First Order Logic

Artificial Intelligence, Santa Clara University 2016
Thomas Schwarz, SJ
The nature of language

• Sapir-Whorf hypothesis (linguistic relativity):
  • Correlation between language and thought
    • Language determinism
      • Language entirely determines the range of possible cognitive processes of an individual
  • Example of constructivism
    • Human faculties and concepts are largely influenced by socially constructed and learned categories
The nature of language

• Natural language is a medium for communication
  • Not merely for representation
• Meaning of language depends on context
• Ambiguity (e.g. meaning of spring) does not impair thinking
The nature of language

• Formal logic:
  • Two different representations do not matter
• Thinking / Learning
  • Outcome can depend on representation
Combining Formal and Natural Languages

- More expressive logic
  - Nouns and noun phrases — objects
  - Verbs and verb phrases — relations
    - Special case: Function
      - One value for a given input
    - Special case: Property
      - Unary relation
Combining Formal and Natural Languages

• One plus two equals three
  • Objects: one, two, three
  • Relation: equals
  • Function: plus

• Equals(Plus(one, two), three)
Combining Formal and Natural Languages

• One plus two equals three
  • Objects: one, two, three
  • Relation: equals
  • Function: plus

• Equals(Plus(one, two), three)
Combining Formal and Natural Languages

• Squares neighboring the wumpus are smelly
  • Objects: wumpus, squares
  • Relation: neighboring
  • Property: smelly

• Neighboring(square, wumpus) implies Smelly(square)
Combining Formal and Natural Languages

- Evil king John ruled England in 1200
  - Objects: John, England, 1200
  - Relation: Rule
  - Property: Evil, King
Syntax and Semantics of First-Order Logic

• Symbols and interpretation
  • Constant symbols: objects
  • Predicate symbols: relations
  • Function symbols: functions

• Interpretation: what do symbols stand for
  • Intended interpretation
    • Richard ~ Richard Lionhearted
    • John ~ John Lackland
    • Brother ~ male sibling relation
    • ...
Syntax and Semantics of First-Order Logic

Sentence → AtomicSentence
  | Sentence Connective Sentence
  | Quantifier Variable,...Sentence
  | ¬Sentence
  | (Sentence)
AtomicSentence → Predicate(Term,...) | Term=Term | Predicate

Term → Function (Term,...)
  | Constant
  | Variable

Connective → ⇒ | ∧ | ∨ | ⇔
Quantifier → ∀ | ∃
Constant → A | X₁ | John | ...
Variable → a | x | s | ...
Predicate → Before | HanColor | Raining | ...
Function → Mother | LeftLegOf | ...
Syntax and Semantics of First-Order Logic

• Atomic sentences
  • predicate symbol
  • optionally followed by a parenthesized list of terms

• Examples:
  • Brother(Richard, John)
  • Married(Father(Richard), Mother(John))
Syntax and Semantics of First-Order Logic

- Complex sentences
  - Obtained using logical connectives
- Example:
  - not Brother(LeftLeg(John), Richard)
  - Brother(John, Richard) and Brother(Richard, John)
Syntax and Semantics of First-Order Logic

• Quantifiers

  • Universal quantification

    \[ \forall x \quad \text{King}(x) \Rightarrow \text{Person}(x) \]

  • \( x \) is a variable

  • Term with no variable is called a ground term

• \( \forall x \ P(x) \) true in a model

• IFF true in all extended interpretations

• which specifies a domain element to which \( x \) refers
Syntax and Semantics of First-Order Logic

### Universal quantification

\[ \forall (\text{variables}) (\text{sentence}) \]

Everyone at Berkeley is smart:
\[ \forall x \ At(x,\text{Berkeley}) \Rightarrow \text{Smart}(x) \]

\[ \forall x \ P \text{ is true in a model } m \text{ iff } P \text{ is true with } x \text{ being each possible object in the model} \]

Roughly speaking, equivalent to the conjunction of instantiations of \( P \)

\[ (At(\text{KingJohn},\text{Berkeley}) \Rightarrow \text{Smart}(\text{KingJohn})) \]
\[ \land (At(\text{Richard},\text{Berkeley}) \Rightarrow \text{Smart}(\text{Richard})) \]
\[ \land (At(\text{Berkeley},\text{Berkeley}) \Rightarrow \text{Smart}(\text{Berkeley})) \]
\[ \land \ldots \]
Syntax and Semantics of First-Order Logic

• Existential quantification

\[ \exists x \quad \text{Crown}(x) \land \text{OnHead}(x, \text{John}) \]

