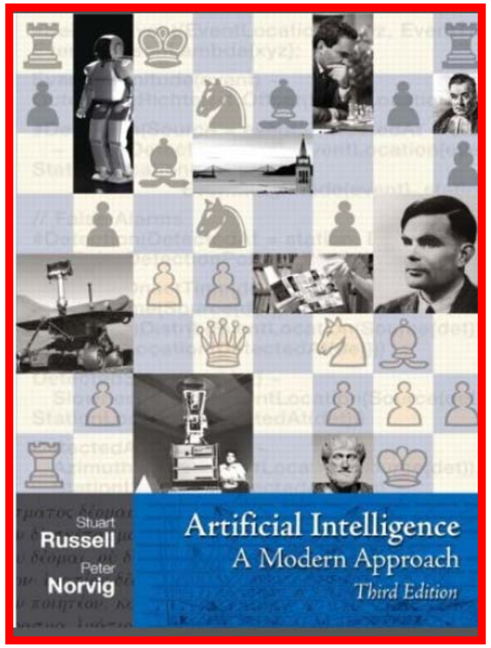


(CHAPTER-8)
FIRST-ORDER LOGIC

Yanmei Zheng

Outline

- Why FOL?
- Syntax and semantics of FOL
- Using FOL
- Wumpus world in FOL
- Knowledge engineering in FOL

First Order Logic

- Objects, relation, function
- Variables, constants, functions, predicates, quantifier, equality, connectives
- Term, atomic sentence , complex sentences

Review： Chapter 7

- Logical agents apply inference to a knowledge base to derive new information and make decisions
- Basic concept of logic
  - Syntax: formal structure of sentences
  - Semantics: truth of sentences wrt(with regard to) models

Review: Chapter 7

- 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

Propositional logic

- Propositional logic is declarative

  Propositional logic allows partial/disjunctive/negated information

  - ✓ (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)

First Order Logic

- **Some P.L. weakness:**
  1. **Limited ability to express knowledge and lose much of their meanings.**
  2. **Not all statements can be represented.**

     All men are mortals.

     Some dogs like cats.
- **Thus, need a more general form of logic capable of representing the details.**

Wumpus world using propositional logic {8.1}

$\neg P_{1.1}, \neg W_{1.1}, \neg B_{1.1}, \neg S_{1.1}$

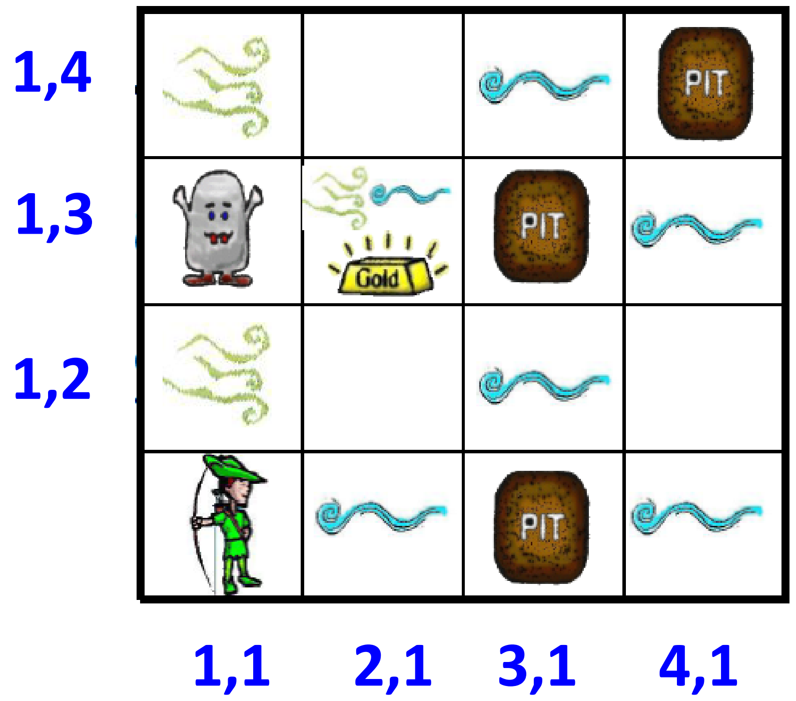
$B_{2.1}, \neg S_{2.1}$

$B_{2.1} \Leftrightarrow P_{1.1} \vee P_{2.2} \vee P_{3.1}$

$S_{2.1} \Leftrightarrow W_{1.1} \vee W_{2.2} \vee W_{3.1}$

$P_{2.2} \vee P_{3.1}, \neg W_{2.2}, \neg W_{3.1}$

…



"pits cause breezes in adjacent squares" needs 16 PL sentences, such as $B_{2.1} \Leftrightarrow P_{1.1} \vee P_{2.2} \vee P_{3.1}$

Many distinct proposition symbols, many sentences

How does PL say these? {8.1}

- In PL (Propositional Logic), in order to say
  - ✓ The adjacent squares of pit are breezy.
  - ✓ All students are smart.
  - ✓ Some students work hard.
  - ✓ Hardworking students have good marks.
  - ✓ If someone is someone else's grandfather, then
  - ✓ someone else has a grandson.
- we have to_____________
- With FOL (First-order Logic), we can say each of these situations with one sentence

How does PL say these? {8.1}

- In PL (Propositional Logic), in order to say
  - The adjacent squares of pit are breezy.
  - All students are smart.
  - Some students work hard.
  - Hardworking students have good marks.
  - If someone is someone else's grandfather, then someone else has a grandson.
- we have to enumerate

- With FOL (First-order Logic), we can say each of these situations with one sentence

The Others

Language	Ontological Commitment (What exists in the world)	Epistemological Commitment (What an agent believes about facts)
Proposition 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 of {0,1}
Fuzzy logic	Facts,with degree of true of {0,1}	Known interval value

First Order Logic

- Propositional logic combines atoms
  - An atom contains no propositional connectives
  - Have no structure (today_is_wet, john_likes_apples)
- Predicates allow us to talk about objects
  - Properties: is_wet(today)
  - Relations: likes(john, apples)
  - True or false
- In predicate logic each atom is a predicate
  - e.g. first order logic, higher-order logic

First Order Logic

- More expressive logic than propositional
- Enhances processing by allowing the use of variables and functions.

  Use symbols that represent

  constants are objects:

  john, apples

  Predicates are properties and relations:

  likes(john, apples)

  variables represent any object:

  likes(X, apples)

  Functions transform objects:

  likes(john, fruit_of(apple_tree))

- Operate on these symbols using PL operators.

First-order logic

- Propositional logic assumes the world contains facts
- First-order logic (like natural language) assumes the world contains
  - ✓ Objects: people, houses, numbers, colors, baseball games, wars, …
    Relations: red, round, prime, brother of, bigger than, part of, comes between, …
  - ✓ Functions: father of, best friend, one more than, plus, …

First Order Logic

- Constant

- Specific objects or properties about a problem.
- Begin with lower case.
- Example: ahmad, elephant and temperature
- ahmad represent object Ahmad. Can also use A or X instead of Ahmad.

First Order Logic

- Predicates

- Divide proposition into 2 parts:
- predicate: assertion about object
- argument: represent the object
- Example: To represent Bob teach Artificial Intelligence(AI),
  - teach (bob, AI)
  - teach is a ***predicate***, denoting relationship between arguments. The 1st letter must be in lower case.

First Order Logic

- Variables

- Represent general classes of objects or properties
- Written as symbols beginning **with upper case**.
- To capture the proposition Bob teach AI, we write:
  - teach (X,Y)
  - X = bob and Y = AI

First Order Logic

- <u>Function</u>

- Permits symbol to be used to represent function.
- A function denotes a mapping from entities of a set to a unique element of another set.
  - father(bob) = zakaria   mother(bob) = zaharah.
  - Can be also used within predicates. For example:
  - husband (father(bob), mother(bob)) = husband(zakaria,zaharah)

Predicate and function {8.2.2}

- There is a correspondence between
  - function, which return values
  - predicates, which are true or false

- Function: father_of(mary) = bill
- Predicate:father_of(marry,bill)

FOL: objects, relation, functions {8.1}

- First-order logic assumes the world contains
  - ✓ **Objects**: people, numbers, game, wars…
  - ✓ **Relation**: red, prime, bigger than, part of,…
  - ✓ **Function**: father of, one's best friend, plus…
- "one plus two equals three"

  Objects: ____________________

  Relations: ____________________

  Functions: ____________________
- "square neighboring the Wumpus are smelly"

  Objects: ____________________

  Relations: ____________________

  Functions: ____________________

Dr.Yanmei Zheng

FOL: objects, relation, functions {8.1}

- First-order logic assumes the world contains
  - ✓ Objects: people, numbers, game, wars…
  - ✓ Relation: red, prime, bigger than, part of,…
  - ✓ Function: father of, one's best friend, plus…
- "one plus two equals three"

  Objects: one, two, three, one plus two

  Relations: equals

  Functions: plus
- "square neighboring the Wumpus are smelly"

  Objects: Wumpus, square

  Relations: neighboring, smelly (properties)

  Functions: _ _

Dr.Yanmei Zheng

First Order Logic

- <u>Operations</u>
- PC uses the same operators found in P.L.
- Proposition: John likes Mary likes(john,mary)

