Meaning Representation and Semantic Analysis

Ling 571
Deep Processing Techniques for NLP
February 11, 2015

Roadmap

- Meaning representation:
  - Event representations
- Semantic Analysis
  - Compositionality and rule-to-rule
  - Semantic attachments
    - Basic
    - Refinements
  - Quantifier scope
  - Earley Parsing and Semantics

FOL Syntax Summary

Formula	$\rightarrow$	*AtomicFormula*
	$\mid$	*Formula Connective Formula*
	$\mid$	*Quantifier Variable, . . . Formula*
	$\mid$	$\neg$ *Formula*
	$\mid$	(*Formula*)
AtomicFormula	$\rightarrow$	*Predicate*(*Term*, . . .)
Term	$\rightarrow$	*Function*(*Term*, . . .)
	$\mid$	*Constant*
	$\mid$	*Variable*
Connective	$\rightarrow$	$\wedge \mid \vee \mid \Rightarrow$
Quantifier	$\rightarrow$	$\forall \mid \exists$
Constant	$\rightarrow$	*A* $\mid$ *VegetarianFood* $\mid$ *Maharani* $\cdots$
Variable	$\rightarrow$	$x \mid y \mid \cdots$
Predicate	$\rightarrow$	*Serves* $\mid$ *Near* $\mid \cdots$
Function	$\rightarrow$	*LocationOf* $\mid$ *CuisineOf* $\mid \cdots$

Representing Events

- Initially, single predicate with some arguments
  - Serves(Maharani,IndianFood)

Representing Events

- Initially, single predicate with some arguments
  - Serves(Maharani,IndianFood)
  - Assume # ags = # elements in subcategorization frame

Representing Events

- Initially, single predicate with some arguments
  - Serves(Maharani,IndianFood)
  - Assume # ags = # elements in subcategorization frame
- Example:
  - I ate.
  - I ate a turkey sandwich.
  - I ate a turkey sandwich at my desk.
  - I ate at my desk.
  - I ate lunch.
  - I ate a turkey sandwich for lunch.
  - I ate a turkey sandwich for lunch at my desk.

Events

- Issues?

Events

- Issues?
  - Arity – how can we deal with different #s of arguments?

Events

- Issues?
  - Arity – how can we deal with different #s of arguments?
- One predicate per frame
  - $Eating_1$(Speaker)
  - $Eating_2$(Speaker,TS)
  - $Eating_3$(Speaker,TS,Desk)
  - $Eating_4$(Speaker,Desk)
  - $Eating_5$(Speaker,TS,Lunch)
  - $Eating_6$(Speaker,TS,Lunch,Desk)

Events (Cont'd)

- Good idea?

Events (Cont'd)

- Good idea?
  - Despite the names, actually unrelated predicates

Events (Cont’d)

- Good idea?
  - Despite the names, actually unrelated predicates
    - Can’t derive obvious info
      - E.g. I ate a turkey sandwich for lunch at my desk
        - Entails all other sentences

Events (Cont'd)

- Good idea?
  - Despite the names, actually unrelated predicates
    - Can't derive obvious info
      - E.g. I ate a turkey sandwich for lunch at my desk
        - Entails all other sentences
    - Can't directly associate with other predicates

Events (Cont'd)

- Good idea?
  - Despite the names, actually unrelated predicates
    - Can't derive obvious info
      - E.g. I ate a turkey sandwich for lunch at my desk
        - Entails all other sentences
    - Can't directly associate with other predicates
- Could write rules to implement implications

Events (Cont'd)

- Good idea?
  - Despite the names, actually unrelated predicates
    - Can't derive obvious info
      - E.g. I ate a turkey sandwich for lunch at my desk
        - Entails all other sentences
    - Can't directly associate with other predicates
- Could write rules to implement implications
  - But?
    - Intractable in the large
      - Like the subcat problem generally.

Variabilizing

- Create predicate with maximum possible arguments
  - Include appropriate args
  - Maintains connections

$$\exists w, x, y\, Eating(Speaker, w, x, y)$$

$$\exists w, x\, Eating(Speaker, TS, w, x)$$

$$\exists w\, Eating(Speaker, TS, w, Desk)$$

$$Eating(Speaker, TS, Lunch, Desk)$$

Variabilizing

- Create predicate with maximum possible arguments
  - Include appropriate args
  - Maintains connections

$$\exists w,x,y Eating(Speaker,w,x,y)$$

$$\exists w,x Eating(Speaker,TS,w,x)$$

$$\exists w Eating(Speaker,TS,w,Desk)$$

$$Eating(Speaker,TS,Lunch,Desk)$$

- Better?

Variabilizing

- Create predicate with maximum possible arguments
  - Include appropriate args
  - Maintains connections

$$\exists w, x, y\, Eating(Speaker, w, x, y)$$

$$\exists w, x\, Eating(Speaker, TS, w, x)$$

$$\exists w\, Eating(Speaker, TS, w, Desk)$$

$$Eating(Speaker, TS, Lunch, Desk)$$

- Better?
  - Yes, but
    - Too many commitments – assume all details show up

Variabilizing

- Create predicate with maximum possible arguments
  - Include appropriate args
  - Maintains connections

$$\exists w, x, y\, Eating(Speaker, w, x, y)$$

$$\exists w, x\, Eating(Speaker, TS, w, x)$$

$$\exists w\, Eating(Speaker, TS, w, Desk)$$

$$Eating(Speaker, TS, Lunch, Desk)$$

- Better?
  - Yes, but
    - Too many commitments – assume all details show up
    - Can't individuate – don't know if same event

Events - Finalized

- Neo-Davidsonian representation:
  - Distill event to single argument for event itself
  - Everything else is additional predication

$$\exists e Eating(e) \wedge Eater(e, Speaker) \wedge Eaten(e, TS) \wedge Meal(e, Lunch) \wedge Location(e, Desk)$$

