Course: Introduction to AI

Prof. Ahmed Guessoum
The National Higher School of AI

Chapter 8

First-Order Logic

Outline

- Representation Revisited
  - Combining the best of formal and natural languages
- Syntax and Semantics of First-Order Logic
  - Models for first-order logic
  - Symbols and interpretations
  - Terms
  - Atomic sentences
  - Complex sentences
  - Quantifiers

Outline

- Equality
- An alternative semantics?

- Using First-Order Logic
  - Assertions and queries in first-order logic
  - The kinship domain
  - Numbers, sets, and lists
  - The Wumpus world
- Knowledge Engineering in First-Order Logic
  - The knowledge-engineering process
  - The electronic circuits domain (self study)

Introduction

- In the previous chapter, we saw :
  - How an KB agent can represent its world and deduce what actions to take.
  - That propositional logic can be used as representation language.
  - That PL sufficed to illustrate the basic concepts of logic and KB agents.
- PL does not allow a representation of the knowledge of complex environments in a concise way.
- In this chapter, we examine **First-Order Logic** (FOL), which is sufficiently expressive to represent a good deal of our commonsense knowledge.

Representation Revisited

- Programming languages (C++, Java, etc.) are the most used formal languages.
  - Programs represent only computational processes.
  - Data structures in programmes represent facts; e.g. a 4×4 array to represent the contents of the wumpus world and a statement like World[2,2]←*Pit* to assert that there is a pit in [2,2].
  - Databases allow to store and retrieve data.
  - Programming languages lack any general mechanism for deriving facts from others: an update to a data structure is done by a domain-specific procedure as designed by the programmer from his 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.

Representation Revisited

- Data structures in programs and databases do not allow such statements as:
  - "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]."
- Advantages of Propositional Logic:
  - It is a declarative language: one states what is *true* about the world.
  - Powerful enough to deal with partial information, using disjunction and negation.
  - Its property of **compositionality**: In a compositional language, the meaning of a sentence is a function of the meaning of its parts.
- Drawback of PL: it lacks the expressive power to *concisely* describe an environment with many objects.
- We had to write 1 rule about breezes and pits for each square:

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

Combining the best of formal and natural languages

- In natural language we refer to objects, relations and functions to describe the “world”:
  - Objects: people, houses, numbers, theories, Larbi BenMhidi, colours, chess games, wars, centuries . . .
  - Relations: can be unary relations or **properties** such as red, round, prime, multistoried . . ., or more general n-ary relations such as brother of, bigger than, inside, part of, has colour, occurred after, owns, comes between, . . .
  - Functions: father of, best friend, beginning of, . . .

Combining the best of formal and NL

- Assertions in NL can be thought of as referring to objects and properties or relations.
- "One plus two equals three."
  - Objects: one, two, three; Relation: equals; Function: plus.
- "Squares neighbouring the wumpus are smelly."
  - Objects: wumpus, squares; Property: smelly; Relation: neighbouring.
- "Good Khalifa Jawad ruled The Land in 200."
  - Objects: Jawad, The Land, 200; Relation: ruled; Properties: Good Khalifa.
- The language of **first-order logic**, is built around objects and relations.
- It can express facts about *some* or *all* of the objects in the universe

Syntax and Semantics of FOL

- Recall that models for propositional logic link proposition symbols to truth values.
- Models for FOL are much more interesting.
- They 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. Example:
- Objects: Harun Rachid, The Wise, Khalifa from 149 to 193; his son Khalifa Mohamed Jawad, ruled from 193 to 198; the left legs of Rachid and Jawad; and a crown
- Relations: Rachid and Jawad are brothers.
- Formally, a relation is a set of **tuples** of objects that are related. So: {< Harun Rachid,Khalifa Jawad Mohamed>, < Khalifa Jawad Mohamed, Harun Rachid>}