  Bob likes Mary likes(bob,mary)
- 2 persons like Mary. To account for jealousy:
  - likes (X,Y) AND likes(Z,Y) implies NOT likes (X,Z)

    or
  - $likes(X,Y) \wedge likes(Z,Y) \rightarrow \neg likes(X,Z)$

Term, Sentences {8.2.3-5}

- Term
  - Constant, symbol, function symbol, or variable
  - At(x,cs): X is a variable, CS is a constant.
- atomic sentence
  - Predicate symbol with value true or false
  - Represents a relation between terms
  - At(x, CS) is an atom.
- complex sentence
  - Atom(s) joined together using logical connectives and/or quantifiers

Atomic sentences

Atomic sentence = predicate $(term_1,\ldots,term_n)$
or $term_1 = term_2$

Term = function $(term_1,\ldots,term_n)$
or constant or variable

- E.g., Brother(KingJohn,RichardTheLionheart)
- > (Length(LeftLegOf(Richard)), Length(LeftLegOf(KingJohn)))

Complex sentences

- Complex sentences are made from **atomic** sentences using connectives

$$\neg S, \qquad S_1 \wedge S_2, \qquad S_1 \vee S_2,$$
$$S_1 \Rightarrow S_2, \qquad S_1 \Leftrightarrow S_2$$

E.g. Sibling(KingJohn,Richard) $\Rightarrow$Sibling(Richard,KingJohn)

Syntax of FOL

Sentence $\rightarrow$ **AtomicSentence**
$|$ **(Sentence Connective Sentence)**
$|$ **Quantifier Variable , … Sentence**
$|$ **$\neg$Sentence**
AtomicSentence $\rightarrow$ **Predicate(Term, …)**
$|$ **Term = Term**
Term $\rightarrow$ **Function(Term, …)**
$|$ **Constant**
$|$ **Variable**
Connective $\rightarrow \Rightarrow | \wedge | V | \Leftrightarrow$
Quantifier $\rightarrow \forall | \exists$
Constant $\rightarrow$ $A \mid X_1 \mid$ **john** $|$ **…**
Variable $\rightarrow$ **a** $|$ **x** $|$ **s** $|$ **…**
Predicate $\rightarrow$ **Before** $|$ **HasColor** $|$ **Raining** $|$ **…**
Function $\rightarrow$ **Mother** $|$ **LeftLeg** $|$ **…**

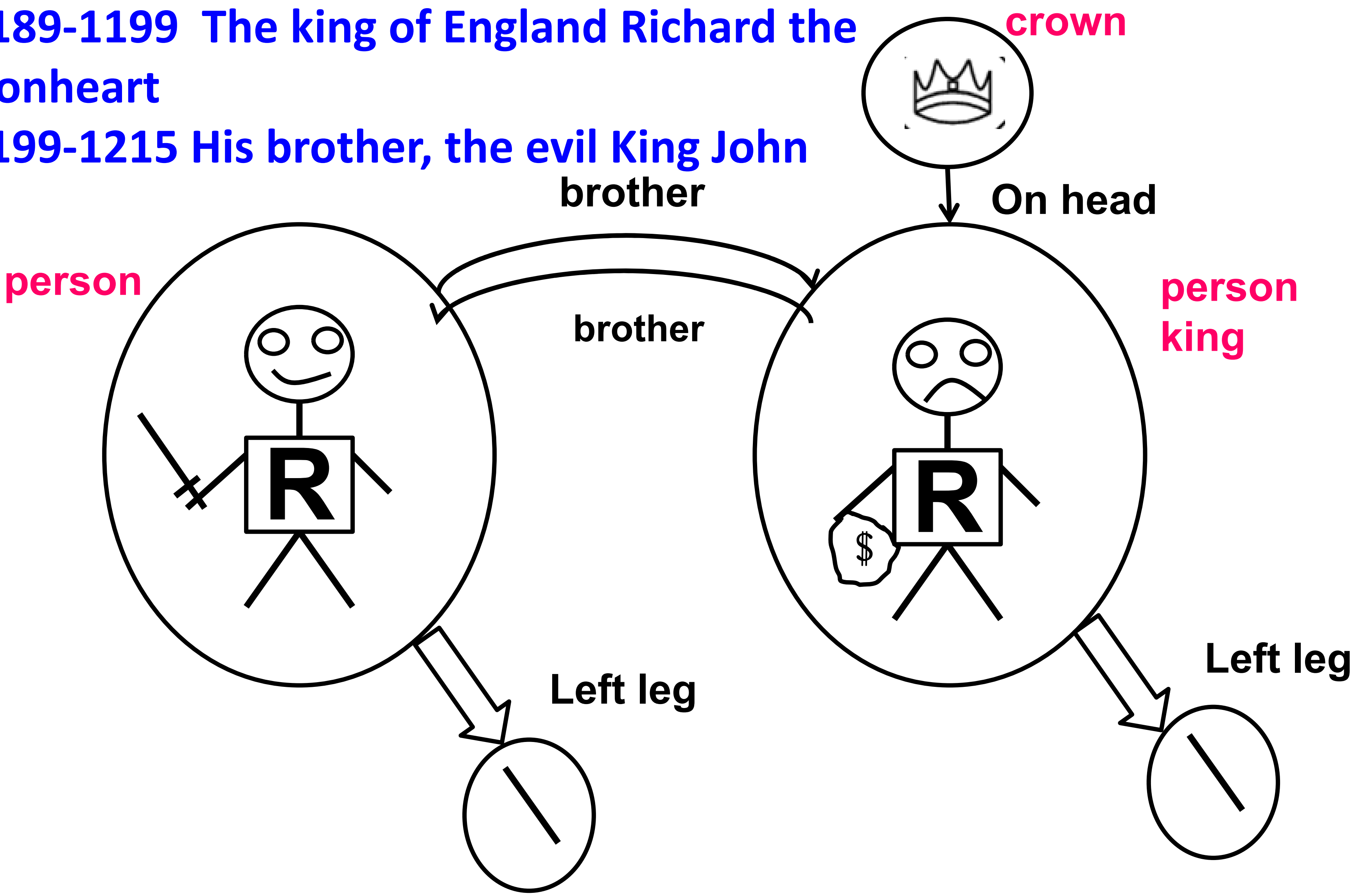
Syntax of Propositional Logic

Sentence $\rightarrow$ **AtomicSentence** $|$ **ComplexSenence**
AtomicSentence $\rightarrow$ **True** $|$ **False** $|$ **Symbol**
symbol $\rightarrow$ **P** $|$ **Q** $|$ **R** $|$ **…**
ComplexSenence $\rightarrow$ **$\neg$Sentence**
$|$ **(Sentence $\wedge$ Sentence)**
$|$ **(Sentence V Sentence)**
$|$ **(Sentence$\Rightarrow$Sentence)**
$|$ **(Sentence$\Leftrightarrow$Sentence)**

Models for FOL: Example

1189-1199 The king of England Richard the Lionheart

1199-1215 His brother, the evil King John



Quantifiers {8.2.6}

- Expressing sentences about collections of objects without enumeration (naming individuals)
  - All Computer Science (CS) students are clever.
  - Someone in the class is sleeping.
- Universal quantification (for all): □
  - □ <variables> <sentence>
- Existential quantification (there exists): □
  - □ < variables> < sentence>

Quantifiers {8.2.6}

- example:

  ✓ All Computer Science (CS) students are clever

  ✓ Someone in the class is sleeping.

Quantifiers {8.2.6}

- **example:**

  ✓ All Computer Science (CS) students are clever

  □ x (At(x, CS)□ Smart (x))

  ✓ Someone in the class is sleeping.

  □ x (Inclass(x) $\wedge$ Sleeping(x))

First Order Logic

- $\forall$ Indicates the expression is TRUE for all values of designated variable.

Example:

- $\forall$ X likes (X,mary)
- means for all values of X, the statement is true, everybody likes Mary

Universal quantification

- $\forall$<variables> <sentence>

  Everyone at HPU is smart:

  $\forall x\ At(x,HPU) \Rightarrow Smart(x)$
- $\forall x$ P is true in a model m iff P is true with x being each possible object in the model
- Suppose the object set is {John, Micheal },

  □ x (At(x, CS)□ Smart (x))

  is equivalent to the conjunction of instantiations of

  (At(John, CS) □ Smart(John)) $\wedge$

  (At(Micheal, CS) □ Smart(Micheal))

Dr.Yanmei Zheng

Universal quantification

- Roughly speaking, equivalent to the **conjunction of instantiations** of *P*

  At(KingJohn, HPU) $\Rightarrow$ Smart(KingJohn) $\wedge$
  At(Richard, HPU) $\Rightarrow$ Smart(Richard) $\wedge$
  At(HPU,HPU) $\Rightarrow$ Smart(HPU) $\wedge$ …

Question: Is it possible to represent the sentence in the following way

$\forall$x At(x,HPU) $\wedge$ Smart(x)

A common mistake to avoid

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

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

  $\forall$x (At(x,HPU) $\wedge$ Smart(x)) $\Leftrightarrow$

  (At(Zhang,HPU) $\wedge$ Smart(Zhang)) $\wedge$

  (At(Li,HPU) $\wedge$ Smart(Li)) $\wedge$

  (At(Wang,HPU) $\wedge$ Smart(Wang)) $\wedge$ ....

  means "Everyone is at HPU and everyone is smart"

  Too strong!