Events - Finalized

- Neo-Davidsonian representation:
  - Distill event to single argument for event itself
  - Everything else is additional predication

$$\exists e Eating(e) \wedge Eater(e, Speaker) \wedge Eaten(e, TS) \wedge Meal(e, Lunch) \wedge Location(e, Desk)$$

- Pros:

Events - Finalized

- Neo-Davidsonian representation:
  - Distill event to single argument for event itself
  - Everything else is additional predication

$$\exists e\, Eating(e) \wedge Eater(e, Speaker) \wedge Eaten(e, TS) \wedge Meal(e, Lunch) \wedge Location(e, Desk)$$

- Pros:
  - No fixed argument structure
    - Dynamically add predicates as necessary

Events - Finalized

- Neo-Davidsonian representation:
  - Distill event to single argument for event itself
  - Everything else is additional predication

$$\exists e Eating(e) \wedge Eater(e, Speaker) \wedge Eaten(e, TS) \wedge Meal(e, Lunch) \wedge Location(e, Desk)$$

- Pros:
  - No fixed argument structure
    - Dynamically add predicates as necessary
  - No extra roles

Events - Finalized

- Neo-Davidsonian representation:
  - Distill event to single argument for event itself
  - Everything else is additional predication

$$\exists e\, Eating(e) \wedge Eater(e, Speaker) \wedge Eaten(e, TS) \wedge Meal(e, Lunch) \wedge Location(e, Desk)$$

- Pros:
  - No fixed argument structure
    - Dynamically add predicates as necessary
  - No extra roles
  - Logical connections can be derived

Meaning Representation for Computational Semantics

- Requirements:
  - Verifiability, Unambiguous representation, Canonical Form, Inference, Variables, Expressiveness
- Solution:
  - First-Order Logic
    - Structure
    - Semantics
    - Event Representation
- Next: Semantic Analysis
  - Deriving a meaning representation for an input

Syntax-driven Semantic Analysis

- Key: Principle of Compositionality
  - Meaning of sentence from meanings of parts
    - E.g. groupings and relations from syntax

Syntax-driven Semantic Analysis

- Key: Principle of Compositionality
  - Meaning of sentence from meanings of parts
    - E.g. groupings and relations from syntax
- Question: Integration?

Syntax-driven Semantic Analysis

- Key: Principle of Compositionality
  - Meaning of sentence from meanings of parts
    - E.g. groupings and relations from syntax
- Question: Integration?
- Solution 1: Pipeline
  - Feed parse tree and sentence to semantic unit

Syntax-driven Semantic Analysis

- Key: Principle of Compositionality
  - Meaning of sentence from meanings of parts
    - E.g. groupings and relations from syntax
- Question: Integration?
- Solution 1: Pipeline
  - Feed parse tree and sentence to semantic unit
  - Sub-Q: Ambiguity:

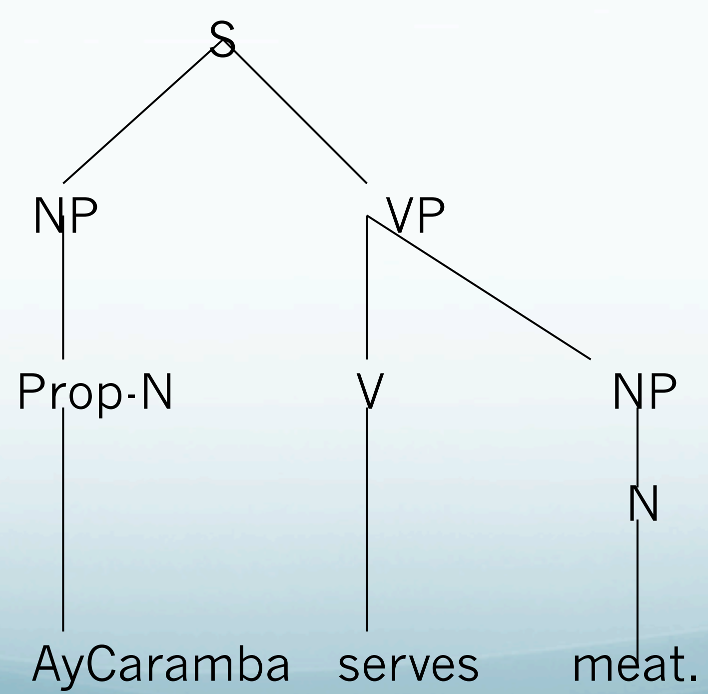
Syntax-driven Semantic Analysis

- Key: Principle of Compositionality
  - Meaning of sentence from meanings of parts
    - E.g. groupings and relations from syntax
- Question: Integration?
- Solution 1: Pipeline
  - Feed parse tree and sentence to semantic unit
  - Sub-Q: Ambiguity:
    - Approach: Keep all analyses, later stages will select

Simple Example

- AyCaramba serves meat.

$$\exists e\ Serving(e) \wedge Server(e, AyCaramba) \wedge Served(e, Meat)$$



Rule-to-Rule

- Issue:

Rule-to-Rule

- Issue:
  - How do we know which pieces of the semantics link to what part of the analysis?