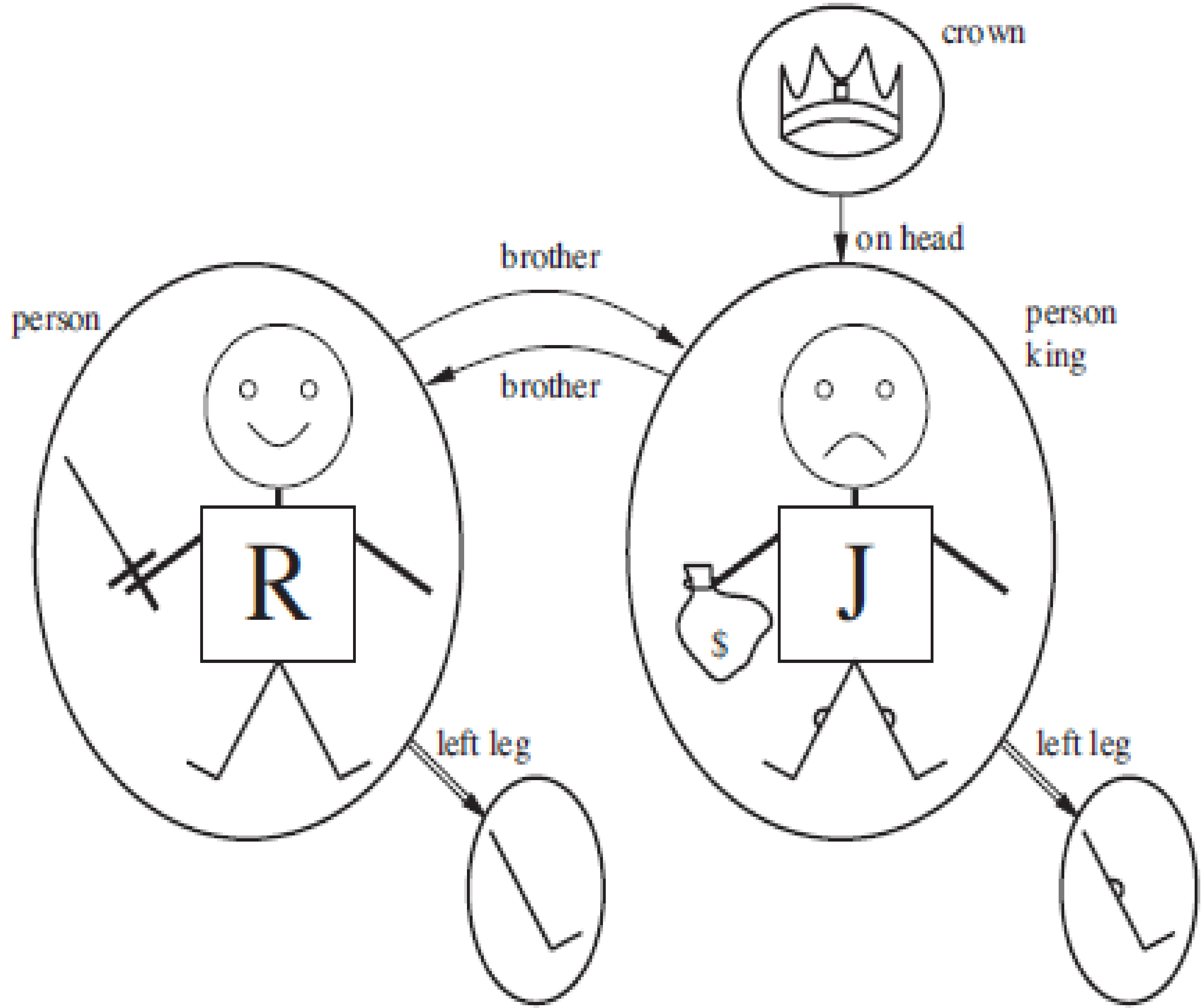


Figure 8.2 A model containing five objects, two binary relations, three unary relations (indicated by labels on the objects), and one unary function, left-leg.

Syntax and Semantics of FOL

- A crown on head of *Jawad*,
  <*the crown, King Jawad*>
- "*brother*" and "*on head*" are binary relations.
- Unary relations:
  - the "person" property is true for *Rachid* and *Jawad;*
  - *"khalifa"* property true for *Jawad* only (say after death of *Rachid*);
  - the "*crown*" property is true only of the *crown*.
- Certain kinds of relationships are best considered as functions (mappings):
  - <Rachid the Wise> → Rachid's left leg
  - <Khalifa Jawad> → Jawad's left leg .

