Computational Semantics: Lambda Calculus

Scott Farrar
CLMA, University of Washington
farrar@u.washington.edu

February 24, 2010

Semantic Analysis
Problems
One Solution: λ-Calculus
λ-calculus and FOL
λ-calculus and compositionality
The semantics of words based on syntactic category

Today's lecture

1. Semantic Analysis
   - Problems
2. One Solution: λ-Calculus
   - λ-calculus and FOL
   - λ-calculus and compositionality
3. The semantics of words based on syntactic category

Scott Farrar
CLMA, University of Washington farrar@u.washington.ed

Semantic Analysis
Problems
One Solution: λ-Calculus
λ-calculus and FOL
λ-calculus and compositionality
The semantics of words based on syntactic category

Semantic analysis

Definition

Semantic analysis is the derivation of a semantic representation from a string of words (perhaps marked up with syntactic structure). In other words, map sentences of NL onto logical formulas.

Map *Jim loves Betty* **to** $love(JIM, BETTY)$

There are several competing approaches for doing this, as there are several competing standards for the *right* semantic representation (use of event vs. relations).

Compositionality

Definition

Recall the principle of **compositionality**: the meaning of a complex expression is a function of the meaning of its parts.

The assumption is that we should be able to assign each "part" a meaning, then build larger structures, guided by the syntax of the language.

The syntax of NL and the syntax of predicate logic **are similar**, but ultimately not one-to-one compatible: translation between the two is a non-trivial task.

Event structure

A sailboat heels.

Event structure

A sailboat heels.

$\exists e\ \exists b\ [SailBoat(b) \wedge HeelingEvent(e) \wedge actor(e, b)]$

Scott Farrar
CLMA, University of Washington farrar@u.washington.ed

Semantic Analysis
Problems
One Solution: λ-Calculus
λ-calculus and FOL
λ-calculus and compositionality
The semantics of words based on syntactic category

Event structure

A sailboat heels.

$\exists e\ \exists b\ [SailBoat(b) \wedge HeelingEvent(e) \wedge actor(e, b)]$

My sailboat is on the bottom.

Event structure

A sailboat heels.

$\exists e\ \exists b\ [SailBoat(b) \wedge HeelingEvent(e) \wedge actor(e, b)]$

My sailboat is on the bottom.

$\exists e\ [SpatialLocating(e) \wedge theme(e, MYSB) \wedge loc(e, SEAFLOOR)]$

Scott Farrar
CLMA, University of Washington farrar@u.washington.ed

Semantic Analysis
Problems
One Solution: λ-Calculus
λ-calculus and FOL
λ-calculus and compositionality
The semantics of words based on syntactic category

Semantic attachments

Consider the problem of two-place predicates in a non-event-style semantics: we need to map *Jim loves Betty* to something like:

$$love(JIM, BETTY)$$

.

Scott Farrar
CLMA, University of Washington farrar@u.washington.ed

Semantic Analysis
Problems
One Solution: λ-Calculus
λ-calculus and FOL
λ-calculus and compositionality
The semantics of words based on syntactic category

Semantic attachments

Consider the problem of two-place predicates in a non-event-style semantics: we need to map *Jim loves Betty* to something like:

$$love(JIM, BETTY)$$

.

Let's assume strict compositionality and say that the meaning of each syntactic constituent contributes to the meaning of the parent constituent. We could come up with something like **XP.sem** to stand for the semantics of some constituent XP.

Semantic attachments

Consider the problem of two-place predicates in a non-event-style semantics: we need to map *Jim loves Betty* to something like:

$$love(JIM, BETTY)$$

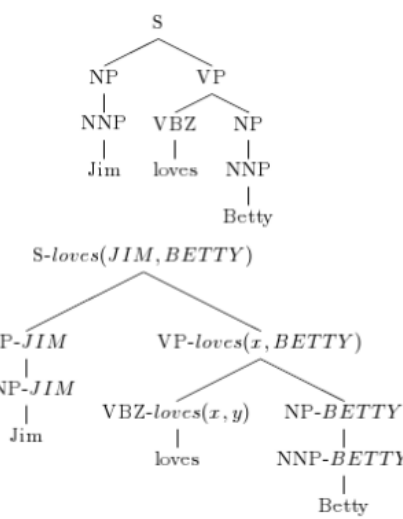
.

Let's assume strict compositionality and say that the meaning of each syntactic constituent contributes to the meaning of the parent constituent. We could come up with something like **XP.sem** to stand for the semantics of some constituent XP.

Definition

Semantic attachment refers to the adornment of phrase structure rules with such semantic information.

Semantic attachments



Example: Semantic attachments

Assume that the + symbol stands for the compositionality operator:

$$loves(JIM, BETTY)$$

Example: Semantic attachments

Assume that the + symbol stands for the compositionality operator:

- S.sem = NP.sem + VP.sem

$$loves(JIM, BETTY)$$

Example: Semantic attachments

Assume that the + symbol stands for the compositionality operator:

- S.sem = NP.sem + VP.sem
- VP.sem = V.sem + NP.sem

$$loves(JIM, BETTY)$$

Scott Farrar
CLMA, University of Washington farrar@u.washington.ed

Semantic Analysis
Problems
One Solution: λ-Calculus
λ-calculus and FOL
λ-calculus and compositionality
The semantics of words based on syntactic category

Example: Semantic attachments

Assume that the + symbol stands for the compositionality operator:

- S.sem = NP.sem + VP.sem
- VP.sem = V.sem + NP.sem
- V.sem = love.sem

$$loves(JIM, BETTY)$$

Scott Farrar
CLMA, University of Washington farrar@u.washington.ed

Semantic Analysis
Problems
One Solution: λ-Calculus
λ-calculus and FOL
λ-calculus and compositionality
The semantics of words based on syntactic category

Example: Semantic attachments

Assume that the + symbol stands for the compositionality operator:

- S.sem = NP.sem + VP.sem
- VP.sem = V.sem + NP.sem
- V.sem = love.sem
- love.sem = $love(x, y)$

$$loves(JIM, BETTY)$$

Scott Farrar
CLMA, University of Washington farrar@u.washington.ed

Semantic Analysis
Problems
One Solution: λ-Calculus
λ-calculus and FOL
λ-calculus and compositionality
The semantics of words based on syntactic category

Example: Semantic attachments

Assume that the + symbol stands for the compositionality operator:

- S.sem = NP.sem + VP.sem
- VP.sem = V.sem + NP.sem
- V.sem = love.sem
- love.sem = $love(x, y)$
- NP.sem = NNP.sem

$$loves(JIM, BETTY)$$

Example: Semantic attachments

Assume that the + symbol stands for the compositionality operator:

- S.sem = NP.sem + VP.sem
- VP.sem = V.sem + NP.sem
- V.sem = love.sem
- love.sem = $love(x, y)$
- NP.sem = NNP.sem
- NNP.sem = Betty.sem or Jim.sem

$$loves(JIM, BETTY)$$

Example: Semantic attachments

Assume that the + symbol stands for the compositionality operator:

- S.sem = NP.sem + VP.sem
- VP.sem = V.sem + NP.sem
- V.sem = love.sem
- love.sem = $love(x, y)$
- NP.sem = NNP.sem
- NNP.sem = Betty.sem or Jim.sem
- Betty.sem = $BETTY$

$$loves(JIM, BETTY)$$

Example: Semantic attachments

Assume that the + symbol stands for the compositionality operator:

- S.sem = NP.sem + VP.sem
- VP.sem = V.sem + NP.sem
- V.sem = love.sem
- love.sem = $love(x, y)$
- NP.sem = NNP.sem
- NNP.sem = Betty.sem or Jim.sem
- Betty.sem = $BETTY$
- Jim.sem = JIM

$$loves(JIM, BETTY)$$

Analysis problem

But what about other examples:

Scott Farrar
CLMA, University of Washington farrar@u.washington.ed

Semantic Analysis
Problems
One Solution: λ-Calculus
λ-calculus and FOL
λ-calculus and compositionality
The semantics of words based on syntactic category

Analysis problem

But what about other examples:

- *Betty is loved by Jim.*

Scott Farrar
CLMA, University of Washington farrar@u.washington.ed

Analysis problem

But what about other examples:

- *Betty is loved by Jim.*
- *It's Jim who loves Betty.*

Analysis problem

But what about other examples:

- *Betty is loved by Jim.*
- *It's Jim who loves Betty.*
- *Betty is the one loved by Jim.*

Analysis problem

But what about other examples:

- *Betty is loved by Jim.*
- *It's Jim who loves Betty.*
- *Betty is the one loved by Jim.*

All clues to how the semantic representation might look are found in the syntactic structure of NL. All this, without even considering the ambiguity problem.

Scott Farrar CLMA, University of Washington farrar@u.washington.ed

Semantic Analysis
Problems
One Solution: λ-Calculus
λ-calculus and FOL
λ-calculus and compositionality
The semantics of words based on syntactic category

Analysis problem

The **analysis problem**: there is no (elegant) way to fill in the arguments of formulas at the level of semantic representation, in a way that is consistent with the syntax. In other words, there is no formal means of combining parts into wholes in standard FOL: .

Even with passive **verbs** for example, we need to get *BETTY* to fill the second argument position of the predicate $love(x, y)$.

Scott Farrar
CLMA, University of Washington farrar@u.washington.ed

Semantic Analysis
Problems
One Solution: λ-Calculus
λ-calculus and FOL
λ-calculus and compositionality
The semantics of words based on syntactic category

Representation problem

Representation problem: no way to represent the meaning for some kinds of constituents.

Scott Farrar
CLMA, University of Washington farrar@u.washington.ed

Semantic Analysis
Problems
One Solution: λ-Calculus
λ-calculus and FOL
λ-calculus and compositionality
The semantics of words based on syntactic category

Representation problem

Representation problem: no way to represent the meaning for some kinds of constituents.

We can very easily express the meaning of full sentences in plain FOL. We can say that a sentence is true given some state of the world.
John kissed Mary is T just in case John really did kiss Mary.

Scott Farrar CLMA, University of Washington farrar@u.washington.ed

Semantic Analysis
Problems
One Solution: λ-Calculus
λ-calculus and FOL
λ-calculus and compositionality
The semantics of words based on syntactic category

Representation problem

Representation problem: no way to represent the meaning for some kinds of constituents.

We can very easily express the meaning of full sentences in plain FOL. We can say that a sentence is true given some state of the world.
John kissed Mary is T just in case John really did kiss Mary.

With standard **truth-conditional semantics**, where the truth of propositions can either be T or F, such logical expressions have a truth value. BUT...

