is true, that is, that M(KB) ⊆ M(α).
- In understanding entailment and inference, it might help to think of The set of all consequences of KB is like a haystack and α as a needle
  - Entailment is like the needle being in the haystack; inference is like finding it.
  - This distinction is embodied in some formal notation:
  - if an inference algorithm *i* can derive α from KB, we write **KB $\vdash_i$ α** , which is pronounced “α is derived from KB by i” or “i derives α from KB."

Representation versus World

- Sentences are physical configurations of the agent, and reasoning is a process of constructing new physical configurations from old ones
- Logical reasoning should ensure that the new configurations represent aspects of the world that actually follow from the aspects that the old configurations represent



Soundness and completeness properties

- An inference algorithm that derives only entailed sentences is called **sound** or **truth preserving**.
  - An unsound inference procedure essentially makes things up as it goes along — it announces the discovery of nonexistent needles, i.e., sentences that are false in some models in which KB is true.
  - It is easy to see that model checking, when it is applicable, is a sound procedure.
- An inference algorithm is **complete** if it can derive any sentence that is entailed
  - For real haystacks, representing a finite set of sentences, it seems obvious that a systematic examination can always decide whether the needle is in the haystack.
  - For many KBs, the haystack of consequences is infinite, and completeness becomes an important issue
  - There are complete inference procedures for logics that are sufficiently expressive to handle many KBs
- If KB is true in the real world, then any sentence α derived from KB by a sound inference procedure is also true in the real world
  - An inference process operates on “syntax”, like internal physical configurations such as bits in registers or patterns of electrical blips in brains
  - This process corresponds to the real-world relationship between real world aspects (see previous figure)

Grounding

- **Grounding** is the connection between logical reasoning processes and the real environment in which the agent exists.
- How do we know that KB is true in the real world?
  - A simple answer is that the agent's sensors create the connection
  - The wumpus-world agent has a smell sensor. The agent program creates a suitable sentence whenever there is a smell. Then, whenever that sentence is in the KB, it is true in the real world.
- The meaning and truth of percept sentences are defined by the processes of sensing and sentence construction that produce them
- What about the rest of the agent's knowledge, such as its belief that wumpuses cause smells in adjacent squares?
  - This is a general rule, derived, perhaps, from perceptual experience but not identical to a statement of that experience.
  - General rules like this are produced by a sentence construction process called learning

Propositional Logic

Proposition symbols

- 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.
- We use symbols that start with an uppercase letter and may contain other letters or subscripts, for example: P, Q, R, W1,3 and *FacingEast*.
- The names are arbitrary but are often chosen to have some mnemonic value
  - We use $W_{1,3}$ to stand for the proposition that the wumpus is in [1,3].
  - Symbols such as $W_{1,3}$ are atomic, i.e., W , 1, and 3 are not meaningful parts of the symbol.)
- There are two **proposition symbols** with fixed meanings:
  - **True** is the always-true proposition
  - **False** is the always-false proposition.
- Complex sentences are constructed from simpler sentences, using parentheses and operators called **logical connectives**.

Logical connectives

- $\neg$ (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**.

Backus–Naur Form (BNF) notation

- The BNF grammar is augmented with an operator precedence list to remove ambiguity when multiple operators are used.
- The “not” operator (¬) has the highest precedence, which means that in the sentence ¬A ∧ B the ¬ binds most tightly, giving us the equivalent of (¬A) ∧ B rather than ¬(A ∧ B).
- The notation for ordinary arithmetic is the same: −2 + 4 is 2, not –6.)
- When appropriate, we also use parentheses and square brackets to clarify the intended sentence structure and improve readability.

$$
\begin{aligned}
\textit{Sentence} &\rightarrow \textit{AtomicSentence} \mid \textit{ComplexSentence} \\
\textit{AtomicSentence} &\rightarrow \textit{True} \mid \textit{False} \mid P \mid Q \mid R \mid \ldots \\
\textit{ComplexSentence} &\rightarrow (\textit{Sentence}) \\
&\mid \neg \textit{Sentence} \\
&\mid \textit{Sentence} \wedge \textit{Sentence} \\
&\mid \textit{Sentence} \vee \textit{Sentence} \\
&\mid \textit{Sentence} \Rightarrow \textit{Sentence} \\
&\mid \textit{Sentence} \Leftrightarrow \textit{Sentence}
\end{aligned}
$$

- Figure shows a BNF (Backus–Naur Form) grammar of sentences in propositional logic, along with operator precedences, from highest to lowest.
- Operator precedence: ¬, ∧,∨, ⇒, ⇔

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.