Predicate Calculus (PC)

- ∃ indicates the expression is **TRUE** for some values of the variable; at least one value exist that makes the statement true:
- ∃ X likes (X,mary)

Existential quantification

- ∃<variables> <sentence>
- Someone at HPU is smart:

  $\exists x\ (At(x,HPU) \wedge Smart(x))$

- ∃x P is true in a model m iff P is true with x being some possible object in the model

Existential quantification

- Roughly speaking, equivalent to the **disjunction** of **instantiations** of P

$\mathrm{At}(\mathrm{KingJohn},\mathrm{HPU}) \wedge \mathrm{Smart}(\mathrm{KingJohn}) \vee$
$\mathrm{At}(\mathrm{Richard},\mathrm{HPU}) \wedge \mathrm{Smart}(\mathrm{Richard}) \vee$
$\mathrm{At}(\mathrm{HPU},\mathrm{HPU}) \wedge \mathrm{Smart}(\mathrm{HPU}) \vee \ldots\ldots$

?? $\exists x\ \mathrm{At}(x,\mathrm{HPU}) \Rightarrow \mathrm{Smart}(x)$

Another common mistake

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

  $\exists x$ At(x,HPU) $\Rightarrow$ Smart(x) $\Leftrightarrow$

  (At(Zhang, HPU) $\Rightarrow$ Smart(Zhang))$\vee$
  (At(Li, HPU) $\Rightarrow$ Smart(Li)) $\vee$
  (At(Wang, HPU) $\Rightarrow$ Smart(Wang) $\vee$…
- The sentence is true if there is anyone who is not at HPU!
- Too weak!

Predicate Calculus (PC)

- Parentheses are used to indicate the scope of quantification
- $\forall$ X (likes(X,mary) $\wedge$ nice(mary) $\rightarrow$ nice (X))

  Determine all instances of X who like Mary and if Mary is nice, then it is implied that those who like Mary are also nice.

Properties of quantifiers

- $\forall x\ \forall y$ is the same as $\forall y\ \forall x$
- $\exists x\ \exists y$ is the same as $\exists y\ \exists x$
- $\exists x\ \forall y$ is not the same as $\forall y\ \exists x$
- $\exists x\ \forall y$ Loves(x,y)
  - ✓ "There is a person who loves everyone in the world"
- $\forall y\ \exists x$ Loves(x,y)
  - ✓ "Everyone in the world is loved by at least one person"

Properties of quantifiers

- **Quantifier duality:** Each can be expressed using the other

$\forall x$ Likes(x,IceCream) $\Leftrightarrow$ $\neg\exists x\ \neg$Likes(x,IceCream)

$\exists x$ Likes(x,Broccoli) $\Leftrightarrow$ $\neg\forall x\ \neg$Likes(x,Broccoli)

Equality

- $term_1 = term_2$ is true under a given interpretation if and only if $term_1$ and $term_2$ refer to the same object
- E.g., definition of Sibling in terms of Parent:

$$\forall x,y\ Sibling(x,y) \Leftrightarrow [\neg(x = y) \wedge \exists m,f\ \neg(m = f) \wedge Parent(m,x) \wedge Parent(f,x) \wedge Parent(m,y) \wedge Parent(f,y)]$$

Equality {8.2.7}

- use the equality symbol to signify that two terms refer to the same object. For example,

  Father (John)=Henry

- To say that Richard has at least two brothers, we would write

  □ x, y Brother (x, Richard) ∧ Brother(y， Richard) ∧ ¬(x=y)

- The notation x ≠ y is sometimes used as an

Abbreviation for ¬((x=y).

Equality {8.2.7}

- Brother(John, Richard)∧Brother(Geoffrey, Richard)
- Can it express "Richard's brothers are John and Geoffrey "?
- The correct translation:

  $\mathit{Brother}(\mathit{John}, \mathit{Richard})$

  $\land\ \mathit{Brother}(\mathit{Geoffrey}, \mathit{Richard})$

  $\land\ \mathit{John} \neq \mathit{Geoffrey}$

  $\land\ \forall X\ \mathit{Brother}(x, \mathit{Richard}) \Longrightarrow (x=\mathit{John} \lor x=\mathit{Geoffre\ y})$.

Syntax of FOL: Basic elements

Constants:	KingJohn, 2, HPU,...
Predicates:	Brother, >,...
Functions:	Sqrt, LeftLegOf,...
Variables:	x, y, a, b,...
Connectives:	$\neg, \Rightarrow, \wedge, \vee, \Leftrightarrow$
Equality:	=
Quantifiers:	$\forall, \exists$

An alternative semantics(8.2.8)

- This seems much more cumbersome than the corresponding natural language expression.
- Humans may make mistakes in translating their knowledge into first-order logic, resulting in unintuitive behaviors from logical reasoning systems that use the knowledge.
- Can we devise a semantics that allows a more straightforward logical expression?

An alternative semantics {8.2.8}

- Database semantics
  - Unique-names assumption: we insist that every constant symbol refer to a distinct object.
  - The closed-world assumption: we assume that atomic sentences not known to be true are in fact false.
  - Domain closure: each model contains no more domain elements than those named by the constant symbols.
- Distinguish the database semantics from the standard semantics of first-order logic.
- Brother(John, Richard) $\wedge$ Brother(Geoffrey, Richard) state that Richard’s two brothers are John and Geoffrey"

Assertions and Queries in FOL

- Sentences are added to a knowledge base using **TELL**:
  - TELL(KB, *King(John)*).
  - TELL(KB, *Person(Richard)*).
  - TELL(KB, $\forall\, x King(x) \Longrightarrow Person(x)$)
- Ask questions of the knowledge base using **ASK**.
  - ASK(KB, *King(John)*).
- Want to know what value of x makes the sentence true, need a different function, **ASKVARS**
  - ASKVARS(*KB, Person(x)).*

The kinship domain

The kinship domain

- **Unary predicates:** Male, Female
- **Binary predicates:** Parent, Sibling , Brother , Sister , Child , Daughter , Son , Spouse , Wife , Husband , Grandparent , Grandchild , Cousin , Aunt , Uncle
- **Functions:** Mother , Father

The kinship domain

- Some axioms are definitions:
  - One’s mother is one’s female parent:

  □ m, c Mother(c)=m□ Female(m)∧Parent(m, c)

- One’s husband is one’s male spouse :
  - □ ┌ , h Husband(h, w) □ Male(h) ∧ Spouse(h, w)

- Male and female are disjoint categories:
  - □ x Male(x) □ ¬Female(x)

The kinship domain

- Parent and child are inverse relations
  - □ p, c Parent(p, c)□ Child(c, p)
- A grandparent is a parent of one’s parent:
  - □ g, c Grandparent(g, c) □ □ p Parent(g, p)∧Parent(p, c)
- A sibling is another child of one’s parents:
  - □ x, y Sibling(x, y) □ x≠y ∧ □ p Parent(p,x) ∧ Parent(p, y)

The kinship domain

- Not all axioms are definitions.
- Some predicates have no complete definition because we do not know enough to characterize them fully.
  - □ x person(x)□ ...
- Fortunately, first-order logic allows us to make use the *Person* predicate without completely defining it. Instead, we can write partial specifications of properties that every person has and properties that make something a person:
  - □ x Person(x) □ ...
  - □ x ... □ Person(x)

The kinship domain

- Axioms can also be “just plain facts”
  - Male(Jim), Spouse(Jim, Laura)
- Often, one finds that expected answers are not forthcoming.
  - for example, from Male(George) and Spouse(George, Laura), one expects Female(Laura), but this does not follow from the axioms given earlier.

The kinship domain

- **Not all logical sentences about a domain are axioms.**
  - **Some are theorems, that is, they are entailed by axioms.**
  - **From axioms:**□ *x, y Sibling(x, y)*□ *x≠y* ⋀□ *p Parent(p, x)* ⋀*Parent(p, y)*
  - **we can get the theorem:** □ *x, y Sibling(x, y)* □ *Sibling(y, x)*.
  - **From a purely logical point of view, a knowledge base need contain only axioms and no theorems.**
  - **From a practical point of view, theorems are essential to reduce the computational cost of deriving new sentences.**

{8.3}

I married a widow who had a grown-up daughter. My father, who visited us quite often, fell in love with my step-daughter and married her. Hence, my father became my son-in-law, and my step-daughter became my mother. Some months later, my wife gave birth to a son, who became the brother-in-law of my father as well as my uncle. The wife of my father, that is my stepdaughter, also had a son. Thereby, I got a brother and at the same time a grandson. My wife is my grandmother, since she is my mother's mother. Hence, I am my wife's husband and at the same time her step-grandson; in other words, I am my own grandfather.

- objects? relations?

Example model {8.3}

- **Objects: I, my father, widow, daughter**
- **Relation:**
  - marriage relation
  - parent relation
  - grandparent relation
- **marriage relation:**
  - {<I, widow>, <my father, daughter>}
  - then marriage(I, widow)is true
  - marriage(my father, widow)is false

Natural numbers {8.3.3}

- Natural numbers are recursively defined as
  - NatNum(0)
  - □ n NatNum(n)□ NatNum(S(n))
  - Successor
- We need axioms to constrain the successor function
  - □ n 0≠S(n)
  - □ n, m m≠n □ S(m)≠S(n)
- Define addition in terms of the successor function
  - □ m NatNum(m)□ +(0, m) = m
  - □ m, n NatNum(m)∧NatNum(n)□ +(S(m),n)= S(+(m,n))
  - Prefix

Natural numbers {8.3.3}

- **Infix** notation, and write S(n) as n+ 1, then
  - □ m, n NatNum(m)∧NatNum(n)□ +(S(m), n)=S(+(m, n))

  Becomes

  □ m, n NatNum(m) ∧ NatNum(n)□ (m+1)+n=(m+n)+1
- The use of infix notation is an example of **syntactic sugar**, that is , an extension to or abbreviation of the standard syntax that does not change the semantics.

