Bridging the Gap between Language and the World

In order to link NL to a knowledge base, we might want to design a formal way to represent meaning. Desiderata for a meaning representation language:

- Represent the state of the world, i.e., a knowledge base
- Query the knowledge base (e.g., verify that a statement is true, or answer a question)
- Handle ambiguity, vagueness, and non-canonical forms
- Support inference and reasoning

Example: "I wanna eat someplace that's close to UW" and "something not too spicy"
In order to link NL to a knowledge base, we might want to design a formal way to represent meaning. Desiderata for a meaning representation language:

▶ represent the state of the world, i.e., a knowledge base
Bridging the Gap between Language and the World

In order to link NL to a knowledge base, we might want to design a formal way to represent meaning. Desiderata for a meaning representation language:

▶ represent the state of the world, i.e., a knowledge base
▶ query the knowledge base (e.g., verify that a statement is true, or answer a question)
In order to link NL to a knowledge base, we might want to design a formal way to represent meaning. Desiderata for a **meaning representation language**:

- represent the state of the world, i.e., a knowledge base
- query the knowledge base (e.g., verify that a statement is true, or answer a question)
- handle ambiguity, vagueness, and non-canonical forms
  - “I wanna eat someplace that’s close to UW”
  - “something not too spicy”
In order to link NL to a knowledge base, we might want to design a formal way to represent meaning. Desiderata for a meaning representation language:

- represent the state of the world, i.e., a knowledge base
- query the knowledge base (e.g., verify that a statement is true, or answer a question)
- handle ambiguity, vagueness, and non-canonical forms
  - “I wanna eat someplace that’s close to UW”
  - “something not too spicy”
- support inference and reasoning
  - “can Karen eat at Schultzy’s?”
Bridging the Gap between Language and the World

In order to link NL to a knowledge base, we might want to design a formal way to represent meaning.

Desiderata for a meaning representation language:

▶ represent the state of the world, i.e., a knowledge base
▶ query the knowledge base (e.g., verify that a statement is true, or answer a question)
▶ handle ambiguity, vagueness, and non-canonical forms
  ▶ “I wanna eat someplace that’s close to UW”
  ▶ “something not too spicy”
▶ support inference and reasoning
  ▶ “can Karen eat at Schultzy’s?”

Eventually (but not today):

▶ deal with non-literal meanings
▶ expressiveness across a wide range of subject matter
A (Tiny) World Model

- **Domain:** Adrian, Brook, Chris, Donald, Schultzy’s Sausage, Din Tai Fung, Banana Leaf, American, Chinese, Thai
- **Property:** Din Tai Fung has a long wait, Schultzy’s is noisy; Alice, Bob, and Charles are human
- **Relations:** Schultzy’s serves American, Din Tai Fung serves Chinese, and Banana Leaf serves Thai

Simple questions are easy:

- Is Schultzy’s noisy?
- Does Din Tai Fung serve Thai?
A (Tiny) World Model

- **Domain:** Adrian, Brook, Chris, Donald, Schultzy’s Sausage, Din Tai Fung, Banana Leaf, American, Chinese, Thai
  \[a, b, c, d, ss, dtf, bl, am, ch, th\]

- **Property:** Din Tai Fung has a long wait, Schultzy’s is noisy; Alice, Bob, and Charles are human
  \[Longwait = \{dtf\}, Noisy = \{ss\}, Human = \{a, b, c\}\]

- **Relations:** Schultzy’s serves American, Din Tai Fung serves Chinese, and Banana Leaf serves Thai
  \[Serves = \{(ss, am), (dtf, ch), (bl, th)\}, Likes = \{(a, ss), (a, dtf), \ldots\}\]

Simple questions are easy:

- Is Schultzy’s noisy?
- Does Din Tai Fung serve Thai?
A Quick Tour of First-Order Logic

