Fundamentals of Artificial Intelligence – First-Order Logic

Matthias Althoff

TU München

Winter semester 2023/24

Organization

1. Representation Revisited
2. Syntax and Semantics of First-Order Logic
3. Using First-Order Logic
4. Knowledge Engineering in First-Order Logic

The content is covered in the AI book by the section “First-Order Logic”.

Learning Outcomes

- You understand the advantages and disadvantages of *first-order logic* compared to *propositional logic*.
- You understand the difference between *objects*, *relations*, and *functions* in first-order logic.
- You can create and evaluate sentences in first-order logic. Specifically, you can create *terms*, *atomic sentences*, *complex sentences*, and use *quantification*.
- You understand and can apply *nested quantification*.
- You understand the difference between *assertions* and *queries* in first-order logic.
- You understand and are able to engineer knowledge in first-order logic.

Advantages and Disadvantages of Propositional Logic

- ☺ Propositional logic is **declarative** (in contrast to procedural as in many programming languages, e.g., C): pieces of syntax correspond to facts.
- ☺ Propositional logic allows partial/disjunctive/negated information; e.g.. "There is a pit in [2,2] or [3,1]"
  (unlike most data structures and databases).
- ☺ Propositional logic is **compositional**:
  meaning of $B_{1,1} \wedge P_{1,2}$ is derived from meaning of $B_{1,1}$ and of $P_{1,2}$.
- ☺ Meaning in propositional logic is **context-independent**
  (unlike natural language, where meaning depends on context).
- ☹ Propositional logic has very limited expressive power
  (unlike natural language);
  e.g., cannot say "pits cause breezes in adjacent squares" except by writing one sentence for each square.

First-Order Logic (FOL)

Whereas propositional logic assumes the world contains *facts*,
first-order logic (like natural language) assumes the world contains

- **Objects**: corresponds to nouns in natural language.
  Examples: people, houses, numbers, theories, Angela Merkel, colors, football games, ...
- **Relations**: corresponds to verbs and adjectives. Relations can be unary or n-ary.
  Unary examples: red, round, bogus, prime, multi-storied, ...
  N-ary examples: brother of, bigger than, inside, part of, occurred after, owns, comes between, ...
- **Functions**: relations where each input is related to exactly one output.
  Examples: father of, best friend, third inning of, one more than, end of, ...

Objects, Relations, and Functions: Examples

- **"One plus two equals three."**
  Objects: one, two, three, one plus two
  Relations: equals
  Functions: plus
- **"Squares neighboring the Wumpus are smelly."**
  Objects: squares, Wumpus
  Relations: smelly (unary), neighboring (n-ary)
  Functions: -
- **"Evil King John ruled England in 1200."**
  Objects: King John, England, 1200
  Relations: evil (unary), ruled (n-ary)
  Functions: -

Logics in General

Not relevant for the exam

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
Fuzzy logic	facts + degree of truth	known interval value

Syntax of FOL: Basic Elements

Syntactic element	Representation of	Examples
Constants	Objects	*KingJohn*, 2, *TUM*, ...
Predicates	Relations	*Brother*, $>$, ...
Functions	Functions	*Sqrt*, *LeftLegOf*, ...

Syntactic element	Examples
Variables	$x, y, a, b, \ldots$
Connectives	$\wedge, \vee, \neg, \Rightarrow, \Leftrightarrow$
Equality	$=$
Quantifiers	$\forall, \exists$

Syntax of FOL: Backus-Naur Form

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

Operator precedence (descending order): $\neg, =, \wedge, \vee, \Rightarrow, \Leftrightarrow$.
Convention: Constants are uppercase and variables are lowercase.