Representation problem

What about constituents like VPs: *kissed Opra*. The semantics would something like VP.sem, or $kiss(x, OPRA)$

Scott Farrar
CLMA, University of Washington farrar@u.washington.ed

Semantic Analysis
Problems
One Solution: λ-Calculus
λ-calculus and FOL
λ-calculus and compositionality
The semantics of words based on syntactic category

Representation problem

What about constituents like VPs: *kissed Opra*. The semantics would something like VP.sem, or $kiss(x, OPRA)$

Scott Farrar
CLMA, University of Washington farrar@u.washington.ed

Semantic Analysis
Problems
One Solution: λ-Calculus
λ-calculus and FOL
λ-calculus and compositionality
The semantics of words based on syntactic category

Representation problem

What about constituents like VPs: *kissed Opra*. The semantics would something like VP.sem, or $kiss(x, OPRA)$

- But $kiss(x, OPRA)$ has no truth value. This is because there are unbound variables: x has no connection to the UD. Such **open sentences** are neither T or F.

Representation problem

What about constituents like VPs: *kissed Opra*. The semantics would something like VP.sem, or $kiss(x, OPRA)$

- But $kiss(x, OPRA)$ has no truth value. This is because there are unbound variables: x has no connection to the UD. Such **open sentences** are neither T or F.
- Intuitively however, we know what a NL predicate/VP means: e.g., ... *kissed Opra* means something like a " kissing Opra event", reguardless of who does the kissing.

Representation problem

What about constituents like VPs: *kissed Opra*. The semantics would something like VP.sem, or $kiss(x, OPRA)$

- But $kiss(x, OPRA)$ has no truth value. This is because there are unbound variables: x has no connection to the UD. Such **open sentences** are neither T or F.
- Intuitively however, we know what a NL predicate/VP means: e.g., ... *kissed Opra* means something like a " kissing Opra event", reguardless of who does the kissing.
- But we cannot express the meaning of this in FOL given our current machinery, since we'll always have an unbound variable.

Summary

In summary then, we have at least two problems for compositionality:

1. Analysis problem: No systematic way to use syntax to guide the construction of a semantic representation
2. Representation problem: Unsatisfying approach to representing the meanings of certain constituents; deriving truth values for certain kinds of constituents is ill defined.

Today's lecture

1. Semantic Analysis
   - Problems
2. One Solution: λ-Calculus
   - λ-calculus and FOL
   - λ-calculus and compositionality
3. The semantics of words based on syntactic category

back to Church

Alonzo Church created a calculus for describing arbitrary functions, called λ-**calculus**. (It was developed to give a functional foundation for mathematics.) It wasn't picked up by mathematicians, but it did become a versatile tool for computer scientists.

Scott Farrar
CLMA, University of Washington farrar@u.washington.ed

Semantic Analysis
Problems
One Solution: λ-Calculus
λ-calculus and FOL
λ-calculus and compositionality
The semantics of words based on syntactic category

back to Church

Alonzo Church created a calculus for describing arbitrary functions, called λ-**calculus**. (It was developed to give a functional foundation for mathematics.) It wasn't picked up by mathematicians, but it did become a versatile tool for computer scientists.

Remember **Lisp**? The second oldest high-level programming language, and still used today (invented by John McCarthy, 1958). Lisp (pure Lisp at least) deals exclusively with functions, and functions can be created on the fly and without names.

back to Church

Alonzo Church created a calculus for describing arbitrary functions, called λ-**calculus**. (It was developed to give a functional foundation for mathematics.) It wasn't picked up by mathematicians, but it did become a versatile tool for computer scientists.

Remember **Lisp**? The second oldest high-level programming language, and still used today (invented by John McCarthy, 1958). Lisp (pure Lisp at least) deals exclusively with functions, and functions can be created on the fly and without names.

In Lisp, this expression evaluates to an anonymous function: `(lambda` (x y) (+ x y)), read as "the pair x and y are mapped to $x + y$".

back to Church

Alonzo Church created a calculus for describing arbitrary functions, called λ-**calculus**. (It was developed to give a functional foundation for mathematics.) It wasn't picked up by mathematicians, but it did become a versatile tool for computer scientists.

Remember **Lisp**? The second oldest high-level programming language, and still used today (invented by John McCarthy, 1958). Lisp (pure Lisp at least) deals exclusively with functions, and functions can be created on the fly and without names.

In Lisp, this expression evaluates to an anonymous function: `(lambda (x y) (+ x y))`, read as "the pair x and y are mapped to $x + y$".

Semantic Analysis
Problems
One Solution: λ-Calculus
λ-calculus and FOL
λ-calculus and compositionality
The semantics of words based on syntactic category

back to Church

Alonzo Church created a calculus for describing arbitrary functions, called λ-**calculus**. (It was developed to give a functional foundation for mathematics.) It wasn't picked up by mathematicians, but it did become a versatile tool for computer scientists.

Remember **Lisp**? The second oldest high-level programming language, and still used today (invented by John McCarthy, 1958). Lisp (pure Lisp at least) deals exclusively with functions, and functions can be created on the fly and without names.