- **Term**: a constant \(ss\) or a variable
- **Formula**: defined inductively . . .
  - If \(R\) is an n-ary relation and \(t_1, \ldots, t_n\) are terms, then \(R(t_1, \ldots, t_n)\) is a formula.
  - If \(\phi\) is a formula, then its negation, \(\neg\phi\), is a formula.
  - If \(\phi\) and \(\psi\) are formulas, then binary logical connectives can be used to create formulas:
    - \(\phi \land \psi\)
    - \(\phi \lor \psi\)
    - \(\phi \rightarrow \psi\)
    - \(\phi \oplus \psi\)
  - If \(\phi\) is a formula and \(v\) is a variable, then quantifiers can be used to create formulas:
    - Universal quantifier: \(\forall v, \phi\)
    - Existential quantifier: \(\exists v, \phi\)

Note: Leaving out functions, because we don’t need them in a single lecture on FOL for NL.
1. Schultzy’s is not loud
2. Some human likes Chinese
3. If a person likes Thai, then they aren’t friends with Donald
4. \( \forall x, Restaurant(x) \Rightarrow (Longwait(x) \lor \neg Likes(a, x)) \)
5. \( \forall x, \exists y, \neg Likes(x, y) \)
6. \( \exists y, \forall x, \neg Likes(x, y) \)
1. Schultzy’s is not loud \( \neg \text{Noisy}(ss) \)

2. Some human likes Chinese

3. If a person likes Thai, then they aren’t friends with Donald

4. \( \forall x, \text{Restaurant}(x) \Rightarrow (\text{Longwait}(x) \lor \neg \text{Likes}(a, x)) \)

5. \( \forall x, \exists y, \neg \text{Likes}(x, y) \)

6. \( \exists y, \forall x, \neg \text{Likes}(x, y) \)
1. Schultzy’s is not loud \( \neg \text{Noisy}(ss) \)
2. Some human likes Chinese \( \exists x, \text{Human}(x) \land \text{Likes}(x, \text{ch}) \)
3. If a person likes Thai, then they aren’t friends with Donald
4. \( \forall x, \text{Restaurant}(x) \Rightarrow (\text{Longwait}(x) \lor \neg \text{Likes}(a, x)) \)
5. \( \forall x, \exists y, \neg \text{Likes}(x, y) \)
6. \( \exists y, \forall x, \neg \text{Likes}(x, y) \)
1. Schultzy’s is not loud
   \( \neg \text{Noisy}(ss) \)

2. Some human likes Chinese
   \( \exists x, \text{Human}(x) \land \text{Likes}(x, \text{ch}) \)

3. If a person likes Thai, then they aren’t friends with Donald
   \( \forall x, \text{Human}(x) \land \text{Likes}(x, \text{th}) \Rightarrow \neg \text{Friends}(x, d) \)

4. \( \forall x, \text{Restaurant}(x) \Rightarrow (\text{Longwait}(x) \lor \neg \text{Likes}(a, x)) \)

5. \( \forall x, \exists y, \neg \text{Likes}(x, y) \)

6. \( \exists y, \forall x, \neg \text{Likes}(x, y) \)
1. Schultzy’s is not loud  \( \neg \text{Noisy}(ss) \)

2. Some human likes Chinese  \( \exists x, \text{Human}(x) \land \text{Likes}(x, \text{ch}) \)

3. If a person likes Thai, then they aren’t friends with Donald
   \( \forall x, \text{Human}(x) \land \text{Likes}(x, \text{th}) \Rightarrow \neg \text{Friends}(x, d) \)

4. \( \forall x, \text{Restaurant}(x) \Rightarrow (\text{Longwait}(x) \lor \neg \text{Likes}(a, x)) \)
   Every restaurant has a long wait or is disliked by Adrian.

5. \( \forall x, \exists y, \neg \text{Likes}(x, y) \)

6. \( \exists y, \forall x, \neg \text{Likes}(x, y) \)
Translating Between FOL and NL

1. Schultzy’s is not loud \( \neg \text{Noisy}(ss) \)

2. Some human likes Chinese \( \exists x, \text{Human}(x) \land \text{Likes}(x, ch) \)

3. If a person likes Thai, then they aren’t friends with Donald
\[ \forall x, \text{Human}(x) \land \text{Likes}(x, th) \Rightarrow \neg \text{Friends}(x, d) \]

4. Every restaurant has a long wait or is disliked by Adrian.
\[ \forall x, \text{Restaurant}(x) \Rightarrow (\text{Longwait}(x) \lor \neg \text{Likes}(a, x)) \]