Syntax of FOL: Examples

- **You can fool all of the people some of the time**:

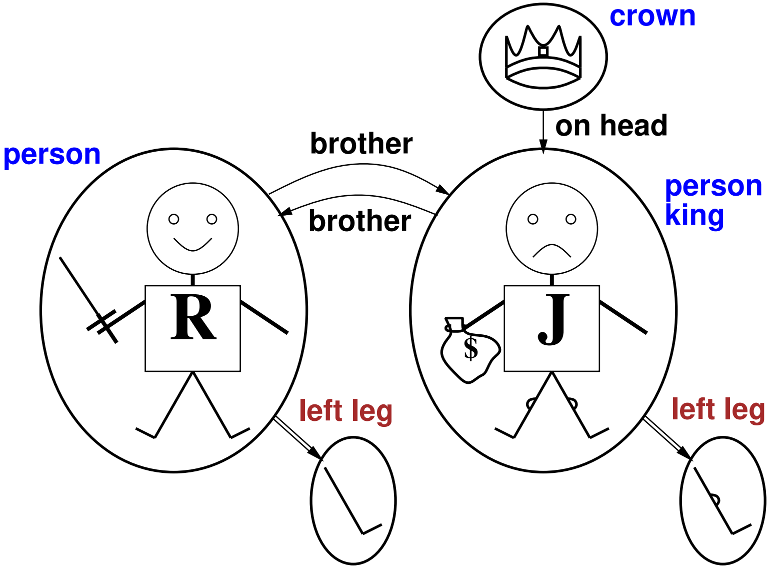
$$\underbrace{\underset{\text{all}}{\forall x},\ \underset{\text{some}}{\exists t}\ \underbrace{\underbrace{person}_{Predicate}(\underbrace{x}_{Var.})}_{AtomicSentence} \Rightarrow \underbrace{\underbrace{time}_{Predicate}(\underbrace{t}_{Var.})}_{AtomicSentence} \wedge \underbrace{\underbrace{can-fool}_{Predicate}(\underbrace{x,t}_{Var.})}_{AtomicSentence}}_{ComplexSentence}$$

- **No car over** 10 **years old will be repaired if it is severely damaged**:

$$\underbrace{\forall x\ \underbrace{\underbrace{car}_{Pred.}(\underbrace{x}_{Var.})}_{AtomicSent.} \wedge (\underbrace{\underbrace{age}_{Funct.}(\underbrace{x}_{Var.}) > 10)}_{Predicate\&AtomicSent.} \Rightarrow (\underbrace{\underbrace{sevDam}_{Predicate}(\underbrace{x}_{Var.})}_{AtomicSentence} \Rightarrow \neg \underbrace{\underbrace{repair}_{Predicate}(\underbrace{x}_{Var.})}_{AtomicSentence})}_{ComplexSentence}$$

$$\equiv \forall x\ car(x) \wedge (age(x) > 10) \wedge sevDam(x) \Rightarrow \neg repair(x)$$

Running Example



Terms

Backus-Naur Form of terms

$$Term ::= Function(Term, \ldots) \mid Constant \mid Variable$$

- A term is a logical expression that refers to an object.
- Constant symbols are therefore terms, but it is not always convenient to have a distinct symbol, e.g., *LeftLeg*(*John*) instead of *LeftLegOfJohn*.
- Think of a term just as a complicated kind of name, rather than a "subroutine call" that returns a value.

Atomic Sentences (1)

Backus-Naur Form of atomic sentences

$$AtomicSentence ::= Predicate \mid Predicate(Term, \ldots) \mid Term = Term$$

- An atomic sentence is formed from a predicate symbol optionally followed by a parenthesized list of terms. A predicate can be seen as a function that only returns *true* or *false*.

  Example: $Brother(Richard, John)$ meaning that Richard is the brother of John.

- Atomic sentences can have complex terms as arguments.

  Example: $Married(Father(Richard), Mother(John))$ meaning that the father of Richard is married to the mother of John.

Atomic Sentences (2)

- A further option to form an atomic sentence is by using equality of terms.
- Equality can signify that two terms refer to the same object.

  Example: $Father(John) = Henry$.
- Equality can also be used to insist that two terms are not the same object.

  Example:
  $\exists\, x,\, y \quad Brother(x, Richard) \land Brother(y, Richard) \land \neg(x = y)$.

