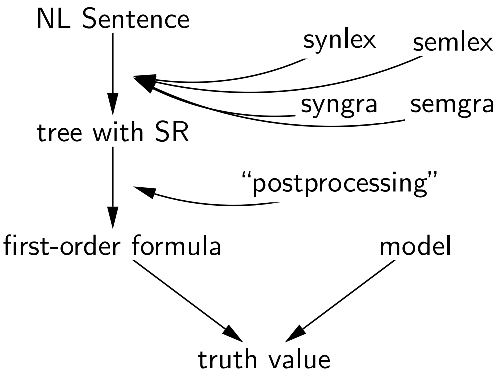
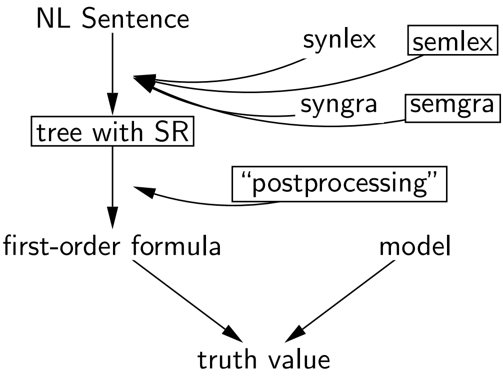
Scope Ambiguities, Montague and Cooper Storage

Kilian Evang

May 21, 2008


NL Sentence
synlex
semlex
syngra
semgra
tree with SR
"postprocessing"
first-order formula
model
truth value


NL Sentence
synlex
semlex
syngra
semgra
tree with SR
"postprocessing"
first-order formula
model
truth value

Note (1):

Semantic representations that are assigned to lexical items and internal nodes in the tree can be anything – currently it's lambda expressions.

Note (2):

Is the syntactic grammar a "black box"?
Yes and no. Though any semantic rules adhering to the interface can be used with it, the parsing process is guided by syntactic and semantic rules at the same time. Example:

```
s(s(NP_st,VP_st),[coord:no,sem:Sem])-->
   np(NP_st,[coord:_,num:Num,gap:[],sem:NP]),
   vp(VP_st,[coord:_,inf:fin,num:Num,gap:[],sem:VP]),
   {combine(s:Sem,[np:NP,vp:VP])}.
```

Scope ambiguities

- arise in sentences containing more than one quantifying noun phrase (QNP)
- *Every criminal hates a man*
- $\forall x(criminal(x) \rightarrow \exists y(man(y) \wedge hate(x, y)))$
- $\exists y(man(y) \wedge \forall x(criminal(x) \rightarrow hate(x, y)))$
- Only the first reading is produced by our system

Scope ambiguities (cont.)

- Semantically, the two quantifiers can be applied in either order.
- Problem: In our system, the order is determined by syntax (example)

Montague's Solution

Montague's Solution

To generate a reading were some QNP has wide scope,

- replace it with a placeholder pronoun
  e.g. *it-1*, semantics: $\lambda w.(w@z_3)$
- process the sentence as usual (you get a formula with a free variable)
- lambda abstract over the formula with respect to the free variable and apply the semantic representation of the original QNP to it

Montague's Solution (cont.)

- can be viewed syntactically as moving the QNP to a syntactic top position, hence a.k.a *quantifier raising*

Montague's Solution (cont.)

Can be applied to multiple QNPs, meaning:

- every QNP *may* be replaced with a placeholder pronoun whose semantic representation has the form $\lambda w.(w@z_i)$ where i is some unique index

Note: Need to keep track of which index belongs to which QNP!

Montague's Solution (cont.)

Can be applied to multiple QNPs, meaning:

- every QNP *may* be replaced with a placeholder pronoun whose semantic representation has the form $\lambda w.(w@z_i)$ where i is some unique index
- the resulting formula for the sentence contains free variables

Note: Need to keep track of which index belongs to which QNP!

Montague's Solution (cont.)

Can be applied to multiple QNPs, meaning:

- every QNP *may* be replaced with a placeholder pronoun whose semantic representation has the form $\lambda w.(w@z_i)$ where i is some unique index
- the resulting formula for the sentence contains free variables
- to get a sentential formula, the free variables are removed one by one, in any order, by lambda abstracting over the formula with respect to the free variable and apply the semantic representation of the appropriate QNP to it

Note: Need to keep track of which index belongs to which QNP!

