Semantics and Pragmatics of NLP Propositional Logic, Predicates and Functions

Alex Lascarides & Ewan Klein

School of Informatics
University of Edinburgh

10 January 2008

Why Bother?

Aim:

1. To associate NL expressions with semantic representations;
2. to evaluate the truth or falsity of semantic representations relative to a knowledge base;
3. to compute inferences over semantic representations.

Strategy:

- Deal with task (1) later, but assume the target is FOL...
- Achieve tasks (2)–(3) by associating FOL with models and rules of inference.

Logics: Syntax and Semantics

1. A Vocabulary (aka lexicon)
   - determines what we can talk about
2. Syntax
   - Uses vocabulary and syntactic rules to define the set of well-formed formulas (WFFS)
   - determines *how* we can talk about things
3. Semantics
   - Compositional (uses recursion)
   - Truth, Satisfaction, Entailment.

The Language of Propositional Logic, version 1

Basic expressions:

1. Propositional variables $p, q, r, p_0, p_1, \ldots$.
2. Boolean connectives:
   - $\neg$ (negation)
   - $\wedge$ (and)
   - $\vee$ (or)
   - $\rightarrow$ (if…then)

Rules of syntax:

1. Every propositional variable is a well-formed formula (WFF).
2. If ϕ and ψ are WFFs, then so are: $\neg\phi$, $(\phi \wedge \psi)$, $(\phi \vee \psi)$, $(\phi \rightarrow \psi)$.

Models for Propositional Logic, version 1

- Interpretation Function: A mapping V from each propositional variable to the set of truth values {0, 1}.

A Valuation

$$V(p) = 1$$
$$V(q) = 0$$
$$V(r) = 1$$

A model M for propositional logic is just a valuation V. For an arbitrary WFF ϕ, we write $M \models \phi$ to mean ϕ is true in model M.

Models for Propositional Logic, version 1

Recursive definition of truth in a model $M = V$.

$$
\begin{array}{lll}
M \models p_i & \text{iff} & V(p_i) = 1 \\
M \models \neg\phi & \text{iff} & M \not\models \phi \\
M \models \phi \wedge \psi & \text{iff} & M \models \phi \text{ and } M \models \psi \\
M \models \phi \vee \psi & \text{iff} & M \models \phi \text{ or } M \models \psi \\
M \models \phi \rightarrow \psi & \text{iff} & M \not\models \phi \text{ or } M \models \psi
\end{array}
$$

Adding Predicates to the Language

FOL designed to talk about various relationships and properties that hold among individuals.

Terms	Unary Predicates	Binary Predicates
john	dog	chase
mary	girl	kiss
kim	run	
fido	smile	

NB: nouns and intransitive verbs treated the same.

The vocabulary constrains the class of models (that is, the kinds of situation we want to describe).

Models for FOL, version 1

- Domain: The collection D of entities we can talk about;
- Interpretation Function: A mapping V from each symbol in the vocabulary to its semantic value.
- The arity of a symbol s determines what kind of value $V(s)$ should be.

Valuations

$V(\text{fido}) \in D$
$V(\text{dog}) \subseteq D$
$V(\text{chase}) \subseteq D \times D$

Valuations for Terms and Predicates

A Valuation

$M = \langle D, V \rangle$, where:

$D = \{d_1, d_2, d_3, d_4\}$

$$\begin{array}{ll}
V(\text{john}) = d_1 & V(\text{dog}) = \{d_4\} \\
V(\text{mary}) = d_2 & V(\text{girl}) = \{d_2, d_3\} \\
V(\text{kim}) = d_3 & V(\text{run}) = \{d_4\} \\
V(\text{fido}) = d_4 & V(\text{smile}) = \{d_1\} \\
V(\text{chase}) = \{(d_2, d_3), (d_3, d_4)\} & \\
V(\text{kiss}) = \{(d_2, d_1), (d_1, d_2)\} &
\end{array}$$

$M \models R(\tau_1, \ldots, \tau_n)$ iff $(V(\tau_1), \ldots V(\tau_n)) \in V(R)$

Alternative Approach to Predicates

- We take function expressions as basic to our language, corresponding to functions in the model.
- It's helpful to regard the function expressions as typed; e.g., $\alpha^{\sigma \rightarrow \tau}$ combines with expressions of type σ to yield expressions of type τ.

A Boolean-valued Function Expression

$\text{dog}^{\text{IND} \rightarrow \text{BOOL}}$

i.e., combines with terms to yield expressions with Boolean values (WFFs).