Complex Sentences

Backus-Naur Form of complex sentences

$$
\begin{aligned}
ComplexSentence ::= &(Sentence) \mid [Sentence] \mid \neg Sentence \mid \\
&Sentence \wedge Sentence \mid Sentence \vee Sentence \mid \\
&Sentence \Rightarrow Sentence \mid Sentence \Leftrightarrow Sentence \mid \\
&Quantifier\ Variable, \ldots\ Sentence
\end{aligned}
$$

- We can use **logical connectives** to construct more complex sentences using the syntax from propositional logics.
- **Examples**:
  $\neg Brother(LeftLeg(Richard), John)$ √
  $Brother(Richard, John) \wedge Brother(John, Richard)$
  $King(Richard) \vee King(John)$
  $\neg King(Richard) \Rightarrow King(John)$

Universal Quantification ($\forall$)

- Expressing general rules in propositional logic is difficult.
  Example: "Squares neighboring the Wumpus are smelly", requiring to list all cases in propositional logic.
- In FOL this is easy, e.g., "All kings are persons" can be written using the variable x as
$$\forall x \quad King(x) \Rightarrow Person(x).$$
- A universally quantified expression is true if it is true for every object.

The running example has 5 objects:

$x \rightarrow$ Richard the Lionheart

$x \rightarrow$ King John

$x \rightarrow$ Richard's left leg

$x \rightarrow$ John's left leg

$x \rightarrow$ the crown

The sentence $King(x) \Rightarrow Person(x)$ is true for all objects and thus
$\forall x \quad King(x) \Rightarrow Person(x)$ is true.

Existential Quantification ($\exists$)

- In FOL, the existential quantifier is used to express a statement about some object.
  Example: "King John has a crown on his head" can be written as
  $$\exists x \quad Crown(x) \land OnHead(x, John).$$
- An existentially quantified expression is true if it is true for at least one object.

The running example has 5 objects:

$x \rightarrow$ Richard the Lionheart

$x \rightarrow$ King John

$x \rightarrow$ Richard's left leg

$x \rightarrow$ John's left leg

$x \rightarrow$ the crown

The sentence $Crown(x) \land OnHead(x, John)$ is true for the fifth object and thus $\exists x \quad Crown(x) \land OnHead(x, John)$ is true.

Common Mistakes to Avoid

- **Typically, $\Rightarrow$ is the main connective with $\forall$**
  Common mistake: using $\land$ as the main connective with $\forall$:

$$\forall x \quad At(x, TUM) \land Smart(x)$$

  means "Everyone is at TUM and everyone is smart" instead of "Everyone at TUM is smart".

- **Typically, $\land$ is the main connective with $\exists$**
  Common mistake: using $\Rightarrow$ as the main connective with $\exists$:

$$\exists x \quad At(x, TUM) \Rightarrow Smart(x)$$

  is true if there is anyone who is not at TUM instead of "Someone at TUM is smart".

Scope of Quantifiers

- The scope of a quantifier is the range in the formula where the quantifier "engages in".
- Parentheses make the scope explicit:

$$\forall x (\exists y \, Brother(x, y) \land \forall y \, Sibling(x, y))$$

- The scope of quantifiers is often implicit:

$$\forall x \exists y \exists z \, P(x, y, z)$$

is the same as

$$\forall x (\exists y (\exists z \, P(x, y, z)))$$

and

$$\forall x \exists y \exists z \, P_1(x, y, z) \land P_2(x, y, z)$$

is the same as

$$\forall x \exists y \exists z \, (P_1(x, y, z) \land P_2(x, y, z)).$$

Different Quantifiers Associated with the Same Variable

- Confusion arises when two quantifiers are used with the same variable name, e.g.,

$$\forall x \Big(Crown(x) \vee (\exists x \quad Brother(Richard, x)) \Big),$$

  where the x is universally and existentially quantified.
- The rule is that a variable is bound by the innermost quantifier.
- In the above example, $\forall x$ has no effect for $Brother(Richard, x)$.
- Often better: Use different variable names with nested quantifiers:

$$\forall x \Big(Crown(x) \vee (\exists z \quad Brother(Richard, z)) \Big).$$

Scope of Quantifiers: Examples

- What does this mean?

$$\exists x\, Smart(x) \land Nice(x).$$

"Someone is smart and nice."

- What does this mean?

$$\exists x\, Smart(x) \land \exists x\, Nice(x).$$

"Someone is smart and someone is nice." This is identical to

$$\exists x\, Smart(x) \land \exists y\, Nice(y).$$

- How to write the following sentence without parentheses?

$$\exists x\, (Smart(x) \land \exists x\, (Nice(x))).$$

This is identical to

$$\exists x\, Smart(x) \land \exists x\, Nice(x).$$

Nested Quantifiers

- We often want to use multiple quantifiers.
- **Quantifiers of the same type:**

  $\forall x \forall y$ is the same as $\forall y \forall x$.
  $\exists x \exists y$ is the same as $\exists y \exists x$.