5. Everybody has something they don’t like.
\[ \forall x, \exists y, \neg \text{Likes}(x, y) \]

6. Every restaurant has a long wait or is disliked by Adrian.
\[ \exists y, \forall x, \neg \text{Likes}(x, y) \]
Translating Between FOL and NL

1. Schultzy’s is not loud
   \( \neg \text{Noisy}(ss) \)

2. Some human likes Chinese
   \( \exists x, \text{Human}(x) \land \text{Likes}(x, \text{ch}) \)

3. If a person likes Thai, then they aren’t friends with Donald
   \( \forall x, \text{Human}(x) \land \text{Likes}(x, \text{th}) \Rightarrow \neg \text{Friends}(x, d) \)

4. \( \forall x, \text{Restaurant}(x) \Rightarrow (\text{Longwait}(x) \lor \neg \text{Likes}(a, x)) \)
   Every restaurant has a long wait or is disliked by Adrian.

5. \( \forall x, \exists y, \neg \text{Likes}(x, y) \)
   Everybody has something they don’t like.

6. \( \exists y, \forall x, \neg \text{Likes}(x, y) \)
   There exists something that nobody likes.
Logical Semantics
(Montague, 1970)

The denotation of a NL sentence is the set of conditions that must hold in the (model) world for the sentence to be true.

Every restaurant has a long wait or Adrian doesn’t like it.

is true if and only if

$$\forall x, \text{Restaurant}(x) \Rightarrow (\text{Longwait}(x) \lor \neg \text{Likes}(a, x))$$

is true.

This is sometimes called the **logical form** of the NL sentence.
The Principle of Compositionality

The meaning of a NL phrase is determined by the meanings of its sub-phrases.
The Principle of Compositionality

The meaning of a NL phrase is determined by the meanings of its sub-phrases.

I.e., semantics is derived from syntax.
The Principle of Compositionality

The meaning of a NL phrase is determined by the meanings of its sub-phrases.

I.e., semantics is derived from syntax.

We need a way to express semantics of phrases, and compose them together!
λ-Calculus

(Much more powerful than what we’ll see today; ask your PL friends.)

Informally, two extensions:

- **λ-abstraction** is another way to “scope” variables.
  - If \( \phi \) is a FOL formula and \( v \) is a variable, then \( \lambda v.\phi \) is a \( \lambda \)-term, meaning: an unnamed function from values (of \( v \)) to formulas (usually involving \( v \)).

- **application** of such functions: if we have \( \lambda v.\phi \) and \( \psi \), then \([\lambda v.\phi](\psi)\) is a formula.
  - It can be **reduced** by substituting \( \psi \) in for every instance of \( v \) in \( \phi \).
λ-Calculus

(Much more powerful than what we’ll see today; ask your PL friends.)

Informally, two extensions:

▶ **λ-abstraction** is another way to “scope” variables.
  ▶ If φ is a FOL formula and v is a variable, then λv.φ is a λ-term, meaning: an unnamed function from values (of v) to formulas (usually involving v)

▶ **application** of such functions: if we have λv.φ and ψ, then [λv.φ](ψ) is a formula.
  ▶ It can be **reduced** by substituting ψ in for every instance of v in φ.

Example:
λx>Likes(x, dtf) maps things to statements that they like Din Tai Fung
λ-Calculus

(Much more powerful than what we’ll see today; ask your PL friends.)

Informally, two extensions:

- **λ-abstraction** is another way to “scope” variables.
  - If \( \phi \) is a FOL formula and \( v \) is a variable, then \( \lambda v.\phi \) is a λ-term, meaning: an unnamed function from values (of \( v \)) to formulas (usually involving \( v \))

- **application** of such functions: if we have \( \lambda v.\phi \) and \( \psi \), then \([\lambda v.\phi](\psi)\) is a formula.
  - It can be reduced by substituting \( \psi \) in for every instance of \( v \) in \( \phi \).

Example:
\[
[\lambda x.\text{Likes}(x, dtf)](c) \text{ reduces to } \text{Likes}(c, dtf)
\]
λ-Calculus

(Much more powerful than what we’ll see today; ask your PL friends.)