Rule-to-Rule

- Issue:
  - How do we know which pieces of the semantics link to what part of the analysis?
  - Need detailed information about sentence, parse tree

Rule-to-Rule

- Issue:
  - How do we know which pieces of the semantics link to what part of the analysis?
  - Need detailed information about sentence, parse tree
    - Infinitely many sentences & parse trees
    - Semantic mapping function per parse tree → intractable
- Solution:

Rule-to-Rule

- Issue:
  - How do we know which pieces of the semantics link to what part of the analysis?
  - Need detailed information about sentence, parse tree
    - Infinitely many sentences & parse trees
    - Semantic mapping function per parse tree → intractable
- Solution:
  - Tie semantics to finite components of grammar
    - E.g. rules & lexicon

Rule-to-Rule

- Issue:
  - How do we know which pieces of the semantics link to what part of the analysis?
  - Need detailed information about sentence, parse tree
    - Infinitely many sentences & parse trees
    - Semantic mapping function per parse tree → intractable
- Solution:
  - Tie semantics to finite components of grammar
    - E.g. rules & lexicon
  - Augment grammar rules with semantic info
    - Aka "attachments"
      - Specify how RHS elements compose to LHS

Semantic Attachments

- Basic structure:
  - A→ a_1....a_n {f(a_j.sem,...a_k.sem)}
  - A.sem

Semantic Attachments

- Basic structure:
  - A→ $a_1 \ldots a_n$ $\{f(a_j.sem, \ldots a_k.sem)\}$
  - A.sem
- Language for semantic attachments

Semantic Attachments

- Basic structure:
  - A→ $a_1....a_n$ {f(a_j.sem,...a_k.sem)}
  - A.sem
- Language for semantic attachments
  - Arbitrary programming language fragments?

Semantic Attachments

- Basic structure:
  - A→ a_1....a_n {f(a_j.sem,...a_k.sem)}
  - A.sem
- Language for semantic attachments
  - Arbitrary programming language fragments?
    - Arbitrary power but hard to map to logical form
    - No obvious relation between syntactic, semantic elements

Semantic Attachments

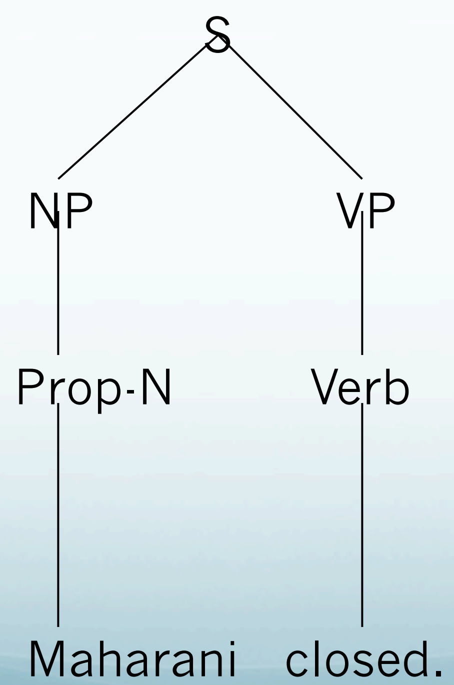
- Basic structure:
  - A→ $a_1 \ldots a_n$ {f(a_j.sem,…a_k.sem)}
  - A.sem
- Language for semantic attachments
  - Arbitrary programming language fragments?
    - Arbitrary power but hard to map to logical form
    - No obvious relation between syntactic, semantic elements
  - Lambda calculus
    - Extends First Order Predicate Calculus (FOPC) with function application

Semantic Attachments

- Basic structure:
  - A→ a_1....a_n {f(a_j.sem,...a_k.sem)}
  - A.sem
- Language for semantic attachments
  - Arbitrary programming language fragments?
    - Arbitrary power but hard to map to logical form
    - No obvious relation between syntactic, semantic elements
  - Lambda calculus
    - Extends First Order Predicate Calculus (FOPC) with function application
  - Feature-based model + unification
- Focus on lambda calculus approach

Basic example

- Input: Maharani closed.
- Target output: Closed(Maharani)



Semantic Analysis Example

- Semantic attachments:
  - Each CFG production gets semantic attachment
- Maharani

Semantic Analysis Example

- Semantic attachments:
  - Each CFG production gets semantic attachment
- Maharani
  - ProperNoun → Maharani

Semantic Analysis Example

- Semantic attachments:
  - Each CFG production gets semantic attachment
- Maharani
  - ProperNoun → Maharani {Maharani}
    - FOL constant to refer to object
  - NP → ProperNoun

Semantic Analysis Example

- Semantic attachments:
  - Each CFG production gets semantic attachment
- Maharani
  - ProperNoun → Maharani {Maharani}
    - FOL constant to refer to object
  - NP → ProperNoun {ProperNoun.sem}
    - No additional semantic info added

Semantic Attachment Example

- Phrase semantics is function of SA of children

- More complex functions are parameterized
  - E.g. Verb → closed

Semantic Attachment Example

- Phrase semantics is function of SA of children

- More complex functions are parameterized
  - E.g. Verb → closed { λx.Closed(x) }
    - Unary predicate:
      - 1 arg = subject, not yet specified
  -

Semantic Attachment Example

- Phrase semantics is function of SA of children

- More complex functions are parameterized
  - E.g. Verb → closed { λ x.Closed(x) }
    - Unary predicate:
      - 1 arg = subject, not yet specified
  - VP → Verb

Semantic Attachment Example

- Phrase semantics is function of SA of children

- More complex functions are parameterized
  - E.g. Verb → closed { λ x.Closed(x) }
    - Unary predicate:
      - 1 arg = subject, not yet specified
  - VP → Verb {Verb.sem}
    - No added information
  - S → NP VP

Semantic Attachment Example

- Phrase semantics is function of SA of children

- More complex functions are parameterized
  - E.g. Verb → closed { λ x.Closed(x) }
    - Unary predicate:
      - 1 arg = subject, not yet specified
  - VP → Verb {Verb.sem}
    - No added information
  - S → NP VP {VP.sem(NP.sem)}
    - Application= λ x.Closed(x)(Maharani)

Semantic Attachment Example

- Phrase semantics is function of SA of children

- More complex functions are parameterized
  - E.g. Verb → closed { λx.Closed(x) }
    - Unary predicate:
      - 1 arg = subject, not yet specified
  - VP → Verb {Verb.sem}
    - No added information
  - S → NP VP {VP.sem(NP.sem)}
    - Application= λx.Closed(x)(Maharani) = Closed(Maharani)

Semantic Attachment

- General pattern:
  - Grammar (non-terminal) rules mostly lambda reductions
    - Functor and arguments
  - Most representation resides in lexicon

Refining Representation

- Add
  - Neo-Davidsonian event-style model
  - Complex quantification
- Example II
  - Input: Every restaurant closed.
  - Target:

$$\forall x Restaurant(x) \Rightarrow \exists e Closed(e) \wedge ClosedThing(e,x)$$

Refining Representation

- Idea: $\forall x \, Restaurant(x)$
  - Good enough?

