NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

CZ3005 Artificial Intelligence

Week 10a – First-Order Logic

Yu Han

han.yu@ntu.edu.sg

Nanyang Assistant Professor
School of Computer Science and Engineering
Nanyang Technological University

Recap

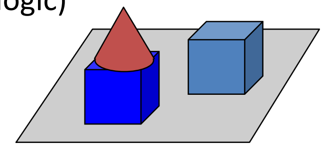
- A **sentence** is either <u>True</u> or <u>False</u> under an interpretation (or a world).
  - When it is <u>True</u>, we say that it is <u>satisfiable</u> (or is a model).
  - When it is <u>False</u>, then we say that it is <u>unsatisfiable</u>.
  - When a sentence is satisfiable under <u>all interpretations</u> (or worlds), then we say that it is a <u>valid</u> sentence or a <u>*tautology*</u>.
- Given that *N* objects are used in a **KB**, there will be 2^N possible interpretations (or worlds).

Recap

- **Soundness** and **Completeness**
  - Refer to an <u>inference procedure</u>
  - We do not say a sentence is sound or complete. An <u>inference procedure</u> is <u>sound</u> if we can derive <u>True</u> sentence from <u>True</u> sentences. That is, it implements entailment.
    - MP implements entailments
  - An <u>inference procedure</u> is <u>complete</u> if we can derive the proof of all entailed sentences (or valid sentences).

Representing Knowledge

- **Knowledge-based agent**
  - Have representations of the world in which they operate
  - Use those representations to infer what actions to take
- **Ontological commitments**
  - The world as <u>facts</u> (propositional logic)
  - The world as <u>objects</u> (first-order logic) with <u>properties</u> *about* each object, and <u>relations</u> *between* objects
    - e.g. the blocks world:
    - Objects: cubes, cylinders, cones, …
      Properties: shape, colour, location, …
      Relations: above, under, next-to, …



First-Order Logic (FOL)

- **A very powerful KR scheme**
  - Essential representation of the world
    - Deal with objects, properties, and relations.
  - Simple, generic representation
    - Does not deal with specialized concepts such as categories, time, and events.
  - Universal language
    - Can express anything that can be programmed.
  - Most studied and best understood
    - More powerful proposals still debated.
    - Less powerful schemes too limited.

Propositional vs. First-Order Logic

- Aristotle's syllogism
  - Socrates is a man. All men are mortal. Therefore Socrates is mortal.

Statement	Propositional Logic	First-Order Logic
"Socrates is a man."	SocratesMan, S43	Man(Socrates), P52(S21)
"Plato is a man."	PlatoMan, S157	Man(Plato), P52(S99)
"All men are mortal."	MortalMan S421 Man ⇒ Mortal S9⇒S4	Man(x) ⇒ Mortal(x), P52(V1) ⇒ P66(V1)
"Socrates is mortal."	MortalSocrates S957 S43 ∧ S421 |– S957 ?!?	Mortal(Socrates) V1←S21, … |– P66(S21)

Syntax and Semantics of FOL

- **Sentences**
  - Built from quantifiers, predicate symbols, and terms
- **Terms**
  - Represent objects
  - Built from variables, constant and function symbols
- **Constant symbols**
  - Refer to ("name") particular objects of the world
    - The object is specified by the interpretation
    - e.g. "John" is a constant , may refer to "John, king of England from 1199 to 1216 and younger brother of Richard Lionheart",
      or my uncle, or …

Syntax and Semantics of FOL

- **Variables**
  - Refer to any object of the world
    - e.g. x, person, … as in Brother(KingJohn, person).
  - Can be substituted by a constant symbol
    - e.g. person ← Richard, yielding Brother(KingJohn, Richard).
- **Terms**
  - Logical expressions referring to objects
    - Include constant symbols ("names") and variables.
    - Make use of function symbols.
      e.g. LeftLegOf(KingJohn) to refer to his leg without naming it
  - Compositional interpretation
    - e.g. LeftLegOf(), KingJohn –> LeftLegOf(KingJohn).

