

# Grundlagen der künstlichen Intelligenz – First-Order Logic

Matthias Althoff

TU München

November 14, 2019

KI\_11/15

KI\_11/21

# 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

Exam - related

- 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

*i.e. you ask KB & ask accordingly*

- ☺ Propositional logic is **declarative** (in contrast to procedural as in many programming languages, e.g., C): pieces of syntax correspond to facts. *no programming needed*
- ☺ 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**: *more unique* 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.”** *Compositional object*  
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

Language	<i>nature of being</i> <b>Ontological Commitment</b> <b>(What exists in the world)</b>	<i>its methods, validity &amp; scope</i> <b>Epistemological Commitment</b> <b>(What an agent believes about facts)</b>
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	<i>KingJohn</i> , 2, <i>TUM</i> , ...
<u>Predicates</u>	<u>Relations</u>	<i>Brother</i> , $>$ , ...
Functions	Functions	<i>Sqrt</i> , <i>LeftLegOf</i> , ...

Syntactic element	Examples
Variables	$x, y, a, b, \dots$
<u>Connectives</u>	$\wedge, \vee, \neg, \Rightarrow, \Leftrightarrow$
Equality	$=$
Quantifiers	$\forall, \exists$

Universal

Existential



Syntax of FOL: **Backus-Naur Form**Important

$$\text{Sentence} ::= \text{AtomicSentence} \mid \text{ComplexSentence}$$

$$\underline{\text{AtomicSentence}} ::= \text{Predicate} \mid \text{Predicate}(\text{Term}, \dots) \mid \text{Term} = \text{Term}$$

$$\text{ComplexSentence} ::= (\text{Sentence}) \mid [\text{Sentence}] \mid \neg \text{Sentence} \mid$$

$$\text{Sentence} \wedge \text{Sentence} \mid \text{Sentence} \vee \text{Sentence} \mid$$

$$\text{Sentence} \Rightarrow \text{Sentence} \mid \text{Sentence} \Leftrightarrow \text{Sentence} \mid$$

$$\text{Quantifier Variable}, \dots \text{Sentence}$$

$$\underline{\text{Term}} ::= \text{Function}(\text{Term}, \dots) \mid \text{Constant} \mid \text{Variable}$$

$$\text{Quantifier} ::= \forall \mid \exists$$

$$\text{Constant} ::= A \mid X_1 \mid \text{John} \mid \dots$$

$$\text{Variable} ::= a \mid x \mid s \mid \dots$$

$$\text{Predicate} ::= \text{True} \mid \text{False} \mid \text{After} \mid \text{Loves} \mid \dots$$

$$\text{Function} ::= \text{Mother} \mid \text{LeftLeg} \mid \dots$$

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:

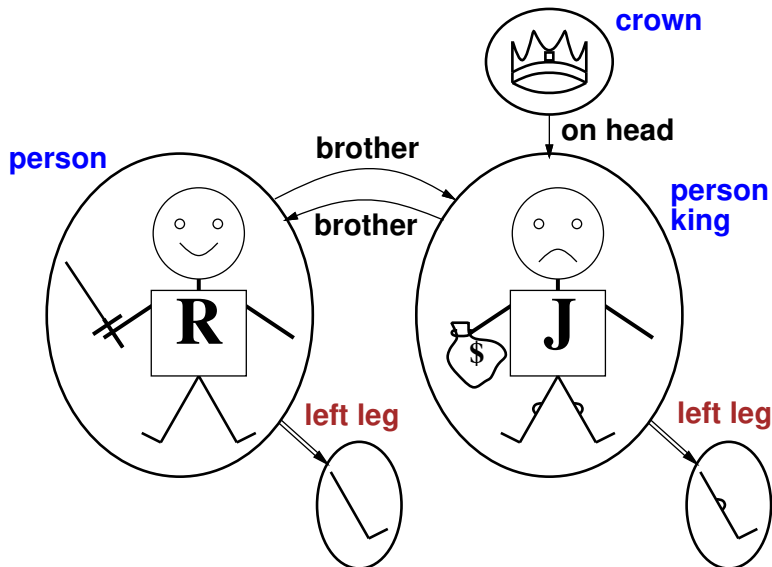
$$\underbrace{\forall x, \exists t \quad \underbrace{\underbrace{\text{person}(x)}_{\text{Predicate}} \underbrace{(x)}_{\text{Var.}}}_{\text{AtomicSentence}} \Rightarrow \underbrace{\underbrace{\text{time}(t)}_{\text{Predicate}} \underbrace{(t)}_{\text{Var.}}}_{\text{AtomicSentence}} \wedge \underbrace{\underbrace{\text{can-fool}(x,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

