Overview

I Artificial Intelligence

II Problem Solving

III Knowledge, Reasoning, Planning

- 7. Logical Agents
- **8. First-Order Logic**
- 9. Inference in First-Order Logic
- 10. Knowledge Representation
- 11. Automated Planning

IV Uncertain Knowledge and Reasoning

V Machine Learning

VI Communicating, Perceiving, and Acting

VII Conclusions

First-Order Logic

- Representation Revisited
- Syntax and Semantics of First-Order Logic
- Using First-Order Logic
- Knowledge Engineering in First-Order Logic

Nature of Representations

- Programming languages
- Propositional logic
- Language of Thought

Representations in Programming Languages

- Largest class of formal languages in common use
- Data Structure: Facts (variables), e.g. array like „World[2,2] == Pit“ in wumpus world
- Disadvantages in comparison to (propositional) logic
  - No general mechanism for deriving facts from other facts
    - Needs domain-specific procedures
  - No easy way to represent partial information, e.g „there is a pit in [2,2] or [3,1]“

Propositional Logic

- **Declarative language:** Truth relation between sentences and possible words; separation of data and inference
- **Partial information** can be represented using disjunction and negation
- **Compositionality:** Meaning of a sentence is a function of the meaning of the parts
- Disadvantages in comparison to natural language
  - Lacks expressiveness
    - Cannot represent statements like „All squares adjacent to pits are breezy“
    - Needs a separate rule for each square like „$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$“

Language of Thought

- Natural languages are very expressive
- Modern view: They **serve for communication** rather than pure representation
  - Meaning results from sentence and context (e.g. „Look“)
  - Contains ambiguity (e.g. „spring“ as season, technical term, or water origin)
    - Context usually solves ambiguity
- **Sapir-Whorf hypothesis**:
  - „Our understanding of the world is strongly influenced by the language we speak.“
  - Example: Bridge is masculine in Spanish and feminine in German. When asking to describe a picture of a bridge, Spanish speakers chose terms like „big, dangerous, strong, towering“ whereas German speakers chose „beautiful, elegant, fragile, slender“.
  - „Framing“ of the same question with different words results in differences in surveys
  - In learning tasks, „Occam‘s Razor“ or „simplicity first“ prefers the most succinct theory, whereby simplicity depends on the representation (i.e. language).

First-Order Logic

- More expressive than propositional logic, but less expressive than natural languages
- World contains not only facts, but also objects with attributes (unary and n-ary relations)
  - **Objects:** e.g. people, houses, squares, pits
  - **Relations:**
    - unary relations (properties) like red(x), green(x), round(x)
    - n-ary relations: brother-of(x,y) , bigger-than(x,y), inside(x,y), part-of(x,y)
  - **Functions:** Relations, in which there is only one value for a given input, e.g. father-of(x)

Examples for First-Order Logic (FOL)

- Example 1: One plus two equals three
  - Objects: One, two, three
  - Relation: equals
  - Function: plus
  - FOL: equals (plus (one, two), three)
- Example 2: Squares adjacent to a wumpus are smelly.
  - Objects: Square, wumpus
  - Relation: adjacent
  - Property: smelly
  - FOL: $\forall$ x square (x) $\wedge$ adjacent (x, wumpus) $\Rightarrow$ smelly (x)
- Example 3: Evil king John ruled England in 1200
  - Objects: John, England, 1200; Relation: ruled-during; Properties: evil, king
  - FOL: Ruled-During (John, England, 1200) $\wedge$ evil (John) $\wedge$ king (John)

Other Logics

Logics can be classified according to different commitments:

- **Ontological commitments:** Related to reality
- **Epistemological commitments:** Related to knowledge of agents