Montague's Solution – How to Implement

- additional syntactic rules for introducing placeholder pronouns
- additional semantic rules for lambda abstracting over semantic representations with free variables
- additional syntactic rules for combining "raised" QNPs with sentences with placeholders

Mess with syntax to solve a semantic problem?

Cooper's Solution

Cooper's Solution

- don't apply QNPs during parsing, just collect them
- *Every criminal hates a man*: Somebody hates somebody, and then there is some information about QNPs.
- This is a *store*:
  $\langle love(z_6, z_7),$
  $(\lambda u.\forall x(criminal(x) \rightarrow u@x), 6),$
  $(\lambda u.\forall y(man(y) \wedge u@y), 7))\rangle$
- core representation, freezer

Representations are Stores

The lambda expressions in the lexicon are just put into sequences, e.g.
hates: $\langle \lambda z.\lambda u.(z@\lambda v.hate(u,v))\rangle$
The freezer is initially empty.

Storage (Cooper)

If the store
$\langle\phi, (\beta, j), \ldots, (\beta\prime, k)\rangle$
is a semantic representation for a quantified NP, then the store
$\langle\lambda u.(u@z_i), (\phi, i), (\beta, j), \ldots, (\beta\prime, k)\rangle$,
where i is some unique index,
is also a representation for that NP.

Retrieval (Cooper)

Let σ_1 and σ_2 be (possibly empty) sequences of binding operators. If the store $\langle \phi, \sigma_1, (\beta, i), \sigma_2 \rangle$ is associated with an expression of category S, then the store $\langle \beta @ \lambda z_i . \phi, \sigma_1, \sigma_2 \rangle$ is also associated with this expression.

Implementation

Representing structures in Prolog

- index binding operators as terms of the form `bo(Quant,Index)`
- indexes represented as Prolog variables (simpler than in theory)
- stores as lists - example: `walk(X),bo(lam(P,all(Y,imp(boxer(Y),app(P,Y)))),X]`

Changing the machinery

1. semantic lexicon: make store-based semantic representations
2. semantic rules: combining stores, applying storage
3. semantic rules: retrieval

Semantic Lexicon: Store-Based Semantic Representations

```
semLex(iv,M):-
  M = [symbol:Sym,
       sem:[lam(X,Formula)]],
  compose(Formula,Sym,[X]).
```

semLexStorage.pl

Semantic Rules: Combining Stores, Applying Storage

```
combine(vp:[app(A,B)|S],[av:[A],vp:[B|S]]).


combine(np:[app(app(B,A),C)|S3],[np:[A|S1],
   coord:[B],np:[C|S2]]):-
   appendLists(S1,S2,S3).


combine(np:[lam(P,app(P,X)),bo(app(A,B),X)|S],
                                  [det:[A],n:[B|S]]).
combine(np:[app(A,B)|S],[det:[A],n:[B|S]]).
```

semRulesCooper.pl

Semantic Rules: Retrieval

Retrieval takes place at the end, i.e. at the sentence level.

```
combine(s:S,[np:[A|S1],vp:[B|S2]]):-
   appendLists(S1,S2,S3),
   sRetrieval([app(A,B)|S3],Retrieved),
   betaConvert(Retrieved,S).
```

semRulesCooper.pl

```
sRetrieval([S],S).

sRetrieval([Sem|Store],S):-
   selectFromList(bo(Q,X),Store,NewStore),
   sRetrieval([app(Q,lam(X,Sem))|NewStore],S).
```

cooperStorage.pl

The Top-Level Predicate

```
cooperStorage:-
   readLine(Sentence),
   setof(Sem,t([sem:Sem],Sentence,[]),Sems1),
   filterAlphabeticVariants(Sems1,Sems2),
   printRepresentations(Sems2).
```

cooperStorage.pl

Filtering Alphabetic Variants

```
filterAlphabeticVariants(L1,L2):-
   selectFromList(X,L1,L3),
   memberList(Y,L3),
   alphabeticVariants(X,Y), !,
   filterAlphabeticVariants(L3,L2).

filterAlphabeticVariants(L,L).
```

cooperStorage.pl

Why is Storage Optional?

Conclusion

	Lambda	Montague	Cooper
Semantic representations	λ-expressions	λ-expressions	storages
Additional operations during parsing		replace QNPs by indexed pronouns	extend storage
Addtional operations after parsing		λ-abstract, apply	retrieve, filter

References

Patrick Blackburn and Johan Bos.
Representation and Inference for Natural Language. A First Course in Computational Semantics, chapter 3.1–3.3.
CSLI Publications, 2005.