Refining Representation

- Idea: $\forall x \, Restaurant(x)$
  - Good enough?
    - No: roughly ‘everything is a restaurant’
    - Saying something about all restaurants – nuclear scope

Refining Representation

- Idea: $\forall x \, Restaurant(x)$
  - Good enough?
    - No: roughly ‘everything is a restaurant’
    - Saying something about all restaurants – nuclear scope
- Solution: Dummy predicate
  $\forall x \, Restaurant(x) \Rightarrow Q(x)$
  - Good enough?

Refining Representation

- Idea: $\forall x \, Restaurant(x)$
  - Good enough?
    - No: roughly 'everything is a restaurant'
    - Saying something about all restaurants – nuclear scope
- Solution: Dummy predicate
  $\forall x \, Restaurant(x) \Rightarrow Q(x)$
  - Good enough?
    - No: no way to get Q(x) from elsewhere in sentence

Refining Representation

- Idea: $\forall x \, Restaurant(x)$
  - Good enough?
    - No: roughly 'everything is a restaurant'
    - Saying something about all restaurants – nuclear scope
- Solution: Dummy predicate

  $\forall x \, Restaurant(x) \Rightarrow Q(x)$
  - Good enough?
    - No: no way to get Q(x) from elsewhere in sentence
- Solution: Lambda

Refining Representation

- Idea: $\forall x \, Restaurant(x)$
  - Good enough?
    - No: roughly 'everything is a restaurant'
    - Saying something about all restaurants – nuclear scope
- Solution: Dummy predicate
  $\forall x \, Restaurant(x) \Rightarrow Q(x)$
  - Good enough?
    - No: no way to get Q(x) from elsewhere in sentence
- Solution: Lambda

$$\lambda Q.\forall x \, Restaurant(x) \Rightarrow Q(x)$$

Updating Attachments

- Noun → restaurant

Updating Attachments

- Noun → restaurant {λ x.Restaurant(x)}
- Nom → Noun

Updating Attachments

- Noun → restaurant $\{\lambda x.Restaurant(x)\}$
- Nom → Noun { Noun.sem }
- Det → Every

Updating Attachments

- Noun → restaurant $\{\lambda x.Restaurant(x)\}$
- Nom → Noun { Noun.sem }
- Det → Every $\{\lambda P.\lambda Q.\forall x P(x) \Rightarrow Q(x)\}$
- NP → Det Nom

Updating Attachments

- Noun → restaurant { λ x.Restaurant(x)}
- Nom → Noun { Noun.sem }
- Det → Every { $\lambda P.\lambda Q.\forall x P(x) \Rightarrow Q(x)$ }
- NP → Det Nom { Det.sem(Nom.sem) }

$$\lambda P.\lambda Q.\forall x P(x) \Rightarrow Q(x)(\lambda x.\mathrm{Re}\,staurant(x))$$

$$\lambda P.\lambda Q.\forall x P(x) \Rightarrow Q(x)(\lambda x.\mathrm{Re}\,staurant(x))$$

$$\lambda P.\lambda Q.\forall x P(x) \Rightarrow Q(x)(\lambda y.\mathrm{Re}\,staurant(y))$$

$$\lambda P.\lambda Q.\forall x P(x) \Rightarrow Q(x)(\lambda x.\mathrm{Re}\,staurant(x))$$

$$\lambda P.\lambda Q.\forall x P(x) \Rightarrow Q(x)(\lambda y.\mathrm{Re}\,staurant(y))$$

$$\lambda Q.\forall x \lambda y.\mathrm{Re}\,staurant(y)(x) \Rightarrow Q(x)$$

$$\lambda P.\lambda Q.\forall x P(x) \Rightarrow Q(x)(\lambda x.\mathrm{Re}\,staurant(x))$$

$$\lambda P.\lambda Q.\forall x P(x) \Rightarrow Q(x)(\lambda y.\mathrm{Re}\,staurant(y))$$

$$\lambda Q.\forall x \lambda y.\mathrm{Re}\,staurant(y)(x) \Rightarrow Q(x)$$

$$\lambda Q.\forall x\,\mathrm{Re}\,staurant(x) \Rightarrow Q(x)$$

Full Representation

- Verb → close

Full Representation

- Verb → close $\{\lambda x.\exists e Closed(e) \wedge ClosedThing(e,x)\}$
- VP → Verb

Full Representation

- Verb → close $\{\lambda x.\exists e Closed(e) \wedge ClosedThing(e,x)\}$
- VP → Verb { Verb.sem }
- S → NP VP

Full Representation

- Verb → close $\{\lambda x.\exists e Closed(e) \wedge ClosedThing(e,x)\}$
- VP → Verb { Verb.sem }
- S → NP VP { NP.sem(VP.sem) }

$$\lambda Q.\forall x \operatorname{Re}staurant(x) \Rightarrow Q(x)(\lambda y.\exists e Closed(e) \wedge ClosedThing(e,y))$$

Full Representation

- Verb → close $\{\lambda x.\exists e Closed(e) \wedge ClosedThing(e,x)\}$
- VP → Verb { Verb.sem }
- S → NP VP { NP.sem(VP.sem) }

$$\lambda Q.\forall x \mathrm{Re}\,staurant(x) \Rightarrow Q(x)(\lambda y.\exists e Closed(e) \wedge ClosedThing(e,y))$$

$$\forall x \mathrm{Re}\,staurant(x) \Rightarrow \lambda y.\exists e Closed(e) \wedge ClosedThing(e,y)(x)$$

Full Representation

- Verb → close $\{\lambda x.\exists e Closed(e) \wedge ClosedThing(e,x)\}$
- VP → Verb { Verb.sem }
- S → NP VP { NP.sem(VP.sem) }

$$\lambda Q.\forall x Restaurant(x) \Rightarrow Q(x)(\lambda y.\exists e Closed(e) \wedge ClosedThing(e,y))$$

$$\forall x Restaurant(x) \Rightarrow \lambda y.\exists e Closed(e) \wedge ClosedThing(e,y)(x)$$

$$\forall x Restaurant(x) \Rightarrow \exists e Closed(e) \wedge ClosedThing(e,x)$$

Generalizing Attachments

- ProperNoun → Maharani {Maharani}

- Does this work in the new style?

