Predicate Logic (II) & Semantic Type

Yimei Xiang
yxiang@fas.harvard.edu
25 February 2014

1 Review

1.1 Set theory

1.2 Propositional logic

• Connectives

• Syntax of propositional logic:
  – A recursive definition of well-formed formulas
  – Abbreviation rules

• Semantics of propositional logic:
  – Truth tables
  – Logical equivalence
  – Tautologies, contradictions, contingencies
  – Indirect reasoning (Deduction ad absurdum)

1.3 Predicate logic (I)

• Vocabulary: individual constants, individual variables, predicates, connectives, quantifiers, constituency labels

• Translations
  i. Arity of a predicate (e.g. *come* vs. *friends* vs. *give*)
  ii. quantification expressions: the representation form of universal quantification and existential quantification have to be an implication and a conjunction, respectively
     
     (1) a. $\exists x[P(x) \land Q(x)]$
     b. $\forall x[P(x) \rightarrow Q(x)]$
  iii. for universal quantification, pay attention to the restriction (viz. the antecedent part)
iv. Scope patterns: pay attention to the scope relation between quantifiers and propositional connectives (e.g. negation)

(2)  a. John didn’t find some book.
    b. John didn’t find any book.
    c. Every boy loves a girl.

- Well-formed formulas of predicate logic (a simpler version)
  Vacuous quantification (e.g. $\forall x P(j)$)

- Scope, bound and free, closed and open

2 The semantics of predicate logic

2.1 Interpretation functions and modals

- Models
  Expressions are interpreted in models. A model $M$ is a pair $(D, I)$, where $D$ is the domain, a (nonempty) set of individuals, and $I$ is an interpretation function: an assignment of semantic values to every basic expression (constant) in the language.
  Models are distinguished both by the objects in their domains and by the values assigned to the expressions of the language by $I$ by the particular way that the words of the language are “linked” to the things in the world.

- Domain
  In order to judge the truth value of the following sentence, it is necessary to know what we are talking about, viz. what the domain of discourse is.

(3) Everyone is friendly.

- Interpretation functions
  An interpretation relates $L$ to the world (or a possible world) by giving the extensions/values of the expressions of the language, i.e. the objects of the world that are designated by the expressions of $L$.
  The interpretation of an arbitrary expression $\alpha$ relative to $M$: $[\alpha]^M$
  $[c]^M$ is called the interpretation of a constant $c$, or its reference/denotation, and if $e$ is the entity in $D$ s.t. $[c]^M = e$, then $c$ is said to be one of $e$’s names ($e$ may have several different names.)
• Example: a toy language L

The toy language L only has three categories of expressions: names, one-place predicates, and two-place predicates.

<table>
<thead>
<tr>
<th>Category</th>
<th>Basic expressions</th>
<th>NL counterpart</th>
</tr>
</thead>
<tbody>
<tr>
<td>Names</td>
<td>s, a, t, m</td>
<td>Sharon, Anna, Tiphanie, Martin</td>
</tr>
<tr>
<td>One place (unary) predicates</td>
<td>H, C</td>
<td>Happy, cries</td>
</tr>
<tr>
<td>Two place (binary) predicates</td>
<td>D, K</td>
<td>dislike, know</td>
</tr>
</tbody>
</table>

(4) \( M_1 = \langle D_1, I_1 \rangle \), where

a. \( D_1 = \{ \text{Sharon, Anna, Tiphanie, Martin} \} \)

b. \( I_1 \) determines the following mapping mapping between names and predicate terms in L and objects in \( D_1 \)

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Predicate</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>Sharon</td>
<td>H</td>
<td>{Sharon, Anna}</td>
</tr>
<tr>
<td>a</td>
<td>Anna</td>
<td>C</td>
<td>{Sharon, Anna, Tiphanie}</td>
</tr>
<tr>
<td>t</td>
<td>Tiphanie</td>
<td>D</td>
<td>{ (Sharon, Martin), (Anna, Tiphanie)}</td>
</tr>
<tr>
<td>m</td>
<td>Martin</td>
<td>K</td>
<td>{ (Sharon, Martin), (Anna, Tiphanie), (Tiphanie, Sharon)}</td>
</tr>
</tbody>
</table>

Composition rules of L (part 1)

(5) a. If \( P \) is a one place predicate and \( \alpha \) is a name, then \([P(\alpha)]^M = 1 \) iff \([\alpha]^M \in [P]^M \).