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

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

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

# Running Example



## Terms

①

think of it as way of creating new objects  
 ↗ instead of introducing new constants

Backus-Naur Form of terms

$$\text{Term} ::= \text{Function}(\text{Term}, \dots) \mid \text{Constant} \mid \text{Variable}$$

- 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.
- The semantics of terms is straightforward and thus not further detailed.

# Atomic Sentences (1)

## Backus-Naur Form of atomic sentences

$$\textit{AtomicSentence} ::= \textit{Predicate} \mid \textit{Predicate}(\textit{Term}, \dots) \mid \textit{Term} = \textit{Term}$$

- 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:**

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

not the same  
person

# Complex Sentences

## Backus-Naur Form of complex sentences

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

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

# Universal Quantification (∀)

④

All variables

- It is difficult to express general rules in propositional logic, e.g., “Squares neighboring the Wumpus are smelly”, which required explicitly listing all cases in propositional logic.
- In FOL this is very easy, e.g., “All kings are persons” can be written as

$$\forall x \text{ King}(x) \Rightarrow \text{Person}(x),$$

where the symbol  $x$  is called a variable (see slide 9).

~~\*~~ 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  $\text{King}(x) \Rightarrow \text{Person}(x)$  is true for all objects and thus  $\forall x \text{ King}(x) \Rightarrow \text{Person}(x)$  is true.



# Existential Quantification ( $\exists$ ) <sup>5</sup>

- In FOL, the existential quantifier is used to express a statement about some object; e.g., “King John has a crown on his head” can be written as

$$\exists x \text{ Crown}(x) \wedge \text{OnHead}(x, \text{John}).$$

- ~~\*~~ 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  $\text{Crown}(x) \wedge \text{OnHead}(x, \text{John})$  is true for the fifth object and thus

$\exists x \text{ Crown}(x) \wedge \text{OnHead}(x, \text{John})$  is true.

## Common Mistakes to Avoid

- Typically,  $\Rightarrow$  is the main connective with  $\forall$

*Common mistake:* using  $\wedge$  as the main connective with  $\forall$ :

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

means “Everyone is at TUM and everyone is smart” instead of  
 “Everyone at TUM is smart”.  $\forall x \quad At(x, TUM) \Rightarrow Smart(x)$

- Typically,  $\wedge$  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”.  $\exists x \quad At(x, TUM) \wedge Smart(x)$

# Scope of Quantifiers

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

$$\forall x(\exists y \text{ Brother}(x, y) \wedge \forall y \text{ Sibling}(x, y))$$


- ~~✗~~ 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

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


is the same as

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

# Scope of Quantifiers: Examples

- What does this mean?

$$\exists x \text{ Smart}(x) \wedge \text{Nice}(x).$$

“Someone is smart and nice.”

- What does this mean?

$$\exists x \text{ Smart}(x) \wedge \exists x \text{ Nice}(x).$$

“Someone is smart and someone is nice.” This is identical to

$$\exists x \text{ Smart}(x) \wedge \exists y \text{ Nice}(y).$$

- How to write the following sentence without parentheses?

$$\exists x (\text{Smart}(x) \wedge \exists x (\text{Nice}(x))).$$

This is identical to

$$\exists x \text{ Smart}(x) \wedge \exists x \text{ Nice}(x).$$

## Nested Quantifiers

- We often want to express more complex sentences using 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:

$\forall (x, y) \text{ Brother}(x, y) \Rightarrow \text{Sibling}(x, y)$  is the same as

$\forall x \forall y \text{ Brother}(x, y) \Rightarrow \text{Sibling}(x, y)$

- **Quantifiers of different type:**

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

\* Example:      $x$  loves  $y$

$\forall y (\exists x \text{ Loves}(x, y))$  "Everyone is loved by at least one person".

$\exists x (\forall y \text{ Loves}(x, y))$  "There is a person who loves everyone".

## Different Quantifiers Associated with the Same Variable

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

$$\forall x \left( \text{Crown}(x) \vee \left( \exists x \text{ Brother}(\text{Richard}, x) \right) \right),$$

where the  $x$  is universally and existentially quantified.