- The models are purely mathematical objects with no necessary connection to wumpus worlds.
  - $P_{1,2}$ is just a symbol;
  - it might mean “there is a pit in [1,2]” or “I’m in Paris today and tomorrow.”

- The semantics 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, so we need to specify
  - how to compute the truth of atomic sentences
  - how to compute the truth of sentences formed with each of the five connectives.

- **Atomic sentences** are 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.

Complex sentences

- There five rules, which hold for any subsentences P and Q (atomic or complex) in any model **m**
  - • ¬P is true iff P is false in m ("iff" means "if and only if")
  - • P ∧ Q is true iff both P and Q are true in m
  - • P ∨ Q is true iff either P or Q is true in m
  - • P ⇒ Q is true unless P is true and Q is false in m
  - • P ⇔ 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

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*

A simple knowledge base

- To construct a knowledge base for the wumpus world we focus first on the immutable aspects of the wumpus world, leaving the mutable aspects for a later section.
- 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].

Each sentence R_i is labelled so we can refer to them

- There is no pit in [1,1]
  - R_1 : $\neg P_{1,1}$
- A square is breezy if and only if there is a pit in a neighboring square
  - R_2 : $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
  - R_3 : $B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$
- The breeze percepts for the first two squares visited in the specific world the agent
  - R_4 : $\neg B_{1,1}$
  - R_5 : $B_{2,1}$

A simple inference procedure

- Our goal now is to decide whether KB |== α for some sentence α.
  - For example, is ¬$P_{1,2}$ entailed by our KB?
- The model-checking approach that is a direct implementation of the definition of entailment
  - 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.
- In the wumpus-world example, the relevant proposition symbols are $B_{1,1}$, $B_{2,1}$, $P_{1,1}$, $P_{1,2}$, $P_{2,1}$, $P_{2,2}$, and $P_{3,1}$
  - With seven symbols, there are 2^7 = 128 possible models
  - in three of these, KB is true.
  - In those three models, ¬$P_{1,2}$ is true, hence there is no pit in [1,2].
  - On the other hand, $P_{2,2}$ 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]

General algorithm for deciding entailment

Like the BACKTRACKING-SEARCH algorithm TT-ENTAILS? performs a recursive enumeration of a finite space of assignments to symbols.

- The algorithm is **sound** because it implements directly the definition of entailment
- It is **complete** because it works for any KB and α and always terminates

```
function TT-ENTAILS?(KB, α) returns true or false
  inputs: KB, the knowledge base, a sentence in propositional logic
          α, the query, a sentence in propositional logic

  symbols ← a list of the proposition symbols in KB and α
  return TT-CHECK-ALL(KB, α, symbols, { })

function TT-CHECK-ALL(KB, α, symbols, model) returns true or false
  if EMPTY?(symbols) then
     if PL-TRUE?(KB, model) then return PL-TRUE?(α, model)
     else return true        // when KB is false, always return true
  else
     P ← FIRST(symbols)
     rest ← REST(symbols)
     return (TT-CHECK-ALL(KB, α, rest, model ∪ {P = true})
             and
             TT-CHECK-ALL(KB, α, rest, model ∪ {P = false })
```

Propositional theorem proving

- We have seen than entailment can be determined by **model checking**
  - Enumerating models and showing that the sentence must hold in all models.
- Entailment can also be done by **theorem proving**
  - Applying rules of inference directly to the sentences in our knowledge base to construct a proof of the desired sentence without consulting models.
- If the number of models is large but the length of the proof is short, then theorem proving can be more efficient than model checking.