Symbols and interpretations

$$
\begin{array}{rcl}
Sentence & \rightarrow & AtomicSentence \mid ComplexSentence \\
AtomicSentence & \rightarrow & Predicate \mid Predicate(Term, \ldots) \mid Term = Term \\
ComplexSentence & \rightarrow & (\ Sentence\ ) \mid [\ Sentence\ ] \\
& \mid & \neg\ Sentence \\
& \mid & Sentence \land Sentence \\
& \mid & Sentence \lor Sentence \\
& \mid & Sentence \Rightarrow Sentence \\
& \mid & Sentence \Leftrightarrow Sentence \\
& \mid & Quantifier\ Variable, \ldots\ Sentence \\
\\
Term & \rightarrow & Function(Term, \ldots) \\
& \mid & Constant \\
& \mid & Variable \\
\\
Quantifier & \rightarrow & \forall \mid \exists \\
Constant & \rightarrow & A \mid X_1 \mid John \mid \cdots \\
Variable & \rightarrow & a \mid x \mid s \mid \cdots \\
Predicate & \rightarrow & True \mid False \mid After \mid Loves \mid Raining \mid \cdots \\
Function & \rightarrow & Mother \mid LeftLeg \mid \cdots \\
\textsc{Operator Precedence} & : & \neg, =, \land, \lor, \Rightarrow, \Leftrightarrow
\end{array}
$$

Figure 8.3 The syntax of first-order logic with equality, specified in Backus–Naur form

Symbols and interpretations

For example,

- constant symbols: *Rachid* and *Jawad*;
- Predicate symbols: *Brother* , *OnHead*, *Person*, *Khalifa*, and *Crown*;
- Function symbol: *LeftLeg*.
- Each predicate and function symbol comes with an **arity** that fixes the number of arguments.
- As in propositional logic, every model must provide the information required to determine if any given sentence is true or false.
- ➔ in addition to its objects, relations, and functions, each model includes an **interpretation** that specifies exactly which objects, relations and functions are referred to by the constant, predicate, and function symbols.

Symbols and interpretations

- **Intended interpretation**: what each symbol is really referring to.
  - *Rachid*: Harun Rachid the wise Khalifa
  - *Jawad*: Khalifa Mohamed Jawad
  - *Brother:* refers to the brotherhood relation
  - *OnHead*: refers to the "on head" relation that holds between the crown and Khalifa Jawad.
  - *Likewise for* Person, Khalifa, and Crown
  - *LeftLeg* function….
- Obviously, other interpretations of the symbols, relations, and functions are possible. We keep the "intuitive" one.
- In summary, a model in FOL consists of a set of objects and an interpretation that maps constant symbols to objects, predicate symbols to relations on those objects, and function symbols to functions on those objects.

Terms & Atomic Sentences

- **<u>Terms</u>**: The formal semantics of terms is straightforward. Consider a term $f(t_1, \ldots, t_n)$.
  - The function symbol *f* refers to some function in the model (call it *F*);
  - the argument terms refer to objects in the domain (call them $d_1, \ldots, d_n$); and
  - the term as a whole refers to the object that is the value of the function *F* applied to $d_1, \ldots, d_n$.
  - Example: *LeftLeg(Jawad)* refers to *Khalifa Jawad's* left leg.
- **<u>Atomic sentences</u>** (or **atom** for short): it is formed from a predicate symbol optionally followed by a parenthesized list of terms, e.g. *Brother (Rachid , Jawad).*
- Atomic sentences can have complex terms as arguments:

  Married(Father (Rachid), Mother (Jawad))

Atomic & Complex sentences

- *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.*
- More complex sentences can be constructed using **logical connectives** with the same syntax and semantics as in propositional calculus.
  - $\neg$Brother (LeftLeg(Rachid), Jawad)
  - Brother (Rachid, Jawad) $\wedge$ Brother (Jawad, Rachid)
  - Khalifa(Rachid) $\vee$ Khalifa(Jawad)
  - $\neg$Khalifa(Rachid) $\Rightarrow$ Khalifa(Jawad)
- **<u>Quantifiers</u>:** Universal and existential
- **Universal quantification ($\forall$)**: in FOL, they allow to easily express statements like “Squares neighbouring the wumpus are smelly” and “All khalifas are persons”:
  - $\forall$ x khalifa(x) $\Rightarrow$ Person(x) .

Atomic & Complex sentences

- By convention, variables are lowercase letters.
- A variable is a term ➔ a variable can be the argument of a function. E.g., *LeftLeg(x)*.
- A term with no variables is called a **ground term.**
- **$\forall$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.
- E.g. consider the model shown in Figure 8.2 and the intended interpretation that goes with it. The interpretation can be extended in five ways:

x → Rachid the Wise, x → Khalifa Jawad,

x → Rachid's left leg, x → Jawad's left leg,

x → the crown.

