Artificial Intelligence 1

Roman Barták
Department of Theoretical Computer Science and Mathematical Logic

Knowledge Representation: First-Order Logic

We are designing **knowledge-based agents** – they combine and recombine information about the world with current observations to uncover hidden aspects of the world and use them for action selection.

How to represent **knowledge?**

- so far **propositional logic**
- today **first-order predicate logic**

We are looking for a **formal language** that can

- **represent knowledge**
- **reason with knowledge**

What about **programming languages** (C++, Java, …)?

- this is the most widely used class of formal languages
- facts are described via **data structures**
  - array world[4,4]
- **programs** describe how to do computations (changing data structures)
  - world[2,2] $\leftarrow$ pit
- How to infer new information from existing facts?
  - ad-hoc procedures changing data structures $\rightarrow$ a **procedural approach**
  - a **declarative approach** separates knowledge and inference mechanism (moreover, inference is general and problem independent)
- How to represent knowledge such as "pit at [2,2] or [3,1]"?
  - variables in computer programs have unique values

Natural languages

Can we use **natural languages** (English, Czech, ...) to represent knowledge?

- That would be great but there is no precise formal semantics for these languages!
- Currently, natural languages are seen as a **medium for communication** rather than for pure representation.
    - the sentence itself does not code information, it also depends on **context**
        - "Look!"
    - another problem is **ambiguity** of natural languages
        - spring, …

> **"... if thought corrupts language, language can also corrupt thought."**
>
> George Orwell, *Politics and the English Language,* 1946

Propositional logic is declarative with compositional semantic that is context-independent and unambiguous.

However, some properties are cumbersome (not easy to model).

- Wumpus world: there is breeze next to a pit
  - $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
  - $B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$
  - …

Let us take inspiration from natural languages:

- we have nouns representing **objects** (pit, square, …)
- verbs express **relations** between the objects (is next to, …)
- some relations are in fact **functions** (is a father of)

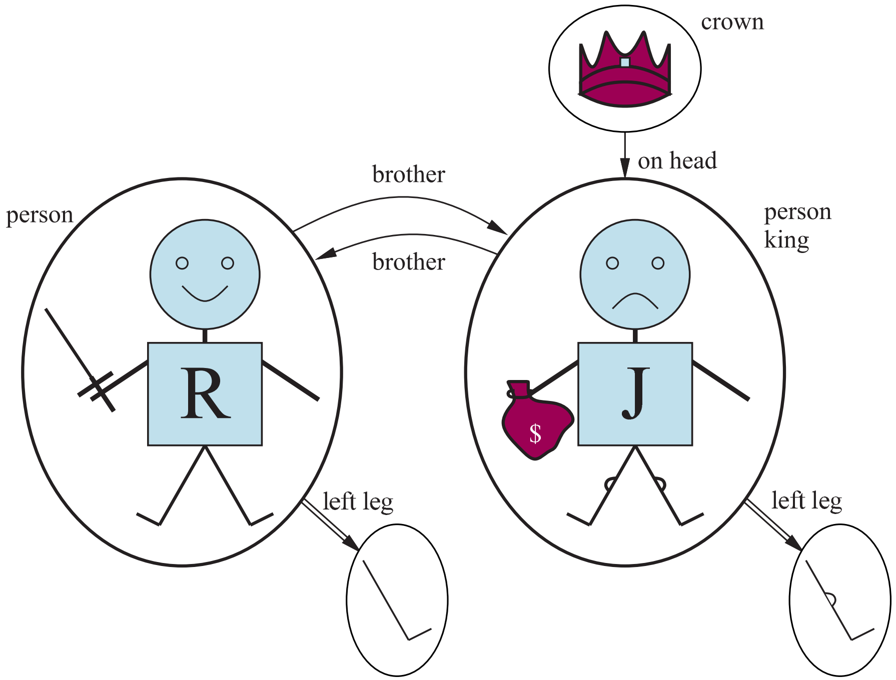
Instead of pure facts (propositional logic) we will work with objects, relations, and functions. We will also express facts about some or all objects (**first-order predicate logic – FOL**).