Generalizing Attachments

- ProperNoun → Maharani {Maharani}

- Does this work in the new style?
  - No, we turned the NP/VP application around

Generalizing Attachments

  - ProperNoun → Maharani {Maharani}
- Does this work in the new style?
  - No, we turned the NP/VP application around
- New style: $\lambda x.x(\text{Maharani})$

More

- Determiner
- Det → a

More

- Determiner
- Det → a { $\lambda P.\lambda Q.\exists x P(x) \wedge Q(x)$ }
- a restaurant

More

- Determiner
- Det → a { $\lambda P.\lambda Q.\exists x P(x) \wedge Q(x)$ }
- a restaurant $\lambda Q.\exists x \mathrm{Re}\, staurant(x) \wedge Q(x)$
- Transitive verb:
  - VP → Verb NP

More

- Determiner
- Det → a $\{ \quad \lambda P.\lambda Q.\exists x P(x) \wedge Q(x) \quad \}$
- a restaurant $\lambda Q.\exists x \mathrm{Re}\, staurant(x) \wedge Q(x)$

- Transitive verb:
  - VP → Verb NP { Verb.sem(NP.sem) }
  - Verb → opened

More

- Determiner
- Det -> a { $\lambda P.\lambda Q.\exists x P(x) \wedge Q(x)$ }
- a restaurant $\lambda Q.\exists x \mathrm{Re}\, staurant(x) \wedge Q(x)$
- Transitive verb:
  - VP -> Verb NP { Verb.sem(NP.sem) }
  - Verb -> opened

$$\lambda w.\lambda z.w(\lambda x.\exists e Opened(e) \wedge Opener(e,z) \wedge OpenedThing(e,x))$$

Matthew opened a restaurant

- Proper_Noun → Matthew

Matthew opened a restaurant

- Proper_Noun → Matthew {λ x.x(Matthew)}

Matthew opened a restaurant

- Proper_Noun → Matthew {λx.x(Matthew)}
- VP → Verb NP {Verb.sem(NP.sem)}

Matthew opened a restaurant

- Proper_Noun → Matthew {λ x.x(Matthew)}
- VP → Verb NP {Verb.sem(NP.sem)}

$$\lambda w.\lambda z.w(\lambda x.\exists e Opened(e) \wedge Opener(e,z) \wedge OpenedThing(e,x))$$

- $(\lambda Q.\exists x \, Restaurant(x) \wedge Q(x))$

Matthew opened a restaurant

- Proper_Noun → Matthew {λ x.x(Matthew)}
- VP → Verb NP {Verb.sem(NP.sem)}

$$\lambda w.\lambda z.w(\lambda x.\exists e Opened(e) \wedge Opener(e,z) \wedge OpenedThing(e,x))$$

- $(\lambda Q.\exists y \, Restaurant(y) \wedge Q(y))$

Matthew opened a restaurant

- Proper_Noun → Matthew {λ x.x(Matthew)}
- VP → Verb NP {Verb.sem(NP.sem)}

$$\lambda w.\lambda z.w(\lambda x.\exists e Opened(e) \wedge Opener(e,z) \wedge OpenedThing(e,x))$$

- $(\lambda Q.\exists y \mathrm{Re}\, staurant(y) \wedge Q(y))$

$$\lambda z.\lambda Q.\exists y \mathrm{Re}\, staurant(y) \wedge Q(y)$$

$$(\lambda x.\exists e Opened(e) \wedge Opener(e,z) \wedge OpenedThing(e,x))$$

Matthew opened a restaurant

- Proper_Noun → Matthew {λ x.x(Matthew)}
- VP → Verb NP {Verb.sem(NP.sem)}

$$\lambda w.\lambda z.w(\lambda x.\exists e Opened(e) \wedge Opener(e,z) \wedge OpenedThing(e,x))$$

- $(\lambda Q.\exists y \mathrm{Re}\, staurant(y) \wedge Q(y))$

$$\lambda z.\lambda Q.\exists y \mathrm{Re}\, staurant(y) \wedge Q(y)$$

$$(\lambda x.\exists e Opened(e) \wedge Opener(e,z) \wedge OpenedThing(e,x))$$

$$\lambda z.\exists y \mathrm{Re}\, staurant(y) \wedge$$

$$\lambda x..\exists e Opened(e) \wedge Opener(e,z) \wedge OpenedThing(e,x)(y)$$

Matthew opened a restaurant

- Proper_Noun → Matthew {λ x.x(Matthew))
- VP → Verb NP {Verb.sem(NP.sem)}

$$\lambda w.\lambda z.w(\lambda x.\exists e Opened(e) \wedge Opener(e,z) \wedge OpenedThing(e,x))$$