Atomic & Complex sentences

- The universally quantified sentence
$\forall x$ Khalifa(x) $\Rightarrow$ Person(x) is true in the original model if the sentence Khalifa (x) $\Rightarrow$ Person(x) is true under each of the five extended interpretations.
- That is, the universally quantified sentence is equivalent to asserting the following five sentences:

1. Rachid the Wise is a Khalifa $\Rightarrow$ Rachid the Wise is a person.
2. Jawad is a Khalifa $\Rightarrow$ Jawad is a person.
3. Rachid's left leg is a Khalifa $\Rightarrow$ Rachid's left leg is a person.
4. Jawad's left leg is a Khalifa $\Rightarrow$ Jawad's left leg is a person.
5. The crown is a Khalifa $\Rightarrow$ the crown is a person.

- Note that the 2^{nd} statement is indeed intended but not the others. But since they are all implications, even if the premise is false, the conclusion is true.

Atomic & Complex sentences

- **Existential quantification (∃):** e.g. Khalifa Jawad has a crown on his head:

  $\exists\, x\ Crown(x) \wedge OnHead(x, Jawad)$.

- <u>∃x P is true</u> in a given model <u>if P is true in *at least one* extended interpretation</u> that assigns x to a domain element.
- That is, at least one of the following is true:

1. *Rachid the Wise is a crown ∧ Rachid the Wise is on Jawad's head;*
2. *Khalifa Jawad is a crown ∧ Khalifa Jawad is on Jawad's head;*
3. *Rachid's left leg is a crown ∧ Rachid's left leg is on Jawad's head;*
4. *Jawad's left leg is a crown ∧ Jawad's left leg is on Jawad's head;*
5. <u>*The crown is a crown ∧ the crown is on Jawad's head.*</u>

- The fifth assertion is true, so the existentially quantified sentence is true.
- Just as ⇒ appears to be the natural connective to use with ∀, ∧ is the natural connective to use with ∃.

Atomic & Complex sentences

Nested quantifiers:

- "Brothers are siblings" $\forall x \forall y$ Brother $(x, y) \Rightarrow$ Sibling(x, y)
- Siblinghood is a symmetric relationship:
  $\forall x, y$ Sibling$(x, y) \Leftrightarrow$ Sibling(y, x)
- "Everybody loves somebody" : $\forall x \exists y$ Loves(x, y)
- "There is someone who is loved by everyone":
  $\exists y \forall x$ 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 (Rachid, x)))
- The rule is that a variable belongs to the innermost quantifier that mentions it; then it will not be subject to any other quantification.
- So the innermost x is not impacted by $\forall x$ ➔ the sentence is equivalent to $\forall x$ (Crown$(x) \lor (\exists z$ Brother (Rachid, z)))

Atomic & Complex sentences

- **Connections between ∀ and ∃:** through negation
- $\forall x \neg Likes(x, Peas)$ is equivalent to $\neg \exists x\ Likes(x, Peas)$
- "Everyone likes ice cream": $\forall x\ Likes(x, IceCream)$ is equivalent to $\neg \exists x \neg Likes(x, IceCream)$
- De Morgan's rules:

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

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

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

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

- **<u>Equality:</u>** this symbol is used to say <u>two terms refer to the same object</u>. E.g., Father(Jawad)= Hassan
- To say that Rachid has at least two brothers, we would write

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

An alternative semantics?

- Suppose that we believe that Rachid has two brothers, Jawad and Ghassan.
- Writing Brother (Jawad, Rachid) ∧ Brother (Ghassan, Rachid) is not good enough!
- It does not exclude other brothers.
- The correct translation of "Rachid's brothers are Jawad and Ghassan" is as follows:

  Brother (Jawad, Rachid) ∧ Brother (Ghassan, Rachid) ∧
  Jawad ≠ Ghassan ∧ ∀x Brother (x, Rachid) ⇒
  (x= Jawad ∨ x= Ghassan)

- Can we devise a semantics that allows a more straightforward logical expression?

An alternative semantics?

1. **Unique-names assumption:** Every constant symbol is taken to refer to a distinct object.
2. **Closed-world assumption:** Atomic sentences not known to be true are assumed to be false.
3. **Domain closure**: Each model contains no more domain elements than those named by the constant symbols.