Informally, two extensions:

- **λ-abstraction** is another way to “scope” variables.
  - If $\phi$ is a FOL formula and $v$ is a variable, then $\lambda v.\phi$ is a λ-term, meaning: an unnamed function from values (of $v$) to formulas (usually involving $v$)

- **application** of such functions: if we have $\lambda v.\phi$ and $\psi$, then $[\lambda v.\phi](\psi)$ is a formula.
  - It can be **reduced** by substituting $\psi$ in for every instance of $v$ in $\phi$.

Example:

$\lambda x.\lambda y.\text{Friends}(x, y)$ maps things $x$ to maps of things $y$ to statements that $x$ and $y$ are friends
\(\lambda\)-Calculus

(Much more powerful than what we’ll see today; ask your PL friends.)

Informally, two extensions:

- **\(\lambda\)-abstraction** is another way to “scope” variables.
  - If \(\phi\) is a FOL formula and \(v\) is a variable, then \(\lambda v.\phi\) is a \(\lambda\)-term, meaning: an unnamed function from values (of \(v\)) to formulas (usually involving \(v\)).

- **application** of such functions: if we have \(\lambda v.\phi\) and \(\psi\), then \([\lambda v.\phi](\psi)\) is a formula.
  - It can be reduced by substituting \(\psi\) in for every instance of \(v\) in \(\phi\).

Example:

\([\lambda x.\lambda y.\text{Friends}(x, y)](b)\) reduces to \(\lambda y.\text{Friends}(b, y)\)
(Much more powerful than what we’ll see today; ask your PL friends.)

Informally, two extensions:

- **λ-abstraction** is another way to “scope” variables.
  - If \( \phi \) is a FOL formula and \( v \) is a variable, then \( \lambda v.\phi \) is a λ-term, meaning: an unnamed function from values (of \( v \)) to formulas (usually involving \( v \))

- **application** of such functions: if we have \( \lambda v.\phi \) and \( \psi \), then \( [\lambda v.\phi](\psi) \) is a formula.
  - It can be reduced by substituting \( \psi \) in for every instance of \( v \) in \( \phi \).

Example:

\[
[[\lambda x.\lambda y.\text{Friends}(x,y)](b)](a) \text{ reduces to } [\lambda y.\text{Friends}(b,y)](a),
\]
which reduces to \( \text{Friends}(b,a) \)
Semantic Attachments to CFG

- **NNP** → Adrian \( \{ a \} \)
- **VBZ** → likes \( \{ \lambda f. \lambda y. \forall x f(x) \Rightarrow Likes(y, x) \} \)
- **JJ** → expensive \( \{ \lambda x. Expensive(x) \} \)
- **NNS** → restaurants \( \{ \lambda x. Restaurant(x) \} \)
- **NP** → NNP \( \{ \text{NNP.sem} \} \)
- **NP** → JJ NNS \( \{ \lambda x. JJ.sem(x) \land \text{NNS.sem}(x) \} \)
- **VP** → VBZ NP \( \{ \text{VBZ.sem}(\text{NP.sem}) \} \)
- **S** → NP VP \( \{ \text{VP.sem}(\text{NP.sem}) \} \)
Example

Adrian likes expensive restaurants.
Example

```
S : VP.sem(NP.sem)
  |
NP : NNP.sem
    |
NNP : a
      |
Adrian
  |
VP : VBZ.sem(NP.sem)
    |
NP : \lambda v. JJ.sem(v) \land NNS.sem(v)
      |
VBZ : ...
        |
likes
  |
JJ : \lambda z. Expensive(z)
      |
expensive
  |
NNS : \lambda w. Restaurant(w)
      |
restaurants
```
Example

\[
S : \text{VP.sem}(\text{NP.sem})
\]

\[
\text{NP} : \text{NNP.sem}
\]

\[
\text{NNP} : a
\]

\[
\text{Adrian}
\]

\[
\text{VP} : \text{VBZ.sem}(\text{NP.sem})
\]

\[
\text{VBZ} : \ldots
\]

\[
\text{NP} : \lambda v. \text{Expensive}(v) \land \text{Restaurant}(v)
\]

\[
\text{JJ} : \lambda z. \text{Expensive}(z)
\]

\[
\text{NNS} : \lambda w. \text{Restaurant}(w)
\]