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

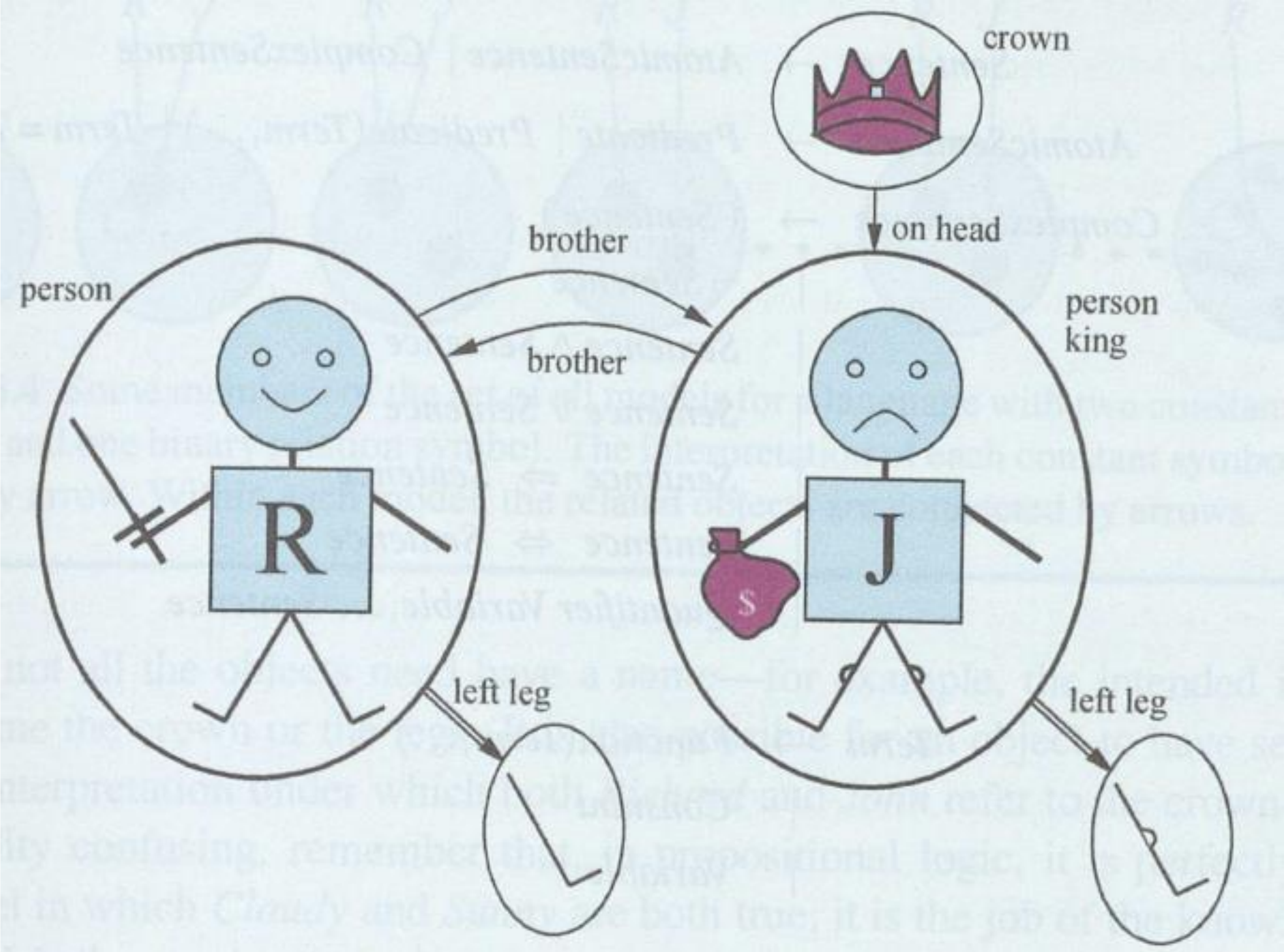
Models in Logic

- *Semantic of representation language: Defines truth of sentences with respect to each possible world (model)*
  - **Possible world:** (Potentially) real environment
  - **Model:** Mathematical abstraction with a truth value (true or false) for each sentence
    - In propositional logic, each variable (proposition symbol) has a truth value
    - In first-order logic, models are more interesting, since objects and relations must be related to something meaningful
      - There are many (nearly infinite many) models for a FOL description
      - Usually, only one model is intended