In Lisp, this expression evaluates to an anonymous function: `(lambda (x y) (+ x y))`, read as "the pair x and y are mapped to $x + y$".
Otherwise, we'd have a named function, something like: $add(x, y)$

Scott Farrar
CLMA, University of Washington farrar@u.washington.ed

Semantic Analysis
Problems
One Solution: λ-Calculus
λ-calculus and FOL
λ-calculus and compositionality
The semantics of words based on syntactic category

Functions and arguments

More generally, we can describe what's going on by assuming that every expression is either a function or argument. For instance, suppose we want to create: $(+\ x\ y)$:

Scott Farrar
CLMA, University of Washington farrar@u.washington.ed

Semantic Analysis
Problems
One Solution: λ-Calculus
λ-calculus and FOL
λ-calculus and compositionality
The semantics of words based on syntactic category

Functions and arguments

More generally, we can describe what's going on by assuming that every expression is either a function or argument. For instance, suppose we want to create: $(+\ x\ y)$:

- Start with three symbols: $+$, x, and y

Scott Farrar CLMA, University of Washington farrar@u.washington.ed

Semantic Analysis
Problems
One Solution: λ-Calculus
λ-calculus and FOL
λ-calculus and compositionality
The semantics of words based on syntactic category

Functions and arguments

More generally, we can describe what's going on by assuming that every expression is either a function or argument. For instance, suppose we want to create: $(+\ x\ y)$:

- Start with three symbols: $+$, x, and y
- Treat each symbol as either a function or argument

Scott Farrar
CLMA, University of Washington farrar@u.washington.ed

Semantic Analysis
Problems
One Solution: λ-Calculus
λ-calculus and FOL
λ-calculus and compositionality
The semantics of words based on syntactic category

Functions and arguments

More generally, we can describe what's going on by assuming that every expression is either a function or argument. For instance, suppose we want to create: $(+\ x\ y)$:

- Start with three symbols: $+$, x, and y
- Treat each symbol as either a function or argument
- $+\ x$ yields $(+\ x)$

Scott Farrar
CLMA, University of Washington farrar@u.washington.ed

Semantic Analysis
Problems
One Solution: λ-Calculus
λ-calculus and FOL
λ-calculus and compositionality
The semantics of words based on syntactic category

Functions and arguments

More generally, we can describe what's going on by assuming that every expression is either a function or argument. For instance, suppose we want to create: $(+\ x\ y)$:

- Start with three symbols: $+$, x, and y
- Treat each symbol as either a function or argument
- $+\ x$ yields $(+\ x)$
- $(+\ x)\ y$ yields $((+x)y)$

Functions and arguments

More generally, we can describe what's going on by assuming that every expression is either a function or argument. For instance, suppose we want to create: $(+\ x\ y)$:

- Start with three symbols: $+$, x, and y
- Treat each symbol as either a function or argument
- $+\ x$ yields $(+\ x)$
- $(+\ x)\ y$ yields $((+x)y)$

Thus, when an expression (function) is applied to another expression (argument), a third expression (result) is obtained.

λ-calculus: Formal definition

Definition

Expressions in the language Λ are composed of:

- variables $\{a, b, c, \ldots, x, y, z\}$
- abstraction symbols: λ and ., the dot
- parentheses: (and)

λ-terms. $T \in \Lambda$ iff one of the following holds:

1. T is a member of a countable set of variables
2. T is of the form (MN) where M and N are in Λ.
3. T is of the form $(\lambda X.Y)$ where X is a variable and Y is in Λ.

Λ is the smallest language with this property.

(MN) is called an **application** and $\lambda X.Y$ is an **abstraction**.

Examples

The following are all examples of λ expressions:

Examples

The following are all examples of λ expressions:

1. $\lambda x \,.\, x$

Scott Farrar
CLMA, University of Washington farrar@u.washington.ed

Semantic Analysis
Problems
One Solution: λ-Calculus
λ-calculus and FOL
λ-calculus and compositionality
The semantics of words based on syntactic category

Examples

The following are all examples of λ expressions:

1. $\lambda x \,.\, x$
2. $\lambda x \,.\, y\ (\lambda\ x \,.\, z\ x)$

Examples

The following are all examples of λ expressions:

1. $\lambda x \, . \, x$
2. $\lambda x \, . \, y \, (\lambda \, x \, . \, z \, x)$
3. $\lambda x \, . \, x \, (y)$

Examples

The following are all examples of λ expressions:

1. $\lambda x \,.\, x$
2. $\lambda x \,.\, y\ (\lambda\ x \,.\, z\ x)$
3. $\lambda x \,.\, x\ (y)$

Scott Farrar
CLMA, University of Washington farrar@u.washington.ed

Semantic Analysis
Problems
One Solution: λ-Calculus
λ-calculus and FOL
λ-calculus and compositionality
The semantics of words based on syntactic category

λ-calculus and FOL

Standard definitions of FOL can be augmented with λ-calculus. The point is that we can use standard FOL formulas as functions and create new FOL formulas compositionally.

λ-calculus and FOL

Standard definitions of FOL can be augmented with λ-calculus. The point is that we can use standard FOL formulas as functions and create new FOL formulas compositionally.

Definition

If in some formula a variable is bound by the λ operator, the formula is called a **lambda expression**.

λ-calculus and FOL

Standard definitions of FOL can be augmented with λ-calculus. The point is that we can use standard FOL formulas as functions and create new FOL formulas compositionally.

Definition

If in some formula a variable is bound by the λ operator, the formula is called a **lambda expression**.

