Semantics and Pragmatics of NLP
Propositional Logic, Predicates and Functions

Alex Lascarides & Ewan Klein

School of Informatics
University of Edinburgh

10 January 2008
1 Motivation

2 Propositional Logic

3 Predicates and Functions

4 Implementing Function expressions in NLTK
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.
A Vocabulary (aka lexicon)
- determines what we can talk about

Syntax
- Uses vocabulary and syntactic rules to define the set of well-formed formulas (WFFs)
- determines how we can talk about things

Semantics
- Compositional (uses recursion)
- Truth, Satisfaction, Entailment.
Basic expressions:

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

Rules of syntax:

1. Every propositional variable is a well-formed formula (WFF).
2. If $\phi$ and $\psi$ are WFFs, then so are: $\neg\phi$, $(\phi \land \psi)$, $(\phi \lor \psi)$, $(\phi \rightarrow \psi)$. 
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 $\phi$, we write $M \models \phi$ to mean $\phi$ is true in model $M$. 
Recursive definition of truth in a model $M = V$.

- $M \models p_i$ \iff $V(p_i) = 1$
- $M \models \neg \phi$ \iff $M \not\models \phi$
- $M \models \phi \land \psi$ \iff $M \models \phi$ and $M \models \psi$
- $M \models \phi \lor \psi$ \iff $M \models \phi$ or $M \models \psi$
- $M \models \phi \rightarrow \psi$ \iff $M \not\models \phi$ or $M \models \psi$
Adding Predicates to the Language

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

<table>
<thead>
<tr>
<th>Terms</th>
<th>Unary Predicates</th>
<th>Binary Predicates</th>
</tr>
</thead>
<tbody>
<tr>
<td>john</td>
<td>dog</td>
<td>chase</td>
</tr>
<tr>
<td>mary</td>
<td>girl</td>
<td>kiss</td>
</tr>
<tr>
<td>kim</td>
<td>run</td>
<td></td>
</tr>
<tr>
<td>fido</td>
<td>smile</td>
<td></td>
</tr>
</tbody>
</table>

**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).
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

\[ M = \langle D, V \rangle, \text{ where:} \]

\[ D = \{ d_1, d_2, d_3, d_4 \} \]

\[ V(\text{john}) = d_1 \]
\[ V(\text{mary}) = d_2 \]
\[ V(\text{kim}) = d_3 \]
\[ V(\text{fido}) = d_4 \]
\[ V(\text{chase}) = \{(d_2, d_3), (d_3, d_4)\} \]
\[ V(\text{kiss}) = \{(d_2, d_1), (d_1, d_2)\} \]

\[ M \models R(\tau_1, \ldots, \tau_n) \text{ iff } (V(\tau_1), \ldots, V(\tau_n)) \in V(R) \]
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 $\sigma$ to yield expressions of type $\tau$.

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 $\sigma$ to yield expressions of type $\tau$.

**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 $\sigma$ to yield expressions of type $\tau$.

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 $\sigma$ to yield expressions of type $\tau$.

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.

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

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.

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

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

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

$V(\text{dog}) \in \{0, 1\}^D = \{f \mid f : D \mapsto \{0, 1\}\}$
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 \( \sigma \) to the denotations of expressions of type \( \tau \).

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

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

\[
V(\text{dog}) \in \{0, 1\}^D = \{f \mid f : D \mapsto \{0, 1\}\}
\]
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 $\sigma$ to the denotations of expressions of type $\tau$.

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.

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

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

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

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

- If $\alpha$ is of type $\sigma \rightarrow \tau$ and $\beta$ is of type $\sigma$, then $(\alpha \beta)$ is of type $\tau$.
- 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

$$\begin{align*}
\text{(dog fido)} & \quad \text{BOOL} \\
\text{dog} & \quad \text{TERM} \rightarrow \text{BOOL} \\
\text{fido} & \quad \text{TERM}
\end{align*}$$
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**

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

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

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

\[ V(\text{dog})(V(\text{kim})) = 0 \]
\[ V(\text{dog})(V(\text{fido})) = 1 \]
Accessing items by their names, e.g., dictionary

Defining entries:

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

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

Accessing:

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

- Accessing items by their names, e.g., dictionary
- Defining entries:
  ```python
  >>> d = {}
  >>> d['colourless'] = 'adj'
  >>> d['furiously'] = 'adv'
  >>> d['ideas'] = 'n'
  ```
- `{}` is an empty dictionary; 'colourless' is a key; 'adj' is a value.
- Accessing:
  ```python
  >>> d.keys()
  ['furiously', 'colourless', 'ideas']
  >>> d['ideas']
  'n'
  >>> d
  {'furiously': 'adv', 'colourless': 'adj', 'ideas': 'n'}
  ```
Accessing items by their names, e.g., dictionary

Defining entries:

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

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

Accessing:

```python
>>> d.keys()
['furiously', 'colourless', 'ideas']
>>> d['ideas']
'n'
>>> d
{'furiously': 'adv', 'colourless': 'adj', 'ideas': 'n'}
```
Accessing items by their names, e.g., dictionary

Defining entries:

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

{} is an empty dictionary; 'colourless' is a **key**; 'adj' is a **value**.

Accessing:

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

```python
>>> 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.
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.

```python
>>> 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.
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.

```python
>>> 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

```python
>>> from nltk.sem import Valuation
>>> val = Valuation({'Mary': 'd2', 'Fido': 'd4', 'dog': {'d4': True}})
>>> 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

```python
>>> from nltk.sem import Valuation
>>> val = Valuation({'Mary': 'd2', 'Fido': 'd4', 'dog': True})
>>> val
{'Fido': 'd4', 'Mary': 'd2', 'dog': True}
>>> val['dog']
{'d4': True}
>>> val['dog'][val['Fido']]
True
>>> val['dog'][val['Mary']]  # KeyError: 'd2'
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.