Logical equivalence and validity

- **Logical equivalence**: two sentences α and β are logically equivalent if they are true in the same set of models.
  - We write this as α ≡ β
  - Note that ≡ is used to make claims about sentences while ⇔ is used as part of a sentence
  - For example, we can easily show (using truth tables) that P ∧ Q and Q ∧ P are logically equivalent
- The equivalences play much the same role in logic as arithmetic identities do in ordinary mathematics.
- An alternative definition of equivalence is as follows
  - any two sentences α and β are equivalent if and only if each of them entails the other:

$$\alpha \equiv \beta \quad \text{if and only if} \quad \alpha \models \beta \text{ and } \beta \models \alpha$$

- **Validity**: a sentence is valid if it is true in all models. For example, the sentence P∨¬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.
- What good are valid sentences? From our definition of entailment, we can derive the deduction theorem

For any sentences α and β, α |== β if and only if the sentence (α ⇒ β) is valid

Satisfiability

- A sentence is satisfiable if it is true in, or satisfied by, ***some*** model.
  - 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, as shown previously
- Satisfiability can be checked by enumerating the possible models until one is found that satisfies the sentence
- Many problems in computer science are really satisfiability problems
  - For example, the CSP problems ask whether the constraints are satisfiable by some assignment
- Validity and satisfiability are of course connected: α is valid iff ¬α is unsatisfiable
  - Contrapositively, α is satisfiable iff ¬α is not valid
  - Useful result: α |== β if and only if the sentence (α ∧ ¬β) is unsatisfiable.

- Proving β from α by checking the unsatisfiability of (α ∧ ¬β) corresponds exactly to the standard mathematical proof technique of reductio *ad absurdum*. It is also called proof by **refutation** or proof by **contradiction.**
  - One assumes a sentence β to be false and shows that this leads to a contradiction with known axioms α.
  - This contradiction is exactly what is meant by saying that the sentence (α ∧ ¬β) is unsatisfiable.

First-Order Logic

Representation Revisited

- Programming languages (such as C++ or Python) are the largest class of formal languages in common use.
  - Data structures within programs can be used to represent facts
  - For example, a program could use a 4 × 4 array to represent the contents of the wumpus world
  - The statement *World[2,2]* ← *Pit* is a fairly natural way to assert that there is a pit in square [2,2]
  - Putting together a string of such statements is sufficient for running a simulation of the wumpus world
- However, programming languages lack a general mechanism for deriving facts from other facts
  - each update to a data structure is done by a domain-specific procedure whose details are derived by the programmer from his or her own knowledge of the domain.
  - This procedural approach can be contrasted with the declarative nature of propositional logic, in which knowledge and inference are separate, and inference is entirely domain independent
  - SQL language take a mix of declarative and procedural knowledge.
- A second drawback of data structures in programs is the lack of any easy way to say, for example, “There is a pit in [2,2] or [3,1]” or “If the wumpus is in [1,1] then he is not in [2,2].”
  - Programs can store a single value for each variable, and some systems allow the value to be “unknown,” but they lack the expressiveness required to directly handle partial information.

Propositional logic

- Propositional logic is a declarative language because its semantics is based on a truth relation between sentences and possible worlds.
  - It also has sufficient expressive power to deal with partial information, using disjunction and negation.
  - Propositional logic has the **compositionality** property that is desirable in representation languages
- In a compositional language, the meaning of a sentence is a function of the meaning of its parts.
  - For example, the meaning of "$S_{1,4} \wedge S_{1,2}$ " is related to the meanings of "$S_{1,4}$ " and "$S_{1,2}$."
  - It would be very strange if "S1,4 " meant that there is a stench in square [1,4] and "$S_{1,2}$ " meant that there is a stench in square [1,2], but "$S_{1,4} \wedge S_{1,2}$ " meant that France and Poland drew 1–1 in last week's ice hockey qualifying match
- However, propositional logic, as a factored representation, lacks the expressive power to concisely describe an environment with many objects.
  - We are forced to write a separate rule about breezes and pits for each square, e.g., $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
  - In English, it seems easy enough to say, once and for all, "Squares adjacent to pits are breezy." The syntax and semantics of English make it possible to describe the environment concisely: English, like first-order logic, is a structured representation.

The best of formal and natural languages

- We can adopt the foundation of propositional logic and build a more expressive logic on that foundation
  - Declarative, compositional semantics that is context-independent and unambiguous
  - Borrowing representational ideas from natural language while avoiding its drawbacks.
- When we look at the syntax of natural language, the most obvious elements **are nouns and noun phrases** that refer to objects (squares, pits, wumpuses) and **verbs and verb phrases** along with **adjectives and adverbs** that refer to relations among objects (is breezy, is adjacent to, shoots).
- Some of these relations are functions—relations in which there is only one "value" for a given "input." It is easy to start listing examples of objects, relations, and functions:
  - **Objects**: people, houses, numbers, theories, Ronald McDonald, colors, baseball games, wars, centuries, ...
  - **Relations**: these can be unary relations or properties such as red, round, bogus, prime, multistoried etc., or more general n-ary relations such as brother of, bigger than, inside, part of, has color, occurred after, owns, comes between, ...
  - **Functions**: father of, best friend, third inning of, one more than, beginning of, ...
- Indeed, almost any assertion can be thought of as referring to objects and properties or relations, e.g.
  - Evil King John ruled England in 1200."
  - Objects: John, England, 1200; Relation: ruled during; Properties: evil, king.