Syntactically, a λ-expression looks just like any other quantified expression:

$$\lambda x.red(x)$$

$$\forall y.boat(y)$$

$$\exists z.floats(z)$$

Application expressions

To symbolize compositionality, we can create a new formula from $\lambda x.dog(x)$ by treating it as a **function** and then applying it to an **argument**:

$$\lambda x.dog(x)(FIDO)$$

Scott Farrar
CLMA, University of Washington farrar@u.washington.ed

Semantic Analysis
Problems
One Solution: λ-Calculus
λ-calculus and FOL
λ-calculus and compositionality
The semantics of words based on syntactic category

Application expressions

To symbolize compositionality, we can create a new formula from $\lambda x.dog(x)$ by treating it as a **function** and then applying it to an **argument**:

$$\lambda x.dog(x)(FIDO)$$

Scott Farrar
CLMA, University of Washington farrar@u.washington.ed

Semantic Analysis
Problems
One Solution: λ-Calculus
λ-calculus and FOL
λ-calculus and compositionality
The semantics of words based on syntactic category

Application expressions

To symbolize compositionality, we can create a new formula from $\lambda x.dog(x)$ by treating it as a **function** and then applying it to an **argument**:

$$\lambda x.dog(x)(FIDO)$$

The result is:

$$dog(FIDO)$$

Application expressions

To symbolize compositionality, we can create a new formula from $\lambda x.dog(x)$ by treating it as a **function** and then applying it to an **argument**:

$$\lambda x.dog(x)(FIDO)$$

The result is:

$$dog(FIDO)$$

Definition

Given some application expression $F\ A$ the function can be reduced by a process called β-**reduction**, such that the result is F with all occurrences of variables bound by λ replaced by A. (The terminology has roots in the original papers of Church and Kleene.)

NLTK notes

The λ operator is represented by the single back slash \, and is indicated with a raw string:

4. `\ x . dog(x) (FIDO)`

The Python string is the equivalent of the following application expression:
$\lambda\ x.dog\ x\ (FIDO)$

Scope of λ

In the augmented FOL, the λ operator ranges over sets and individuals, not just individuals as with $\forall$ and $\exists$.

example

```
(\P. P) (walk(x))
```

reduces to:

```
boat(x)
```


Scott Farrar
CLMA, University of Washington farrar@u.washington.ed

Semantic Analysis
Problems
One Solution: λ-Calculus
λ-calculus and FOL
λ-calculus and compositionality
The semantics of words based on syntactic category

Summary of terminology

- **abstraction**: the process of creating a λ function from a predicate logic formula.

Summary of terminology

- **abstraction**: the process of creating a λ function from a predicate logic formula.
- λ **expression**: one with variables bound by the λ operator, sometimes called a λ function.

Scott Farrar CLMA, University of Washington farrar@u.washington.ed

Semantic Analysis
Problems
One Solution: λ-Calculus
λ-calculus and FOL
λ-calculus and compositionality
The semantics of words based on syntactic category

Summary of terminology

- **abstraction**: the process of creating a λ function from a predicate logic formula.
- λ **expression**: one with variables bound by the λ operator, sometimes called a λ function.
- **application expression**: one with a function and an argument.

Summary of terminology

- **abstraction**: the process of creating a λ function from a predicate logic formula.
- λ **expression**: one with variables bound by the λ operator, sometimes called a λ function.
- **application expression**: one with a function and an argument.
- β-**reduction**: where subparts of a function are evaluated and rewritten until the function itself is reduced to a simpler form.

Steps in compositionality

Steps in compositionally deriving a semantic representation:

Scott Farrar
CLMA, University of Washington farrar@u.washington.ed

Semantic Analysis
Problems
One Solution: λ-Calculus
λ-calculus and FOL
λ-calculus and compositionality
The semantics of words based on syntactic category

Steps in compositionality

Steps in compositionally deriving a semantic representation:

1. Express the semantics of each constituent in terms of lambda expressions;

Steps in compositionality

Steps in compositionally deriving a semantic representation:

1. Express the semantics of each constituent in terms of lambda expressions;
2. Determine which expression is the function and which is the argument;

Scott Farrar CLMA, University of Washington farrar@u.washington.ed

Semantic Analysis
Problems
One Solution: λ-Calculus
λ-calculus and FOL
λ-calculus and compositionality
The semantics of words based on syntactic category

Steps in compositionality

Steps in compositionally deriving a semantic representation:

1. Express the semantics of each constituent in terms of lambda expressions;
2. Determine which expression is the function and which is the argument;
3. Apply the function to the argument;

Semantic Analysis
Problems
One Solution: λ-Calculus
λ-calculus and FOL
λ-calculus and compositionality
The semantics of words based on syntactic category

Steps in compositionality

Steps in compositionally deriving a semantic representation:

1. Express the semantics of each constituent in terms of lambda expressions;
2. Determine which expression is the function and which is the argument;
3. Apply the function to the argument;
4. β-reduce the conjoined elements to arrive at the final semantic representation.

Example: *Sue bikes*

$$Sue\ bikes \Rightarrow bikes(SUE)$$

Given:

Scott Farrar
CLMA, University of Washington farrar@u.washington.ed

Semantic Analysis
Problems
One Solution: λ-Calculus
λ-calculus and FOL
λ-calculus and compositionality
The semantics of words based on syntactic category

Example: *Sue bikes*

Sue bikes $\Rightarrow$ *bikes*(*SUE*)

Given:

- λ-expression for *Sue*: `\ P . P (SUE)`

Example: *Sue bikes*

Sue bikes $\Rightarrow$ *bikes*(*SUE*)

Given:

- λ-expression for *Sue*: `\ P . P (SUE)`
- λ-expression for *bikes*: `\ x. bikes(x)`

Example: *Sue bikes*

Sue bikes $\Rightarrow$ *bikes*(*SUE*)

Given:

- λ-expression for *Sue*: `\ P . P (SUE)`
- λ-expression for *bikes*: `\ x.  bikes(x)`

Derivation

Example: *Sue bikes*

Sue bikes $\Rightarrow$ *bikes*(*SUE*)

Given:

- λ-expression for *Sue*: `\ P . P (SUE)`
- λ-expression for *bikes*: `\ x. bikes(x)`

Derivation

- An application expression:
  `\ P . P (SUE) ( \ x . bikes(x))`

Example: *Sue bikes*

Sue bikes $\Rightarrow$ *bikes*(*SUE*)

Given:

- λ-expression for *Sue*: `\ P . P (SUE)`
- λ-expression for *bikes*: `\ x.  bikes(x)`

Derivation

- An application expression:
  `\ P . P (SUE) ( \ x .  bikes(x))`
- `\ x.  bikes(x) (SUE)` by β-reduction

Example: *Sue bikes*

Sue bikes $\Rightarrow$ *bikes*(*SUE*)

Given:

- λ-expression for *Sue*: `\ P . P (SUE)`
- λ-expression for *bikes*: `\ x.  bikes(x)`

Derivation

- An application expression:
  `\ P . P (SUE) ( \ x .  bikes(x))`
- `\ x.  bikes(x) (SUE)` by β-reduction
- `bikes(SUE)` by β-reduction

Today's lecture

1. Semantic Analysis
   - Problems
2. One Solution: λ-Calculus
   - λ-calculus and FOL
   - λ-calculus and compositionality
3. The semantics of words based on syntactic category

Scott Farrar
CLMA, University of Washington farrar@u.washington.ed

Semantic Analysis
Problems
One Solution: λ-Calculus
λ-calculus and FOL
λ-calculus and compositionality
The semantics of words based on syntactic category

General strategy for using λ-calculus

The point is to enrich each lexical entry with a semantics, and then derive the semantic representation of the entire sentence or phrase.

We'll need to express the semantics of everything using λ-calculus. Namely, we'll need to express the semantics of lexical items using the functional notation. $NNP \rightarrow Sue$
$NNP[sem = \backslash\ \texttt{S . S (SUE)}\ ] \rightarrow Sue$

Intransitive verbs

Intransitive verbs, in non-event style FOL, are mapped to unary predicates. The semantic attachment for *run* would be $\lambda x.run(x)$, a predicate waiting for an argument. *Bill runs*:

Intransitive verbs

Intransitive verbs, in non-event style FOL, are mapped to unary predicates. The semantic attachment for *run* would be $\lambda x.run(x)$, a predicate waiting for an argument. *Bill runs*:

\ `x.run(x) (BILL)` reduces to: `run(BILL)`

Intransitive verbs

Intransitive verbs, in non-event style FOL, are mapped to unary predicates. The semantic attachment for *run* would be $\lambda x.run(x)$, a predicate waiting for an argument. *Bill runs*:

`\ x.run(x) (BILL)` reduces to: `run(BILL)`

But, Bill comes before the verb in the syntax.

Proper nouns

Ordinarily. the semantic attachment for *Bill* would be a constant like *BILL*, as proper nouns are (non-logical) constants, i.e., always arguments of other expressions. But in a λ system, the semantic attachment is `\ P . P (BILL)`

Scott Farrar
CLMA, University of Washington farrar@u.washington.ed

Semantic Analysis
Problems
One Solution: λ-Calculus
λ-calculus and FOL
λ-calculus and compositionality
The semantics of words based on syntactic category

Proper nouns

Ordinarily. the semantic attachment for *Bill* would be a constant like *BILL*, as proper nouns are (non-logical) constants, i.e., always arguments of other expressions. But in a λ system, the semantic attachment is `\ P . P (BILL)`

Why? Because we need the semantics of a proper noun to be a **function** in order to get our representations to come out correctly.

Intransitive verbs, proper order

We need to preserve the order from the syntax. For *Bill runs*, we need to find a semantic representation for the word *Bill* and then for *runs*:

Intransitive verbs, proper order

We need to preserve the order from the syntax. For *Bill runs*, we need to find a semantic representation for the word *Bill* and then for *runs*:

`(\ P. P) (BILL) (\ x.run(x))` reduces to:

`\ x.run(x) (BILL)`

`run(BILL)`

Scott Farrar
CLMA, University of Washington farrar@u.washington.ed

Semantic Analysis
Problems
One Solution: λ-Calculus
λ-calculus and FOL
λ-calculus and compositionality
The semantics of words based on syntactic category

Intransitive verbs, proper order

We need to preserve the order from the syntax. For *Bill runs*, we need to find a semantic representation for the word *Bill* and then for *runs*:

`(\ P. P) (BILL) (\ x.run(x))` reduces to:

`\ x.run(x) (BILL)`

`run(BILL)`

Thus, order from the syntax can be used as is, which makes things much easier for compositionality.

Transitive Verbs

More care has to be taken to specify the order of reduction for the semantics of transitive verbs and di-transitive verbs. These are respectively binary and ternary predicates in FOL. For a transitive verb like *love*:

Transitive Verbs

More care has to be taken to specify the order of reduction for the semantics of transitive verbs and di-transitive verbs. These are respectively binary and ternary predicates in FOL. For a transitive verb like *love*:

- `\ y. \ x. love(x,y)`

Transitive Verbs

More care has to be taken to specify the order of reduction for the semantics of transitive verbs and di-transitive verbs. These are respectively binary and ternary predicates in FOL. For a transitive verb like *love*:

- `\ y.  \ x.  love(x,y)`
- `\ y.  \ x.  love(x,y) (BETTY)`

Scott Farrar
CLMA, University of Washington farrar@u.washington.ed

Semantic Analysis
Problems
One Solution: λ-Calculus
λ-calculus and FOL
λ-calculus and compositionality
The semantics of words based on syntactic category