Propositional logic	facts that hold or not
First-order predicate logic	facts, objects and relations that hold between them
Temporal logic	facts, objects, relations, and times when they hold
Fuzzy logic	facts with degree of truth

First-order logic: syntax

• constants	John, 2, Crown, ...
• predicates	Brother, >, ...
• functions	Sqrt, LeftLeg, ...
• variables	x, y, a, b, ...
• connectives	$\neg, \Rightarrow, \wedge, \vee, \Leftrightarrow$
• equality	$=$
• quantifiers	$\forall, \exists$

First-order logic: an example



- **constants** (names of objects):
  - Richard, John, TheCrown
- **function symbols**:
  - LeftLeg
- **terms** (another form to name objects)
  - LeftLeg(John)
- **predicate symbols**:
  - Brother, OnHead, Person, King, Crown
- **atomic sentences** (describe relations between objects):
  - Brother(Richard,John)
- **complex sentences**:
  - King(Richard) $\vee$ King(John)
  - $\neg$King(Richard) $\Rightarrow$ King(John)
- **quantifiers** (help to define sentences over more objects):
  - $\forall$x (King(x) $\Rightarrow$ Person(x))
    Beware: $\forall$x (King(x) $\wedge$ Person(x)) !!!
  - $\exists$x (Crown(x) $\wedge$ OnHead(x,John))
    Beware: $\exists$x (Crown(x) $\Rightarrow$ OnHead(x,John)) !!!

- $\forall$x,y (Brother(x,y) $\Rightarrow$ Brother(y,x))
- $\exists$x,y (Brother(x,Richard) $\wedge$ Brother(y,Richard))
- $\exists$x,y (Brother(x,Richard) $\wedge$ Brother(y,Richard) $\wedge$ $\neg$(x=y))
  Equality says that two terms refer to the same object (Father(John) = Henry).

Universal quantifier $\forall$x P

- **P** is true for any object **x**
- corresponds to a conjunction of all formulas P
  - P(John) $\wedge$ P(Richard) $\wedge$ P(TheCrown) $\wedge$ P(LeftLeg(John)) $\wedge$ …
- Typically connected with implication (to select the objects for which the sentence holds)
  - $\forall$x King(x) $\Rightarrow$ Person(x)

Existential quantifier $\exists$x P

- there is an object **x** such that **P** holds for it
- corresponds to a disjunction of all formulas P
  - P(John) $\vee$ P(Richard) $\vee$ P(TheCrown) $\vee$ P(LeftLeg(John)) $\vee$ …

Relations between quantifiers

- $\forall$x $\forall$y is identical to $\forall$y $\forall$x
  $\exists$x $\exists$y is identical to $\exists$y $\exists$x
- $\exists$x $\forall$y is not identical to $\forall$y $\exists$x ($\exists$x $\forall$y Loves(x,y) vs.$\forall$y $\exists$x Loves(x,y))
- $\forall$x P is identical to $\neg\exists$x $\neg$P
  $\exists$x P is identical to $\neg\forall$x $\neg$P

Similarly to propositional logic we will use **operations TELL** to add a sentence to knowledge base:

- TELL(KB, King(John))
- TELL(KB,∀x (King(x) ⇒ Person(x)))
- We are typically adding **axioms** (**facts** as atomic sentences, **definitions** using ⇔ and other complex sentences) and sometime even **theorems** (can be deduced from axioms, but they "speed up" further inference).

and **operations ASK** for querying the sentences entailed by KB:

- ASK(KB, King(John))
- ASK(KB, Person(John))
- ASK(KB, ∃x Person(x))

a database query

we need some inference here

in addition to YES/NO answers we also ask for the value of x for which the sentence holds – substitution {x/John}

The domain of family relationships (kinship).

Objects = people

Unary predicates: *Male*, *Female*
Binary predicates (kinship relations): *Parent*, *Sibling*, *Child*, *Grandparent*, …
Functions: *Mother*, *Father*

Axioms:

Plain facts:

$Male(Jim)$

Definitions:

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

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

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

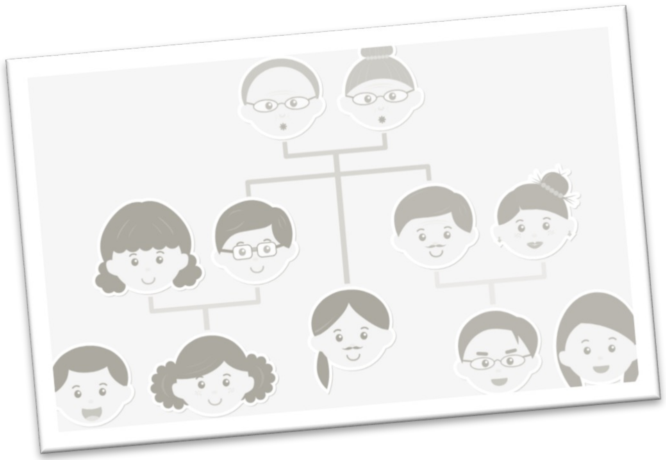
General information (but not definition)