• Needs to be true for at least one domain object in each extended interpretation
Syntax and Semantics of First-Order Logic

Existential quantification

\[ \exists \text{(variables)} \ (\text{sentence}) \]

Someone at Stanford is smart:
\[ \exists x \ \text{At}(x, \text{Stanford}) \land \text{Smart}(x) \]

\[ \exists x \ P \text{ is true in a model } m \text{ iff } P \text{ is true with } x \text{ being some possible object in the model} \]

Roughly speaking, equivalent to the disjunction of instantiations of \( P \)

\[ (\text{At}(\text{KingJohn}, \text{Stanford}) \land \text{Smart}(\text{KingJohn})) \]
\[ \lor (\text{At}(\text{Richard}, \text{Stanford}) \land \text{Smart}(\text{Richard})) \]
\[ \lor (\text{At}(\text{Stanford}, \text{Stanford}) \land \text{Smart}(\text{Stanford})) \]
\[ \lor \ldots \]
Syntax and Semantics of First-Order Logic

- Perils of translations to First Order Logic
  - How to express that Dick has two brothers, Ernie and Fred
    \[ \text{Brother}(\text{Dick}, \text{Ernie}) \land \text{Brother}(\text{Dick}, \text{Fred}) \]
    - is false
  - Need to prevent that Ernie and Fred refer to the same object
  - Does not exclude that Dick has more brothers
    \[ \text{Brother}(\text{Dick, Ernie}) \land \text{Brother}(\text{Dick, Fred}) \land \text{Ernie} \neq \text{Fred} \land \forall x \quad \text{Brother}(\text{Dick, } x) \Rightarrow (x = \text{Ernie} \lor x = \text{Fred}) \]
Syntax and Semantics of First-Order Logic

Another common mistake to avoid

Typically, $\wedge$ is the main connective with $\exists$

Common mistake: using $\Rightarrow$ as the main connective with $\exists$:

$\exists x \ At(x, Stanford) \Rightarrow Smart(x)$

is true if there is anyone who is not at Stanford!
Syntax and Semantics of First-Order Logic

- Database systems: *Database semantics*
  - *unique names assumption*
    - Every constant refers to a different object
  - *closed-world assumption*
    - Atomic sentences not known to be true are in fact false
  - *domain closure:*
    - There are no more domain elements than those named by constant symbols
Monty Python and The Art of Fallacy

Cast

– Sir Bedevere the Wise, master of (odd) logic
– King Arthur
– Villager 1, witch-hunter
– Villager 2, ex-newt
– Villager 3, one-line wonder
– All, the rest of you scoundrels, mongrels, and nere-do-wells.
An example from Monty Python by way of Russell & Norvig

• FIRST VILLAGER: We have found a witch. May we burn her?
• ALL: A witch! Burn her!
• BEDEVERE: Why do you think she is a witch?
• SECOND VILLAGER: She turned me into a newt.
• B: A newt?
• V2 (after looking at himself for some time): I got better.
• ALL: Burn her anyway.
• B: Quiet! Quiet! There are ways of telling whether she is a witch.
Monty Python cont.

• **B**: Tell me… what do you do with witches?
• **ALL**: Burn them!
• **B**: And what do you burn, apart from witches?
• **Third Villager**: …wood?
• **B**: So **why do witches burn**?
• **V2 (after a beat)**: because they’re made of wood?
• **B**: Good.
• **ALL**: I see. Yes, of course.
B: So how can we tell if she is made of wood?
V1: Make a bridge out of her.
B: Ah… but can you not also make bridges out of stone?
ALL: Yes, of course… um… er…
B: Does wood sink in water?
ALL: No, no, it floats. Throw her in the pond.
B: Wait. Wait… tell me, what also floats on water?
ALL: Bread? No, no no. Apples… gravy… very small rocks…
B: No, no, no,
• **KING ARTHUR**: A duck!
• *(They all turn and look at Arthur. Bedevere looks up, very impressed.)*
• **B**: Exactly. So… logically…
• **V1 (beginning to pick up the thread)**: If she… weighs the same as a duck… she’s made of wood.
• **B**: And therefore?
• **ALL**: A witch!
Monty Python Fallacy #1

• $\forall x \text{ witch}(x) \rightarrow \text{burns}(x)$
• $\forall x \text{ wood}(x) \rightarrow \text{burns}(x)$
• ------------------------------
• $\therefore \forall z \text{ witch}(x) \rightarrow \text{wood}(x)$

• $p \rightarrow q$
• $r \rightarrow q$
• --------
• $p \rightarrow r$ Fallacy: Affirming the conclusion
Monty Python Near-Fallacy #2