  Why? Consecutive quantifiers of the same type can be written as one quantifier with several variables:
  $\forall (x, y) \quad Brother(x, y) \Rightarrow Sibling(x, y)$ is the same as
  $\forall x \forall y \quad Brother(x, y) \Rightarrow Sibling(x, y)$
- **Quantifiers of different type:**

  $\exists x \forall y$ is **not** the same as $\forall y \exists x$.

  Example:
  $\forall y (\exists x \quad Loves(x, y))$ "Everyone is loved by at least one person".
  $\exists x (\forall y \quad Loves(x, y))$ "There is a person who loves everyone".

Connections between $\forall$ and $\exists$

Quantifier duality: $\forall$ and $\exists$ can be expressed by each other using negation:

$\forall x \quad Likes(x, IceCream)$ is equivalent to $\neg\exists x \quad \neg Likes(x, IceCream)$

$\exists x \quad Likes(x, Broccoli)$ is equivalent to $\neg\forall x \quad \neg Likes(x, Broccoli)$

De Morgan rules for quantified sentences

$$\begin{aligned}
\forall x \quad \neg P &\equiv \quad \neg\exists x \quad P \\
\neg\forall x \quad P &\equiv \quad \exists x \quad \neg P \\
\forall x \quad P &\equiv \quad \neg\exists x \quad \neg P \\
\exists x \quad P &\equiv \quad \neg\forall x \quad \neg P
\end{aligned}$$

We do not need both $\forall$ and $\exists$, just as we do not need both $\wedge$ and $\vee$, but we keep it for readability.

Tweedback Questions

- **Brothers are siblings:**
  - A $\forall x, y \quad Brother(x, y) \wedge Sibling(x, y).$
  - B $\forall x, y \quad Brother(x, y) \Rightarrow Sibling(x, y).$
- **One's mother is one's female parent:**
  - A $\forall x, y \quad Mother(x, y) \Leftrightarrow (Female(x) \wedge Parent(x, y)).$
  - B $\forall x, y \quad Mother(x, y) \Leftrightarrow (Female(x) \Rightarrow Parent(x, y)).$
- **A first cousin is a child of a parent's sibling:**
  - A
    $$\forall x, y \quad FirstCousin(x, y) \Leftrightarrow$$
    $$\forall p, ps \quad Parent(p, x) \wedge Sibling(ps, p) \wedge Parent(ps, y).$$
  - B
    $$\forall x, y \quad FirstCousin(x, y) \Leftrightarrow$$
    $$\exists p, ps \quad Parent(p, x) \wedge Sibling(ps, p) \wedge Parent(ps, y).$$

Assertions and Queries in First-Order Logic

- We add sentences to a knowledge base using `Tell`, called **assertions**:

$$\text{John is a king: } \mathtt{Tell}(KB, King(John))$$

$$\text{Richard is a person: } \mathtt{Tell}(KB, Person(Richard))$$

$$\text{All kings are persons: } \mathtt{Tell}(KB, \forall x \quad King(x) \Rightarrow Person(x))$$

- We ask questions of the knowledge base using `Ask`, called **queries**:

$$\mathtt{Ask}(KB, King(John)) \text{ returns } \mathit{true}.$$

- For some queries a yes/no answer is undesirable:

$$\mathtt{Ask}(KB, \exists x \quad Person(x)).$$

  With `AskVars` we obtain a stream of answers:

$$\mathtt{AskVars}(KB, \exists x \quad Person(x)),$$

  which returns $\{x/John\}$ and $\{x/Richard\}$.

Example 1: Formalizing Overtaking in First-Order Logic

Not relevant for the exam

German Traffic Code *Straßenverkehrsordnung* §5(4)

When changing the lane to the left lane during overtaking, no following road users shall be endangered. [...] During overtaking, the driver has to change from the fast lane to the right lane as soon as possible. The road user being overtaken shall not be obstructed.