Predicate and Function Symbols

- **Predicate symbols**
  - Refer to particular relations on objects
    - Binary relation specified by the interpretation
      e.g. Brother(KingJohn, RichardLionheart) –> T or F
  - A *n-ary* relation if defined by a set of *n-tuples*
    - Collection of objects arranged in a fixed order
      e.g. { ⟨KingJohn, RichardLionheart⟩, ⟨KingJohn, Henry⟩, ... }

Returns T|F

- **Function symbols**
  - Refer to functional relations on objects
    - Many-to-one relation specified by the interpretation
      e.g. BrotherOf(KingJohn) –> a person, e.g. Richard (*not* T/F)
  - Defined by a set of *n+1-tuples*
    - Last element is the function value for the first *n* elements.

Returns anything but T|F

Function Symbol Example

- A <u>function</u> of arity *n* takes *n* <u>objects</u> of type $W_1,\ldots,W_n$ as inputs and returns an <u>object</u> of type *W.*
- Example:
  - Plus(3, 4)=7

Object Terms

Functional Term

Predicate Symbol Example

- <u>Predicates</u> are like functions except that their return type is <u>True</u> or <u>False</u>.
- Example:
  - Greater-Than(3, 4)=False

Sentences in FOL

- **Atomic sentences**
  - State facts, using terms and predicate symbols
    - e.g. Brother(Richard, John).
  - Can have complex terms as arguments
    - e.g. Married(FatherOf(Richard), MotherOf(John)).
  - Have a truth value
    - Depends on both the interpretation and the world.
- **Complex sentences**
  - Combine sentences with connectives
    - e.g. Father(Henry, KingJohn) ∧ Mother(Mary, KingJohn)
  - Connectives identical to propositional logic
    - i.e.: ∧, ∨, ⇔, ⇒, ¬

Sentence Equivalence

- ***There are many ways to write a logical statement in FOL***
  - Example
    - $A \Rightarrow B$ equivalent to $\neg A \vee B$
      "rule form" *"complementary cases"*

      Dog(x) ⇒ Mammal(x) ¬Dog(x) ∨ Mammal(x)
      "dogs are mammals" "either it's not a dog or it's a mammal"

    - $A \wedge B \Rightarrow C$ equivalent to $A \Rightarrow (B \Rightarrow C)$

    - Proof: $A \wedge B \Rightarrow C \Leftrightarrow \neg(A \wedge B) \vee C \Leftrightarrow (\neg A \vee \neg B) \vee C$
      $\Leftrightarrow \mathbf{\neg A \vee \neg B \vee C} \Leftrightarrow \neg A \vee (\neg B \vee C)$
      $\Leftrightarrow \neg A \vee (B \Rightarrow C) \Leftrightarrow A \Rightarrow (B \Rightarrow C)$

      $\neg P \vee Q \Leftrightarrow P \Rightarrow Q$

Sentences in Normal Form

- ***There is only one way to write a logical statement using a Normal Form of FOL***
  - Example

    without ⇒

    $\neg B \Rightarrow \neg A$
    - $A \Rightarrow B$, $A \wedge B \Rightarrow C$ equivalent to $\neg A \vee B$, $\neg A \vee \neg B \vee C$
      "Implicative Normal Form" "Conjunctive Normal Form"
- ***Rewriting logical sentences allows to determine whether they are equivalent or not***
  - Example
    - $A \wedge B \Rightarrow C$ and $A \Rightarrow (B \Rightarrow C)$
    - both have the same CNF: $\neg A \vee \neg B \vee C$

Sentence Verification

- ***Rewriting logical sentences helps to understand their meaning***
  - Example
    - Owns(x,y) ⇒ (Dog(y) ⇒ AnimalLover(x)) $A \Rightarrow (B \Rightarrow C)$
    - Owns(x,y) ∧ Dog(y) ⇒ AnimalLover(x) $A \wedge B \Rightarrow C$
      "A dog owner is an animal lover"
- ***Rewriting logical sentences helps to verify their meaning is as intended***
  - Example
    - *"Dogs all have the same enemies"*
      Dog(x) ∧ Enemy(z, x) ⇒ (Dog(y) ⇒ Enemy(z, y)) same as
      Dog(x) ∧ Dog(y) ∧ Enemy(z, x) ⇒ Enemy(z, y)

Universal Quantifier ∀

- **Express properties of collections of objects**
  - Make a statement about *every* objects w/out enumerating
    - e.g. "All kings are mortal
      King(Henry) ⇒ Mortal(Henry) ∧
      King(John) ⇒ Mortal(John) ∧
      King(Richard) ⇒ Mortal(Richard) ∧
      King(London) ⇒ Mortal(London) ∧
      ...
  - instead: ∀ x, King(x) ⇒ Mortal(x)
    - Note: the semantics of the implication says F ⇒ F is TRUE.
    - Thus, for those individuals that satisfy the premise King(x), the rule asserts the conclusion Mortal(x)
    - But, for those individuals that do not satisfy the premise, the rule makes no assertion.