- $(\lambda Q.\exists y \mathrm{Re}\, staurant(y) \wedge Q(y))$

$$\lambda z.\lambda Q.\exists y \mathrm{Re}\, staurant(y) \wedge Q(y)$$

$$(\lambda x.\exists e Opened(e) \wedge Opener(e,z) \wedge OpenedThing(e,x))$$

$$\lambda z.\exists y \mathrm{Re}\, staurant(y) \wedge$$

$$\exists e Opened(e) \wedge Opener(e,z) \wedge OpenedThing(e,y)$$

Matthew opened a restaurant

- Proper_Noun → Matthew {λx.x(Matthew)}
- S → NP VP {NP.sem(VP.sem)}
- λx.x(Matthew)

$$(\lambda z.\exists y Restaurant(y) \wedge$$

$$\exists e Opened(e) \wedge Opener(e,z) \wedge OpenedThing(e,y))$$

Matthew opened a restaurant

- Proper_Noun → Matthew {λ x.x(Matthew)}
- S → NP VP {NP.sem(VP.sem)}
- λ x.x(Matthew)

$$(\lambda z.\exists y Restaurant(y) \wedge$$
$$\exists e Opened(e) \wedge Opener(e,z) \wedge OpenedThing(e,y))$$

$$(\lambda z.\exists y Restaurant(y) \wedge$$
$$\exists e Opened(e) \wedge Opener(e,z) \wedge OpenedThing(e,y))(Matthew)$$

Matthew opened a restaurant

$$(\lambda z.\exists y \, Restaurant(y) \wedge$$
$$\exists e Opened(e) \wedge Opener(e,z) \wedge OpenedThing(e,y))(Matthew)$$

$$\exists y \, Restaurant(y) \wedge$$
$$\exists e Opened(e) \wedge Opener(e, Matthew) \wedge OpenedThing(e,y)$$

Strategy for Semantic Attachments

- General approach:
  - Create complex, lambda expressions with lexical items
    - Introduce quantifiers, predicates, terms
  - Percolate up semantics from child if non-branching
  - Apply semantics of one child to other through lambda
    - Combine elements, but don't introduce new

Sample Attachments

Grammar Rule	Semantic Attachment
$S \rightarrow NP\ VP$	$\{NP.sem(VP.sem)\}$
$NP \rightarrow Det\ Nominal$	$\{Det.sem(Nominal.sem)\}$
$NP \rightarrow ProperNoun$	$\{ProperNoun.sem\}$
$Nominal \rightarrow Noun$	$\{Noun.sem\}$
$VP \rightarrow Verb$	$\{Verb.sem\}$
$VP \rightarrow Verb\ NP$	$\{Verb.sem(NP.sem)\}$
$Det \rightarrow every$	$\{\lambda P.\lambda Q.\forall x P(x) \Rightarrow Q(x)\}$
$Det \rightarrow a$	$\{\lambda P.\lambda Q.\exists x P(x) \wedge Q(x)\}$
$Noun \rightarrow restaurant$	$\{\lambda r.Restaurant(r)\}$
$ProperNoun \rightarrow Matthew$	$\{\lambda m.m(Matthew)\}$
$ProperNoun \rightarrow Franco$	$\{\lambda f.f(Franco)\}$
$ProperNoun \rightarrow Frasca$	$\{\lambda f.f(Frasca)\}$
$Verb \rightarrow closed$	$\{\lambda x.\exists e Closed(e) \wedge ClosedThing(e,x)\}$
$Verb \rightarrow opened$	$\{\lambda w.\lambda z.w(\lambda x.\exists e Opened(e) \wedge Opener(e,z) \wedge Opened(e,x))\}$

Semantics Learning

- Zettlemoyer & Collins, 2005, 2007, etc; Mooney 2007
- Given semantic representation and corpus of parsed sentences
  - Learn mapping from sentences to logical form
    - Structured perceptron
    - Applied to ATIS corpus sentences
- Similar approaches to: learning instructions from computer manuals, game play from walkthroughs, robocup/soccer play from commentary

Quantifier Scope

- Ambiguity:
  - *Every restaurant has a menu*

$$\forall x\, Restaurant(x) \Rightarrow \exists y(Menu(y) \wedge (\exists e Having(e) \wedge Haver(e,x) \wedge Had(e,y)))$$

    - efficiently and recover all alternatives.

Quantifier Scope

- Ambiguity:
  - *Every restaurant has a menu*

$$\forall x\, Restaurant(x) \Rightarrow \exists y (Menu(y) \wedge (\exists e Having(e) \wedge Haver(e,x) \wedge Had(e,y)))$$

- Readings:

Quantifier Scope

- Ambiguity:
  - *Every restaurant has a menu*

$$\forall x \,\mathrm{Re}\, staurant(x) \Rightarrow \exists y (Menu(y) \wedge (\exists e Having(e) \wedge Haver(e,x) \wedge Had(e,y)))$$

  - Readings:
    - all have a menu;
    - all have same menu

Quantifier Scope

- Ambiguity:
  - *Every restaurant has a menu*

$$\forall x \, Restaurant(x) \Rightarrow \exists y (Menu(y) \wedge (\exists e Having(e) \wedge Haver(e,x) \wedge Had(e,y)))$$

  - Readings:
    - all have a menu;
    - all have same menu
  - Only derived one

$$\exists y Menu(y) \wedge \forall x (Restaurant(x) \Rightarrow \exists e Having(e) \wedge Haver(e,x) \wedge Had(e,y)))$$

  - Potentially O(n!) scopings (n=# quantifiers)

- There are approaches to describe ambiguity efficiently and recover all alternatives.