- Under the resulting semantics, called **database semantics** to distinguish it from the standard semantics of FOL, the sentence *Brother (Jawad, Rachid)* ∧ *Brother (Ghassan, Rachid)* does indeed state that Rachid's two brothers are Jawad and Ghassan.
- Note: there is no one "correct" semantics for logic.
- The usefulness of any proposed semantics depends on:
  - Its conciseness and intuitiveness, and
  - Ease and naturalness of the corresponding rules of inference

Using First-Order Logic

- **Assertions and queries in first-order logic:** We will use TELL and ASK as for Propositional Logic:
- *Jawad is a Khalifa*, *Rachid is a person*, and *all Khalifas are persons*:
  - *TELL(KB, Khalifa(Jawad))* .
  - *TELL(KB, Person(Rachid))* .
  - *TELL(KB,* $\forall x$ *Khalifa(x)* $\Rightarrow$ *Person(x))* .
- Query the KB using ASK. E.g.,
  - *ASK(KB, Khalifa(Jawad))* returns *true*
- Questions asked with ASK are called **queries** or **goals**.
- A query that is logically entailed by the KB should return *true*.
  - *ASK(KB, Person(Jawad))* returns *true*

Assertions and queries in FOL

- Asking quantified queries:
  - *ASK(KB, ∃x Person(x))* returns *true*
- This is not useful because we would like to know this *x* who is a *Person.*
- Another function, *ASKVARS*, does this:
  - *ASKVARS(KB, Person(x))* can return all such *x*
  - Here we get two answers: *{x/Jawad}* and *{x/Rachid}*
  - Such an answer is called a **substitution** or **binding list**.
- *ASKVARS* will be used with KBs consisting of Horn Clauses only.

The kinship (family relations) domain

- There are binary predicates: *Parent, Sibling, Brother , Sister , Child , Daughter ,* etc.
- Functions: *mother, father*
- Different sentences about the kinship domain, e.g.

$\forall m,c \quad Mother(c)=m \Leftrightarrow Female(m) \wedge Parent(m,c)$

$\forall w,h \quad Husband(h,w) \Leftrightarrow Male(h) \wedge Spouse(h,w)$

$\forall x \quad Male(x) \Leftrightarrow \neg Female(x)$

$\forall p,c \quad Parent(p,c) \Leftrightarrow Child(c,p)$

$\forall g,c \quad Grandparent(g,c) \Leftrightarrow \exists p\ Parent(g,p) \wedge Parent(p,c)$

$\forall x,y \quad Sibling(x,y) \Leftrightarrow x\neq y \wedge \exists p\ Parent(p,x) \wedge Parent(p,y)$

- These sentences can be viewed as **axioms** of the kinship domain.
- These axioms provide the basic factual information from which useful conclusions can be derived.

The kinship (family relations) domain

- The definitions "bottom out" at a basic set of predicates (*Child* , *Spouse*, and *Female*) in terms of which the others are ultimately defined.
- This basic set of (primitive) predicates is not unique.
- **Theorems** are sentences that are entailed by the axioms. E.g. $\forall x,y \;\; Sibling(x,y) \Leftrightarrow Sibling(y,x)$
- It can be entailed from the definition of *sibling*.
- Some axioms are definitions (*Mother, Husband, Male, Parent, Grandparent, Sibling*)
- Other axioms are not definitions; e.g. about the predicate *Person* because no complete information to define it.
- Axioms can also be plain **facts**, such as *Male(Jim)* and *Spouse(Jim, Laura)*.
- Note that $\neg Spouse(George, Laura)$ is not inferred even if we add $Jim \neq George$. An axiom is missing for this.

Numbers, sets, and lists

- Let us describe the theory of **natural numbers**.
- We need
  - a predicate *NatNum* , true for natural numbers
  - a constant symbol 0 and
  - one function symbol, *S* (*successor*).
- **Peano axioms** recursively define natural numbers & addition.
  - $NatNum(0)$.
  - $\forall n\ NatNum(n) \Rightarrow NatNum(S(n))$
- So the natural numbers are $0, S(0), S(S(0))$, etc.
- We also need axioms to constrain the successor function:
  - $\forall n\ \ 0 \neq S(n)$ $\qquad \forall m,n\ \ m=n \Rightarrow S(m)=S(n)$
- We define addition in terms of the successor function:

  $\forall m\ NatNum(m) \Rightarrow +(0,m) = m$.

  $\forall m,n\ NatNum(m) \wedge NatNum(n) \Rightarrow +(S(m), n) = S(+(m, n))$

Numbers, sets, and lists

- Using infix notation for + and *S(n)* as *n+1*, the last axiom becomes:

$$\forall m,n \;\; NatNum(m) \land NatNum(n) \Rightarrow$$
$$(m+1) + n = (m+n) + 1$$

- This axiom reduces addition to a repeated application of the successor function.
- Once we have addition, it is straightforward to define multiplication as repeated addition, exponentiation as repeated multiplication, integer division and remainders, prime numbers, and so on.
- ➔ the whole of number theory can be built up from one constant, one function, one predicate and four axioms.