Transitive Verbs

More care has to be taken to specify the order of reduction for the semantics of transitive verbs and di-transitive verbs. These are respectively binary and ternary predicates in FOL. For a transitive verb like *love*:

- `\ y.  \ x.  love(x,y)`
- `\ y.  \ x.  love(x,y) (BETTY)`
- `\ x.  love(x,BETTY)`

Transitive Verbs

More care has to be taken to specify the order of reduction for the semantics of transitive verbs and di-transitive verbs. These are respectively binary and ternary predicates in FOL. For a transitive verb like *love*:

- `\ y. \ x. love(x,y)`
- `\ y. \ x. love(x,y) (BETTY)`
- `\ x. love(x,BETTY)`
- `\ x. love(x,BETTY) (JIM)`

Transitive Verbs

More care has to be taken to specify the order of reduction for the semantics of transitive verbs and di-transitive verbs. These are respectively binary and ternary predicates in FOL. For a transitive verb like *love*:

- `\ y. \ x. love(x,y)`
- `\ y. \ x. love(x,y) (BETTY)`
- `\ x. love(x,BETTY)`
- `\ x. love(x,BETTY) (JIM)`
- `love(JIM,BETTY)`

Transitive Verbs, proper order

But, how do we deal with the linear order of the NL string? Due to subject and object order, the following will not reduce, since *JIM* is not a function:

Transitive Verbs, proper order

But, how do we deal with the linear order of the NL string? Due to subject and object order, the following will not reduce, since *JIM* is not a function:

Consider:

Transitive Verbs, proper order

But, how do we deal with the linear order of the NL string? Due to subject and object order, the following will not reduce, since *JIM* is not a function:

Consider:

- `\ Q. Q (JIM)` which is another form of simply `JIM`

Transitive Verbs, proper order

But, how do we deal with the linear order of the NL string? Due to subject and object order, the following will not reduce, since *JIM* is not a function:

Consider:

- `\ Q. Q (JIM)` which is another form of simply `JIM`
- `\ X y.  X(\ x.  loves(y,x))`

Transitive Verbs, proper order

But, how do we deal with the linear order of the NL string? Due to subject and object order, the following will not reduce, since *JIM* is not a function:

Consider:

- `\ Q. Q (JIM)` which is another form of simply `JIM`
- `\ X y.  X(\ x.  loves(y,x))`

Transitive Verbs, proper order

But, how do we deal with the linear order of the NL string? Due to subject and object order, the following will not reduce, since *JIM* is not a function:

Consider:

- `\ Q. Q (JIM)` which is another form of simply `JIM`
- `\ X y.  X(\ x.  loves(y,x))`

- `\ X y.  X (\ x.  loves(y,x)) ( \ Q . Q (BETTY))`
  just the inner terms

Transitive Verbs, proper order

But, how do we deal with the linear order of the NL string? Due to subject and object order, the following will not reduce, since *JIM* is not a function:

Consider:

- `\ Q. Q (JIM)` which is another form of simply `JIM`
- `\ X y.  X(\ x.  loves(y,x))`

- `\ X y.  X (\ x.  loves(y,x)) ( \ Q . Q (BETTY))`
  just the inner terms
- `(\ Q . Q (BETTY)) (\ x.  loves(y,x))`

Scott Farrar
CLMA, University of Washington farrar@u.washington.ed

Semantic Analysis
Problems
One Solution: λ-Calculus
λ-calculus and FOL
λ-calculus and compositionality
The semantics of words based on syntactic category

Transitive Verbs, proper order

But, how do we deal with the linear order of the NL string? Due to subject and object order, the following will not reduce, since *JIM* is not a function:

Consider:

- `\ Q. Q (JIM)` which is another form of simply `JIM`
- `\ X y. X(\ x. loves(y,x))`

- `\ X y. X (\ x. loves(y,x)) ( \ Q . Q (BETTY))`

  just the inner terms
- `(\ Q . Q (BETTY)) (\ x. loves(y,x))`
- `\ x. loves(y,x) (BETTY)`

Transitive Verbs, proper order

But, how do we deal with the linear order of the NL string? Due to subject and object order, the following will not reduce, since *JIM* is not a function:

Consider:

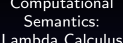
- `\ Q. Q (JIM)` which is another form of simply `JIM`
- `\ X y.  X(\ x.  loves(y,x))`

- `\ X y.  X (\ x.  loves(y,x)) ( \ Q . Q (BETTY))`
  just the inner terms
- `(\ Q . Q (BETTY)) (\ x.  loves(y,x))`
- `\ x.  loves(y,x) (BETTY)`
- `loves(y,BETTY)`

Scott Farrar
CLMA, University of Washington farrar@u.washington.ed

Semantic Analysis
Problems
One Solution: λ-Calculus
λ-calculus and FOL
λ-calculus and compositionality
The semantics of words based on syntactic category

Transitive Verbs, proper order



Scott Farrar
CLMA, University of Washington farrar@u.washington.ed

Transitive Verbs, proper order

- `\ y.loves(y,BETTY)`

Scott Farrar
CLMA, University of Washington farrar@u.washington.ed

Semantic Analysis
Problems
One Solution: λ-Calculus
λ-calculus and FOL
λ-calculus and compositionality
The semantics of words based on syntactic category

Transitive Verbs, proper order

- \ y.loves(y,BETTY)
- \ P . P (JIM) (\ y.loves(y,BETTY))