Logic languages

- The language of First-order logic is built around objects and relations
- First-order logic can also express facts about some or all of the objects in the universe. This enables one to represent general laws or rules, such as the statement “Squares neighboring the wumpus are smelly.”

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

- There are several languages of logic with different facets
  - The ontological commitment allows to start with true statements and infer other true statements
  - The Epistemological commitment states of knowledge that it allows with respect to each fact

Models for First-order logic

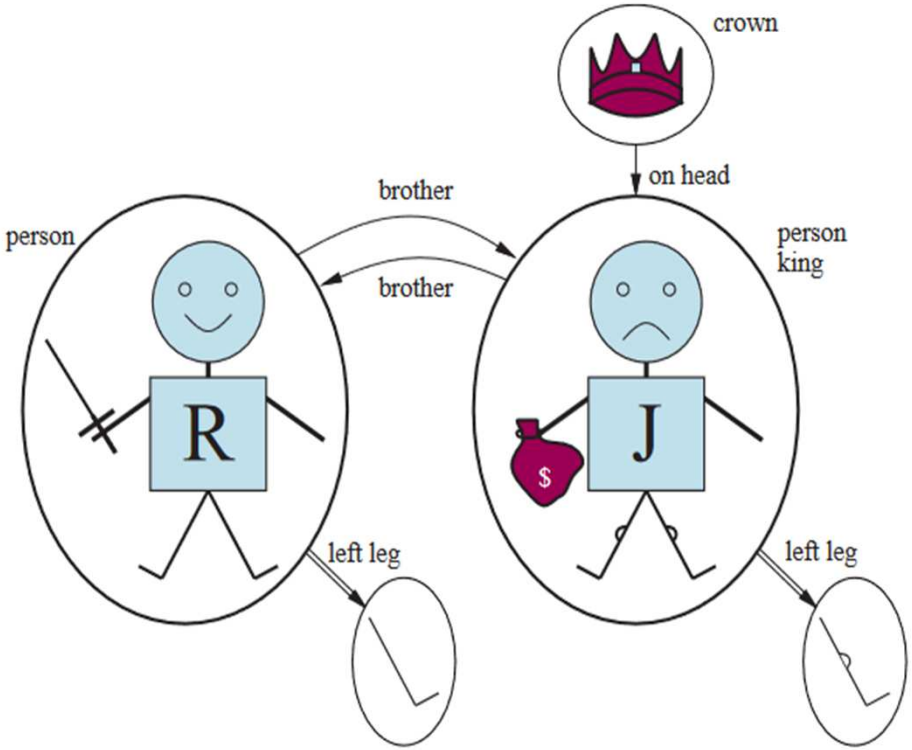
- The models of a logical language are the formal structures that constitute the possible worlds under consideration
  - Each model links the vocabulary of the logical sentences to elements of the possible world, so that the truth of any sentence can be determined
  - Models for propositional logic link proposition symbols to predefined truth values
- Models for first-order have objects in them
  - The domain of a model is the set of objects or domain elements it contains
  - The domain is required to be nonempty—every possible world must contain at least one object
  - Mathematically speaking, it doesn't matter what these objects are
- Next slide figure shows a model with five objects:
  - Richard the Lionheart, King of England from 1189 to 1199
  - His younger brother, the evil King John, who ruled from 1199 to 1215
  - The left legs of Richard and John
  - The crown