- wood(x) → can-build-bridge(x)
- -------------------------------
- ∴ can-build-bridge(x) → wood(x)

• B: Ah… but can you not also make bridges out of stone?
Monty Python Fallacy #3

• ∀x wood(x) → floats(x)
• ∀x duck-weight (x) → floats(x)
• ----------------------------
• ∴ ∀x duck-weight(x) → wood(x)

• p → q
• r → q
• -------
• ∴ r → p
Using first order logic

- First order logic Knowledge Base (KB)
  - Use TELL to add assertions
  - Use ASK for queries (a.k.a. goals)
    - Example: \( \text{ASK}(\text{KB, } \exists x : \text{Person}(x)) \)
  - Find substitutions / binding list:
    - Example: \( \text{ASKVARS}(\text{Person}(x)) \)
    - Answer: \( \{x/\text{John}\}, \{x/\text{Richard}\} \)
Using First-Order Logic

- Binding lists cannot always be supplied
  - KB:
    \[ \text{King(John)} \lor \text{King(Richard)} \]
  - No answer for
    \[ \text{ASKVAR(King(x))} \]
Natural Numbers

• Peano axioms
  • Constant symbol 0
  • Successor function $S$
• Definition of natural numbers

\[
\begin{align*}
\text{NatNum}(0) \\
\forall n \text{NatNum}(n) \Rightarrow \text{NatNum}(S(n))
\end{align*}
\]

• Constraints on successor function

\[
\begin{align*}
\forall n : 0 \neq S(n) \\
\forall n, m : n \neq m \Rightarrow S(n) \neq S(m)
\end{align*}
\]
Using First-Order Logic

• Peano axioms
  • Define addition

\[ \forall n : \text{NatNum}(n) \Rightarrow +(0, n) = n \]
\[ \forall n, m : \text{NatNum}(n) \land \text{NatNum}(m) \Rightarrow +(S(n), m) = S(+ (n, m)) \]
Knowledge Engineering

- Knowledge Engineering Process
  1. Identify the task:
     - Range of questions
     - Kinds of facts
  2. Assemble the relevant knowledge
  3. Decide on a vocabulary of predicates, functions and constants
  4. Encode general knowledge about the domain
Knowledge Engineering

- Knowledge Engineering Process
  5. Encode a description of the specific problem instance
  6. Pose queries to the inference procedure and get answers
  7. Debug the knowledge base
Electronic Circuit Domain

• Step 1: Identify the task
  • Does the circuit add up properly
  • If all inputs are high, what is the output of gate A2
  • Does the circuit contain feedback loops
• Not considered:
  • Timing delays
  • Circuit area
  • Power consumption
  • Production costs
Electronic Circuit Domain

- Step 2: Assemble the relevant knowledge
  - Need to know rules about gates
  - Need to specify connections
- Step 3: Decide on a vocabulary
  - Assert that an object $X$ is a gate
    - $\text{gate}(X1)$
  - Assert the type of a gate
    - $\text{Type}(X1) = \text{XOR}$
  - Talk about circuits
    - $\text{Circuit}(C1)$
Electronic Circuit Domain

- Step 3: Decide on a Vocabulary (cont.)
  - Terminals
    - Terminal(x)
  - Each gate has one output terminal and one or two input terminals
    - First input terminal to X
      - In(1,X)
      - Out(X)
Electronic Circuit Domain

- Step 3: Decide on a Vocabulary (cont.)
  - Function arity gives the number of terminals
    - $\text{Arity}(c, i, j)$
  - Connectivity of terminals is a predicate
    - $\text{Connected}(\text{Out}(1,X1), \text{In}(1,X2))$
  - Use two signal values 0 and 1
  - Use function to obtain the signal at a terminal
    - $\text{Signal}(t)$
Electronic Circuit Domain

- Encode General Knowledge

\[ \forall t_1, t_2 \text{Terminal}(t_1) \wedge \text{Terminal}(t_2) \wedge \text{Connected}(t_1, t_2) \Rightarrow \text{Signal}(t_1) = \text{Signal}(t_2) \]

\[ \forall t \text{Terminal}(t) \Rightarrow \text{Signal}(t) = 1 \vee \text{Signal}(t) = 0 \]

\[ \forall t_1, t_2 \text{Connected}(t_1, t_2) \iff \text{Connected}(t_2, t_1) \]

\[ \forall g \text{Gate}(g) \wedge k = \text{Type}(g) \Rightarrow \]

\[ k = \text{AND} \vee k = \text{OR} \vee k = \text{XOR} \vee k = \text{NOT} \]
Electronic Circuit Domain

