Computational Semantics
Day 4: Dominance Graphs, Round Two

Aljoscha Burchardt
Alexander Koller
Stephan Walter

ESSLLI 2004, Nancy
Overview

- Semantics construction for dominance graphs
- Implementation in our Prolog framework

- Solving dominance graphs
- Implementing the graph solver
Recap: Yesterday

Sentence → Syntax → USR

dominance graphs

semantic repres. 1
semantic repres. 2
semantic repres. 3
semantic repres. 4

lambda structures
"Every swimmer wants a medal."

∃y medal(y) ∧ ∀x.swimmer(x) → want(x,y)
∀x.swimmer(x) → ∃y medal(y) ∧ want(x,y)
Recap: Yesterday

∀x

@ swimmers x

@ want x

@ medal y

∃y

∀x

@ swimmers x

@ want x

@ medal y

@ y

@ want x

@ y
Recap: Yesterday

∀x

∃y

∧

swimmer x

∧

medal y

∧

want x

∧

∃y

∧

medal y

∧

swimmer x

∧

want x
First remaining question:
- How do we construct a dominance graph from a syntactic analysis?

We use Tuesday's modular syntax-semantics framework.

Replace semantic macros and combine rules by new ones.
Semantics Construction Architecture

- Syntax
  - DCG
    - combine-calls
      - lexicon-calls
        - lexicon-facts
    - combine-rules
  - Semantic macros
- Semantics
  - lexicon-facts
  - combine-rules
Semantics Construction: Principles

- We use exactly the same DCG grammar and lexicon facts as on Tuesday.
- For every node in the syntax tree, we derive a dominance graph that represents the semantic readings.
- Prolog representation of dominance graphs:
  \[
  \text{usr}(\text{Nodes, LCs, DCs, BCs})
  \]
- First element of node list is the interface node (or root). Use this to connect the subgraph to other subgraphs.
A Simple Example

```
S
 / \
NP NP
 /  /
PN TV PN
 Ian beats Michael
```
Semantic macros for the example

- Most semantic macros introduce graphs that have exactly one node, which is labelled by the "core semantics".

- Macro for proper names:
  
  \[
  \text{pnSem}(\text{Symbol},
  \usr([\text{Root}],[\text{Root:Symbol}],[[],[]])).
  \]

- Macro for transitive verbs:
  
  \[
  \text{tvSem}(\text{Symbol},
  \usr([\text{Root}],[\text{Root:Symbol}],[[],[]])).
  \]
Semantics construction: The simple example

- lexicon facts
- semantic macros
Combining verbs and NPs

- General rule: The interface node of a graph for a noun phrase is the node that will be plugged into the verb as an argument.

- For proper names, this means we don't need to do any real work:
  
  ```
  combine(np:NP, [pn:NP]).
  ```
Combine rules for verbs

\[
\text{combine}(\text{vp}:V, [\text{tv}:TV, \text{np}:NP]) :-
\begin{align*}
\text{TV} &= \text{usr}([\text{TVRoot}|\_], \_, \_, \_), \\
\text{NP} &= \text{usr}([\text{NPRoot}|\_], \_, \_, \_), \\
\text{NewUsr} &= \text{usr}([\text{Root}], [\text{Root}:(\text{TVRoot}@\text{NPRoot})], \\
& \quad \quad \quad \quad \quad [], [], []), \\
\text{mergeUSR} &= ([\text{NewUsr}, \text{TV}, \text{NP}], V).
\end{align*}
\]

\[
\text{combine}(\text{s}:S, [\text{np}:NP, \text{vp}:VP]) :-
\begin{align*}
\text{NP} &= \text{usr}([\text{TVRoot}|\_], \_, \_, \_), \\
\text{VP} &= \text{usr}([\text{NPRoot}|\_], \_, \_, \_), \\
\text{NewUsr} &= \text{usr}([\text{Root}], [\text{Root}:(\text{VPRoot}@\text{NPRoot})], \\
& \quad \quad \quad \quad \quad [], [], []), \\
\text{mergeUSR} &= ([\text{NewUsr}, \text{NP}, \text{VP}], S).
\end{align*}
\]
Semantics construction: The simple example
Semantics construction: The simple example