Formalizing Overtaking (Objects and Predicates)

Not relevant for the exam

Objects required:

1. **real numbers** — for denoting times
2. **traffic participants** — for identifying other traffic participants
3. **ego** — a constant for ego vehicle

Predicates required:

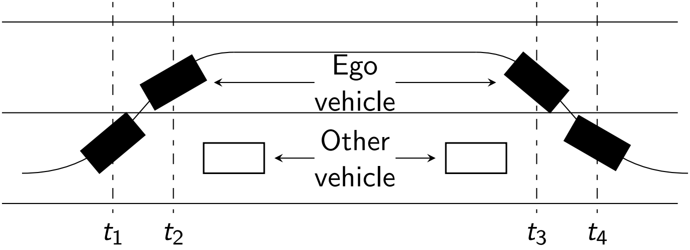
1. $\textit{is-time}(t)$ — true iff t is a real number
2. $\textit{is-traffic-part}(id)$ — true iff id is a traffic participant or **ego**
3. $\textit{sd-rear}(id_1, id_2)$ — id_1 is safely located behind id_2
4. $\textit{follows}(id_1, id_2)$ — id_1 follows id_2

Function required:

- *veh-being-overtaken* — returns the vehicle being overtaken

Formalizing Overtaking (Semantics)

Not relevant for the exam



predicates		**meaning (semantics)**
overtaking(t)	iff	$t \in [t_1, t_4)$
begin-overtake(t)	iff	$t \in [t_1; t_2)$
merging(t)	iff	$t = t_3$
finish-overtake(t)	iff	$t \in [t_3, t_4)$

Formalizing Overtaking (Translation to First-Order Logic)

Not relevant for the exam

1. **When changing the lane to the left lane during overtaking, no following road users shall be endangered.**

$$\forall t, id \quad \textit{is-time}(t) \wedge \textit{begin-overtake}(t) \wedge \textit{follows}(id, ego) \Rightarrow \textit{sd-rear}(id, ego)$$

2. **During overtaking, the driver has to change from the fast lane to the right lane as soon as possible.**

$$\forall t, id \quad \textit{is-time}(t) \wedge \textit{veh-being-overtaken}(t) = id \Rightarrow (\textit{merging}(t) \Leftrightarrow \textit{sd-rear}(id, ego))$$

3. **The road user being overtaken shall not be obstructed.**

$$\forall t, id \quad \textit{is-time}(t) \wedge \textit{veh-being-overtaken}(t) = id \wedge \textit{finish-overtake}(t) \Rightarrow \textit{sd-rear}(id, ego)$$

Example 2: Wumpus World (Sensing and Acting)

Not relevant for the exam

- **Comparison with propositional logic:** The first-order axioms are much more concise.
- **Percept vector:** The agent perceives three elements and a time step, e.g., $Percept([Stench, Breeze, Glitter], 5)$, or $Percept([Stench, None, None], 6)$, where *Percept* is a binary predicate.
- **Percept data implies facts** about the current state, e.g.,
  $\forall\, t, s, g \quad Percept([s, Breeze, g], t) \Rightarrow Breeze(t)$,
  $\forall\, t, s, b \quad Percept([s, b, Glitter], t) \Rightarrow Glitter(t)$.

  Note the quantification of time: In propositional logic, we need copies of sentences for each time step.
- **Actions** are represented by logical terms:
  $Turn(Right)$, $Turn(Left)$, $Forward$, $Shoot$, $Grab$, $Climb$.
- **The best action** is determined by the query
  $\texttt{AskVars}(\exists\, a \quad BestAction(a, 5))$.
- **Simple reflex behavior (see reflex agent)**, e.g.,
  $\forall\, t \quad Glitter(t) \Rightarrow BestAction(Grab, t)$.

Wumpus World (Environment)

Not relevant for the exam

- **Propositional logic:** We would have to “hand code” facts, such as which squares are adjacent.
- **First-order logic:** Complex term in which row and column appear as integers. Adjacency can be defined as:
  $\forall x, y, a, b \quad Adjacent([x, y], [a, b]) \Leftrightarrow$
  $\big(x = a \wedge (y = b - 1 \vee y = b + 1)\big) \vee \big(y = b \wedge (x = a - 1 \vee x = a + 1)\big).$
- **Pit**: Unary predicate that is true of squares containing pits.
- **Wumpus**: Constant since only one exists.
- **Predicate** $At(Agent, s, t)$: Agent is at square s at time t.
- **Wumpus’ location is fixed** by $\forall t \quad At(Wumpus, [2, 2], t)$.
- **Objects can only be at one location at a time:**
  $\forall x, s_1, s_2, t \quad At(x, s_1, t) \wedge At(x, s_2, t) \Rightarrow s_1 = s_2.$