Using the Universal Quantifier

∀ must be used with Rule (⇒)

- **The implication (⇒) is the natural connective to use with the universal quantifier (∀)**
  - Example
    - General form: $\forall x\ P(x) \Rightarrow Q(x)$
      - e.g. $\forall x\ Dog(x) \Rightarrow Mammal(x)$ "all dogs are mammals"
    - Use conjunction? $\forall x\ P(x) \land Q(x)$
      - e.g. $\forall x\ Dog(x) \land Mammal(x)$
      - same as $\forall x\ P(x)$ and $\forall x\ Q(x)$

        e.g. $\forall x\ Dog(x)$ and $\forall x\ Mammal(x)$

        All dogs are mammals. All mammals are dogs. Not correct.

      –> yields a very strong statement (too strong! i.e. *incorrect*)

Existential Quantifier ∃

- **Express properties of some particular objects**
  - Make a statement about <u>one</u> object without naming it
    - e.g., "King John has a brother who is king"
    - $\exists x$, Brother(x, KingJohn) $\land$ King(x)
    - instead of

      Brother(Henry, KingJohn) $\land$ King(Henry) $\lor$
      Brother(London, KingJohn) $\land$ King(London) $\lor$
      Brother(Richard, KingJohn) $\land$ King(Richard) $\lor$
      …

Using the Existential Quantifier

- **The conjunction ($\land$) is the natural connective to use with the existential quantifier (∃)**
  - Example
    - General form: $\exists x\ P(x) \land Q(x)$
      - e.g., $\exists x$ Dog (x) $\land$ Owns(John, x), "John owns a dog"
    - Use Implication? $\exists x\ P(x) \Rightarrow Q(x)$
      - e.g., $\exists x\ Dog(x) \Rightarrow Owns(John, x)$
      - Could be true for all x such that P(x) is false

        e.g., Dog(Garfield_the_cat) ⇒ Owns(John, Garfield_the_cat)

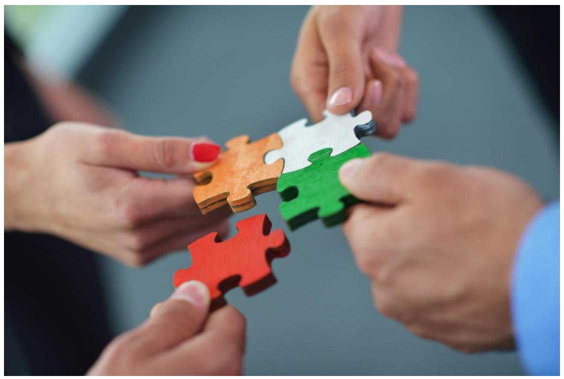
      –> yields a very weak statement (too weak! i.e. *useless*)

Properties of ⇒
if T ⇒ T
F ⇒ T|F
∴ WRONG

F

can be T

Thank you!



NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

CZ3005 Artificial Intelligence

Week 10b – First-Order Logic

Yu Han

han.yu@ntu.edu.sg

Nanyang Assistant Professor
School of Computer Science and Engineering
Nanyang Technological University

Recap

- **First-Order Logic:**
  - Allows descriptions of relations and properties.
  - Allows the descriptions of relations and properties in a compounded manner.
  - Permits the use of constant and variables.
  - Allows general statements to be made.
  - Separation of inference from the representation of knowledge.