Model example

- The objects in the model may be related in various ways
  - Richard and John are brothers.
    {〈Richard the Lionheart, King John〉, 〈King John, Richard the Lionheart〉}
- Formally speaking, a **relation is just the set of tuples of objects that are related**
- A **tuple** is a collection of objects arranged in a fixed order and is written with brackets surrounding the objects
  - The crown is on King John's head, so the "on head" relation contains just one tuple, 〈the crown, King John〉
  - The "brother" and "on head" relations are binary relations, i.e., they relate pairs of objects
  - The model also contains unary relations, or properties: the "person" property is true of both Richard and John
  - the "king" property is true only of John (presumably because Richard is dead at this point);
  - the "crown" property is true only of the crown
- Certain kinds of relationships are best considered as **functions**, in that a given object must be related to exactly one object in this way. For example, each person has one left leg, so the model has a unary "left leg" function — a mapping from a one-element tuple to an object—that includes the following mappings:
  - 〈Richard the Lionheart〉 → Richard's left leg
  - 〈King John〉 → John's left leg

Total functions

- First-order logic models require **total functions**
  - there must be a value for every input tuple
- The crown must have a left leg and so must each of the left legs
  - There is a technical solution to this awkward problem involving an additional “invisible” object that is the left leg of everything that has no left leg, including itself
  - Fortunately, as long as one makes no assertions about the left legs of things that have no left legs, these technicalities are of no import.
- We have described the elements that populate models for first-order logic
- The other essential part of a model is the link between those elements and the vocabulary of the logical sentences.



Symbols and interpretations

- 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
  - **Function symbols**, 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*
  - the function symbol *LeftLeg*

- As with proposition symbols, the choice of names is entirely up to the user. Each predicate and function symbol comes with an arity that fixes the number of arguments.

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

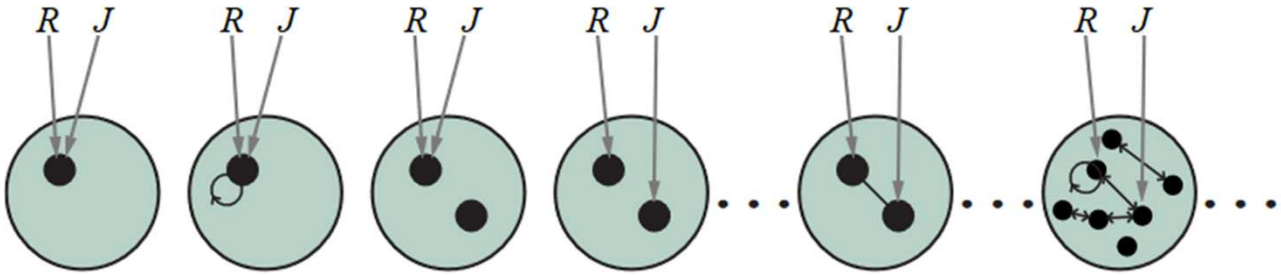
The syntax of first-order logic with equality, (BNF)

- Every model must provide the information required to determine if any given sentence is true or false.
- Thus, in addition to its objects, relations, and functions, each model includes an interpretation
- An **interpretation** specifies exactly which objects, relations and functions are referred to by the constant, predicate, and function symbols.

- The operator precedence is ¬, ∧,∨, ⇒, ⇔

Interpretation

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



Intended interpretation

- Notice that not all the objects need have a name
  - the intended interpretation does not name the crown or the legs.
- It is also possible for an object to have several names
  - there is an interpretation under which both *Richard* and *John* refer to the crown.
- In propositional logic, it is perfectly possible to have a model in which *Cloudy* and *Sunny* are both true;
  - it is the job of the knowledge base to rule out models that are inconsistent with our knowledge
- Just as with propositional logic, entailment, validity, and so on are defined in terms of all possible models
- Because the number of first-order models is unbounded, we cannot check entailment by enumerating them all, as we can do for propositional logic
  - Even if the number of objects is restricted, the number of combinations can be very large.
  - For the previousexample, there are 137,506,194,466 models with six or fewer objects.

Terms

- A term is a logical expression that refers to an object.
  - Constant symbols are terms
  - 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).
- A **complex term** is formed by a function symbol followed by a parenthesized list of terms as arguments
  - There is no LeftLeg subroutine that takes a person as input and returns a leg.
  - We can reason about left legs, e.g., stating the general rule that everyone has one and then deducing that John must have one, without ever providing a definition of LeftLeg.
- Consider a term $f(t_1, ..., t_n)$, where f refers to some function in the model (call it F)
  - the argument terms refer to objects in the domain $d_1, ..., d_n$
- The term to the object that is the value of the function F applied to $d_1, ..., d_n$.
  - For example, suppose the *LeftLeg* function symbol refers to the function *LeftLeg* and *John* refers to King John, then *LeftLeg(John)* refers to King John’s left leg.
  - In this way, the interpretation fixes the referent of every term.