Models of First-Order Logic

A model containing 5 objects, 2 binary relations (brother, on-head), 3 unary relations (person, king, crown) and one unary function (left-leg)

- Besides the intended interpretation there are many other possible interpretations, e.g. mapping the constant symbol Richard (R) to the crown (or to another of the 5 objects).
- To rule out unintended interpretations, additional statements may be necessary (e.g. that Richard and John are distinct)



Syntax of First-Order Logic

- **Term:** Refer to an object, e.g. John, Leftleg (John)
- **AtomicSentence** (atom): Predicate optionally with terms, e.g. Brother (Richard, John)
- **ComplexSentence**: Uses the same logical connectives as propositional logic , e.g. ¬King(Richard) ⇒ King(John)
- **Quantifiers**: Express properties over collections of objects:
  - **Univeral Quantifier**: ∀ (for all …)
  - **Existential Quantier**: ∃ (There exists …)
  - Order of quantifier important, e.g.
    - ∀ x (∃ y Loves (x,y)) versus ∃ x (∀y Loves (x,y))
  - Connections similar to de Morgan‘s rules, e.g.
    - ∀ x Likes (x, IceCream) is equivalent to ¬∃ y ¬Likes (x, IceCream)
- **Equality**: 2 terms refer to same object, e.g. Father(John) = Henry

Sentence	→	*AtomicSentence* | *ComplexSentence*
AtomicSentence	→	*Predicate* | *Predicate*(*Term*, ...) | *Term* = *Term*
ComplexSentence	→	(*Sentence*)
	|	¬ *Sentence*
	|	*Sentence* ∧ *Sentence*
	|	*Sentence* ∨ *Sentence*
	|	*Sentence* ⇒ *Sentence*
	|	*Sentence* ⇔ *Sentence*
	|	*Quantifier Variable*, ... *Sentence*
Term	→	*Function*(*Term*, ...)
	|	*Constant*
	|	*Variable*
Quantifier	→	∀ | ∃
Constant	→	*A* | X_1 | *John* | ⋯
Variable	→	*a* | *x* | *s* | ⋯
Predicate	→	*True* | *False* | *After* | *Loves* | *Raining* | ⋯
Function	→	*Mother* | *LeftLeg* | ⋯

OPERATOR PRECEDENCE : ¬, =, ∧, ∨, ⇒, ⇔

Capturing Semantics

- Assume, Richard has two brothers, John and Geoffry.
  - Brother (John, Richard) $\land$ Brother (Geoffry, Richard)
    - Not correct in First-Order Logic: It does not exclude, that their is only one brother if (John = Geoffry) or that there are further brothers
    - Might lead to wrong conclusions!
  - Brother (John, Richard) $\land$ Brother (Geoffry, Richard) $\land$ John $\neq$ Geoffry $\land$ $\forall$ x Brother (x, Richard) $\Rightarrow$ (x = John) $\lor$ (x = Georffy)
    - Correct but cumbersome
- Alternate semantic: **Database semantic** with 3 more assumptions to allow first statement:
  - **Unique name assumption:** Every constant refers to a different object
  - **Closed-world assumption:** Unknown sentences are false
  - **Domain closure:** There are no more elements than those named by constant symbols

Using First-Order Logic (1)