Recap

- **Normal Forms**
  - P=>Q can be rewritten as ¬ P v Q
- **Auto documentation with predicate naming**
  - Reading from left to right
  - e.g., grandfather(Philip, William)
- **For all** quantifier (universal for all models) is used with implication connective
  - ∀ **with =>**
- **Existential** quantifier (satisfiable for at least one model) is used with and connective
  - ∃ with ^

Recap

Generation of complex sentences

Composition via

Connectives – expressiveness

i.e. complex relations

Generalization – universal statement

– existential statement

Nesting and Mixing Quantifiers

- **Combining ∀ and ∃**
  - Express more complex sentences
    - e.g., “if x is the parent of y, then y is the child of x”:
    - ∀ x, ∀ y Parent(x, y) ⇒ Child(y, x)
    - “everyone has a parent”: ∀ x, ∃ y Parent(y, x)
  - Semantics depends on quantifiers ordering
    - e.g., ∃ y, ∀ x Parent(y, x)
    - “there is someone who is everybody’s parent” ?
- **Well-formed formula (WFF)**
  - Sentences with all variables properly quantified

For all x & all y if x is the parent of y, y must be parent of x

For all x, there must exists y such that y is the parent of x

there exists a y such that for all x, y is the parent of x

Connections between Quantifiers

- **Equivalences**
  - Using the negation *(hence only one quantifier is needed)*

there does not exists any x such that P(x) is true

$$\forall x \; P(x) \Leftrightarrow \neg \exists x \neg P(x)$$

- e.g. ”everyone is mortal”:

$$\forall x \; Mortal(x) \Leftrightarrow \neg \exists x \neg Mortal(x)$$

- De Morgan’s Laws
  -

$\forall x\, P \Leftrightarrow \neg\exists x\, \neg P$	$P \wedge Q \Leftrightarrow \neg(\neg P \vee \neg Q)$
$\forall x\, \neg P \Leftrightarrow \neg\exists x\, P$	$\neg P \wedge \neg Q \Leftrightarrow \neg(P \vee Q)$
$\neg\forall x\, P \Leftrightarrow \exists x\, \neg P$	$\neg(P \wedge Q) \Leftrightarrow \neg P \vee \neg Q$
$\neg\forall x\, \neg P \Leftrightarrow \exists x\, P$	$\neg(\neg P \wedge \neg Q) \Leftrightarrow P \vee Q$

Equality Predicate Symbol

- **Need for equality**
  - State that two terms refer to the <u>same object</u>
    - e.g., Father(John) = Henry, or
    - =(Father(John), Henry)
  - Useful to define properties
    - e.g. “King John has two brothers”:
    - ∃ x, y Brother(x, KingJohn) ∧ Brother(y, KingJohn) ∧ ¬(x=y)

distinguish two different persons.

Grammar of First-Order Logic

(Backus-Naur Form)

Sentence	→	AtomicSentence | (Sentence) | Sentence Connective Sentence | ¬Sentence | Quantifier Variable, … Sentence
AtomicSentence	→	Predicate(Term, …) | Term = Term
Term	→	Function(Term, …) | Constant | Variable
Connective	→	∧ | ∨ | ⇔ | ⇒
Quantifier	→	∀ | ∃
Constant	→	A | X_1 | John | …
Variable	→	a | x | person | …
Predicate	→	P() | Colour() | Before() | …
Function	→	F() | MotherOf() | SquareRootOf() | …

Using First-Order Logic

- **Knowledge domain**
  - A part of the world we want to express knowledge about
- **Example of the kinship domain**
  - Objects: people e.g., Elizabeth, Charles, William, etc.
  - Properties: gender i.e., male, female
    Unary predicates: Male() and Female()
  - Relations: kinship e.g., motherhood, brotherhood, etc.
    Binary predicates: Parent(), Sibling(), Brother(), Child(), etc.
    Functions: MotherOf(), FatherOf() ← Unary
  - > Express facts e.g., Charles is a male
    and rules e.g., the mother of a parent is a grandmother

Sample Functions and Predicates

- **Functions**
  - ∀ x,y FatherOf(x)=y ⇔ Parent(y,x) ∧ Male(y)
  - ∀ x,y MotherOf(x)=y ⇔ Parent(y,x) ∧ Female(y)
- **Predicates**
  - ∀ x,y Parent(x,y) ⇔ Child(y,x) → transit two levels of parents.
  - ∀ x,y Grandparent(x,y) ⇔ ∃ z, Parent(x,z) ∧ Parent(z,y)
  - ∀ x,y Sibling(x,y) ⇔ ¬x=y ∧ ∃ z, Parent(z,x) ∧ Parent(z,y)
  - ∀ x Male(x) ⇔ ¬Female(x)

this to defined like this "recursive"