b. If \( Q \) is a two place predicate and \( \alpha \) and \( \beta \) are names, then \([P(\alpha, \beta)]^M = 1 \) iff \([\alpha]^M, [\beta]^M \in [Q]^M \).

c. If \( \phi \) is a formula, then \([-\phi]^M = 1 \) i \( [\phi]^M = 0 \).

d. If \( \phi \) and \( \psi \) are formulas then \([\phi \land \psi]^M = 1 \) iff both \([\phi]^M \) and \([\psi]^M = 1 \).

e. If \( \phi \) and \( \psi \) are formulas then \([\phi \lor \psi]^M = 1 \) iff

f. If \( \phi \) and \( \psi \) are formulas then \([\phi \rightarrow \psi]^M = 1 \) iff

g. If \( \phi \) and \( \psi \) are formulas then \([\phi \leftrightarrow \psi]^M = 1 \) iff

Exercise 1:

Give one sentence with a negation which is true in \( M_1 \).

Give one sentence with an implication which is true in \( M_1 \).
2.2 Assignment function

- So far we have assumed that interpretations of basic expressions are given by \( I \), which assigns values in \( D \) to names and predicates. The interpretation of individual variables requires a further semantic component, called an assignment function, notated \( g \). The assignment function assigns individuals in \( D \) to individual variables in formulas.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( g_1 )</td>
<td>( x )</td>
<td>Anna</td>
<td></td>
</tr>
<tr>
<td>( y )</td>
<td></td>
<td>Sharon</td>
<td>( g_2 )</td>
</tr>
<tr>
<td>( z )</td>
<td></td>
<td>Martin</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Tiphanie</td>
</tr>
</tbody>
</table>

- Composition rules of \( L \) (Part 2)

    (6) a. If \( \phi \) is a formula, then \( [\forall x \phi]^{M,g} = 1 \) iff \( [\phi]^{M,g[d/x]} = 1 \) for all \( d \in D \).

    b. If \( \phi \) is a formula, then \( [\exists x \phi]^{M,g} = 1 \) iff \( [\phi]^{M,g[d/x]} = 1 \) for some \( d \in D \).

2.3 Properties of relations

- Reflexivity

If \( \forall x R(x, x) \) holds in \( M \), then \( R \) is reflexive in \( M \).

- Symmetry

If \( \forall x \forall y (R(x, y) \rightarrow R(y, x)) \) holds in \( M \), then \( R \) is symmetric in \( M \).

- Transitivity

If \( \forall x \forall y \forall z (R(x, y) \land R(y, z) \rightarrow R(x, z)) \) holds in \( M \), then \( R \) is transitive in \( M \).

- Converse

A relation \( R \) is said to be the converse of another relation \( S \), if \( R(x, y) \) is true whenever \( S(y, x) \) is true.

E.g. ‘parent of’ vs. ‘children of’; ‘is seen by’ vs. ‘see’.

**Exercise 2:** How to represent non-reflexive, non-symmetric, non-transitive? How to represent irreflexive, asymmetric and intransitive?
3 Semantic Types

3.1 Fregean principle and categorical grammar

- **Fregean principle:**

  The meaning of a complex expression should be a function of the meaning of its parts.

  Frege distinguished between *saturated* and *unsaturated* meanings.

  (7) Sue snores.

  This sentence can be split into two parts:

  \[
  S \quad \text{Sue} \quad \text{snores}
  \]

  Frege’s suggestion was to treat one part of the sentence as saturated and the other part as unsaturated, and then assume that the meaning of (7) is the result of applying the unsaturated part of the sentence to the saturated part. This process is called *functional application*.

  In set-theoretic terms, we can think of this idea as follows:

  - Unsaturated parts are functions (functions that take arguments and output values)
  - Saturated parts are arguments (arguments for functions)

- **Categorical grammar**

  Linguistic communication essentially involves two things:

  i. Picking out some entity in the world; → **NAME (N)**
  
     Snores is the *functor* of a function (S/N) which right-concatenates the lexical item *runs* with a NAME(N) to make a SENTENCE(S).