$\forall x\ (Person(x) \Rightarrow \ldots)$

$\forall x\ (\ldots \Rightarrow Person(x))$

Theorems:

$\forall x,y\ Sibling(x,y) \Leftrightarrow Sibling(y,x)$



The numbers domain

The domain for numbers can also be constructed from a tiny kernel of **(Peano) axioms**.

Predicate: *NatNum*

Constant symbol: *0*

Function symbol: *S* (successor)

Natural numbers are defined recursively:

$NatNum(0)$

$\forall n \; NatNum(n) \Rightarrow NatNum(S(n))$

Axioms constraining the successor function:

$\forall n \; 0 \neq S(n)$

$\forall m,n \; m \neq n \Rightarrow S(m) \neq S(n)$

Definition of **addition**:

(m+1)+n = (m+n)+1

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

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

Knowledge engineering

Knowledge engineering deals with the process of knowledge-base construction.

A **knowledge engineer** is someone who:

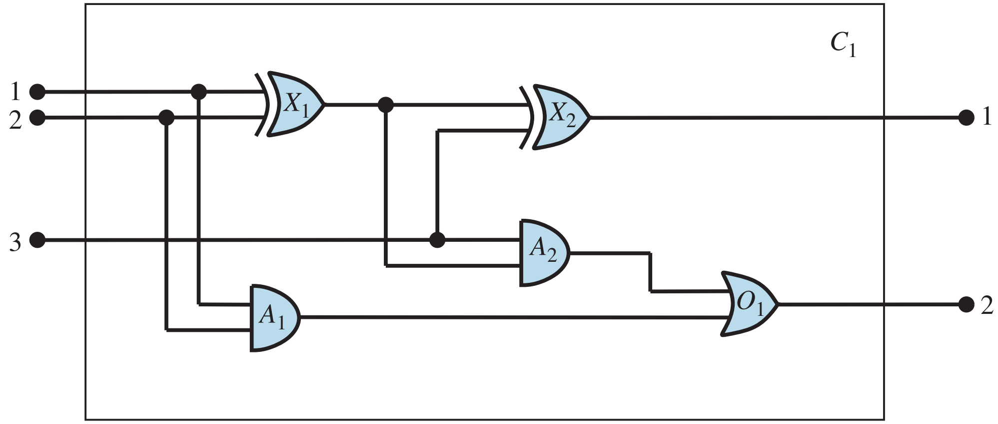
- **investigates** a particular domain
  - How do the things work?
  - This is usually done in co-operation with a problem expert.
- **learns** what **concepts** are important in that domain
  - Which will be the queries asked and what do we need to find answers?
- **creates** a formal **representation** of the objects and relations in the domain
  - How to encode facts and axioms so the computer can do inference?

Knowledge-engineering process

1. **identify the task**
   - What is the range of questions?
   - Wumpus: action selection or asking about the contents of the environment?
2. **assemble the relevant knowledge (knowledge acquisition)**
   - How does the domain actually work?
   - Wumpus: what does it mean to feel stench and breeze?
3. **decide on a vocabulary of predicates, functions, and constants**
   - How to translate domain-level concepts to logic-level names?
   - Wumpus: is a pit an object or a function of the square?
   - The result is an **ontology** of the domain (vocabulary of notions).
4. **encode general knowledge about the domain**
   - Which axioms hold in the domain?
   - Wumpus: breeze means a pit in the neighbourhood square
5. **encode a description of the specific problem instance**
   - What is the current state of the world?
   - Wumpus: the agent is at square (1,1) looking to the right
6. **pose queries to the inference procedure and get answers**
   - How does the inference procedure operate on our KB?
   - Wumpus: is cell (2,2) really safe?
7. **debug the knowledge base**
   - What is missing in the knowledge base?
   - Wumpus: there is a single wumpus in the cave

Digital circuits

- 1 and 2 are input bits, 3 is a carry bit
- 1 is output bit for sum, 2 is output bit for carry

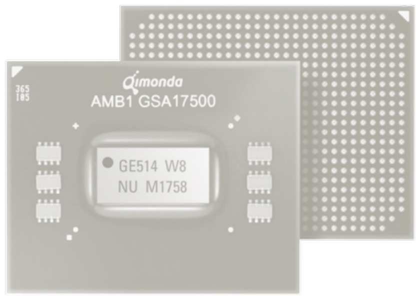


What is important in the domain?