\[
\lambda v. \left[ \begin{array}{l}
\lambda z. \text{Expensive}(z) \\
\text{JJ.sem}
\end{array} \right] (v) \land \left[ \begin{array}{l}
\lambda w. \text{Restaurant}(w) \\
\text{NNS.sem}
\end{array} \right] (v)
\]
Example

\[
\text{VP : } \text{VBZ.sem(NP.sem)}
\]

\[
\text{VBZ : } \lambda f. \lambda y. \forall x f(x) \Rightarrow \text{Likes}(y, x) \quad \text{NP : } \lambda v. \text{Expensive}(v) \land \text{Restaurant}(v)
\]

likes

expensive restaurants
Example

\[ \text{VP} : \lambda y. \forall x, \text{Expensive}(x) \land \text{Restaurant}(x) \Rightarrow \text{Likes}(y, x) \]

\[ \text{VBZ} : \lambda f. \lambda y. \forall x f(x) \Rightarrow \text{Likes}(y, x) \]

\[ \text{NP} : \lambda v. \text{Expensive}(v) \land \text{Restaurant}(v) \]

\[ \text{likes} \]

\[ \text{expensive restaurants} \]

\[ \lambda f. \lambda y. \forall x f(x) \Rightarrow \text{Likes}(y, x) \]

\[ \lambda v. \text{Expensive}(v) \land \text{Restaurant}(v) \]

\[ \lambda y. \forall x, \text{Expensive}(x) \land \text{Restaurant}(x) \Rightarrow \text{Likes}(y, x) \]
Example

$S : \text{VP.sem}(\text{NP.sem})$

$\text{NP} : \text{NNP.sem} \quad \text{VP} : \lambda y. \forall x, \text{Expensive}(x) \land \text{Restaurant}(x) \Rightarrow \text{Likes}(y, x)$

$\text{NNP} : a$

$\text{Adrian}$

likes expensive restaurants
Example

\[ S : \text{VP.sem(NP.sem)} \]

\[ \begin{align*}
\text{NP} & : \ a \\
\text{NNP} & : \ a \\
\text{Adrian} & \\
\text{VP} & : \lambda y. \forall x, \text{Expensive}(x) \land \text{Restaurant}(x) \Rightarrow \text{Likes}(y, x) \\
\text{likes expensive restaurants} & 
\end{align*} \]
Example

\[ S : \forall x, \text{Expensive}(x) \land \text{Restaurant}(x) \Rightarrow \text{Likes}(a, x) \]

NP : \( a \)  

VP : \( \lambda y. \forall x, \text{Expensive}(x) \land \text{Restaurant}(x) \Rightarrow \text{Likes}(y, x) \)

NNP : \( a \)  

Adrian

\[ \forall x, \text{Expensive}(x) \land \text{Restaurant}(x) \Rightarrow \text{Likes}(a, x) \]
“The boy wants to visit New York City.”

Designed for (1) annotation-ability and (2) eventual use in machine translation.
Combinatory Categorial Grammar
(Steedman, 2000)

CCG is a grammatical formalism that is well-suited for tying together syntax and semantics.

Formally, it is more powerful than CFG—it can represent context-sensitive languages (which we do not have time to define formally).
CCG Types

Instead of the "N" of CFGs, CCGs can have an infinitely large set of structured categories (called **types**).

- **Primitive types:** typically S, NP, N, and maybe more
- **Complex types,** built with “slashes,” for example:
  - S/NP is “an S, except that it lacks an NP to the right”
  - S\NP is “an S, except that it lacks an NP to its left”
  - (S\NP)/NP is “an S, except that it lacks an NP to its right, and its left”

You can think of complex types as functions, e.g., S/NP maps NPs to Ss.
CCG Combinators

Instead of the production rules of CFGs, CCGs have a very small set of generic **combinators** that tell us how we can put types together.