Natural numbers {8.3.3}

- Once we have addition, it is straightforward to define multiplication as repeated addition, exponentiation as repeated multiplication, integer division and reminders, prime numbers, and so on.
- Thus, the whole of number theory(including cryptography) can be built up from one constant, one function, one predicate and four axioms.

Sets{8.3.3}

We use the normal vocabulary of set theory as syntactic sugar.

- The **empty set** is a constant written as{}
- One **unary predicate**, Set, which is true of sets.
- The **binary predicates** are $x \in s$ (x is a member of set s), and$s_1 \subseteq s_2$ (set s_1 is a subset, not necessarily proper, of set s_2).
- The binary functions are:
  - $s_1 \cap s_2$ the intersection of two set
  - $s_1 \cup s_2$ the union of two set
  - $\{x \mid s\}$ the set resulting from adjoining element x to set s

Sets{8.3.3}

One possible set of axioms :

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

$$\forall s\ Set(s) \Leftrightarrow (s=\{\}) \vee (\exists x, s_2\ Set(s_2) \wedge s=\{x \mid s_2\})$$

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

$$\neg \exists x, s \{x \mid s\}=\{\ \}$$

- Adjoining an element already in the set has no effect

$$\forall x,s\ x \in s \Leftrightarrow s=\{x \mid s\}$$

Sets{8.3.3}

- The only members of a set are the elements that were adjoined into it. We express this recursively, saying that *x* is a member of *s* if and only if *s* is equal to some set s_2 adjoined with some element *y*, where either *y* is the same as *x* or x is a member of s_2

$$\forall x,s\ x\in s \Leftrightarrow \exists y, s_2 (s=\{y \mid s_2\} \wedge (x=y \vee x\in s_2))$$

- A set is a subset of another set if and only if all of the first set's members are members of the second set

$$\forall s_1, s_2\ \ s_1 \subseteq s_2 \Leftrightarrow (\forall x\ x\in s_1 \Rightarrow x \in s_2)$$

Sets{8.3.3}

- Two sets are equal if and only if each is a subset of the other

$$\forall\ s_1, s_2\ \ s_1 = s_2 \Leftrightarrow (s_1 \subseteq s_2 \wedge s_2 \subseteq s_1)$$

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

$$\forall\ x, s_1, s_2\ \ x \in (s_1 \cap s_2) \Leftrightarrow (x \in s_1 \wedge x \in s_2)$$

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

$$\forall\ x, s_1, s_2\ \ x \in (s_1 \cup s_2) \Leftrightarrow (x \in s_1 \vee x \in s_2)$$

Lists {8.3.3}

- Lists are similar to set. The difference are that list are ordered and the same element can appear more than once in a list.

Interacting with FOL KBs

- Suppose a wumpus-world agent is using an FOL KB and perceives a smell and a breeze (but no glitter) at t=5:

  Tell(KB,Percept([Smell,Breeze,None],5))

  Ask(KB,∃a BestAction(a,5))

  i.e., does the KB entail some best action at t=5?

- Answer: Yes, {a/Shoot} ← substitution (binding list)

Interacting with FOL KBs

- Given a sentence S and a substitution σ,
- Sσ denotes the result of plugging σ into S; e.g.,

  S = Smarter(x,y)

  σ = {x/Hillary,y/Bill}

  Sσ = Smarter(Hillary,Bill)

- Ask(KB,S) returns some/all σ such that KB ╞ σ

Knowledge base for the wumpus world

- **Perception**
  - ✓ $\forall t,s,b$ **Percept([s,b,Glitter],t)** $\Rightarrow$ **Glitter(t)**
- **Reflex**
  - ✓ $\forall t$ **Glitter(t)** $\Rightarrow$ **BestAction(Grab,t)**

Deducing hidden properties

- $\forall$x,y,a,b *Adjacent*([x,y],[a,b]) $\Leftrightarrow$
  [a,b] $\in$ {[x+1,y], [x-1,y],[x,y+1],[x,y-1]}

  Properties of squares:
- $\forall$s,t *At*(Agent,s,t) $\wedge$ Breeze(t) $\Rightarrow$ Breezy(s)

Squares are breezy near a pit:

- ✓ Diagnostic rule---infer cause from effect

  $\forall$s Breezy(s) $\Rightarrow$ $\exists$ r Adjacent(r,s) $\wedge$ Pit(r)

- ✓ Causal rule---infer effect from cause

  $\forall$r Pit(r) $\Rightarrow$ [$\forall$s Adjacent(r,s) $\Rightarrow$ Breezy(s)]

Reasoning with logic

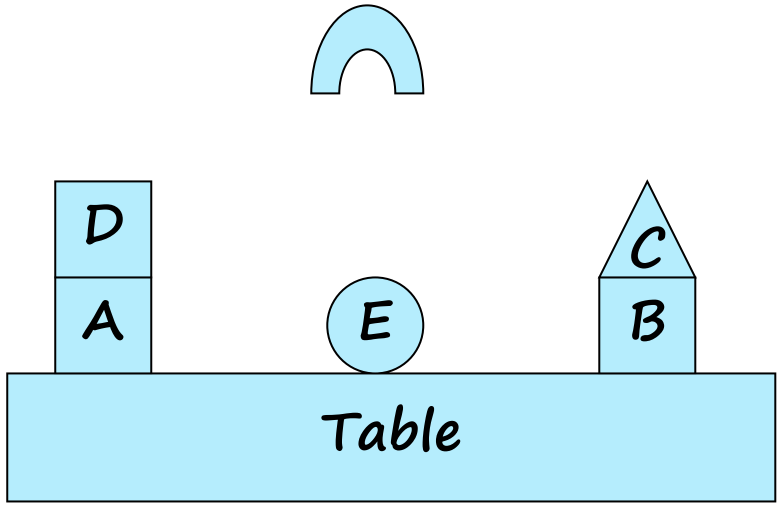
- PC can provide reasoning capability to intelligent system
- Reasoning requires the ability to infer conclusions from available facts.
- One simple form of inference is modus ponens
  - IF A is true

    AND $A \rightarrow B$ is true

    THEN B is true

Reasoning with logic

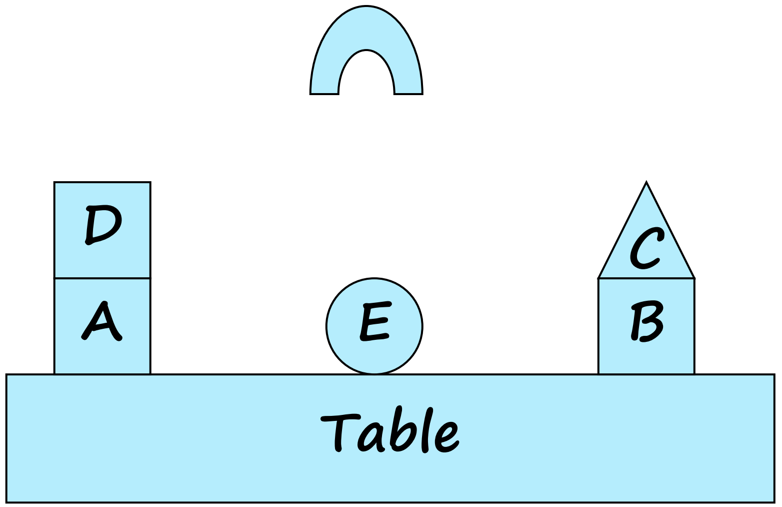
- **Robot Control Example**
- **The function of the robot is move a specified block to some specified location**



Reasoning with logic

Robot Control Example

- Description of the block world using PC using the following logical assertions:
  1. cube(a), cube(b), cube(d), pyramid(c), sphere(e), hand(hand), table (table1)
  2. on(a,table1), on(b,table1), on(d,a), on(c,b)
  3. holding(hand,nothing)



Reasoning with logic

- <u>Robot Control Example</u>

- The goal might be to put some block on other block, for example put block b onto block a:

  put_on(b,a)

- To accomplish this the robot need to obtain block b and make certain that block a is clear:

  $\text{hand\_holding}(b) \wedge \text{clear}(a) \rightarrow \text{put\_on}(b,a)$

- To move any block, in variable form:

  $\forall X\ \exists Y\ (\text{hand\_holding}(X) \wedge \text{clear}(Y)\ \text{Ynot} = c\ \text{Ynot}=e \rightarrow \text{put\_on}(X,Y))$

  where X is the block to be move and Y is the target block

Reasoning with logic

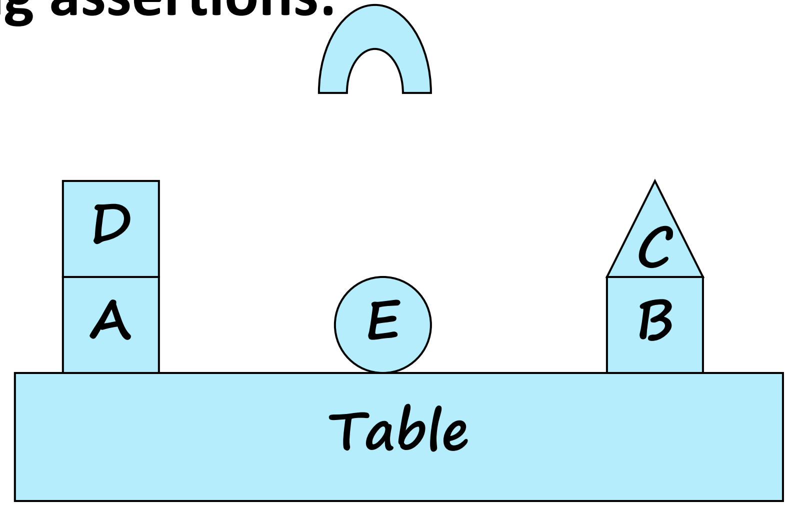
- Robot Control Example

- One of the robot task when instructed to pick up and move some blocks is to determine if it is clear.
- If not clear, need to remove any item on the block:

  $\forall X (\neg\exists Y \, on(Y,X) \rightarrow clear(X))$

  For all X, X is clear if there does not exist a Y such that Y is on X, would produce the following assertions:

  clear(c), clear(d)

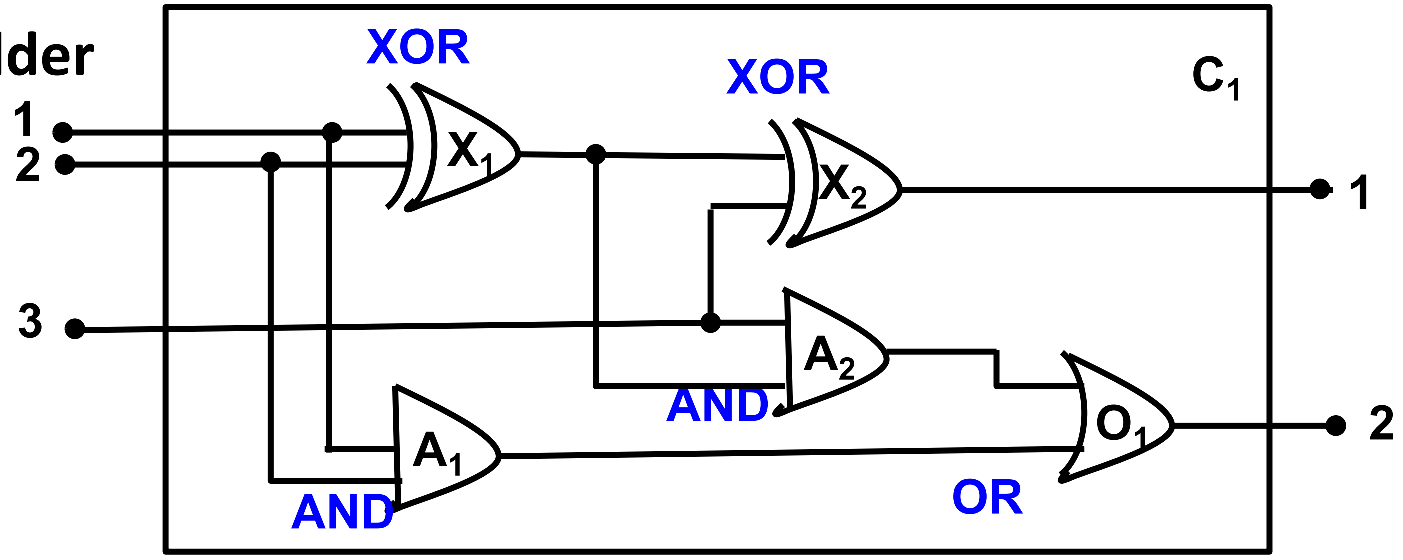


Knowledge engineering in FOL

- Identify the task
- Assemble the relevant knowledge
- Decide on a vocabulary of predicates, functions, and constants
- Encode general knowledge about the domain
- Encode a description of the specific problem instance
- Pose queries to the inference procedure and get answers
- Debug the knowledge base

The electronic circuits domain

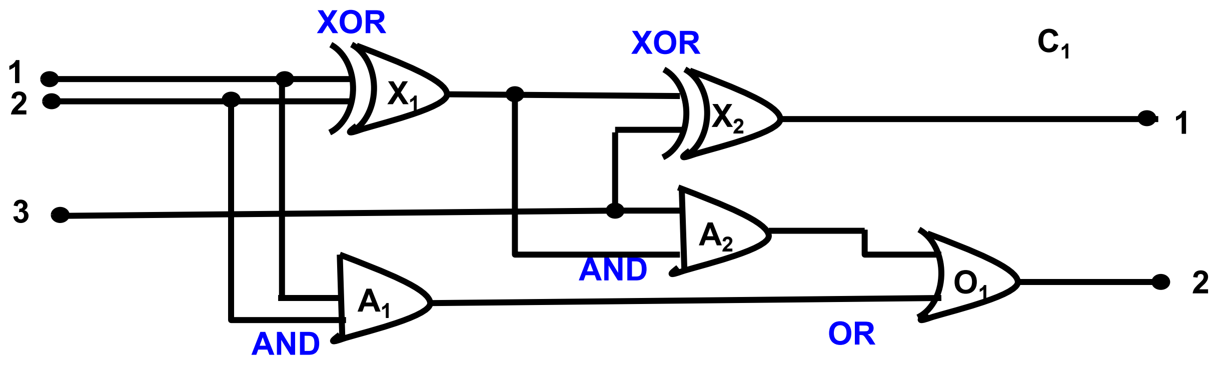
One-bit full adder



1. The first two inputs are the two bits to be added
2. The third inputs is a carry bit.
3. The first output is the sum
4. The second output is a carry bit for the next adder.
5. The circuit contains two XOR gates, two AND gates, and one OR gate.

The electronic circuits domain

1. **Identify the task**
   - ✓ Does the circuit actually add properly? (circuit verification)
2. **Assemble the relevant knowledge**
   - ✓ Composed of wires and gates; Types of gates (AND, OR, XOR, NOT)
   - ✓ Irrelevant: size, shape, color, cost of gates



Dr.Yanmei Zheng

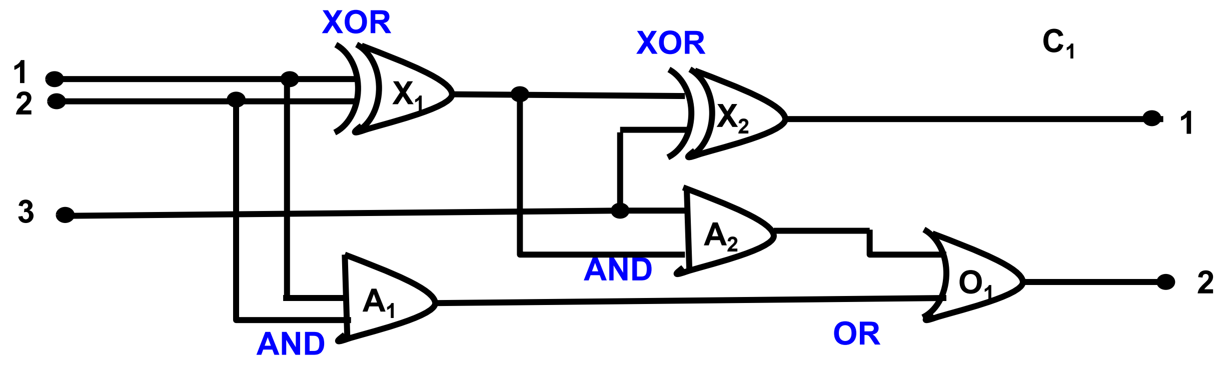
The electronic circuits domain

3. Decide on a vocabulary， choose functions, predicates, and constants to represent them.

Gate(X_1)

Type(X_1) = XOR   Type(X_1, XOR)   XOR(X_1)

Circuit(C_1)

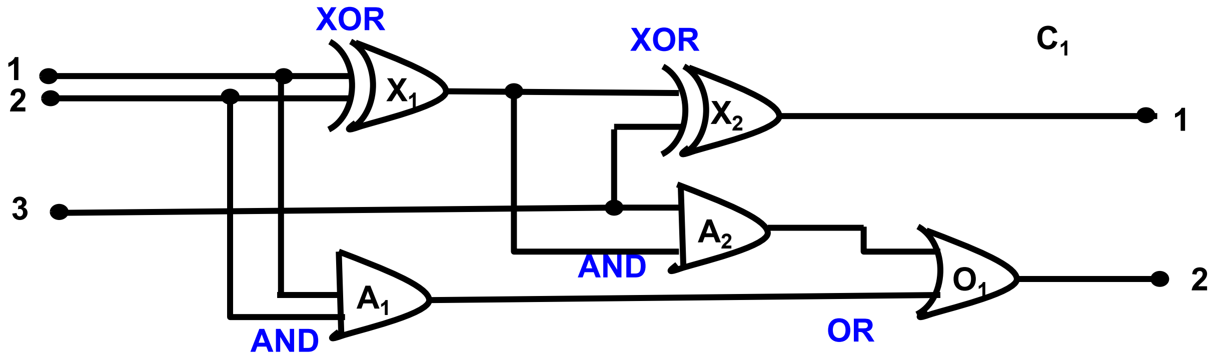


Dr.Yanmei Zheng

The electronic circuits domain

3. Decide on a vocabulary, **choose functions, predicates, and constants to represent them.**
   - ✓ *Terminal(x)* identify a **terminal.**
   - ✓ *In(1,X_1)* denote the first input terminal for gate X_1

     Arity(c,i,j) says the circuit *c* has input *i* and output terminals *j* .
   - ✓ *Connected(Out(1,X_1), In(1,X_2))*
   - ✓ *Signal(t)* denotes the signal value for the terminal *t* ~~On(t)~~