Example: Kinship-domain

- **Unary predicates:** Male, Female, …
- **Binary predicates:** Parent, Sibling, Brother, Sister, Child, Daughter, Son, Spouse, Wife, Husband, Grandparent, Grandchild, Cousin, Aunt, Uncle, …
- **Functions:** Father, Mother, …
- **Axioms:**
  - $\forall$ m,c Mother (c) = m $\Leftrightarrow$ Female (m) $\wedge$ Parent (m, c)
  - $\forall$ w,h Husband (h,w) $\Leftrightarrow$ Male (h) $\wedge$ Spouse (h, w)
  - $\forall$ p,c Parent (p,c) $\Leftrightarrow$ Child (c,p)
  - $\forall$ g,c Grandparent (g,c) $\Leftrightarrow$ $\exists$p Parent (g,p) $\wedge$ Parent (p, c)
  - $\forall$ x,y Sibling (x,y) $\Leftrightarrow$ x $\neq$ y $\wedge$ $\exists$p Parent (p,x) $\wedge$ Parent (p, y)
  - …
- **Theorems** (are entailed in the axioms):
  - $\forall$ x,y Sibling (x,y) $\Leftrightarrow$ Sibling (y,x), …
- Axioms for facts, e.g. Male (Jim) and Spouse (Jim, Laura), …

Using First-Order Logic (2)

Example: Numbers, sets, and lists

- Peano axioms define natural numbers (Natnum) and addition:
  - NatNum(0)
  - $\forall$n NatNum (n) $\Rightarrow$ NatNum (S (n))
  - $\forall$n 0 $\neq$ S(n)
  - $\forall$m,n m $\neq$ n $\Rightarrow$ S(m) $\neq$ S(n)
  - $\forall$m NatNum (m) $\Rightarrow$ + (0,m) = m
  - $\forall$m,n NatNum (m) $\wedge$ NatNum (n) $\Rightarrow$ + (S(m),n) = S(+ (m,n))
    - in infix-notation: $\forall$m,n NatNum (m) $\wedge$ NatNum (n) $\Rightarrow$ (m+1) + n = (m+n) + 1
- Similar for sets and lists

Using First-Order Logic (3)

Example: Wumpus world (much easier than in propositional logic)

- A typical percept at time step 5 would be: Percept (Stench, Breeze, Glitter, None, None), 5)
  - Percept is a binary predicate and Stench, Breeze, etc. are constants
- Actions:
  - Turn (Right), Turn (Left), Forward, Shoot, Crab, Climb
- To determine, which action is the best, the agent program executes a query:
  - AskVars (KB, BestAction(a, 5)
  - Returns a binding list like (a/Grab)
- The raw percept data implies certain facts about the current state:
  - $\forall$t,s,g,w,c Percept ([s, Breeze, g, w, c], t) $\Rightarrow$ Breeze(t)
  - $\forall$t,s,g,w,c Percept ([s, None, g, w, c], t) $\Rightarrow$ $\neg$Breeze(t)
- Simple „reflex behaviour“ can be formalized easily: $\forall$t Glitter (t) $\Rightarrow$ BestAction (Grab, t)
- Defining Ajacency: $\forall$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))

Wumpus example continued

- If an agent is at a square and perceives a breeze, then that square is breezy. Since pits cannot move, „breezy“ needs no time parameter:
  - $\forall s,t\ At(Agent, s, t) \land Breeze\ (t) \Rightarrow Breezy\ (s)$
- However, shooting an arrow needs a time parameter:
  - $\forall t\ HaveArrow\ (t+1) \Leftrightarrow (HaveArrow\ t) \land \neg Action\ (Shoot, t)$

- Causal rules (from causes to effects), e.g.:
  - $\forall s\ [\forall r\ \ Adjacent\ (r, s) \Rightarrow \neg Pit\ (r)] \Rightarrow \neg Breezy\ (s)$
  - $\forall r\ Pit\ (r) \Rightarrow [\forall s\ \ Adjacent\ (r, s) \Rightarrow Breezy\ (s)]$
- Diagnostic rules (from effect to causes), e.g.:
  - $\forall s\ \ Breezy\ (s) \Rightarrow\ \ \exists r\ \ Adjacent\ (r, s) \land Pit\ (r)$
  - $\forall s\ \neg Breezy\ (s) \Rightarrow \neg\exists r\ \ Adjacent\ (r, s) \land Pit\ (r)$ *(equivalent to)* $\forall r\ \ \neg Adjacent\ (r, s) \lor \neg Pit\ (r)$