Numbers, sets, and lists

- **A theory of sets:**
- The empty set is a constant written as $\{\}$.
- There is one unary predicate, ***Set***, which is true of sets.
- The binary predicates are
  - $x \in s$ (x is a member of set s)
  - $s1 \subseteq s2$ (set s1 is a subset, not necessarily proper, of s2).
- The binary functions are
  - $s1 \cap s2$ (intersection of two sets),
  - $s1 \cup s2$ (union of two sets), and
  - $\{x|s\}$ (the set resulting from adjoining element x to set s).
- One possible set of axioms is as follows:

1. The only sets are the empty set and those made by adjoining something to a set:

$$\forall s\ Set(s) \Leftrightarrow (s=\{\}) \vee (\exists x,s2\ \ Set(s2) \wedge s=\{x|s2\})\ .$$

Numbers, sets, and lists

2. The empty set has no elements adjoined into it. In other words, there is no way to decompose {} into a smaller set and an element:

$\neg\exists x,s \quad \{x|s\} = \{\}$

3. Adjoining an element already in the set has no effect:

$\forall x,s \;\; x \in s \Leftrightarrow s=\{x|s\}$

4. The only members of a set are the elements that were adjoined into it. This is expressed recursively, :

$\forall x,s \;\; x \in s \;\Leftrightarrow \exists y,s2 \;\; (s=\{y|s2\} \wedge (x=y \vee x \in s2))$

5. A set is a <u>subset</u> of another set if and only if all of the first set members are members of the second set:

$\forall s1,s2 \;\; s1 \subseteq s2 \;\Leftrightarrow\; (\forall x \;\; x \in s1 \;\Rightarrow x \in s2)$

6. <u>Two sets are equal</u> if and only if each is a subset of the other:

$\forall s1,s2 \;\; (s1=s2) \Leftrightarrow (s1 \subseteq s2 \;\wedge\; s2 \subseteq s1)$

Numbers, sets, and lists

7. An object is in the intersection of two sets if and only if it is a member of both sets:

   $\forall x,s1,s2 \quad x\in (s1 \cap s2) \Leftrightarrow (x\in s1 \wedge x\in s2)$

8. An object is in the union of two sets if and only if it is a member of either set:

   $\forall x,s1,s2 \quad x\in (s1 \cup s2) \Leftrightarrow (x\in s1 \vee x\in s2)$

The Wumpus world

- The percept sentence stored in the KB must include both the percept and the time at which it occurred:

  Percept ([Stench, Breeze, Glitter , None, None], 5)

- The actions in the wumpus world can be represented by logical terms:

  Turn(Right), Turn(Left), Forward , Shoot , Grab, Climb .

- To determine the best action, the agent executes the query

  ASKVARS(∃a BestAction(a, 5))

- This returns a binding list such as {a/Grab}
- The raw percept data implies certain facts about the current state. E.g.:

  $\forall t,s,g,m,c \quad Percept\,([s, Breeze, g,m, c], t) \Rightarrow Breeze(t)$

  $\forall t,s,b,m,c \quad Percept\,([s, b, Glitter,m, c], t) \Rightarrow Glitter\,(t)$

- Etc.

The Wumpus world

- Simple "reflex" behaviour can also be implemented by quantified implication sentences. E.g.

  $\forall t \quad Glitter(t) \Rightarrow BestAction(Grab, t)$
- Given the percept and preceding rules, the following conclusion is reached:

  $BestAction(Grab, 5)$
- <u>Let us represent the environment:</u>
- Adjacency of any two squares can be defined as

$\forall x,y,a,b\ Adjacent([x,y], [a,b]) \Leftrightarrow (x=a \wedge (y=b-1 \vee y=b+1)) \vee (y=b \wedge (x = a-1 \vee x=a+1))$

- *No need to distinguish between pits* ➔ It is simpler to use a unary predicate *Pit* that is true of squares containing pits.
- *A constant Wumpus* will achieve the purpose.
- *At(Agent, s, t)* means that the agent is at square *s* at time *t*.