\[ \forall g \text{Gate}(g) \land \text{Type}(g) = \text{AND} \Rightarrow \]
\[ \text{Signal(Out(1, g))} = 0 \iff \exists n \text{Signal(In(n, g))} = 0 \]

\[ \forall g \quad \text{Gate}(g) \land \text{Type}(g) = \text{OR} \Rightarrow \]
\[ \text{Signal(Out(1, g))} = 1 \iff \exists n \text{Signal(In(n, g))} = 1 \]

\[ \forall g \quad \text{Gate}(g) \land \text{Type}(g) = \text{XOR} \Rightarrow \]
\[ \text{Signal(Out(1, g))} = 1 \iff \text{Signal(In(1, g))} \neq \text{Signal(In(2, g))} \]

\[ \forall g \quad \text{Gate}(g) \land \text{Type}(g) = \text{NOT} \Rightarrow \]
\[ \text{Signal(Out(1, g))} \neq \text{Signal(In(1, g))} \]
Electronic Circuit Domain

\( \forall g \, \text{Gate}(g) \land \text{Type}(g) = \text{NOT} \Rightarrow \text{arity}(g, 1, 1) \)

\( \forall g \, \text{Gate}(g) \land (\text{Type}(g) = \text{AND} \lor \text{Type}(g) = \text{OR} \lor \text{Type}(g) = \text{XOR}) \Rightarrow \text{arity}(g, 2, 1) \)

\( \forall c \forall i \forall j \quad \text{Circuit}(c) \land \text{Arity}(c, i, j) \Rightarrow \\
(\forall n \quad n \leq i \Rightarrow \text{Terminal}(\text{In}(c, n))) \\
\land (\forall n \quad n > i \Rightarrow \neg\text{Terminal}(\text{In}(c, n))) \\
\land (\forall n \quad n \leq j \Rightarrow \text{Terminal}(\text{Out}(c, n))) \\
\land (\forall n \quad n > j \Rightarrow \neg\text{Terminal}(\text{Out}(c, n))) \)
Electronic Circuit Domain

\[ \forall x, y : \text{Gate}(x) \land \text{Terminal}(y) \Rightarrow x \neq y \]
\[ \forall x : \text{Gate}(x) \Rightarrow x \neq 0 \]
\[ \forall x : \text{Gate}(x) \Rightarrow x \neq 1 \]
\[ \forall x : \text{Gate}(x) \Rightarrow x \neq \text{Nothing} \]

\[ \forall x : \text{Terminal}(x) \Rightarrow x \neq 0 \]
\[ \forall x : \text{Terminal}(x) \Rightarrow x \neq 1 \]
\[ \forall x : \text{Terminal}(x) \Rightarrow x \neq \text{Nothing} \]
Electronic Circuit Domain

And ≠ Or
And ≠ Xor
And ≠ Not
Or ≠ Xor
Or ≠ Not
Xor ≠ Not
Nothin ≠ And
Nothing ≠ Or
Nothin ≠ Xor
Nothing ≠ Not
Electronic Circuit Domain

- Encode the specific problem instance

\[
\text{Circuit}(C_1) \land \text{arity}(C_1, 3, 2)
\]
\[
\text{Gate}(X_1) \land \text{Type}(X_1) = \text{Or}
\]
\[
\text{Gate}(X_2) \land \text{Type}(X_2) = \text{Xor}
\]
\[
\text{Gate}(A_1) \land \text{Type}(A_1) = \text{And}
\]
\[
\text{Gate}(A_2) \land \text{Type}(A_2) = \text{And}
\]
\[
\text{Gate}(O_1) \land \text{Type}(O_1) = \text{Or}
\]
Connected(Out(1, X₁), In(1, X₂))
Connected(Out(1, X₁), In(2, A₂))
Connected(Out(1, A₂), In(1, O₁))
Connected(Out(1, A₁), In(2, O₁))
Connected(Out(1, X₂), Out(1, C₁))
Connected(Out(1, O₁), Out(2, C₁))
Connected(In(1, C₁), In(1, X₁))
Connected(In(1, C₁), In(2, A₁))
Connected(In(2, C₁), In(2, X₁))
Connected(In(2, C₁), In(2, A₁))
Connected(In(3, C₁), In(2, A₂))
Connected(In(3, C₁), In(2, X₂))
Electronic Circuit Domain

- Debugging:
  - System currently unable to provide outputs for circuits except for 000 and 110 as inputs
  - Forgot to assert

\[ 1 \neq 0 \]

- This type of errors are very common