Atomic sentences

- We can combine terms and predicate symbols 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)*
- This states, under the intended interpretation given earlier, that Richard the Lionheart is the brother of King John. 4 Atomic sentences can have complex terms as arguments.
- Thus, *Married(Father(Richard), Mother(John))* states that Richard the Lionheart's father is married to King John's mother, again, under a suitable interpretation
- An **atomic sentence is true in a given model** if the relation referred to by the predicate symbol holds among the objects referred to by the arguments
- --

Complex sentences

- We can use logical connectives to construct more complex sentences, with the same syntax and semantics as in propositional calculus.
- Here are four sentences that are true in the example model under our intended interpretation:

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

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

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

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

Quantifiers

- Quantifiers let us express properties of collections of objects, instead of enumerating the objects by name.
- First-order logic contains two standard quantifiers, called ***universal*** and ***existential***.

- Universal quantification (∀)
  - Rules such as “All kings are persons” are the *bread and butter* of first-order logic.
  - “All kings are persons,” is written in first-order logic as $\forall x \; King(x) \Rightarrow Person(x)$
- The universal quantifier ∀ is usually pronounced “For all ...”.
  - Thus, the sentence says, “For all x, if x is a king, then x is a person.”
- The symbol x is called a **variable**. By convention, variables are lowercase letters.
  - A variable is a term all by itself, and can serve as the argument of a function—for example, LeftLeg(x)
- A term with no variables is called a **ground term**.

Universal quantification

- Intuitively, the sentence $\forall x\ P$, where P is any logical sentence, says that P is true for every object x.
  - More precisely, $\forall x\ P$ is true in a given model if P is true in all possible extended interpretations constructed from the interpretation given in the model, where each extended interpretation specifies a domain element to which x refers.

- Consider the example model and the intended interpretation that goes with it. We can extend the interpretation in five ways:
  x → Richard the Lionheart,
  x → King John,
  x → Richard's left leg,
  x → John's left leg,
  x → the crown.

Universally quantified sentences

- The universally quantified sentence $\forall x\ King(x) \Rightarrow Person(x)$ is true in the original model if the sentence $King(x) \Rightarrow Person(x)$ is true under each of the five extended interpretations.

  Richard the Lionheart is a king ⇒ Richard the Lionheart is a person.

  King John is a king ⇒ King John is a person.

  Richard's left leg is a king ⇒ Richard's left leg is a person.

  John's left leg is a king ⇒ John's left leg is a person.

  The crown is a king ⇒ the crown is a person.

- Since, in our model, King John is the only king, the second sentence asserts that he is a person, as we would hope. So, the first assertion is true.
- But the other four assertions are also true in the model and none of these objects is a king.
  - Implication is true whenever its premise is false
  - Thus, by asserting the universally quantified sentence, we end up asserting the conclusion of the rule just for those objects for which the premise is true and saying nothing at all about those objects for which the premise is false.

Existential quantification (∃)

- Universal quantification makes statements about every object.
- Similarly, 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 $\exists x\ Crown(x) \land OnHead(x, John)$
  - ∃x is pronounced “There exists an x such that ...” or “For some x ...”.
- Intuitively, the sentence $\exists x$ P says that P is true for at least one object x. More precisely, $\exists x$ P is true in a given model if P is true in at least one extended interpretation that assigns x to a domain element.
- That is, at least one of the following is true:
  - Richard the Lionheart is a crown ∧ Richard the Lionheart is on John’s head;
  - King John is a crown ∧ King John is on John’s head;
  - Richard’s left leg is a crown ∧ Richard’s left leg is on John’s head;
  - John’s left leg is a crown ∧ John’s left leg is on John’s head;
  - The crown is a crown ∧ the crown is on John’s head.

Strong statements

- The last assertion is true in the model, so the original existentially quantified sentence is true in the model.
  - By our definition, the sentence would also be true in a model in which King John was wearing two crowns.
  - This is entirely consistent with the original sentence “King John has a crown on his head.”
- Just as $\Rightarrow$ appears to be the natural connective to use with $\forall$, $\wedge$ is the natural connective to use with $\exists$.
  - Using $\wedge$ as the main connective with $\forall$ led to an overly strong statement in the example;
  - using $\Rightarrow$ with $\exists$ usually leads to a very weak statement, indeed.