Transitive Verbs, proper order

- `\ y.loves(y,BETTY)`
- `\ P . P (JIM) (\ y.loves(y,BETTY))`
- `\ y.loves(y,BETTY) (JIM)`

Scott Farrar
CLMA, University of Washington farrar@u.washington.ed

Semantic Analysis
Problems
One Solution: λ-Calculus
λ-calculus and FOL
λ-calculus and compositionality
The semantics of words based on syntactic category

Transitive Verbs, proper order

- \ y.loves(y,BETTY)
- \ P . P (JIM) (\ y.loves(y,BETTY))
- \ y.loves(y,BETTY) (JIM)
- loves(JIM,BETTY)

Full transitive verb example

And the mostly unreadable full lambda epression for *Jim loves Betty*:

```
\ P . P (JIM) (\ X y.  X(\ x.  loves(y,x)) ( \ Q
. Q (BETTY)))
```

Nouns

Common nouns work just like intransitive verbs, i.e., the semantic attachment is a unary predicate.

Nouns

Common nouns work just like intransitive verbs, i.e., the semantic attachment is a unary predicate.

For example, the semantic attachment for *dog* would be:

\ x.dog(x) in λ-calculus.

Scott Farrar
CLMA, University of Washington farrar@u.washington.ed

Semantic Analysis
Problems
One Solution: λ-Calculus
λ-calculus and FOL
λ-calculus and compositionality
The semantics of words based on syntactic category

Copulas

The copula (*am*, *is*, *are*, etc.) is a special kind of transitive verb, as it equates the subject and object. We introduce a special binary predicate *eq* for the semantics of the copula:

```
\ X y.  X(\ x.  eq(y,x))
```


Scott Farrar
CLMA, University of Washington farrar@u.washington.ed

Semantic Analysis
Problems
One Solution: λ-Calculus
λ-calculus and FOL
λ-calculus and compositionality
The semantics of words based on syntactic category

Copulas

The copula (*am*, *is*, *are*, etc.) is a special kind of transitive verb, as it equates the subject and object. We introduce a special binary predicate *eq* for the semantics of the copula:
`\ X y.  X(\ x.  eq(y,x))`

The semantics of the copula looks just like the sematnics of any transitive verb (see previous).

Copulas

The copula (*am, is, are*, etc.) is a special kind of transitive verb, as it equates the subject and object. We introduce a special binary predicate *eq* for the semantics of the copula:

```
\ X y.  X(\ x.  eq(y,x))
```

The semantics of the copula looks just like the sematnics of any transitive verb (see previous).

For the negative copula (*ain't, isn't*, etc.) we have a slightly different formula:

```
\ X y.  X (\ x.-eq(y,x))
```

Auxiliaries

An auxiliary verb such as *does* is transparent at the level of semantic representation, at least concerning propositional content. Thus, *does go* would simply be:

```
\ z.  go(z).
```

Auxiliaries

An auxiliary verb such as *does* is transparent at the level of semantic representation, at least concerning propositional content. Thus, *does go* would simply be:

```
\ z.  go(z).
```

If we want to specify a semantics for *does* that will turn out to contribute nothing to higher constituents, this will suffice: The lambda expression for the semantics of an auxiliary contributing nothing would be:

```
\ P x.  P(x) (\ z.  go(z))
```

Auxiliaries

An auxiliary verb such as *does* is transparent at the level of semantic representation, at least concerning propositional content. Thus, *does go* would simply be:

```
\ z.  go(z).
```

If we want to specify a semantics for *does* that will turn out to contribute nothing to higher constituents, this will suffice: The lambda expression for the semantics of an auxiliary contributing nothing would be:

```
\ P x.  P(x) (\ z.  go(z))
```

- *does go*

Scott Farrar
CLMA, University of Washington farrar@u.washington.ed

Semantic Analysis
Problems
One Solution: λ-Calculus
λ-calculus and FOL
λ-calculus and compositionality
The semantics of words based on syntactic category

Auxiliaries

An auxiliary verb such as *does* is transparent at the level of semantic representation, at least concerning propositional content. Thus, *does go* would simply be:

```
\ z.  go(z).
```

If we want to specify a semantics for *does* that will turn out to contribute nothing to higher constituents, this will suffice: The lambda expression for the semantics of an auxiliary contributing nothing would be:

```
\ P x.  P(x) (\ z.  go(z))
```

- *does go*
- `\ P x.  P(x) (\ z.  go(z))`

Auxiliaries

An auxiliary verb such as *does* is transparent at the level of semantic representation, at least concerning propositional content. Thus, *does go* would simply be:

```
\ z.  go(z).
```

If we want to specify a semantics for *does* that will turn out to contribute nothing to higher constituents, this will suffice: The lambda expression for the semantics of an auxiliary contributing nothing would be:

```
\ P x.  P(x) (\ z.  go(z))
```

- *does go*
- `\ P x.  P(x) (\ z.  go(z))`
- `\ x .  (\ z.  go(z)) (x)`

Auxiliaries

An auxiliary verb such as *does* is transparent at the level of semantic representation, at least concerning propositional content. Thus, *does go* would simply be:
`\ z. go(z).`

If we want to specify a semantics for *does* that will turn out to contribute nothing to higher constituents, this will suffice: The lambda expression for the semantics of an auxiliary contributing nothing would be:
`\ P x. P(x) (\ z. go(z))`

- *does go*
- `\ P x. P(x) (\ z. go(z))`
- `\ x . (\ z. go(z)) (x)`
- `\ z. go(z)` (same as we started with; *does* is transparent)