Alternative Approach to Predicates

- We take function expressions as basic to our language, corresponding to functions in the model.
- It's helpful to regard the function expressions as typed; e.g., $\alpha^{\sigma \rightarrow \tau}$ combines with expressions of type σ to yield expressions of type τ.

A Boolean-valued Function Expression

$\text{dog}^{\text{IND} \rightarrow \text{BOOL}}$

i.e., combines with terms to yield expressions with Boolean values (WFFs).

Alternative Approach to Predicates

- We take function expressions as basic to our language, corresponding to functions in the model.
- It's helpful to regard the function expressions as typed; e.g., $\alpha^{\sigma \rightarrow \tau}$ combines with expressions of type σ to yield expressions of type τ.

A Boolean-valued Function Expression

dog$^{\text{IND}\rightarrow\text{BOOL}}$

i.e., combines with terms to yield expressions with Boolean values (WFFs).

Alternative Approach to Predicates

- We take function expressions as basic to our language, corresponding to functions in the model.
- It's helpful to regard the function expressions as typed; e.g., $\alpha^{\sigma \rightarrow \tau}$ combines with expressions of type σ to yield expressions of type τ.

A Boolean-valued Function Expression

$\text{dog}^{\text{IND} \rightarrow \text{BOOL}}$

i.e., combines with terms to yield expressions with Boolean values (WFFs).

Functions in the model

Types are pretty much the same as arities.

- $V(\alpha^{\mathrm{IND}}) \in D$
- $V(\alpha^{\mathrm{BOOL}}) \in \{0, 1\}$
- $V(\alpha^{\sigma \rightarrow \tau}) \in T^S$, which is the set of all functions from the denotations of expressions of type σ to the denotations of expressions of type τ.

Write $f : X \mapsto Y$ for a function which takes arguments from X and maps them to values in Y.

$V(\alpha^{\mathrm{IND} \rightarrow \mathrm{BOOL}})$

$V(\mathrm{dog}) \in \{0, 1\}^D = \{f \mid f : D \mapsto \{0, 1\}\}$

Functions in the model

Types are pretty much the same as arities.

- $V(\alpha^{\text{IND}}) \in D$
- $V(\alpha^{\text{BOOL}}) \in \{0, 1\}$
- $V(\alpha^{\sigma \to \tau}) \in T^S$, which is the set of all functions from the denotations of expressions of type σ to the denotations of expressions of type τ.

Write $f : X \mapsto Y$ for a function which takes arguments from X and maps them to values in Y.

$V(\alpha^{\text{IND} \to \text{BOOL}})$

$V(\text{dog}) \in \{0, 1\}^D = \{f \mid f : D \mapsto \{0, 1\}\}$

Functions in the model

Types are pretty much the same as arities.

- $V(\alpha^{\text{IND}}) \in D$
- $V(\alpha^{\text{BOOL}}) \in \{0, 1\}$
- $V(\alpha^{\sigma \to \tau}) \in T^S$, which is the set of all functions from the denotations of expressions of type σ to the denotations of expressions of type τ.

Write $f : X \mapsto Y$ for a function which takes arguments from X and maps them to values in Y.

$V(\alpha^{\text{IND} \to \text{BOOL}})$

$V(\text{dog}) \in \{0, 1\}^D = \{f \mid f : D \mapsto \{0, 1\}\}$

Functions in the model

Types are pretty much the same as arities.

- $V(\alpha^{\text{IND}}) \in D$
- $V(\alpha^{\text{BOOL}}) \in \{0, 1\}$
- $V(\alpha^{\sigma \rightarrow \tau}) \in T^S$, which is the set of all functions from the denotations of expressions of type σ to the denotations of expressions of type τ.

Write $f : X \mapsto Y$ for a function which takes arguments from X and maps them to values in Y.

$V(\alpha^{\text{IND} \rightarrow \text{BOOL}})$

$V(\text{dog}) \in \{0, 1\}^D = \{f \mid f : D \mapsto \{0, 1\}\}$

Functions in the model

Types are pretty much the same as arities.

- $V(\alpha^{\text{IND}}) \in D$
- $V(\alpha^{\text{BOOL}}) \in \{0, 1\}$
- $V(\alpha^{\sigma \to \tau}) \in T^S$, which is the set of all functions from the denotations of expressions of type σ to the denotations of expressions of type τ.

Write $f : X \mapsto Y$ for a function which takes arguments from X and maps them to values in Y.