- combine rules

S

NP

PN

Ian

TV

beats

NP

PN

Michael

Ian

beat

Michael
Semantics construction: The simple example

- combine rules
Quantifiers

- The graph for a quantifier NP contains a variable node and its binder, linked by dominance and binding edges.
- The interface node of the graph is the variable node!
Semantic macro for determiners

detSem(\text{uni},
\text{usr}([\text{Root}, N1, N2, N3, N4, N5, N6, N7, N8, N9],
[\text{Root}: \lambda \text{lambda}(N1), N1: \lambda \text{lambda}(N2),
N2: \forall \text{forall}(N3), N3: (N4 > N5),
N4: (N6@N7), N5: (N8@N9), N6: \text{var},
N7: \text{var}, N8: \text{var}, N9: \text{var}],
[]),
[\text{bind}(N6, \text{Root}), \text{bind}(N7, N2),
\text{bind}(N8, N1), \text{bind}(N9, N2)]).

$\lambda P \lambda Q \forall x. (P@x \rightarrow Q@x)$
Combine rule for determiners

\[
\text{combine}(\text{np:NP}, [\text{det:DET}, \text{n:N}]) :- \\
\text{DET} = \text{usr}([\text{DETRoot|}_], _, _, _), \\
\text{N} = \text{usr}([\text{NRoot|}_], _, _, _), \\
\text{NewUsr} = \text{usr}(........), \\
\text{mergeUSR}([\text{NewUsr, TV, NP}], V).
\]

- This rule encodes Montague's Trick!
An example with determiners

S
  / \   
 NP   VP
 |    |
 Det N IV
|    |   |
Every swimmer sleeps

NP
  / \   
 Det N
|    |   |
Every swimmer

VP
  / \   
 IV
|    |
sleeps

swimmer

every

sleep
An example with determiners
An example with determiners

\[ \lambda P \lambda Q \forall x. (P@x \rightarrow Q@x) \]

@ swimmer

@ (\lambda y. sleep@y)
Scope ambiguities

Every swimmer wants a medal

S
   NP
      Det N TV Det N
      Every swimmer wants a medal

NP
  VP
     Det NP
    @ swimmer
   @ every

NP
  @ a

NP
  @ medal

NP
  @ want
Scope ambiguities

Every swimmer wants a medal.

Trie representation:

- **S** (Sentence)
  - **NP** (Noun Phrase): Every swimmer
    - **Det** (Determiner): Every
    - **N** (Noun): swimmer
  - **VP** (Verb Phrase): wants
    - **Det** (Determiner): a
    - **N** (Noun): medal
  - **@** (Variable)
  - **λ** (Lambda)
Every swimmer wants a medal.
Semantics construction: Summary

- By plugging new rules into yesterday's syntax-semantics framework, we can compute dominance graphs for English sentences.
- Changed semantic macros to give us dominance graphs for lexicon entries.
- Combine rules plug subgraphs together by connecting their interface nodes.
- Always apply verb semantics to interface variable of an argument NP.
Underspecification in semantics construction

- Combine rule of determiners encodes Montague's Trick.
- Variable and binder are introduced together: No capturing necessary!
- Need fewer lambdas because we can now talk about positions in formulas explicitly.
- All large-scale grammars with semantics use some form of underspecification.
Solving Dominance Graphs

- Now we know
  - how to model scope ambiguities with dominance graphs
  - how to represent dominance graphs in Prolog
  - how to compute dominance graphs for English sentences.

- What's still missing: How to compute the trees (= formulas) that a graph represents?
We have seen yesterday that every solvable graph has an infinite number of solutions (= trees into which it can be embedded).
Solved Forms

- Thus, we aim at enumerating all solved forms of a dominance graph and not all solutions.

- A dominance graph in solved form is a graph whose tree and dominance edges are a forest.