Wumpus World (Combining Knowledge)

Not relevant for the exam

- **Infer properties from agent's percept**:
  $\forall\, s,\ t \quad At(Agent, s, t) \wedge Breeze(t) \Rightarrow Breezy(s)$
- **Infer locations of pits**:
  $\forall\, s \quad Breezy(s) \Leftrightarrow \exists\, r \quad Adjacent(r, s) \wedge Pit(r)$
- **Quantification over time**: Just one successor-state axiom for each predicate, rather than a different copy for each time step, e.g.,
  $\forall\, t \quad HaveArrow(t+1) \Leftrightarrow \big(HaveArrow(t) \wedge \neg Action(Shoot, t)\big).$

Knowledge Engineering Process (1)

We describe the general process of knowledge-base construction, which we exemplify for the Wumpus world:

1. **Identify the task**
   Example: does the knowledge base need to answer questions about actions or only about the environment?
2. **Assemble the relevant knowledge** from experts in their domain.
   Example: what does it mean when a cave is smelly?
3. **Decide on vocabulary of predicates, functions, and constants**.
   The resulting vocabulary is also known as the ontology of a domain.
   Example: Should pits be represented by constants or unary predicates?
4. **Encode general knowledge about the domain**. This often reveals misconceptions, requiring us to go back to step 3.
   Example: $\forall s, t \quad At(Agent, s, t) \wedge Breeze(t) \Rightarrow Breezy(s)$

Knowledge Engineering Process (2)

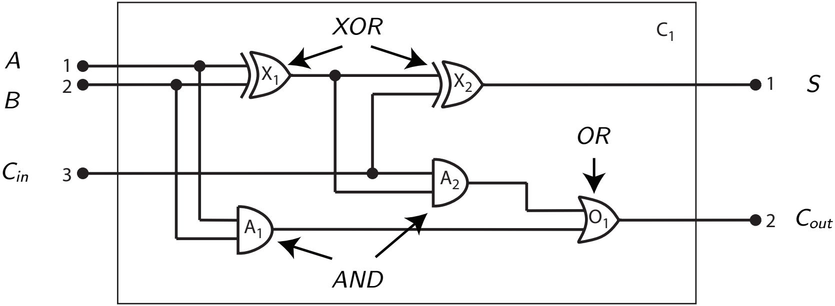
5. **Encode a description of the specific problem instance**. This step is simple if the ontology is well thought out. Problem instances come from sensors or are added as sentences.
   Example: There is no pit in square [1,1]: $\neg Pit([1,1])$.
6. **Pose queries to the inference procedure**. This is the reward: We get answers without writing an application-specific solution algorithm.
   Example: Where are the pits? $\texttt{AskVars}(KB, \exists x \quad Pit(x))$
7. **Debug the knowledge base**. If knowledge is missing, some queries cannot be answered.
   Example: if the knowledge base includes the diagnostic rule

   $$\forall s \quad Smelly(s) \Rightarrow Adjacent\big(Home(Wumpus), s\big),$$

   instead of the biconditional for finding the Wumpus, then the agent will never be able to prove the absence of Wumpus.

Detailed Example: One-Bit Full Adder

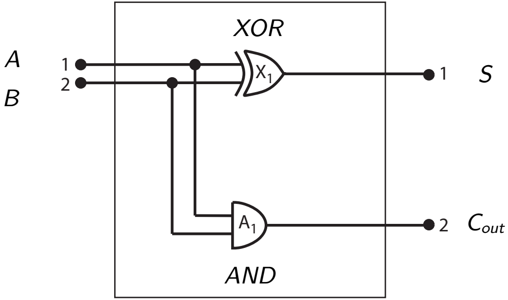
Not relevant for the exam



Short Excursion into Digital Circuits: Half Adder

Not relevant for the exam

A half adder takes as input two boolean variables A and B and produces two outputs: sum S and carry C. The carry represents an overflow into the next digit of a multi-digit addition.