Convention writes the rule differently from CFG: $X \ Y \Rightarrow Z$ means that $X$ and $Y$ combine to form a $Z$ (the “parent” in the tree).
Application Combinator

Forward \((X/Y \ Y \Rightarrow X)\) and backward \((Y \ X\backslash Y \Rightarrow X)\)
Application Combinator

Forward \((X/Y \ Y \Rightarrow X)\) and backward \((Y \ X \backslash Y \Rightarrow X)\)

```
NP
  NP/N
    the
    dog
```

42 / 56
Application Combinator

Forward \((X/Y Y \Rightarrow X)\) and backward \((Y X\backslash Y \Rightarrow X)\)

\[
\text{NP} \\
\text{NP/N} \quad \text{N} \\
\text{the} \quad \text{N/N} \quad \text{N} \\
\text{yellow} \quad \text{dog}
\]
Application Combinator

Forward \((X/Y \ Y \Rightarrow X)\) and backward \((Y \ X/Y \Rightarrow X)\)

```
S
  \[\text{NP}\]
  \[\text{N}\]
  the
dog
  \[\text{NP}/\text{N}\]
  \[\text{NP}/\text{NP}\]
  bit
  John

(S/\text{NP})/\text{NP}
```
Conjunction Combinator

\[
X \text{ and } X \Rightarrow X
\]

\[
\begin{array}{c}
\text{NP} \\
\text{NP} \quad \text{and} \quad \text{NP} \\
\text{cats} \quad \text{dogs}
\end{array}
\]
Conjunction Combinator

\[ X \text{ and } X \Rightarrow X \]
Conjunction Combinator

\[ X \text{ and } X \Rightarrow X \]
Composition Combinator

Forward \((X/Y \quad Y/Z \Rightarrow X/Z)\) and backward \((Y/Z \quad X/Y \Rightarrow X/Z)\)

\[
\begin{align*}
S & \\
\text{NP} & S/\text{NP} \\
\text{I} & (S/\text{NP})/\text{NP} \\
\text{would} & (S/\text{NP})/(S/\text{NP}) \quad \text{prefer} \\
\text{olives} & (S/\text{NP})/\text{NP}
\end{align*}
\]
Composition Combinator

Forward \((X/Y \ Y/Z \Rightarrow X/Z)\) and backward \((Y\ Z \ X\ Y \Rightarrow X\ Z)\)

\[
S
\]

\[
NP
\]

\[
I
\]

\[
(S\ NP)/(S\ NP)
\]

\[
would
\]

\[
(S\ NP)/NP
\]

\[
prefer
\]

\[
NP
\]

\[
olives
\]
Type-Raising Combinator

Forward ($X \Rightarrow Y/(Y\backslash X)$) and backward ($X \Rightarrow Y\backslash(Y/X)$)

```
S

S/NP

S/NP

S/(S\backslash NP)

NP

NP

S/(S\backslash NP)

love

I

S/(S\backslash NP)

NP

Karen

NP

hates

(S\backslash NP)/NP

NP
`
Each combinator also tells us what to do with the semantic attachments.

- **Forward application:** $X/Y : f \quad Y : g \Rightarrow X : f(g)$
- **Forward composition:**
  $X/Y : f \quad Y/Z : g \Rightarrow X/Z : \lambda x.f(g(x))$
- **Forward type-raising:** $X : g \Rightarrow Y/(Y\setminus X) : \lambda f.f(g)$
Most of the work is done in the lexicon!

Syntactic and semantic information is much more formal here.

- Slash categories define where all the syntactic arguments are expected to be
- $\lambda$-expressions define how the expected arguments get “used” to build up a FOL expression

Extensive discussion: Carpenter (1997)
Some Topics We Don’t Have Time For

- Tasks, evaluations, annotated datasets (e.g., CCGbank, Hockenmaier and Steedman, 2007)
- Learning for semantic parsing (Zettlemoyer and Collins, 2005) and CCG parsing (Clark and Curran, 2004a)
- Using CCG to represent other kinds of semantics (e.g., predicate-argument structures; Lewis and Steedman, 2014)
- Integrating continuous representations in semantic parsing (Lewis and Steedman, 2013; Krishnamurthy and Mitchell, 2013)
- Supertagging (Clark and Curran, 2004b) and making semantic parsing efficient (Lewis and Steedman, 2014)
Readings and Reminders

- Steedman (1996)
  - Or take a look at Jurafsky and Martin (2008), compositional semantics chapter
- Submit a suggestion for an exam question by Friday at 5pm.
- Your project is due March 9.
References I


Mark Steedman. A very short introduction to CCG, 1996. URL