Knowledge Engineering

- A knowledge engineer should know:
  - Domain knowledge about objects and relations
  - General knowledge about the knowledge representation and inference procedure
- Seven steps to build a knowledge base:
  1. Identify the questions: Task understanding, PEAS process
  2. Assemble the relevant knowledge
  3. Decide on the vocabulary (ontology) of predicates, functions, constants
  4. Encode general knowledge about the domain
  5. Encode a description of the problem instance
  6. Pose questions to the inference procedure and get answers
  7. Debug and evaluate the knowledge base (should be easier than debugging code)

Example for Knowledge Engeering: A Digital Circuit

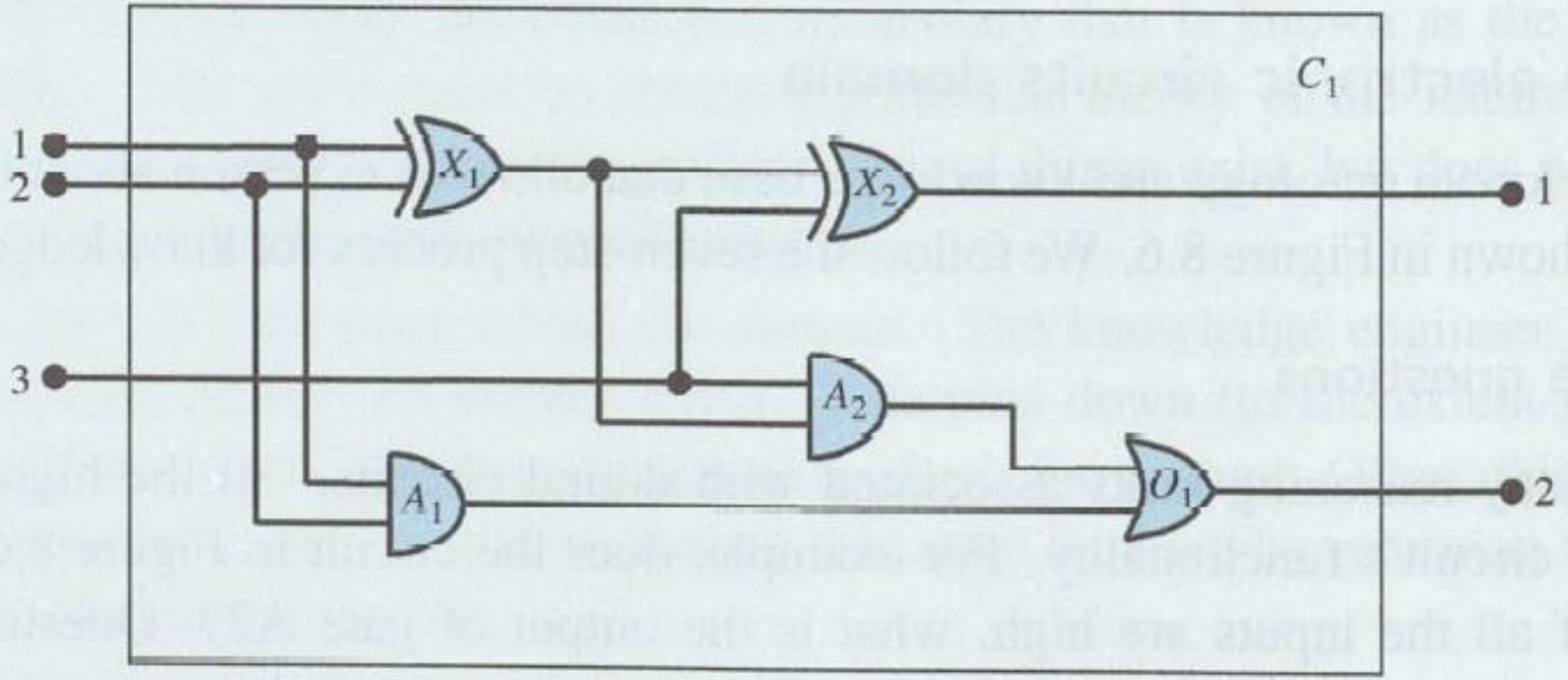


Figure 8.6 A digital circuit C_1, purporting to be a one-bit full adder. The first two inputs are the two bits to be added, and the third input is a carry bit. The first output is the sum, and the second output is a carry bit for the next adder. The circuit contains two XOR gates, two AND gates, and one OR gate.

Task Description

- Possible tasks:
  - Does the circuit work correctly?
    - e.g. if all inputs are high, what is the output at gate A2?
  - Can errors be recognized (diagnosis)?
  - Can the efficiency improved?