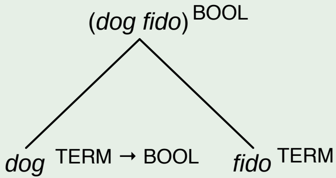
$V(\alpha^{\text{IND}\to\text{BOOL}})$

$V(\text{dog}) \in \{0, 1\}^D = \{f \mid f : D \mapsto \{0, 1\}\}$

Function Application

- If α is of type $\sigma \rightarrow \tau$ and β is of type σ, then $(\alpha\ \beta)$ is of type τ.
- NB funny syntax (from lambda calculus); more common is $\alpha(\beta)$, but we're going to follow the $(\alpha\ \beta)$ notation in NLTK.

Derivation of Typed Expression



Denotation of Function Expressions

- Every set A corresponds to a characteristic function f_A such that $f_A(x) = 1$ iff $x \in A$.
- Equivalently, define $A = \{x \mid f_A(x) = 1\}$.
- So given the denotation $A \subseteq D$ of some unary predicate, we have a corresponding $f_A \in \{0, 1\}^D$.

dog as a function expression

$$V(\text{dog}) = \begin{bmatrix} d_1 & \rightarrow & 0 \\ d_2 & \rightarrow & 0 \\ d_3 & \rightarrow & 0 \\ d_4 & \rightarrow & 1 \end{bmatrix}$$

Denotation of Function Expressions

- Every set A corresponds to a characteristic function f_A such that $f_A(x) = 1$ iff $x \in A$.
- Equivalently, define $A = \{x \mid f_A(x) = 1\}$.
- So given the denotation $A \subseteq D$ of some unary predicate, we have a corresponding $f_A \in \{0, 1\}^D$.

dog as a function expression

$$V(\text{dog}) = \begin{bmatrix} d_1 & \rightarrow & 0 \\ d_2 & \rightarrow & 0 \\ d_3 & \rightarrow & 0 \\ d_4 & \rightarrow & 1 \end{bmatrix}$$

Denotation of Function Expressions

- Every set A corresponds to a characteristic function f_A such that $f_A(x) = 1$ iff $x \in A$.
- Equivalently, define $A = \{x \mid f_A(x) = 1\}$.
- So given the denotation $A \subseteq D$ of some unary predicate, we have a corresponding $f_A \in \{0, 1\}^D$.

dog as a function expression

$$V(\text{dog}) = \begin{bmatrix} d_1 & \rightarrow & 0 \\ d_2 & \rightarrow & 0 \\ d_3 & \rightarrow & 0 \\ d_4 & \rightarrow & 1 \end{bmatrix}$$

Denotation of Function Expressions

- Every set A corresponds to a characteristic function f_A such that $f_A(x) = 1$ iff $x \in A$.
- Equivalently, define $A = \{x \mid f_A(x) = 1\}$.
- So given the denotation $A \subseteq D$ of some unary predicate, we have a corresponding $f_A \in \{0, 1\}^D$.

dog as a function expression

$$V(\text{dog}) = \begin{bmatrix} d_1 & \rightarrow & 0 \\ d_2 & \rightarrow & 0 \\ d_3 & \rightarrow & 0 \\ d_4 & \rightarrow & 1 \end{bmatrix}$$

Denotation of Function Expressions

- Every set A corresponds to a characteristic function f_A such that $f_A(x) = 1$ iff $x \in A$.
- Equivalently, define $A = \{x \mid f_A(x) = 1\}$.
- So given the denotation $A \subseteq D$ of some unary predicate, we have a corresponding $f_A \in \{0, 1\}^D$.

dog as a function expression

$$V(\text{dog}) = \begin{bmatrix} d_1 & \rightarrow & 0 \\ d_2 & \rightarrow & 0 \\ d_3 & \rightarrow & 0 \\ d_4 & \rightarrow & 1 \end{bmatrix}$$

Evaluating Function Application

$$M \models (\alpha^{\text{IND}\rightarrow\text{BOOL}}\ \beta^{\text{IND}}) \text{ iff } V(\alpha)(V(\beta)) = 1$$

Evaluating dog as a function expression

$V(\text{dog})(V(\text{kim})) = 0$
$V(\text{dog})(V(\text{fido})) = 1$

Python Dictionaries

- Accessing items by their names, e.g., dictionary
- Defining entries:

```
>>> d = {}
>>> d['colourless'] = 'adj'
>>> d['furiously'] = 'adv'
>>> d['ideas'] = 'n'
```

- {} is an empty dictionary; 'colourless' is a key; 'adj' is a value.
- Accessing:

```
>>> d.keys()
['furiously', 'colourless', 'ideas']
>>> d['ideas']
'n'
>>> d
{'furiously': 'adv', 'colourless': 'adj', 'ideas': '
```