  ii. Saying something about that entity. → **SENTENCE (S)**

### A simple categorial syntax

<table>
<thead>
<tr>
<th>Basic Categories</th>
<th>Dictionary Expressions</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i) S</td>
<td>none</td>
</tr>
<tr>
<td>(ii) N</td>
<td><em>Arthur, Canute, Tristan, Isolde</em></td>
</tr>
<tr>
<td>Derived Categories</td>
<td>necessary, possibly, not</td>
</tr>
<tr>
<td>(iii) S/S</td>
<td>and, or, if ... then, if</td>
</tr>
<tr>
<td>(iv) S/SS</td>
<td><em>runs</em></td>
</tr>
<tr>
<td>(v) S/N</td>
<td><em>fast, carefully</em></td>
</tr>
<tr>
<td>(vi) (S/N)/(S/N)</td>
<td><em>someone, everyone</em></td>
</tr>
<tr>
<td>(vii) S/(S/N)</td>
<td><em>seeks</em></td>
</tr>
<tr>
<td>(viii) (S/N)/N</td>
<td></td>
</tr>
</tbody>
</table>
3.2 Basic notions

- The categories of syntax correspond in a one-to-one fashion to **semantic types**. In type theory, one starts by assuming that there is a set of types \( T \). This set contains two basic types and it is then recursively defined for complex types. Here we will only focus on *extensional types*.

\[
\begin{align*}
(8) & \quad \text{a. Type } e \\
& \qquad e \text{ is the type of individuals so, } D_e \text{ is a set of objects of type } e. \\
& \qquad \text{b. Type } t \text{ is the type of truth values so, } D_t \text{ is a set of objects of type } t.
\end{align*}
\]

The basic types correspond to the objects that Frege took to be *saturated*. From these basic types, we now recursively define complex types.

\[
\begin{align*}
(9) & \quad \text{a. } e \in T \\
& \qquad \text{b. } t \in T \\
& \qquad \text{c. If } \alpha \in T, \beta \in T, \text{ then } <\alpha, \beta> \in T \\
& \qquad \text{d. Nothing is an element of } T \text{ except on the basis of (a), (b), and (c).}
\end{align*}
\]

With this definition, we can now define an infinity of different types. For example: We will soon discover that for

\[
(10) \quad <e, t>, <e, <e, t>>, <t, t>
\]

each of these types there is an expression in English with that type.

- Complex types, such as those above, correspond to what Frege took to be unsaturated objects, i.e. *functions*.

\[
\begin{align*}
(11) & \quad \text{a. } <e, t> \text{ is a function from objects of type } e \text{ to objects of type } t. \\
& \qquad \text{b. } <e, <e, t>> \text{ is a function from objects of type } e \text{ to a function of type } <e, t>. \\
\end{align*}
\]

With types as part of our inventory, we can now classify expressions according to their type. For example:

\[
(12) \quad \text{a. } [\text{Bob}] \text{ is of type } e. \\
& \qquad \text{b. } [\text{run}] \text{ is of type } <e, t> \\
& \qquad \text{c. } [\text{marry}] \text{ is of type } <e, <e, t>>
\]

The denotations of these expressions obviously depend on the model \( M \), but they each have different domains; we will use the following convenient abbreviations for these domains.

\[
\begin{align*}
(13) & \quad \text{a. } D_e: \text{ The set of individuals/objects } x, x \in D_e \\
& \qquad \text{b. } D_{<e,t>}: \text{ The set of functions } f \text{ from } D_e \text{ to } D_t, f \subseteq D_e \times D_t \\
& \qquad \text{c. } D_{<e,<e,t>>}: \text{ The set of functions } f \text{ from } D_e \text{ to } D_{<e,t>}, f \subseteq D_e \times (D_e \times D_t) \\
& \qquad \text{d. For every type } a, \text{ the set of domain of denotations of that type will be } D_a;
\end{align*}
\]
e. For a complex type, the set $D_{<a,b>}$ will be the set of functions from $D_a$ to $D_b$.

- **Exercise 2:** Classify the following words based on their semantic types:
  
  *not, if...then, beautiful, student, John, Mary, buy, friend, fast, carefully, necessarily*

### 3.3 The Convenience of Types

- With type assignments to expressions of natural language, it is much easier to determine the semantics of new expressions.

Suppose we want to figure out what semantics to give for the preposition *to* as used in the following sentence. Annotating the phrase structure with types makes it extremely easy to tell what type of expression is needed for the derivation to work out.

(14) Sue talked to Bob.
**Exercise 3:** Determine the semantic type of vacuous words like ‘is’ and ‘of’ in the following sentence.

(15) Sam is proud of John.