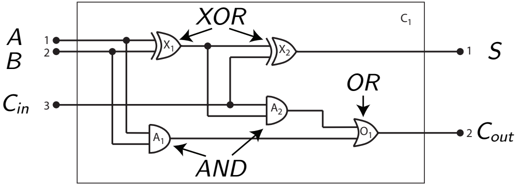
Truth table:

Inputs		Outputs	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Short Excursion into Digital Circuits: Full Adder

Not relevant for the exam

A full adder adds binary numbers and accounts for values carried in as well as out. A one-bit full adder adds the one-bit numbers A, B, and C_{in}.



Truth table:

Inputs			**Outputs**	
A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

Detailed Example: First Steps

Not relevant for the exam

1. **Identify the task**:
   - Does the circuit actually add properly? (circuit verification)
   - Other questions, e.g., timing analysis, are not of interest.
2. **Assemble the relevant knowledge**:
   - Composed of wires and gates; Gate types: AND, OR, XOR, NOT.
   - Irrelevant: size, shape, color, cost of gates.
3. **Decide on a vocabulary**:
   - Each gate, terminal, and circuit is represented by a predicate: $\mathit{Gate}(X_1)$, $\mathit{Terminal}(T_1)$, $\mathit{Circuit}(C_1)$.
   - Each gate can be of one type only (AND, OR, XOR, NOT), so that a function can specify it: $\mathit{Type}(X_1) = \mathit{XOR}$.
     Alternatives: $\mathit{Type}(X1, \mathit{XOR})$ *or* $\mathit{XOR}(X1)$.
   - We use the function $\mathit{In}(1, X_1)/\mathit{Out}(1, X_1)$ to return the first input/output terminal for gate X_1.
   - Pred. $\mathit{Arity}(c, i, j)$ says circuit c has i input and j output terminals.
   - Predicates for gate connections: $\mathit{Connected}(\mathit{Out}(1, X_1), \mathit{In}(1, X_2))$.
   - The function $\mathit{Signal}(t)$ returns the signal value (1 or 0) at terminal t.

Detailed Example: Domain Knowledge (1)

Not relevant for the exam

4. **Encode general knowledge of the domain**:

1. **Connected terminals have the same signal**:

$$\forall\, t_1,\ t_2 \quad Terminal(t_1) \wedge Terminal(t_2) \wedge Connected(t_1, t_2) \Rightarrow Signal(t_1) = Signal(t_2).$$

2. **The signal at every terminal is either** 1 **or** 0:

$$\forall\, t \quad Terminal(t) \Rightarrow Signal(t) = 1 \vee Signal(t) = 0.$$

3. **Connected is commutative**:

$$\forall\, t_1,\ t_2 \quad Connected(t_1, t_2) \Leftrightarrow Connected(t_2, t_1).$$

4. **There are four types of gates**:

$$\forall\, g \quad Gate(g) \wedge Type(g) = k \Rightarrow k = AND \vee k = OR \vee k = XOR \vee k = NOT.$$

Detailed Example: Domain Knowledge (2)

Not relevant for the exam

5. **An AND gate's output is** 0 $\Leftrightarrow$ **any of its inputs is** 0:

$$\forall g \quad Gate(g) \land Type(g) = AND \Rightarrow$$
$$Signal(Out(1,g)) = 0 \Leftrightarrow \exists n \, Signal(In(n,g)) = 0.$$

6. **An OR gate's output is** 1 $\Leftrightarrow$ **any of its inputs is** 1:

$$\forall g \quad Gate(g) \land Type(g) = OR \Rightarrow$$
$$Signal(Out(1,g)) = 1 \Leftrightarrow \exists n \, Signal(In(n,g)) = 1.$$

7. **An XOR gate's output is** 1 $\Leftrightarrow$ **its inputs are different**:

$$\forall g \quad Gate(g) \land Type(g) = XOR \Rightarrow$$
$$Signal(Out(1,g)) = 1 \Leftrightarrow Signal(In(1,g)) \neq Signal(In(2,g)).$$

8. **A NOT gate's output is different from its input**:

$$\forall g \quad Gate(g) \land Type(g) = NOT \Rightarrow$$
$$Signal(Out(1,g)) \neq Signal(In(1,g)).$$

Detailed Example: Domain Knowledge (3)