Dr.Yanmei Zheng

The electronic circuits domain

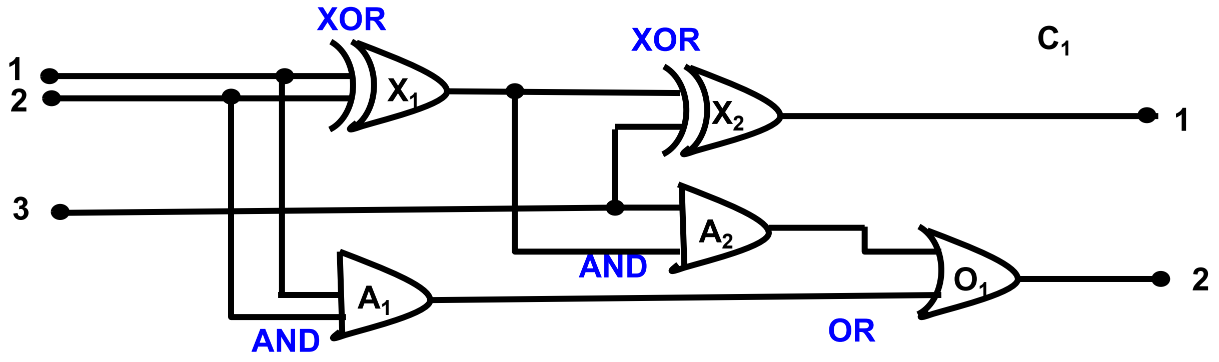
4. **Encode general knowledge of the domain**

- ✓ If two terminals are connected, then they have the same signal:

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

- ✓ The signal at every terminal is either 1 or 0:

$\forall t\ Terminal(t) \Rightarrow Signal(t) = 1 \lor Signal(t) = 0$



Dr.Yanmei Zheng

The electronic circuits domain

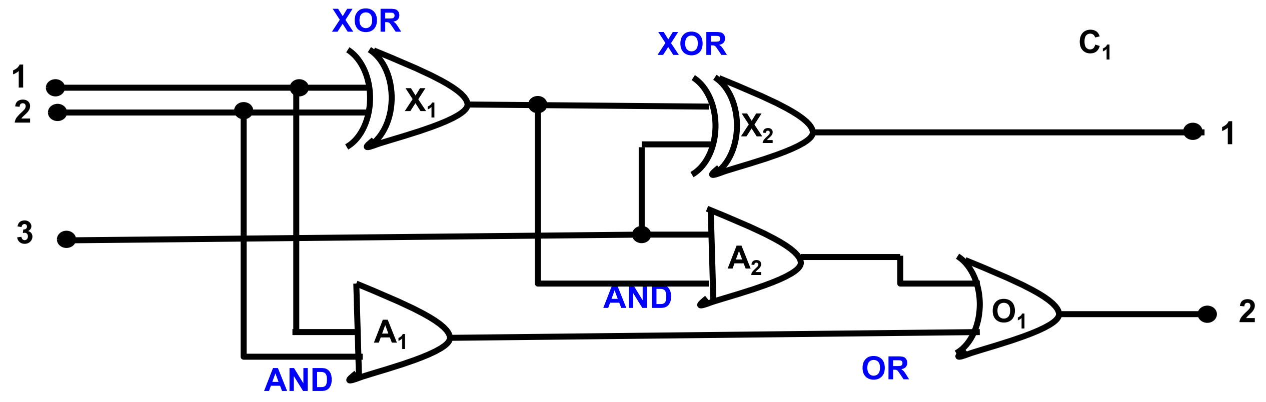
4. Encode general knowledge of the domain

✓ Connected is commutative:

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

✓ There are four types of gates:

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



Dr.Yanmei Zheng

The electronic circuits domain

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

- ✓ An AND gate's output is 0 if and only if any of its inputs is 0:

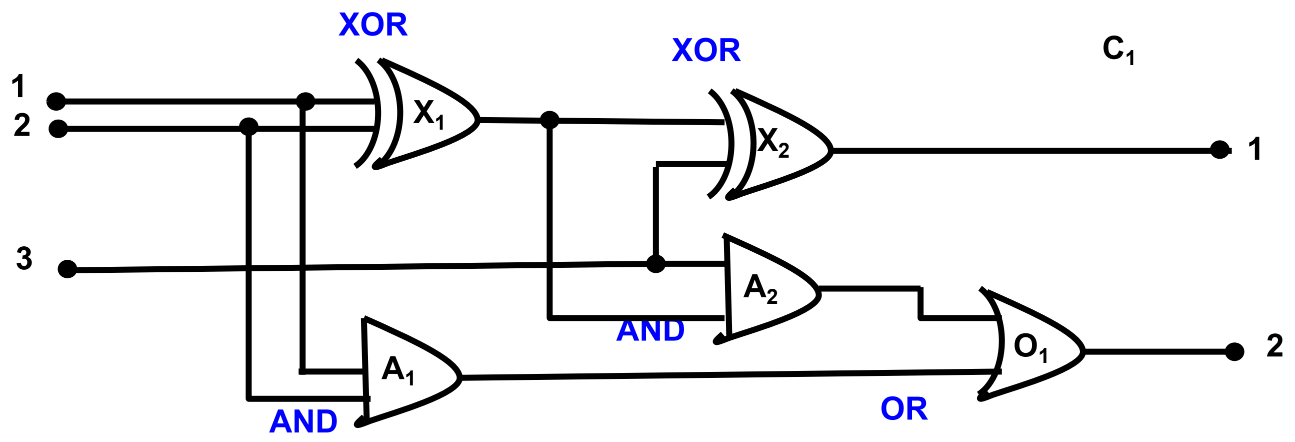
$\forall g\ Gate(g) \wedge Type(g) = AND \Rightarrow$

$Signal(Out(1,g)) = 0 \Leftrightarrow \exists n\ Signal(In(n,g)) = 0$

- ✓ An OR gate's output is 1 if and only if any of its inputs is 1:

$\forall g\ Gate(g) \wedge Type(g) = OR \Rightarrow$

$Signal(Out(1,g)) = 1 \Leftrightarrow \exists n\ Signal(In(n,g)) = 1$



Dr.Yanmei Zheng

The electronic circuits domain

4. Encode general knowledge of the domain

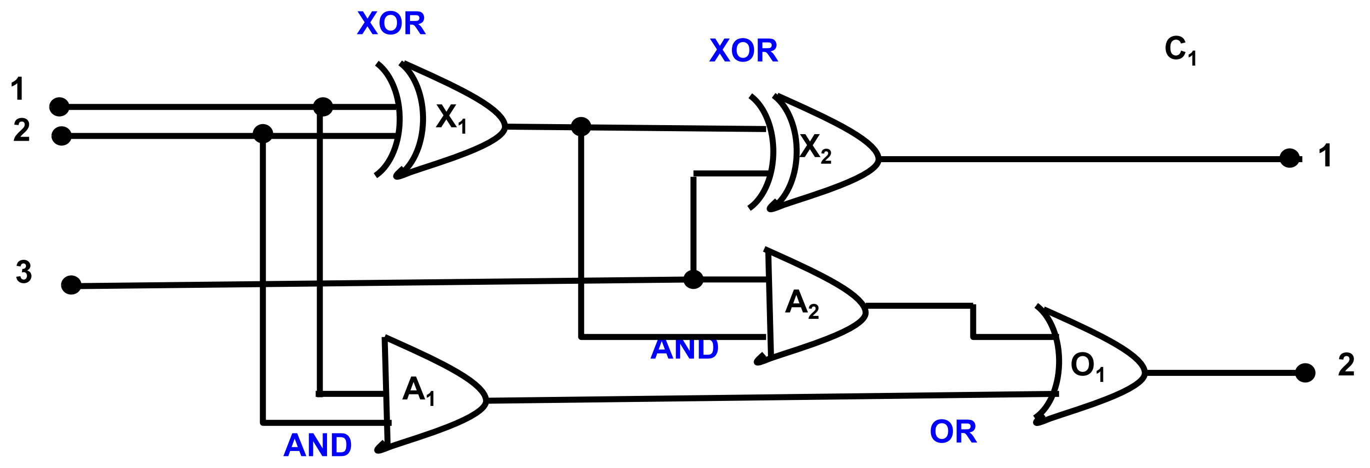
✓ An XOR gate's outputs is 1 if and only if its inputs are different

$\forall g\ Gate(g) \land Type(g) = XOR \Rightarrow$

$Signal(Out(1,g)) = 1 \Leftrightarrow Signal(In(1,g)) \neq Signal(In(2,g))$

✓ An NOT gate's output is different from its input

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



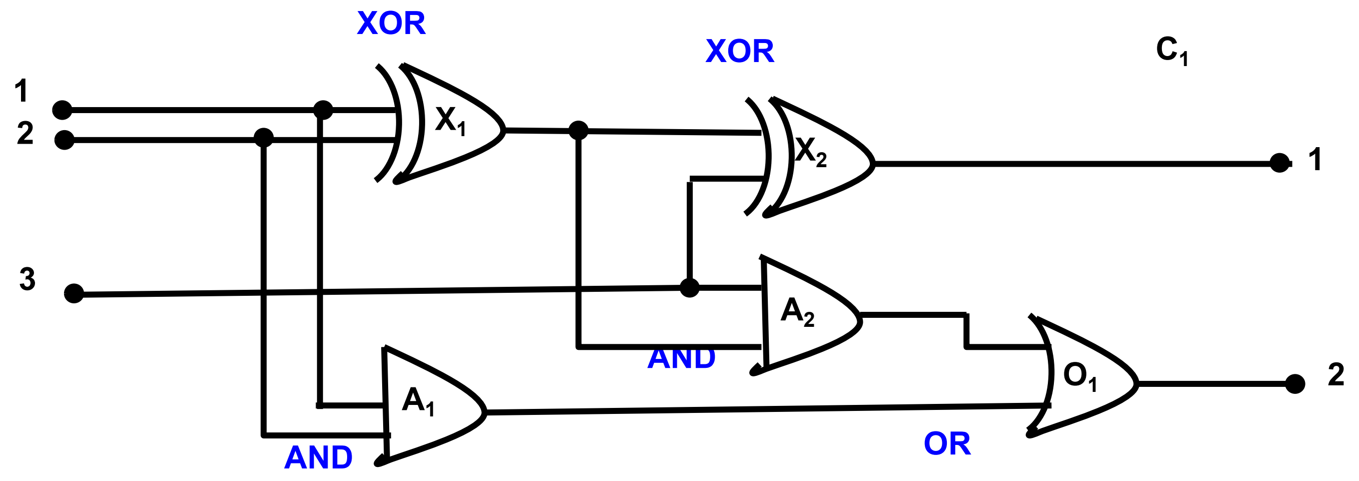
The electronic circuits domain

4. Encode general knowledge of the domain

✓ The gates(except for NOT) have two inputs and one output.