- Does the circuit add properly?
- If the inputs are known, what is the output?
- If desired output is given, what should be the input?

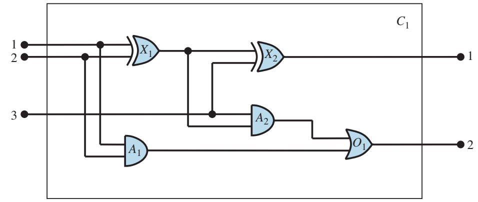
Different queries may require different knowledge!

- What is the cost of the circuit?
- What is the size of the circuit?
- How much energy does the circuit consume?



What do we know about digital circuits?

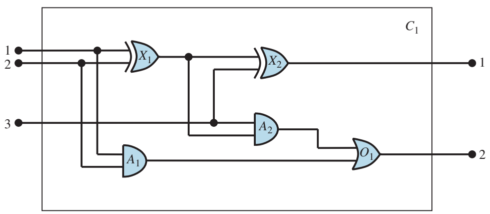
- circuits are composed from wires and gates
- signals 0 and 1 flow along wires
- signals flow to the input terminals of gates
- each gate produces signal on the output terminal
- there are four types of gates: AND, OR, XOR, NOT
- circuits have input and output terminals
- wires are used just as connections between terminals
- signal delay, energy consumption, shape of gates are not assumed



KE process: vocabulary

What constants, predicates, and functions?

- we describe circuits, gates, terminals, signals, and connections
  - **gates** are denoted by **constants X_1, X_2, A_1, ...**
  - the behaviour of each **gate** is determined by its **type**
    - we will use constants AND, OR, XOR, NOT
    - types of gates are described by **functions Type(X_1) = XOR**
    - We can also use predicates Type(X_1,XOR) or XOR(X_1)
      - Beware! We will also need axioms to describe uniqueness of the gate type.
  - **terminals** of gates can also be named by constants (X_1In_1, ...), but then we need to connect them to gates
    - it is better to use **functions In(1, X_1), ...**
  - **wires** can be described by **predicates**
    - **Connected(Out(1, X_1),In(1, X_2)), ...**
    - Beware! We connect the terminals not the gates.
  - **signals** at terminals are determined by a **function**
    - **Signal(g) = 1**



KE example: general knowledge

If two terminals are connected, then they have the same signal.

- $\forall t_1, t_2$ Connected$(t_1, t_2) \Rightarrow$ Signal(t_1) = Signal(t_2)

The signal at every terminal is either 1 or 0.

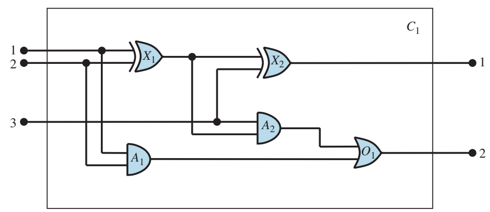
- $\forall t$ Signal$(t) = 1 \vee$ Signal$(t) = 0$
- $1 \neq 0$

The predicate "Connected" is commutative.

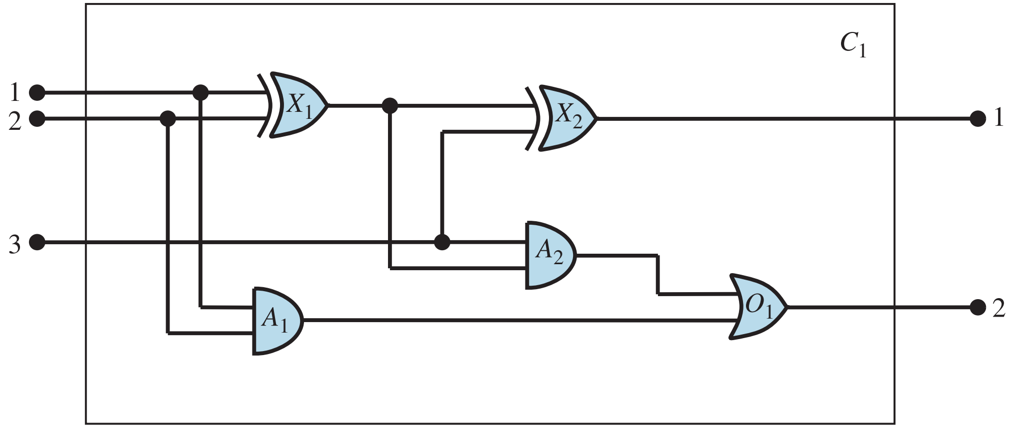
- $\forall t_1, t_2$ Connected$(t_1, t_2) \Rightarrow$ Connected(t_2, t_1)

The gate behaviour is determined by its type.