~~✗~~ The rule is that the variable belongs to the innermost quantifier.

- In the above example,  $\forall x$  has no effect.
- Solution: Use different variable names with nested quantifiers:

$$\forall x \left( \text{Crown}(x) \vee \left( \exists z \text{ Brother}(\text{Richard}, z) \right) \right).$$

Connections between  $\forall$  and  $\exists$ Important

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

$\forall x \text{ Likes}(x, \text{IceCream})$  is equivalent to  $\neg \exists x \neg \text{Likes}(x, \text{IceCream})$

$\exists x \text{ Likes}(x, \text{Broccoli})$  is equivalent to  $\neg \forall x \neg \text{Likes}(x, \text{Broccoli})$

De Morgan rules for quantified sentences

$$\forall x \neg P \equiv \neg \exists x P$$

$$\neg \forall x P \equiv \exists x \neg P$$

$$\forall x P \equiv \neg \exists x \neg P$$

$$\exists x P \equiv \neg \forall x \neg P$$

\* We do not need both  $\forall$  and  $\exists$ , just as we do not need both  $\wedge$  and  $\vee$ , but we keep it for readability. e.g.  $A \wedge B \equiv \neg A \vee \neg B$

# Tweedback Questions

- **Brothers are siblings:**

A  $\forall x, y \quad \text{Brother}(x, y) \wedge \text{Sibling}(x, y).$

B  $\forall x, y \quad \text{Brother}(x, y) \Rightarrow \text{Sibling}(x, y).$

- **One's mother is one's female parent:**

A  $\forall x, y \quad \text{Mother}(x, y) \Leftrightarrow (\text{Female}(x) \wedge \text{Parent}(x, y)).$

B  $\forall x, y \quad \text{Mother}(x, y) \Leftrightarrow (\text{Female}(x) \Rightarrow \text{Parent}(x, y)).$

- **A first cousin is a child of a parent's sibling:**

A

$$\forall x, y \quad \text{FirstCousin}(x, y) \Leftrightarrow$$

$$\forall p, ps \quad \text{Parent}(p, x) \wedge \text{Sibling}(ps, p) \wedge \text{Parent}(ps, y).$$

B

$$\forall x, y \quad \text{FirstCousin}(x, y) \Leftrightarrow$$

$$\exists p, ps \quad \text{Parent}(p, x) \wedge \text{Sibling}(ps, p) \wedge \text{Parent}(ps, y).$$



# Assertions and Queries in First-Order Logic

how to fill &  
retrieve KB

- Sentences are added to a knowledge base using Tell. Such sentences are called assertions, e.g.,

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

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

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

- We can ask questions of the knowledge base using Ask, called queries, e.g.,

$\text{Ask}(\text{KB}, \text{King}(\text{John}))$

returns *true*.

- For some queries a yes/no answer is undesirable, e.g.,

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

With AskVars we obtain a stream of answers:

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

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

## Example 1: Formalizing Overtaking in First-Order Logic

### 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)

## Objects required:

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

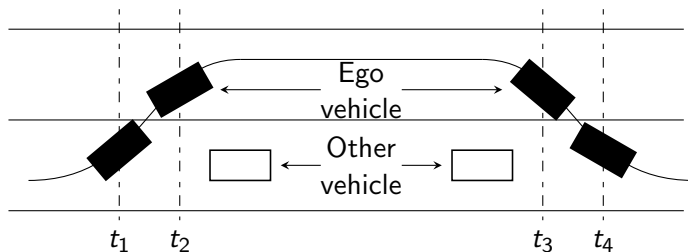
## Predicates required:

- ① *is-time( $t$ )* — true iff  $t$  is a real number
- ② *is-traffic-part( $id$ )* — true iff  $id$  is a traffic participant or **ego**
- ③ *sd-rear( $id_1, id_2$ )* —  $id_1$  is safely located behind  $id_2$  [safe distance]
- ④ *follows( $id_1, id_2$ )* —  $id_1$  follows  $id_2$

## Function required:

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

# Formalizing Overtaking (Semantics)



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

# Formalizing Overtaking (Translation to First-Order Logic)

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

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

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

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

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

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

## Example 2: Wumpus World (Sensing and Acting)

- **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  
 $AskVars(\exists a \quad BestAction(a, 5)).$
- **Simple reflex behavior (see reflex agent)**, e.g.,  
 $\forall t \quad Glitter(t) \Rightarrow BestAction(Grab, t).$

# Wumpus World (Environment)

- **Propositional logic:** Name each square (e.g.,  $Square_{1,2}$ ), but then 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:  

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

- **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.,

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



# Knowledge Engineering Process (1)

We describe the general process of knowledge-base construction:

## ① Identify the task

Example: does the Wumpus knowledge base need to answer questions about actions or is it sufficient to answer questions about the environment?