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

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



Dr.Yanmei Zheng

The electronic circuits domain

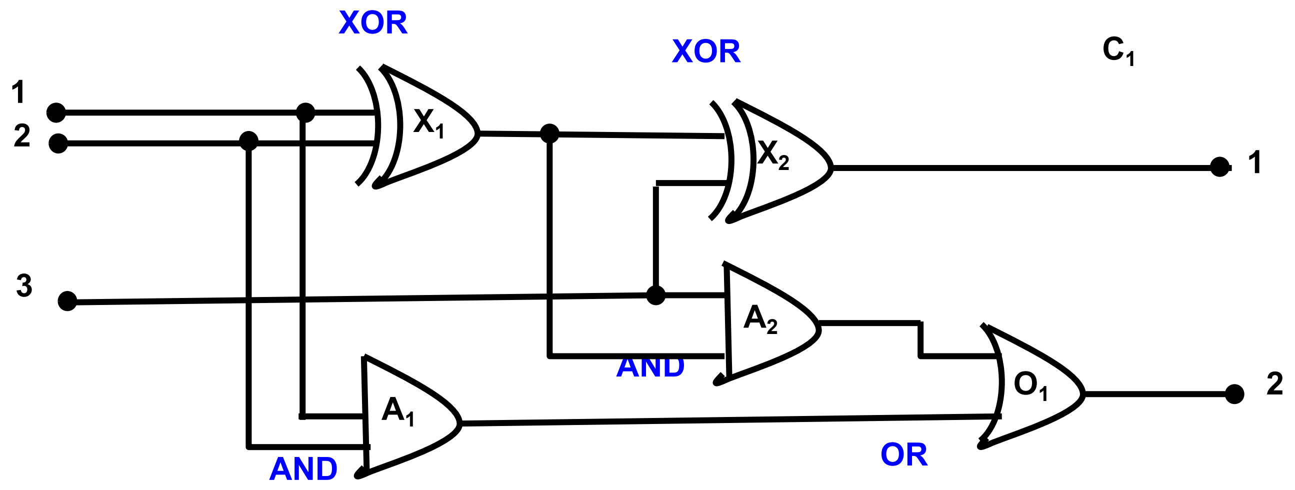
4. Encode general knowledge of the domain

✓ A circuit has terminals, up to its input and output urity, and nothing beyond its arity:

$\forall c,i,j \leq Circuit(c) \wedge Arity(c,i,j) \Rightarrow$

$\forall n(n \leq i \Rightarrow Termimal(In(c,n))) \wedge (n>i \Rightarrow In(c,n)=Nothing) \wedge$

$\forall n(n \quad j \Rightarrow Termimal(Out(c,n))) \wedge (n>j \Rightarrow Out(c,n)=Nothing)$



Dr.Yanmei Zheng

The electronic circuits domain

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

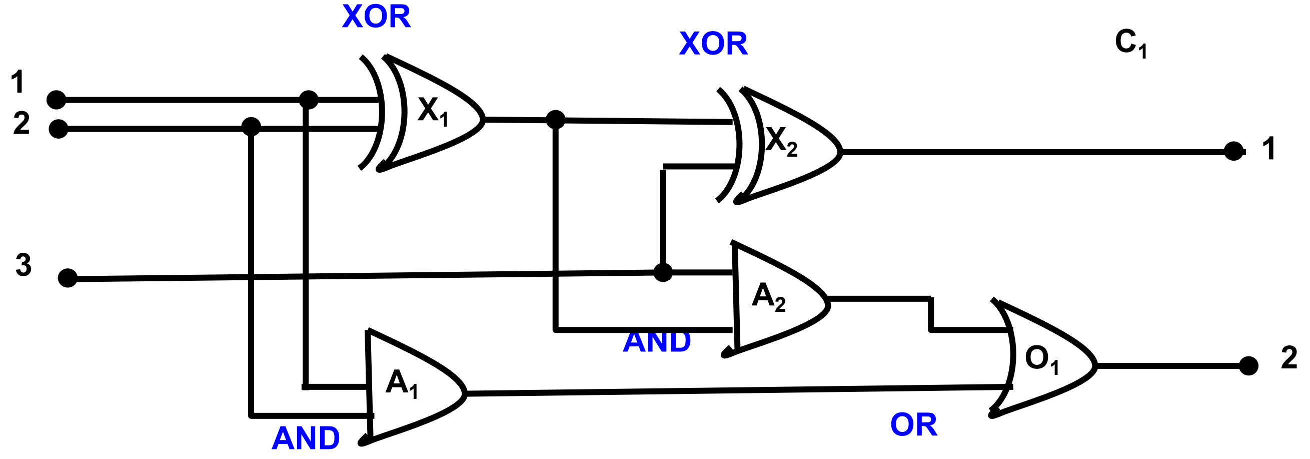
✓ **Gates, terminals, signals, gate types, and Nothing are all distinct.**

∀g,t Gata(g) ∧ Terminal(t) ⇒

$g \neq t \neq 1 \neq 0 \neq OR \neq AND \neq XOR \neq NOT \neq Nothing$

✓ **Gates are circuits.**

∀g Gate(g)⇒Circuit(g)



Dr.Yanmei Zheng

The electronic circuits domain

5. Encode the specific problem instance

✓ **We categorize the circuit and its component gates :**

$Circuit(C_1) \wedge Arity(C_1, 3, 2)$

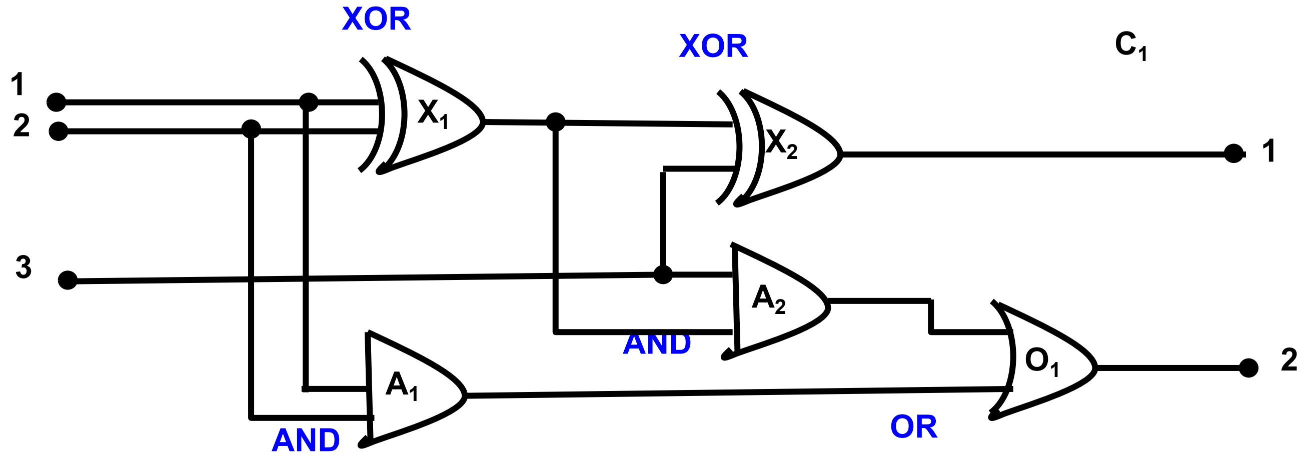
$Gate(X_1) \wedge Type(X_1) = XOR$

$Gate(X_2) \wedge Type(X_2) = XOR$

$Gate(A_1) \wedge Type(A_1) = AND$

$Gate(A_2) \wedge Type(A_2) = AND$

$Gate(O_1) \wedge Type(O_1) = OR$



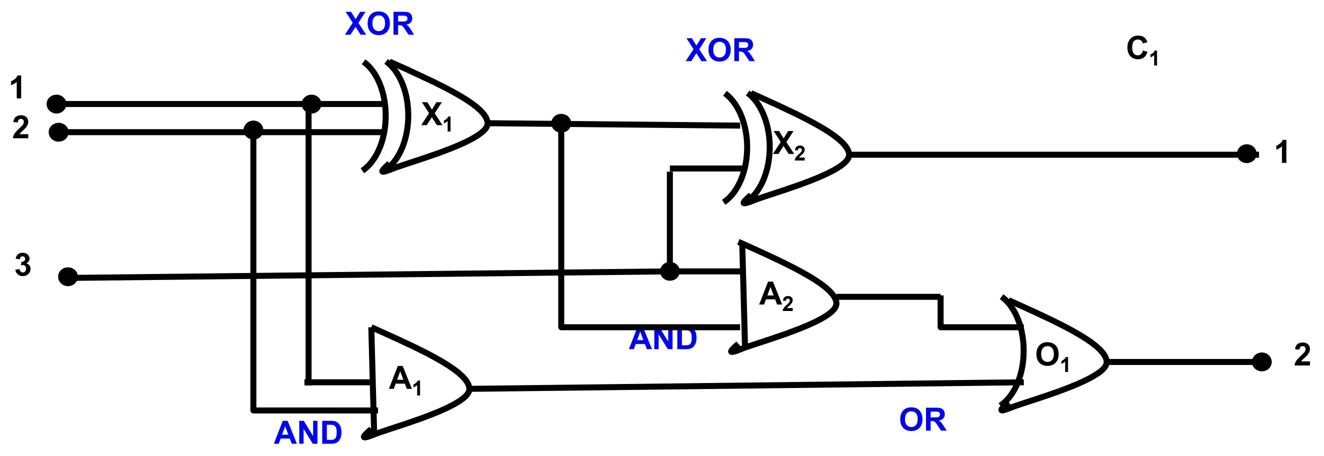
Dr.Yanmei Zheng

The electronic circuits domain

5. Encode the specific problem instance

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))