Use of implication

- Consider the following sentence: $\exists x\ Crown(x) \Rightarrow OnHead(x, John)$
  - On the surface, this might look like a reasonable rendition of our sentence.

- Applying the semantics, we see that the sentence says that at least one of the following assertions is true:
  Richard the Lionheart is a crown ⇒ Richard the Lionheart is on John's head;
  King John is a crown ⇒ King John is on John's head;
  Richard's left leg is a crown ⇒ Richard's left leg is on John's head;
  and so on.

- An implication is true if both premise and conclusion are true, or if its premise is false
  - so if Richard the Lionheart is not a crown, then the first assertion is true and the existential is satisfied.
  - So, an existentially quantified implication sentence is true whenever any object fails to satisfy the premise;
  - hence such sentences really do not say much at all.

Nested quantifiers

- We will often want to express more complex sentences using multiple quantifiers and variables
  - For example, "Brothers are siblings" can be written as $\forall x \ \forall y \ Brother(x, y) \Rightarrow Sibling(x, y)$
  - For example, to say that siblinghood is a symmetric relationship as $\forall x,y \ Sibling(x, y) \Leftrightarrow Sibling(y, x)$
- In other cases we will have mixtures
  - "Everybody loves somebody" - for every person, there is someone that person loves $\forall x \ \exists y \ Loves(x, y)$
  - On the other hand, to say " - there is someone who is loved by everyone," we write $\exists y \ \forall x \ Loves(x, y)$
  - The order of quantification is important. These are equivalent $\forall x \ (\exists y \ Loves(x, y)), \ \exists y \ (\forall x \ Loves(x, y))$

- Some confusion can arise when two quantifiers are used with the same variable name.
  - Consider the sentence $\forall x \ (Crown(x) \lor (\exists x \ Brother(Richard, x)))$
  - Here the x in $Brother(Richard, x)$ is existentially quantified.
  - The rule is that the variable belongs to the innermost quantifier that mentions it, then it will not be subject to any other quantification

Connections between ∀ and ∃

- The two quantifiers are actually intimately connected with each other, through negation
- Asserting that everyone dislikes parsnips is the same as asserting there does not exist someone who likes them, and vice versa:

  $\forall x \neg Likes(x, Parsnips)$ is equivalent to $\neg \exists x\ Likes(x, Parsnips)$
- We can go one step further: "Everyone likes ice cream" means that there is no one who does not like ice cream:

  $\forall x\ Likes(x, IceCream)$ is equivalent to $\neg \exists x \neg Likes(x, IceCream)$
- Because ∀ is really a conjunction over the universe of objects and ∃ is a disjunction, it should not be surprising that they obey De Morgan's rules.
- The De Morgan rules for quantified and unquantified sentences are as follows:

$$\neg\exists x\ P \equiv \forall x\ \neg P \qquad \neg(P \vee Q) \equiv \neg P \wedge \neg Q$$
$$\neg\forall x\ P \equiv \exists x\ \neg P \qquad \neg(P \wedge Q) \equiv \neg P \vee \neg Q$$
$$\forall x\ P \equiv \neg\exists x\ \neg P \qquad P \wedge Q \equiv \neg(\neg P \vee \neg Q)$$
$$\exists x\ P \equiv \neg\forall x\ \neg P \qquad P \vee Q \equiv \neg(\neg P \wedge \neg Q)$$

Equality

- First-order logic includes one more way to make atomic sentences, other than using a predicate and terms as described earlier. We can use the equality symbol to signify that two terms refer to the same object.
  - For example, $Father(John) = Henry$
- The equality symbol can be used to state facts about a given function, as we just did for the Father symbol
- It can also be used with negation to insist that two terms are not the same object. To say that Richard has at least two brothers, we would write

$$\exists x, y\ Brother(x, Richard) \wedge Brother(y, Richard) \wedge \neg(x = y)$$

- The sentence $\exists x, y\ Brother(x, Richard) \wedge Brother(y, Richard)$
  - does not have the intended meaning.
  - It is true in the model, where Richard has only one brother.
  - To see this, consider the extended interpretation in which both x and y are assigned to King John.
  - The addition of ¬(x = y) rules out such models.

Thank you!