## ② Assemble the relevant knowledge from experts in their domain

Example: what does it mean when a cave in Wumpus world 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 in Wumpus world 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?  $AskVars(KB, \exists x \quad Pit(x))$

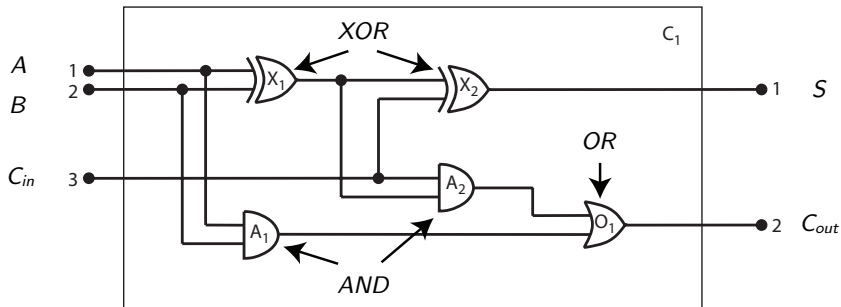
- ⑦ **Debug the knowledge base.** If knowledge is missing, some queries cannot be answered.

Example: if the knowledge base includes a diagnostic rule for finding the Wumpus,

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

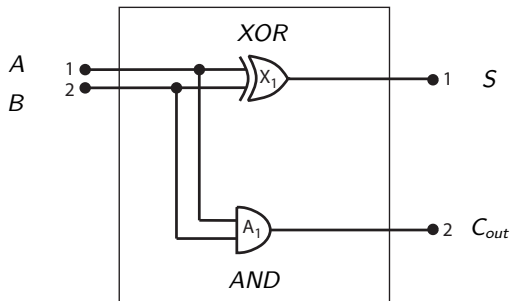
instead of the biconditional, then the agent will never be able to prove the absence of Wumpus.

## Detailed Example: One-Bit Full Adder



## Short Excursion into Digital Circuits: Half Adder

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.



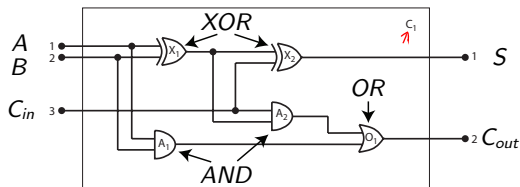
Truth table:

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

A full adder adds binary numbers and accounts for values carried in as well as out. A one-bit full adder adds three one-bit numbers, often written as  $A$ ,  $B$ , 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

## ① Identify the task:

- Does the circuit actually add properly? (circuit verification)
- Other questions, e.g., timing analysis, are not required in this analysis.

## ② Assemble the relevant knowledge:

- Composed of wires and gates; Types of gates: 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:  
 $Gate(X_1)$ ,  $Terminal(T_1)$ ,  $Circuit(C_1)$ .
- Each gate can be of one type only (AND, OR, XOR, NOT), so that a function can specify it:  $Type(X_1) = XOR$ .  
*Alternatives:*  $Type(X_1, XOR)$  or  $XOR(X_1)$ .
- We use the function  $In(1, X_1)/Out(1, X_1)$  to return the first input/output terminal for gate  $X_1$ .
- Pred.  $Arity(c, i, j)$  says circuit  $c$  has  $i$  input and  $j$  output terminals.
- Predicates for gate connections:  $Connected(Out(1, X_1), In(1, X_2))$ .
- The function  $Signal(t)$  returns the signal value (1 or 0) at terminal  $t$ .