Earley Parsing with Semantics

- Implement semantic analysis
  - In parallel with syntactic parsing
    - Enabled by compositional approach
- Required modifications

Earley Parsing with Semantics

- Implement semantic analysis
  - In parallel with syntactic parsing
    - Enabled by compositional approach
- Required modifications
  - Augment grammar rules with semantic field

Earley Parsing with Semantics

- Implement semantic analysis
  - In parallel with syntactic parsing
    - Enabled by compositional approach
- Required modifications
  - Augment grammar rules with semantic field
  - Augment chart states with meaning expression

Earley Parsing with Semantics

- Implement semantic analysis
  - In parallel with syntactic parsing
    - Enabled by compositional approach
- Required modifications
  - Augment grammar rules with semantic field
  - Augment chart states with meaning expression
  - Completer computes semantics
    - Can also fail
      - Blocks semantically invalid parses
    - Can impose extra work

Sidelight: Idioms

- Not purely compositional
  - E.g. kick the bucket = die
  - tip of the iceberg = beginning
- Handling:
  - Mix lexical items with constituents (word nps)
  - Create idiom-specific const. for productivity
  - Allow non-compositional semantic attachments
- Extremely complex: e.g. metaphor

Semantic Analysis

- Applies principle of compositionality
  - Rule-to-rule hypothesis
  - Links semantic attachments to syntactic rules
    - Incrementally ties semantics to parse processing
    - Lambda calculus meaning representations
    - Most complexity pushed into lexical items
    - Non-terminal rules largely lambda applications

Representing Time

- Temporal logic:
  - Includes tense logic to capture verb tense info
- Basic notion:
  - Timeline:
    - From past to future
    - Events associated with points or intervals on line
      - Ordered by positioning on line
    - Current time
      - Relative order gives past/present/future

Temporal Information

- I arrived in New York.
- I am arriving in New York.
- I will arrive in New York.

Temporal Information

- I arrived in New York.
- I am arriving in New York.
- I will arrive in New York.
  - Same event, differ only in tense

$$\exists e Arriving(e) \wedge Arriver(e, Speaker) \wedge Destination(e, NY)$$

Temporal Information

- I arrived in New York.
- I am arriving in New York.
- I will arrive in New York.
  - Same event, differ only in tense

$$\exists e Arriving(e) \wedge Arriver(e, Speaker) \wedge Destination(e, NY)$$

- Create temporal representation based on verb tense
  - Add predication about event variable

Temporal Information

- I arrived in New York.
- I am arriving in New York.
- I will arrive in New York.
  - Same event, differ only in tense

$$\exists e Arriving(e) \wedge Arriver(e, Speaker) \wedge Destination(e, NY)$$

- Create temporal representation based on verb tense
  - Add predication about event variable
  - Temporal variables represent:
    - Interval of event

Temporal Information

- I arrived in New York.
- I am arriving in New York.
- I will arrive in New York.
  - Same event, differ only in tense

$$\exists e Arriving(e) \wedge Arriver(e, Speaker) \wedge Destination(e, NY)$$

- Create temporal representation based on verb tense
  - Add predication about event variable
  - Temporal variables represent:
    - Interval of event
    - End point of event

Temporal Information

- I arrived in New York.
- I am arriving in New York.
- I will arrive in New York.
  - Same event, differ only in tense

$$\exists e Arriving(e) \wedge Arriver(e, Speaker) \wedge Destination(e, NY)$$

- Create temporal representation based on verb tense
  - Add predication about event variable
  - Temporal variables represent:
    - Interval of event
    - End point of event
    - Predicates link end point to current time

Temporal Representation

$$\exists e,i,n\, Arriving(e) \wedge Arriver(e, Speaker) \wedge Destination(e, NY)$$
$$\wedge IntervalOf(e,i) \wedge EndPoint(i,n) \wedge Precedes(n, Now)$$

$$\exists e,i,n\, Arriving(e) \wedge Arriver(e, Speaker) \wedge Destination(e, NY)$$
$$\wedge IntervalOf(e,i) \wedge MemberOf(i, Now)$$

$$\exists e,i,n\, Arriving(e) \wedge Arriver(e, Speaker) \wedge Destination(e, NY)$$
$$\wedge IntervalOf(e,i) \wedge EndPoint(i,n) \wedge Precedes(Now, e)$$

More Temp Rep

- Flight 902 arrived late.
- Flight 902 had arrived late.

- Does the current model cover this?

More Temp Rep

- Flight 902 arrived late.
- Flight 902 had arrived late.

- Does the current model cover this?
  - Not really

More Temp Rep

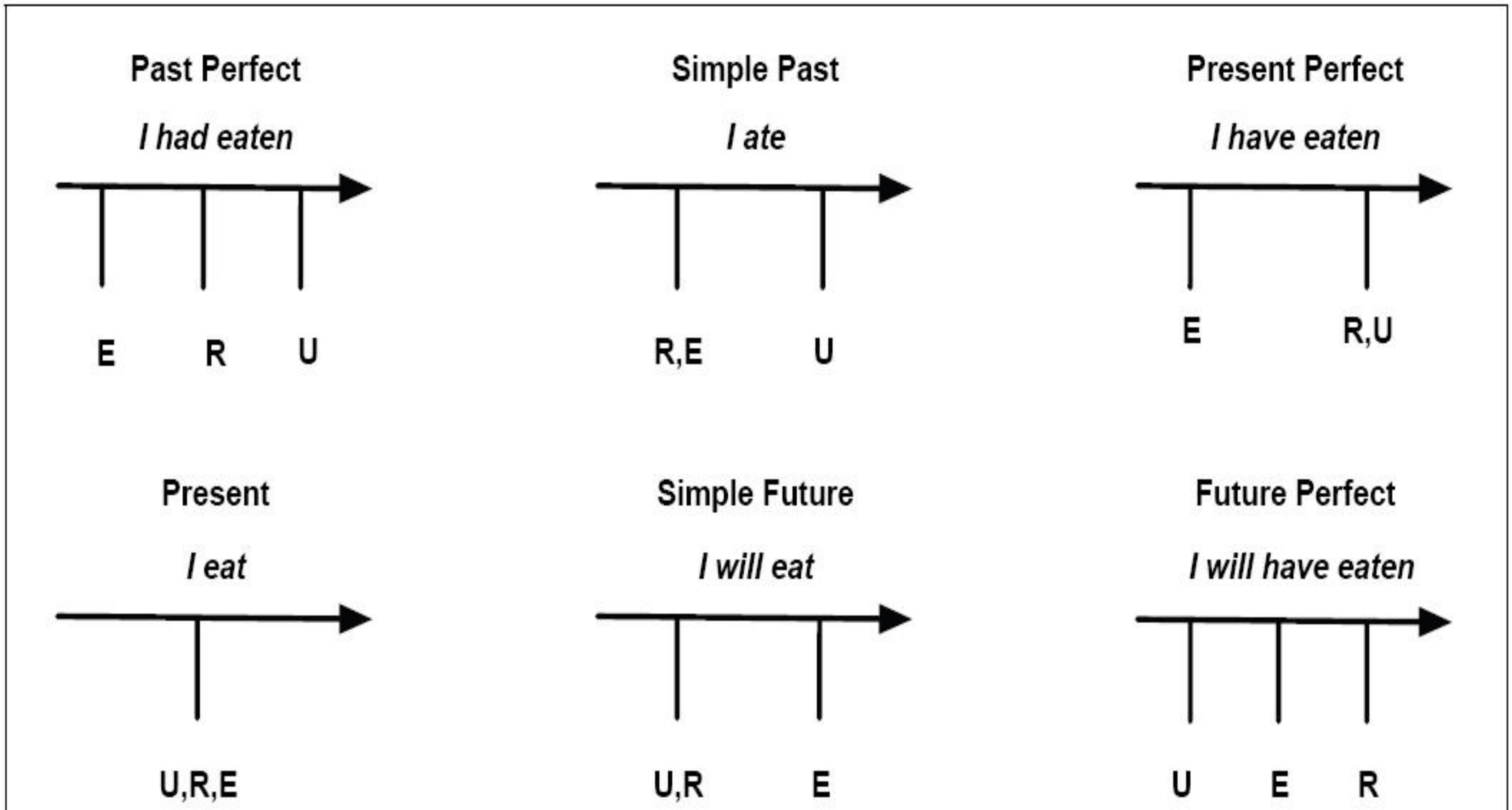
  - Flight 902 arrived late.
  - Flight 902 had arrived late.
- Does the current model cover this?
  - Not really
  - Need additional notion:

More Temp Rep

- Flight 902 arrived late.
- Flight 902 had arrived late.

- Does the current model cover this?
  - Not really
  - Need additional notion:
    - Reference point
      - As well as current time, event time
        - Current model: current = utterance time = reference point

Reichenbach’s Tense Model



Roadmap

- Lexical Semantics
  - Motivation: Word sense disambiguation
  - Meaning at the word level
  - Issues
    - Ambiguity
    - Meaning
    - Meaning structure
      - Relations to other words
      - Subword meaning composition
  - WordNet: Lexical ontology

What is a plant?

There are more kinds of plants and animals in the rainforests than anywhere else on Earth. Over half of the millions of known species of plants and anima live in the rainforest. Many are found nowhere else. There are even plants and animals in the rainforest that we have not yet discovered.

The Paulus company was founded in 1938. Since those days the product rang has been the subject of constant expansions and is brought up continuously t correspond with the state of the art. We're engineering, manufacturing, and commissioning world-wide ready-to-run plants packed with our comprehensive know-how.

Lexical Semantics

- Focus on word meanings:
  - Relations of meaning among words
    - Similarities & differences of meaning in sim context
  - Internal meaning structure of words
    - Basic internal units combine for meaning
- Lexeme: meaning entry in lexicon
  - Orthographic form, phonological form, sense

Sources of Confusion

- Homonymy:
  - Words have same form but different meanings
    - Generally same POS, but unrelated meaning
    - E.g. bank (side of river) vs bank (financial institution)
      - Bank1 vs bank2
    - Homophones: same phonology, diff't orthographic form
      - E.g. two, to, too
    - Homographs: Same orthography, diff't phonology
- Why?
  - Problem for applications: TTS, ASR transcription, IR

Sources of Confusion II

- Polysemy
  - Multiple RELATED senses
    - E.g. bank: money, organ, blood,...
  - Big issue in lexicography
    - # of senses, relations among senses, differentiation
    - E.g. serve breakfast, serve Philadelphia, serve time

Relations between Words

- Synonymy:
  - “same meaning”: substitutability?
  - Issues:
    - Polysemy – same as some sense
    - Shades of meaning – other associations:
      - Price/fare
    - Collocational constraints: e.g. babbling brook
    - Register: social factors: e.g. politeness, formality
- Hyponomy:
  - Isa relations:
    - More General (hypernym) vs more specific (hyponym)
      - E.g. dog vs golden retriever
  - Organize as ontology/taxonomy

WordNet Taxonomy

- Manually constructed lexical database
  - 3 Tree-structured hierarchies
    - Nouns, verbs, adjective+adverb
    - Entries: synonym set, gloss, example use
- Relations between entries:
  - Synonymy: in synset
  - Hypo(per)nym: Isa tree
- Heavily used resource

Word-internal Structure

- Thematic roles:
  - Characterize verbs by their arguments
    - E.g. transport: agent, theme, source, destination
      - They transported grain from the fields to the silo.
    - Deep structure: passive / active: same roles
- Thematic hierarchy
  - E.g. agent > theme > source, dest
    - Provide default surface positions
  - Tie to semantics (e.g. Levin): Interlinguas
    - Cluster verb meanings by set of syntactic alternations
    - Limitations: only NP,PP: other arguments predicates less

Selectional Restrictions

- Semantic constraints on filling of roles
  - E.g. Bill ate chicken
    - Eat: Agent: animate; Theme: Edible
  - Associate with sense
    - Most commonly of verb/event; possibly adj, noun…
- Specifying constraints:
  - Add a term to semantics, e.g. Isa(x,Ediblething)
  - Tie to position in WordNet
    - All hyponyms inherit

Primitive Decompositions

- Jackendoff(1990), Dorr(1999), McCawley (1968)

- Word meaning constructed from primitives
  - Fixed small set of basic primitives
    - E.g. cause, go, become,
    - kill=cause X to become Y
  - Augment with open-ended "manner"
    - Y = not alive
    - E.g. walk vs run
- Fixed primitives/Infinite descriptors