Python Dictionaries

- Accessing items by their names, e.g., dictionary
- Defining entries:

```
>>> d = {}
>>> d['colourless'] = 'adj'
>>> d['furiously'] = 'adv'
>>> d['ideas'] = 'n'
```

- `{}` is an empty dictionary; `'colourless'` is a key; `'adj'` is a value.
- Accessing:

```
>>> d.keys()
['furiously', 'colourless', 'ideas']
>>> d['ideas']
'n'
>>> d
{'furiously': 'adv', 'colourless': 'adj', 'ideas': '
```

Python Dictionaries

- Accessing items by their names, e.g., dictionary
- Defining entries:

```
>>> d = {}
>>> d['colourless'] = 'adj'
>>> d['furiously'] = 'adv'
>>> d['ideas'] = 'n'
```

- `{}` is an empty dictionary; `'colourless'` is a key; `'adj'` is a value.
- Accessing:

```
>>> d.keys()
['furiously', 'colourless', 'ideas']
>>> d['ideas']
'n'
>>> d
{'furiously': 'adv', 'colourless': 'adj', 'ideas': '
```

Python Dictionaries

- Accessing items by their names, e.g., dictionary
- Defining entries:

```
>>> d = {}
>>> d['colourless'] = 'adj'
>>> d['furiously'] = 'adv'
>>> d['ideas'] = 'n'
```

- `{}` is an empty dictionary; `'colourless'` is a key; `'adj'` is a value.
- Accessing:

```
>>> d.keys()
['furiously', 'colourless', 'ideas']
>>> d['ideas']
'n'
>>> d
{'furiously': 'adv', 'colourless': 'adj', 'ideas': '
```

Functions as Dictionaries

- We can use dictionaries to implement functions; the arguments are the keys and the values are the . . . values!
- dog again — we use strings `'d1'` etc for the keys (representing individuals in *D*), and the built-in Boolean types `True` and `False` as values.

```
>>> dog = {}
>>> dog['d1'] = False
>>> dog['d2'] = False
>>> dog['d3'] = False
>>> dog['d4'] = True
>>> dog
{'d4': True, 'd2': False, 'd3': False, 'd1': False}
```

Exercise

Define the function corresponding to the set value of the predicate girl.

Functions as Dictionaries

- We can use dictionaries to implement functions; the arguments are the keys and the values are the . . . values!
- dog again — we use strings `'d1'` etc for the keys (representing individuals in *D*), and the built-in Boolean types `True` and `False` as values.

```
>>> dog = {}
>>> dog['d1'] = False
>>> dog['d2'] = False
>>> dog['d3'] = False
>>> dog['d4'] = True
>>> dog
{'d4': True, 'd2': False, 'd3': False, 'd1': False}
```

Exercise

Define the function corresponding to the set value of the predicate girl.

Functions as Dictionaries

- We can use dictionaries to implement functions; the arguments are the keys and the values are the . . . values!
- dog again — we use strings `'d1'` etc for the keys (representing individuals in D), and the built-in Boolean types `True` and `False` as values.

```
>>> dog = {}
>>> dog['d1'] = False
>>> dog['d2'] = False
>>> dog['d3'] = False
>>> dog['d4'] = True
>>> dog
{'d4': True, 'd2': False, 'd3': False, 'd1': False}
```

Exercise

Define the function corresponding to the set value of the predicate girl.

Valuations in `nltk.sem`, 1


```
>>> from nltk.sem import Valuation
>>> val = Valuation({'Mary': 'd2', 'Fido': 'd4', 'do
>>> val
{'Fido': 'd4', 'Mary': 'd2', 'dog': {'d4': True}}
>>> val['dog']
{'d4': True}
>>> val['dog'][val['Fido']]
True
>>> val['dog'][val['Mary']]
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
KeyError: 'd2'
```

- Omitting the `False` entries:
    - more succinct, but we need a wrapper to get the negative cases.

Valuations in `nltk.sem`, 1


```
>>> from nltk.sem import Valuation
>>> val = Valuation({'Mary': 'd2', 'Fido': 'd4', 'do
>>> val
{'Fido': 'd4', 'Mary': 'd2', 'dog': {'d4': True}}
>>> val['dog']
{'d4': True}
>>> val['dog'][val['Fido']]
True
>>> val['dog'][val['Mary']]
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
KeyError: 'd2'
```

- Omitting the `False` entries:
  - more succinct, but we need a wrapper to get the negative cases.