- **Potential problems**
  - Self-definition (causes infinite recursion)
    - e.g., ∀ x,y Child(x,y) ⇔ Parent(y,x) following the above

In KB Just list child(John, Betty) ;

TELLing and ASKing

- **TELLing the KB**
  - Assertion: add a sentence to the knowledge base
    - e.g.
      TELL(KB, ∀x,y MotherOf(x)=y ⇔ Parent(y,x) ∧ Female(y))
      and so on, then
      TELL(KB, Female(Elizabeth) ∧ Parent(Elizabeth,Charles) ∧ Parent(Charles,William))
- **ASKing the KB**
  - Query: retrieve/infer a sentence from the knowledge base
  - Yes/No answer
    - e.g. ASK(KB, Grandparent(Elizabeth, William))
  - Binding list, or substitution
    - e.g. ASK(KB, ∃x Child(William, x)) yields {x / Charles}

Inferences Rules for FOL

- **Inference rules from Propositional Logic**
  - *Modus Ponens*
    - $$\frac{\alpha \Rightarrow \beta,\ \alpha}{\beta}$$
      if α (T) ⇒ β (T)
  - Double-Negation-Elimination
    - $$\frac{\neg\neg\alpha}{\alpha}$$
      implies
  - And-Elimination
    - $$\frac{\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n}{\alpha_i}$$
      if you link symbols ∧ in αi then if out take out αi should eval to T
  - And-Introduction
    - $$\frac{\alpha_1, \alpha_2, \ldots, \alpha_n}{\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n}$$
      All interpreted as T
      when combined ⇒ become T
  - Or-Introduction
    - $$\frac{\alpha_i}{\alpha_1 \vee \alpha_2 \vee \ldots \vee \alpha_n}$$
      As long as any αi true you can use OR to
  - Resolution
    - $$\frac{\alpha \vee \cancel{\beta},\ \cancel{\neg\beta} \vee \gamma}{\alpha \vee \gamma}$$

Universal Elimination

- $\forall$ x, Likes(x, flower)
- Substituting x by Shirin gives No longer Rule
- Likes(Shirin, flower)
- The substitution should be done by a constant term.
- In this way, the $\forall$ quantifier can be eliminated.

Link them with ∨ and the entire sentence will still be True

Existential Elimination/Introduction

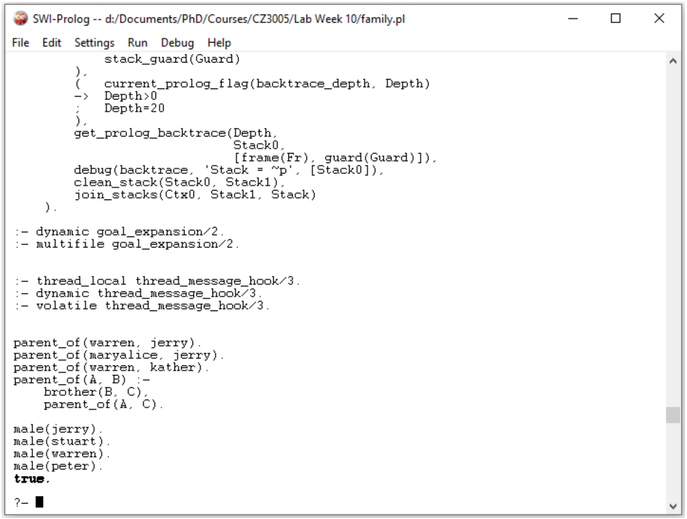
- **Existential Elimination**
  - $\exists$x, likes(x, flower)
  - Can be changed to:
  - likes(Person, flower)
  - As long as the person is not in the knowledge base.
- **Existential Introduction**
  - Likes(Marry, flower)
  - Can be written as:
  - $\exists$x, likes(x, flower)

Working Example with Prolog

- SWI-Prolog offers a comprehensive free Prolog environment.
- Since its start in 1987, SWI-Prolog development has been driven by the needs of real world applications.
- SWI-Prolog is widely used in research and education as well as commercial applications.
- Download it here: https://www.swi-prolog.org/
- Let's see the "royal family" example together

SWI Prolog

Working Example with Prolog



Thank you!