# Detailed Example: Domain Knowledge (1)

## ④ Encode general knowledge of the domain:

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

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

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

$$\forall t \quad \text{Terminal}(t) \Rightarrow \text{Signal}(t) = 1 \vee \text{Signal}(t) = 0.$$

3. Connected is commutative:

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

4. There are four types of gates:

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

## Detailed Example: Domain Knowledge (2)

5. **An AND gate's output is 0 if and only if any of its inputs is 0:**

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

6. **An OR gate's output is 1 if and only if any of its inputs is 1:**

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

7. **An XOR gate's output is 1 if and only if its inputs are different:**

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

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

$$\forall g \quad \text{Gate}(g) \wedge \text{Type}(g) = \text{NOT} \Rightarrow \\ \text{Signal}(\text{Out}(1, g)) \neq \text{Signal}(\text{In}(1, g)).$$



## Detailed Example: Domain Knowledge (3)

9. **The gates (except for NOT) have two inputs and one output:**

$$\forall g \quad \text{Gate}(g) \wedge \text{Type}(g) = \text{NOT} \Rightarrow \text{Arity}(g, 1, 1).$$

$$\forall g \quad \text{Gate}(g) \wedge \text{Type}(g) = k \wedge (k = \text{AND} \vee k = \text{OR} \vee k = \text{XOR}) \Rightarrow \text{Arity}(g, 2, 1).$$

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

$$\begin{aligned} & \forall c, i, j \quad \text{Circuit}(c) \wedge \text{Arity}(c, i, j) \Rightarrow \\ & \forall n \quad (n \leq i \Rightarrow \text{Terminal}(\text{In}(n, c))) \wedge (n > i \Rightarrow \text{In}(n, c) = \text{Nothing}) \wedge \\ & \forall n \quad (n \leq j \Rightarrow \text{Terminal}(\text{Out}(n, c))) \wedge (n > j \Rightarrow \text{Out}(n, c) = \text{Nothing}). \end{aligned}$$

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

$$\begin{aligned} & \forall g, t \quad \text{Gate}(g) \wedge \text{Terminal}(t) \Rightarrow \\ & g \neq t \neq 1 \neq 0 \neq \text{OR} \neq \text{AND} \neq \text{XOR} \neq \text{NOT} \neq \text{Nothing}. \end{aligned}$$

12. **Gates are circuits:**

$$\forall g \quad \text{Gate}(g) \Rightarrow \text{Circuit}(g).$$

## Detailed Example: Encode Problem

- ⑤ **Encode the specific problem instance:** First, we categorize the circuit and its gates:

$$\text{Circuit}(C_1) \wedge \text{Arity}(C_1, 3, 2)$$

$$\text{Gate}(X_1) \wedge \text{Type}(X_1) = \text{XOR}$$

$$\text{Gate}(X_2) \wedge \text{Type}(X_2) = \text{XOR}$$

$$\text{Gate}(A_1) \wedge \text{Type}(A_1) = \text{AND}$$

$$\text{Gate}(A_2) \wedge \text{Type}(A_2) = \text{AND}$$

$$\text{Gate}(O_1) \wedge \text{Type}(O_1) = \text{OR}.$$

Second, we specify the connections:

$$\text{Connected}(\text{Out}(1, X_1), \text{In}(1, X_2))$$

$$\text{Connected}(\text{Out}(1, X_1), \text{In}(2, A_2))$$

$$\text{Connected}(\text{Out}(1, A_2), \text{In}(1, O_1))$$

$$\text{Connected}(\text{Out}(1, A_1), \text{In}(2, O_1))$$

$$\text{Connected}(\text{Out}(1, X_2), \text{Out}(1, C_1))$$

$$\text{Connected}(\text{Out}(1, O_1), \text{Out}(2, C_1))$$

$$\text{Connected}(\text{In}(1, C_1), \text{In}(1, X_1))$$

$$\text{Connected}(\text{In}(1, C_1), \text{In}(1, A_1))$$

$$\text{Connected}(\text{In}(2, C_1), \text{In}(2, X_1))$$

$$\text{Connected}(\text{In}(2, C_1), \text{In}(2, A_1))$$

$$\text{Connected}(\text{In}(3, C_1), \text{In}(2, X_2))$$

$$\text{Connected}(\text{In}(3, C_1), \text{In}(1, A_2))$$

## Detailed Example: Pose Queries

### 6 Pose queries to the inference procedure:

- What combination of inputs causes the first output of  $C_1$  (the sum bit) to be 0 and the second output of  $C_1$  (the carry bit) to be 1?

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

AskVars provides three answers (details on inference are presented in the next lecture):

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

- What are the possible values of all terminals?

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

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.
- *Unlike those in N/N*  
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.