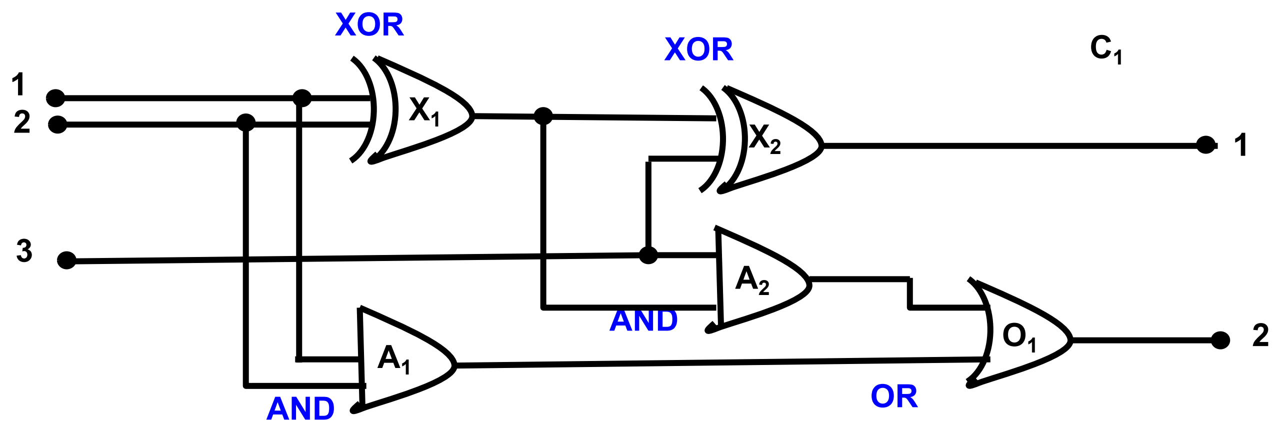
Dr.Yanmei Zheng

The electronic circuits domain

6. 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?

$$\exists i_1,i_2,i_3\ Signal(In(1, C_1)) = i_1 \wedge Signal(In(2,C_1)) = i_2 \wedge Signal(In(3,C_1)) = i_3 \wedge Signal(Out(1,C_1)) = 0 \wedge Signal(Out(2,C_1)) = 1$$



Dr.Yanmei Zheng

The electronic circuits domain

6. Pose queries to the inference procedure

What are the possible sets of values of all the terminals

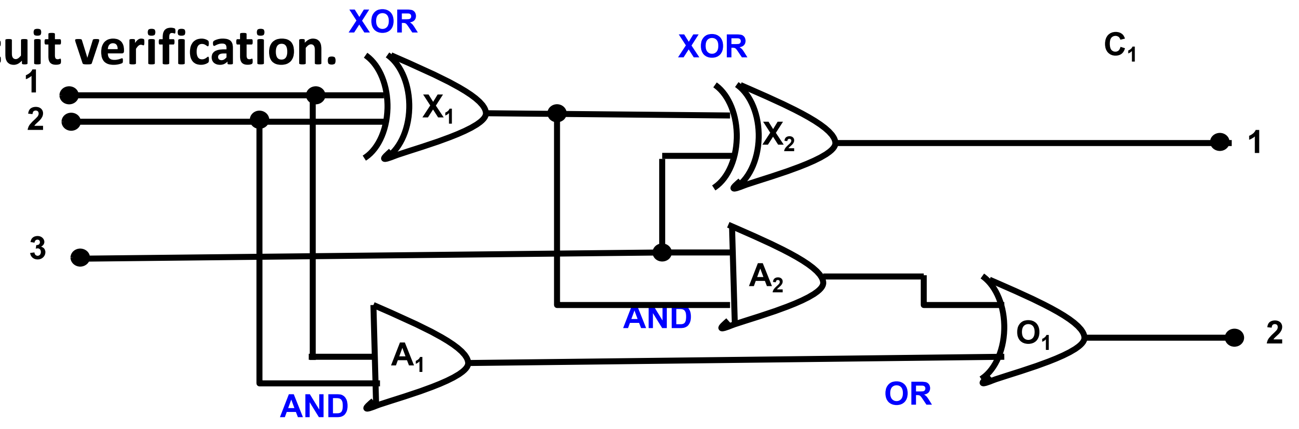
for the adder circuit? $\exists i_1,i_2,i_3,o_1,o_2$ Signal(In(1, C_1)) = $i_1 \wedge$ Signal(In(2,C_1)) =

$i_2 \wedge$ Signal(In(3,C_1)) = $i_3 \wedge$ Signal(Out(1,C_1)) =

$o_1 \wedge$ Signal(Out(2,C_1)) = o_2

This final query will return a complete input-output table for the device, which can be used to check that is does in fact add its inputs correctly.

A simple example of circuit verification.

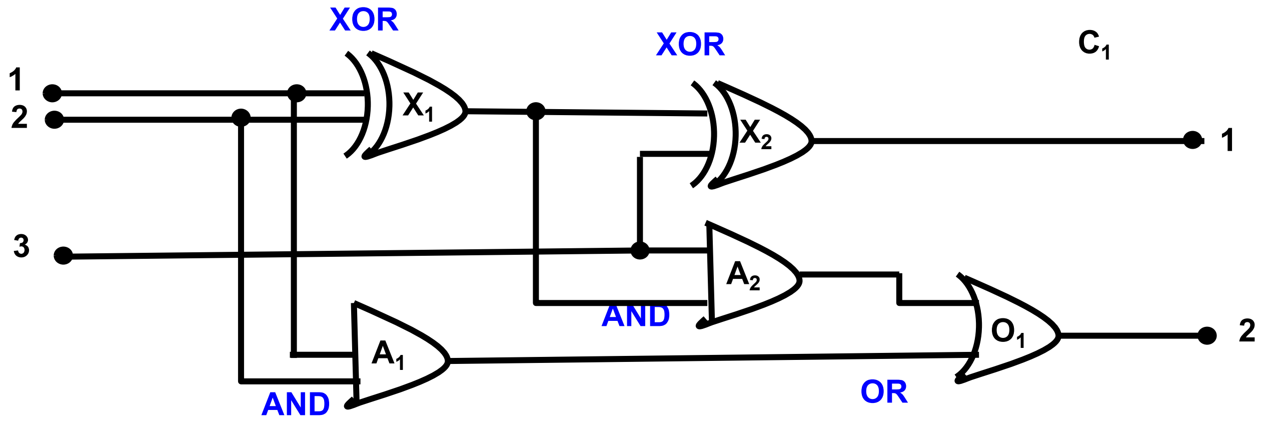


Dr.Yanmei Zheng

The electronic circuits domain

7. Debug the knowledge base

May have omitted assertions like 1 ≠ 0



Simple Example

- Sam, Clyde and Oscar are elephants. We know the following facts about them:
  - ✓ Sam is pink.
  - ✓ Clyde is gray and likes Oscar.
  - ✓ Oscar is either pink or gray(but not both) and likes Sam
- Prove a gray elephant likes a pink elephant.

Given the following facts：

- Pink(S);
- Gray(C);
- Gray(O) $\vee$ Pink(O);
- $\neg$ Gray(O) $\vee \neg$ Pink(O);
- Likes(C,O);
- Likes(O,S);

To prove $(\exists x,y)$ [Gray(x) $\wedge$ Pink(y) $\wedge$ Likes(x,y)]

Higher-order logic

- First-order logic allows us to quantify over objects.
- Higher-order logic also allows quantification over relations and functions.
  - ✓ e.g., “two objects are equal iff all properties applied to them are equivalent”:

    $(\forall x,y)\ ((x=y) \Leftrightarrow (\forall p,\ p(x) \Leftrightarrow p(y)))$
- Higher-order logics are more expressive than first-order; we have little understanding on how to effectively reason with sentences in higher-order logic.

Summary

- **First-order logic:**
  - ✓ objects and relations are semantic primitives
  - ✓ syntax: constants, functions, predicates, equality, quantifiers

- **Increased expressive power: sufficient to define wumpus world**

Logic is a Good Representation

- Fairly easy to do the translation when possible
- Branches of mathematics devoted to it
- It enables us to do logical reasoning
  - Tools and techniques come for free
- Basis for programming languages
  - Prolog uses logic programs (a subset of FOL)

Map of concept

- Objects, relation, function
- Variables, constants, functions, predicates, quantifier, equality, connectives
- Term, atomic sentence , complex sentences

Thank you

End of
Chapter 8