The Wumpus world

- The wumpus's location is fixed ➔ $\forall t\ At(Wumpus, [2, 2], t)$
- To say that objects can be at only one location at a time:

  $\forall x,s1,s2,t \quad At(x, s1, t) \wedge At(x, s2, t) \Rightarrow s1=s2$
- Given its current location, the agent can infer properties of the square from properties of its current percept.
- if the agent is at a square and perceives a breeze, then that square is breezy:

  $\forall s,t \quad At(Agent, s, t) \wedge Breeze(t) \Rightarrow Breezy(s)$
- Note that since *Pits* do not move, *Breezy* has no time argument.
- FOL just needs one axiom:

  $\forall s \quad Breezy(s) \Leftrightarrow \exists r\ Adjacent\ (r, s) \wedge Pit(r)$
- We need just one successor-state axiom for each predicate, rather than a different copy for each time step. E.g. for Arrow:

  $\forall t \quad HaveArrow(t+1) \Leftrightarrow (HaveArrow(t) \wedge \neg Action(Shoot, t))$

The knowledge engineering process

All Knowledge engineering projects include the following steps:

1. *Identify the task:* The knowledge engineer (KE) must:
   - Delineate the range of questions that the KB will support
   - Delineate the kinds of facts that will be available for each problem instance.
   - So here define what knowledge must be represented in order to connect problem instances to answers.
   - This step is analogous to the PEAS process for designing agents.
2. *Assemble the relevant knowledge:*
   - This process is called **knowledge acquisition.**
   - Either the KE is an expert and he/she will write the needed K. or he/she is not, hence the need to sit with an expert to explicit the K.
   - At this stage, the knowledge is not represented formally.

The knowledge engineering process

3. *Decide on a vocabulary of predicates, functions, and constants:*
   - That is, translate the important domain-level concepts into logic-level names.
   - Decide on what should be an object, a predicate, or a function.
   - The result is a vocabulary, known as the **ontology** of the domain.
4. *Encode general knowledge about the domain:* The KE writes down the axioms for all the vocabulary terms.
   - This defines as thoroughly as possible the meaning of the terms.
   - The KE checks these axioms with the expert correcting what needs be, returning to step 3 and iterating through the process.

The knowledge engineering process

5. *Encode a description of the specific problem instance:*
   - This step involves writing simple atomic sentences about instances of concepts that are already part of the ontology.
   - For a logical agent, problem instances are supplied by the sensors.
   - Any other KB is supplied with additional sentences that represent these problem instances (just like input data).
6. *Pose queries to the inference procedure and get answers: i.e.*
   - Let the inference procedure operate on the axioms and problem-specific facts to derive the new K. the KE seeks.
7. *Debug the knowledge base :*
   - I.e. notice places where the chain of reasoning stops unexpectedly.
   - Identify any missing axioms or axioms that are too weak.
   - Add these to the KB and iterate.