- Tasks concerning the structure of the ciruit
  - Does it contain feedback loops?
  - Are there timing delays?
- General tasks:
  - Power consumption, production costs, etc.

➢ The knowledge to be acquired depends on the task!

Assembly of Relevant Knowledge

- Circuits are composed of wires and gates with terminal (for input and output)
  - There are four types of gates: AND, OR, XOR, NOT

- Level of detail depends on the task:
  - Here: Does the circuit work correctly?
  - For diagnosis, we need e.g. statements about wires (which need not be modelled if correct functioning is assumend)
  - For effiency analysis, we need additional knowledge about duration of gates and wires

Decide on Vocabulary

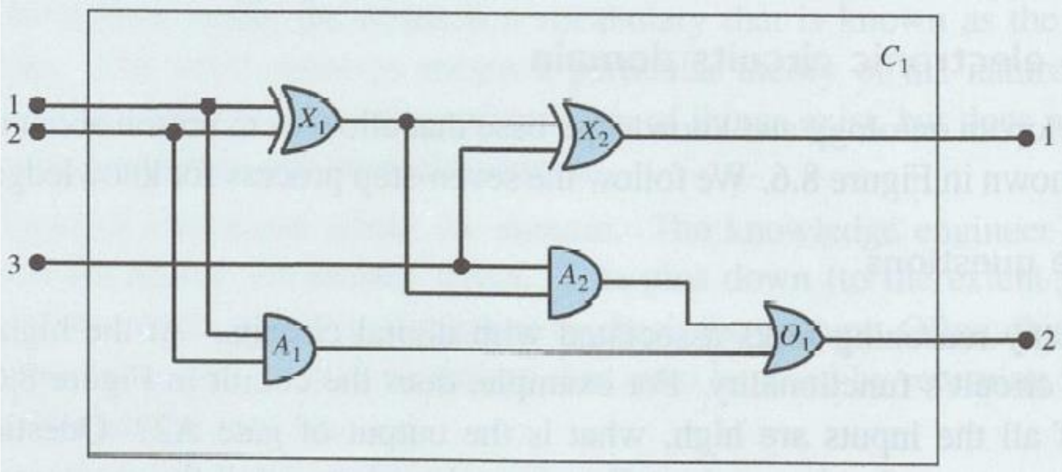
- Representation of gates:
  - As objects named with constants. For gate types we introduce a function „Type“, e.g. Type (X1) = XOR
- Representation of terminals:
  - Instead of constants better with functions of the gates, e.g. In (1, X1), In (2, X1), Out (1, X1) etc.
- Representation of connections between gates:
  - With predicates, e.g. Connected (Out (1, X1), In (1, X2))
- Representation of signals
  - With constants for the signal values 1 and 0 representing „on“ and „off“ and a function signal (t), that denotes the signal value for the terminal t.