- A graph $G'$ is a solved form of $G$ iff $G'$ is in solved form and if there is a path from $u$ to $v$ in $G$ (over tree and dominance edges), there is also a path from $u$ to $v$ in $G'$. 
Can consider solved forms as representatives of classes of solutions that only differ in "irrelevant details".
Solving Dominance Graphs

- Solver algorithm applies three graph simplification rules and then calls itself recursively:
  - Choice
  - Parent Normalisation
  - Redundancy Elimination
- Detect unsolvability: Test for cycles.
- Prolog implementation.
The Choice Rule

- Driving force behind solver is the Choice rule: Which of two trees comes first?
The Choice Rule

- Every application of Choice arranges the dominance parents of one node.
- Eventually, the dominance parents of all nodes will be arranged.
- Choice rule is sound: Every tree that satisfies original graph also satisfies one of the two possible results of the Choice application.
Cleaning Up I: Parent Normalisation

- Parent Normalisation changes a dominance edge \((u,v)\) into a dominance edge \((u,w)\), where \(w\) is the parent of \(v\) over a solid edge.
Cleaning Up II: Redundancy Elimination

- Redundancy Elimination deletes an edge \((u,v)\) whenever there is a path from \(u\) to \(v\) that doesn't use this edge.

![Diagram showing the reduction of a redundant edge in a graph.](image-url)
Detecting Unsolvability

- Every dominance graph that has a cycle (using only tree and dominance edges) is unsolvable.
The Enumeration Algorithm

1. Apply Redundancy Elimination and Parent Normalisation exhaustively.

2. If the graph has a cycle, it is unsolvable.

3. If there is a node with two incoming dominance edges, pick one and apply Choice once. Then continue with Step 1 for each of the resulting graphs.

4. Otherwise, the dominance graph is in solved form.
Search Tree
The Algorithm in Prolog

Input: An USR

Output: List of solved forms

Case 1: Choice applicable

\[
\text{solve(Usr, SFs) :-}
\text{normalize(Usr, NormalUsr),}
\text{distribute(NormalUsr, Dist1, Dist2),}
\text{solve(Dist1, SFs1),}
\text{solve(Dist2, SFs2),}
\text{append(SFs1, SFs2, SFs).}
\]

Case 2: Solved Form

\[
\text{solve(Usr, [NormalUsr]) :-}
\text{normalize(Usr, NormalUsr),}
\text{\+ hasCycle(NormalUsr).}
\]

Case 3: Cycle

\[
\text{solve(_Usr, []).}
\]
The algorithm uses some other predicates. These are all available on the course website.

Here we look at
- distribute
- elimRedundancy
The predicate "distribute"

distribute(usr(Ns,LCs,DCs,BCs),
    usr(Ns,LCs, [dom(X,Y)|DCs],BCs),
    usr(Ns,LCs, [dom(Y,X)|DCs],BCs)) :-

    member(dom(X,Z), DCs),
    member(dom(Y,Z), DCs),
    X \== Y.
The predicate "elimRedundancy"

normalize(Usr, Normal) :-
    parentNormalization(Usr, Lifted),
    elimRedundancy(Lifted, Normal).

elimRedundancy(usr(Ns,LCs,DCs,BCs), Irr) :-
    select(dom(X,Y), DCs, DCsRest),
    reachable(Y,X,usr(Ns,LCs,DCsRest,BCs)),
    !,
    elimRedundancy(usr(Ns,LCs,DCsRest,BCs), Irr).

elimRedundancy(Ur, Usr).
A Note on Efficiency

- The implementation is correct, but:
  - checking for cycles is not a complete unsatisifiability test: Search space may be too large.
  - Redundancy Elimination, Choice, etc. are not implemented efficiently.

- Both problems can be solved. Best current implementations enumerate over 100,000 solved forms per second (Bodirsky et al. 2004).
A Note on Formalisms

- Dominance graphs are equivalent to normal dominance constraints (Althaus et al. 03; Egg et al. 01).
- Hole Semantics (Bos 96) can be encoded into normal dominance constraints (Koller et al. 03).
- MRS (Copestake et al. 99) can be encoded into normal dominance constraints (Niehren & Thater 03; Fuchss et al. 04).
Summary

Semantics construction for dominance graphs:
- use Tuesday's framework
- use interface nodes to combine subgraphs
- clean construction that introduces variables and binders together.

Solving dominance graphs:
- enumerate solved forms
- driving force is Choice rule
- Prolog implementation very concise
- can be made efficient (not in Prolog)
State of the art in underspecification

- Well-understood formalisms.
- Efficient solvers are available.
- Large-scale grammars that compute underspecified semantic descriptions are available: e.g. English Resource Grammar (Copestake & Flickinger, 2000).
- Used, in one form or another, in most major grammars that define semantics.