The electronic circuits domain

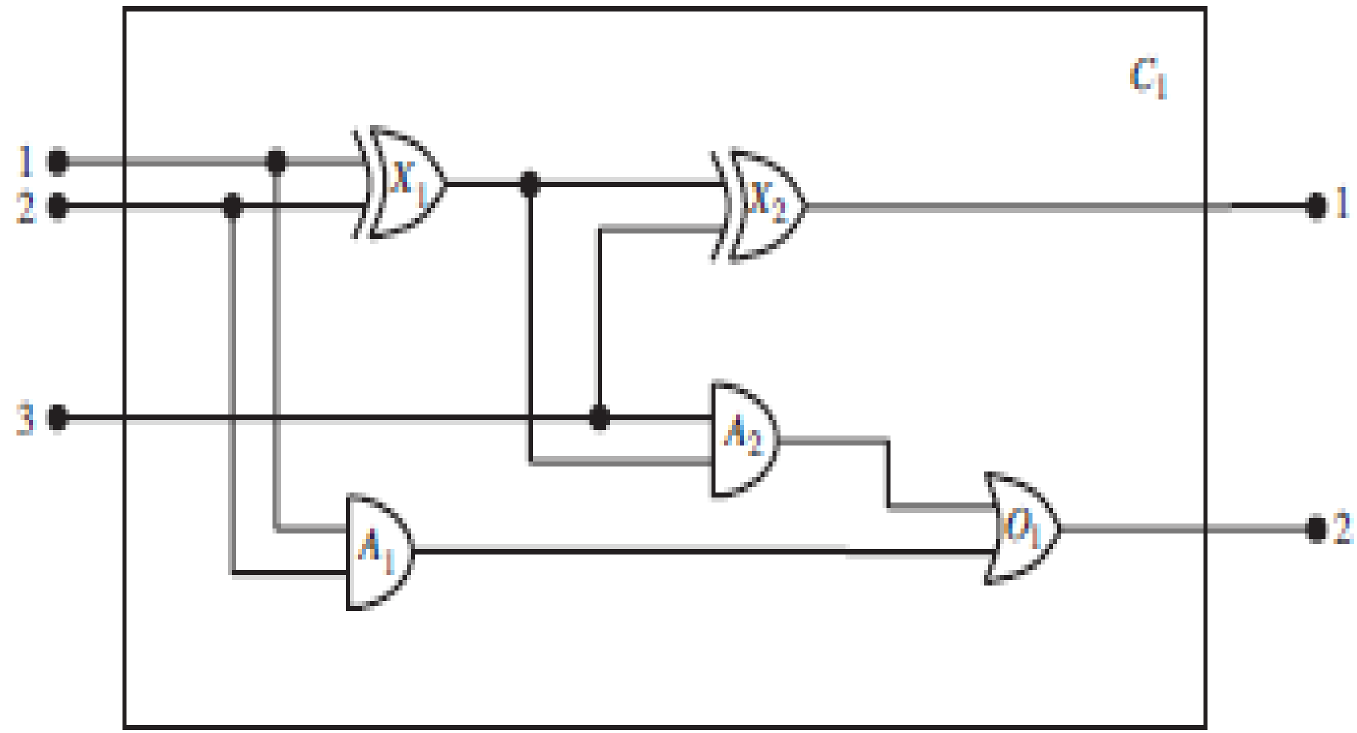
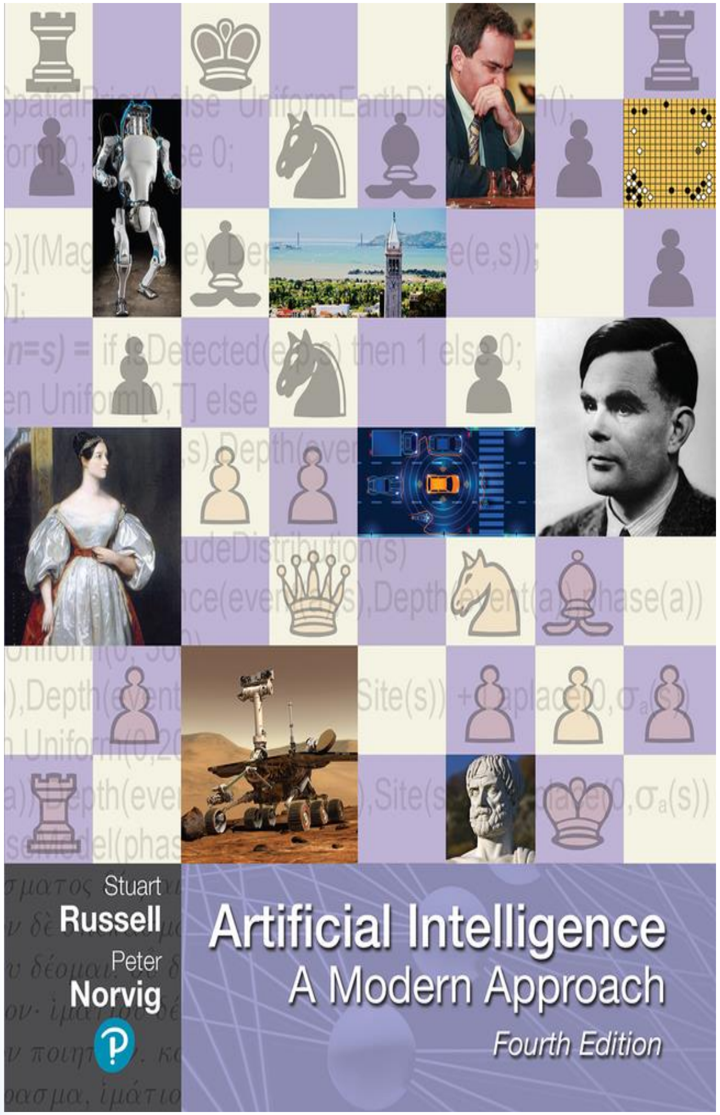


Figure 8.6 A digital circuit C1, purporting to be a one-bit full adder. The first two inputs are the two bits to be added, and the third input is a carry bit. The first output is the sum, and the second output is a carry bit for the next adder. The circuit contains two XOR gates, two AND gates, and one OR gate.

Homework: Read in your textbook Section **8.4.2 "The electronic circuits domain"**.

Slides based on the textbook



- Russel, S. and Norvig, P. (2020) Artificial Intelligence,

A Modern Approach (4th Edition), Pearson Education Limited.