Not relevant for the exam

9. **Gates (excl. NOT) have two inputs and one output**:

$$\forall g \quad Gate(g) \land Type(g) = NOT \Rightarrow Arity(g, 1, 1).$$

$$\forall g \quad Gate(g) \land Type(g) = k \land (k = AND \lor k = OR \lor k = XOR) \Rightarrow Arity(g, 2, 1).$$

10. **A circuit has terminals up to its input and output arity**:

$$\forall c, i, j \quad Circuit(c) \land Arity(c, i, j) \Rightarrow$$
$$\forall n\ (n \leq i \Rightarrow Terminal(In(n, c))) \land (n > i \Rightarrow In(n, c) = Nothing) \land$$
$$\forall n\ (n \leq j \Rightarrow Terminal(Out(n, c))) \land (n > j \Rightarrow Out(n, c) = Nothing).$$

11. **Gates, terminals, signals, gate types, and Nothing are distinct**:

$$\forall g, t \quad Gate(g) \land Terminal(t) \Rightarrow$$
$$g \neq t \neq 1 \neq 0 \neq OR \neq AND \neq XOR \neq NOT \neq Nothing.$$

12. **Gates are circuits**:

$$\forall g \quad Gate(g) \Rightarrow Circuit(g).$$

Detailed Example: Encode Problem

Not relevant for the exam

5. **Encode the specific problem instance**:

$$
\begin{aligned}
&Circuit(C_1) \wedge Arity(C_1, 3, 2)\\
&Gate(X_1) \wedge Type(X_1) = XOR\\
&Gate(X_2) \wedge Type(X_2) = XOR\\
&Gate(A_1) \wedge Type(A_1) = AND\\
&Gate(A_2) \wedge Type(A_2) = AND\\
&Gate(O_1) \wedge Type(O_1) = OR.
\end{aligned}
$$

$$
\begin{array}{ll}
Connected(Out(1, X_1), In(1, X_2)) & Connected(In(1, C_1), In(1, X_1))\\
Connected(Out(1, X_1), In(2, A_2)) & Connected(In(1, C_1), In(1, A_1))\\
Connected(Out(1, A_2), In(1, O_1)) & Connected(In(2, C_1), In(2, X_1))\\
Connected(Out(1, A_1), In(2, O_1)) & Connected(In(2, C_1), In(2, A_1))\\
Connected(Out(1, X_2), Out(1, C_1)) & Connected(In(3, C_1), In(2, X_2))\\
Connected(Out(1, O_1), Out(2, C_1)) & Connected(In(3, C_1), In(1, A_2))
\end{array}
$$

Detailed Example: Pose Queries

Not relevant for the exam

6. **Pose queries to the inference procedure**:
   - What combination of inputs causes the first output of C_1 to be 0 and the second output of C_1 to be 1?

$$\begin{aligned} \exists\, i_1,\, i_2,\, i_3 \quad & Signal(In(1, C_1)) = i_1 \\ \wedge Signal(In(2, C_1)) = i_2 \wedge{} & Signal(In(3, C_1)) = i_3 \\ \wedge Signal(Out(1, C_1)) = 0 \wedge{} & Signal(Out(2, C_1)) = 1. \end{aligned}$$

`AskVars` provides three answers:

$$\{i_1/1,\, i_2/1,\, i_3/0\}, \quad \{i_1/1,\, i_2/0,\, i_3/1\}, \quad \{i_1/0,\, i_2/1,\, i_3/1\}.$$

   - What are the possible values of all terminals?

$$\begin{aligned} \exists\, i_1,\, i_2,\, i_3,\, o_1,\, o_2 \quad & Signal(In(1, C_1)) = i_1 \\ \wedge Signal(In(2, C_1)) = i_2 \wedge{} & Signal(In(3, C_1)) = i_3 \\ \wedge Signal(Out(1, C_1)) = o_1 \wedge{} & Signal(Out(2, C_1)) = o_2. \end{aligned}$$

The result is the complete input/output table from slide 37.

Summary

- **First-order logic is far more powerful than propositional logic**.
- **Knowledge representation languages** should be declarative, compositional, expressive, context independent, and unambiguous.
- **The syntax of first-order logic builds on that of propositional logic**. It adds terms to represent objects, and has universal and existential quantifiers.
- **Developing a knowledge base in first-order logic requires** analyzing the domain, choosing a vocabulary, and encoding the axioms required to support the desired queries.