Encoding General Knowledge of the Domain

1. If two terminals are connected, then they have the same signal:
$$\forall t_1, t_2 \; Terminal(t_1) \wedge Terminal(t_2) \wedge Connected(t_1, t_2) \Rightarrow Signal(t_1) = Signal(t_2) \, .$$
2. The signal at every terminal is either 1 or 0:
$$\forall t \; Terminal(t) \Rightarrow Signal(t) = 1 \vee Signal(t) = 0 \, .$$
3. *Connected* is commutative:
$$\forall t_1, t_2 \; Connected(t_1, t_2) \Leftrightarrow Connected(t_2, t_1) \, .$$
4. There are four types of gates:
$$\forall g \; Gate(g) \wedge k = Type(g) \Rightarrow k = AND \vee k = OR \vee k = XOR \vee k = NOT \, .$$
5. An AND gate's output is 0 if and only if any of its inputs is 0:
$$\forall g \; Gate(g) \wedge Type(g) = AND \Rightarrow Signal(Out(1, g)) = 0 \Leftrightarrow \exists n \; Signal(In(n, g)) = 0 \, .$$
6. An OR gate's output is 1 if and only if any of its inputs is 1:
$$\forall g \; Gate(g) \wedge Type(g) = OR \Rightarrow Signal(Out(1, g)) = 1 \Leftrightarrow \exists n \; Signal(In(n, g)) = 1 \, .$$
7. An XOR gate's output is 1 if and only if its inputs are different:
$$\forall g \; Gate(g) \wedge Type(g) = XOR \Rightarrow Signal(Out(1, g)) = 1 \Leftrightarrow Signal(In(1, g)) \neq Signal(In(2, g)) \, .$$
8. A NOT gate's output is different from its input:
$$\forall g \; Gate(g) \wedge Type(g) = NOT \Rightarrow Signal(Out(1, g)) \neq Signal(In(1, g)) \, .$$
9. The gates (except for NOT) have two inputs and one output.
$$\forall g \; Gate(g) \wedge Type(g) = NOT \Rightarrow Arity(g, 1, 1) \, .$$
$$\forall g \; Gate(g) \wedge k = Type(g) \wedge (k = AND \vee k = OR \vee k = XOR) \Rightarrow Arity(g, 2, 1)$$
10. A circuit has terminals, up to its input and output arity, and nothing beyond its arity:
$$\forall c, i, j \; Circuit(c) \wedge Arity(c, i, j) \Rightarrow$$
$$\forall n \; (n \leq i \Rightarrow Terminal(In(n, c))) \wedge (n > i \Rightarrow In(n, c) = Nothing) \wedge$$
$$\forall n \; (n \leq j \Rightarrow Terminal(Out(n, c))) \wedge (n > j \Rightarrow Out(n, c) = Nothing)$$
11. Gates, terminals, and signals are all distinct.
$$\forall g, t, s \; Gate(g) \wedge Terminal(t) \wedge Signal(s) \Rightarrow g \neq t \wedge g \neq s \wedge t \neq s \, .$$
12. Gates are circuits.
$$\forall g \; Gate(g) \Rightarrow Circuit(g)$$

Encoding the Specific Problem Instance

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



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

Pose Queries to the Inference Procedure

What combinations of inputs would cause 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?

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

The answers are substitutions for the variables i_1, i_2, and i_3 such that the resulting sentence is entailed by the knowledge base. ASKVARS will give us three such substitutions:

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

What are the possible sets of values of all the terminals for the adder circuit?

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

The last query will return a complete input-output table for the circuit (kind of total check)

Debug the Knowledge Base

- The first queries to the inference procedure usually yield buggy answers
- Debugging is simplified by:
  - The inference procedure should be correct
  - There should be an explanation component showing the trace of used rules and facts for answering the query
- Often, errors result from missing axioms (e.g. if we forget to assert that 1 ≠ 0)