- $\forall g$ Type(g) = OR $\Rightarrow$
  Signal(Out(1,g)) = 1 $\Leftrightarrow \exists n$ Signal(In(n,g)) = 1
- $\forall g$ Type(g) = AND $\Rightarrow$
  Signal(Out(1,g)) = 0 $\Leftrightarrow \exists n$ Signal(In(n,g)) = 0
- $\forall g$ Type(g) = XOR $\Rightarrow$
  Signal(Out(1,g)) = 1 $\Leftrightarrow$ Signal(In(1,g)) $\neq$ Signal(In(2,g))
- $\forall g$ Type(g) = NOT $\Rightarrow$
  Signal(Out(1,g)) $\neq$ Signal(In(1,g))



KE process: specific problem instance



$Type(X_1) = XOR$
$Type(X_2) = XOR$
$Type(A_1) = AND$
$Type(A_2) = AND$
$Type(O_1) = OR$

$Connected(Out(1,X_1),In(1,X_2))$
$Connected(Out(1,X_1),In(2,A_2))$
$Connected(Out(1,A_2),In(1,O_1))$
$Connected(Out(1,A_1),In(2,O_1))$
$Connected(Out(1,X_2),Out(1,C_1))$
$Connected(Out(1,O_1),Out(2,C_1))$

$Connected(In(1,C_1),In(1,X_1))$
$Connected(In(1,C_1),In(1,A_1))$
$Connected(In(2,C_1),In(2,X_1))$
$Connected(In(2,C_1),In(2,A_1))$
$Connected(In(3,C_1),In(2,X_2))$
$Connected(In(3,C_1),In(1,A_2))$

KE process: querying and debugging

Query is a **logical formula**.

- What combination of inputs would cause the sum output to be 0 and carry-bit output to be 1?
  - $\exists i_1,i_2,i_3$ $\mathrm{Signal}(\mathrm{In}(1,C_1)) = i_1 \wedge \mathrm{Signal}(\mathrm{In}(2,C_1)) = i_2 \wedge \mathrm{Signal}(\mathrm{In}(3,C_1)) = i_3 \wedge \mathrm{Signal}(\mathrm{Out}(1,C_1)) = 0 \wedge \mathrm{Signal}(\mathrm{Out}(2,C_1)) = 1$

Answer is obtained as **substitutions of variables** i_1,i_2,i_3.

- $\{i_1/1,\ i_2/1,\ i_3/0\}$, $\{i_1/1,\ i_2/0,\ i_3/1\}$, $\{i_1/0,\ i_2/1,\ i_3/1\}$

Debug the knowledge base

- Some queries may give an unexpected (wrong) answer that indicates a problem in the knowledge base (wrong/missing axiom, …).
  - A typical problem is a missing axiom claiming that constants identify different objects.
    - $1 \neq 0$

Example:

- Assume the following claim:
  - *„In summer we will teach courses CS101, CS102, CS106, and EE101"*
  - so in FOL we have the facts
    - Course(CS,101), Course(CS, 102), Course(CS,106), Course(EE,101)
- How many courses will we teach in summer?
  - Something between one and infinity!!

Why?

- We usually assume having a complete information about the world, i.e., what is not explicitly said does not hold – this is called a **closed world assumption** (CWA).
- There is no such assumption in FOL, so we need to complete the knowledge base:

  $$\text{Course}(d,n) \Leftrightarrow [d,n] = [CS,101] \vee [d,n] = [CS,102] \vee [d,n] = [CS,206] \vee [d,n] = [EE,101]$$
- We also assumed that different names (constants) denote different objects – this is called a **unique name assumption** (UNA)
- Again, we need to explicitly describe that objects are different:
  - $[CS,101] \neq [CS,102]$, …

Department of Theoretical Computer Science and Mathematical Logic
bartak@ktiml.mff.cuni.cz