

Generating Speech Recognition Packages

Johan Bos

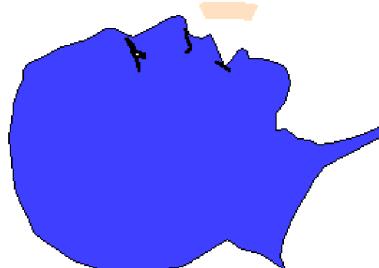
Language Technology Group
The University of Edinburgh

Outline of this talk

- Automatic Speech Recognition (ASR)
- How to build a simple recognition package (incl. demo)
- How to add features for natural language understanding (incl. demo)
- Why this is not a good approach
- How we can do better
 - Linguistically-motivated grammars

Automatic Speech Recognition

Go to the kitchen!

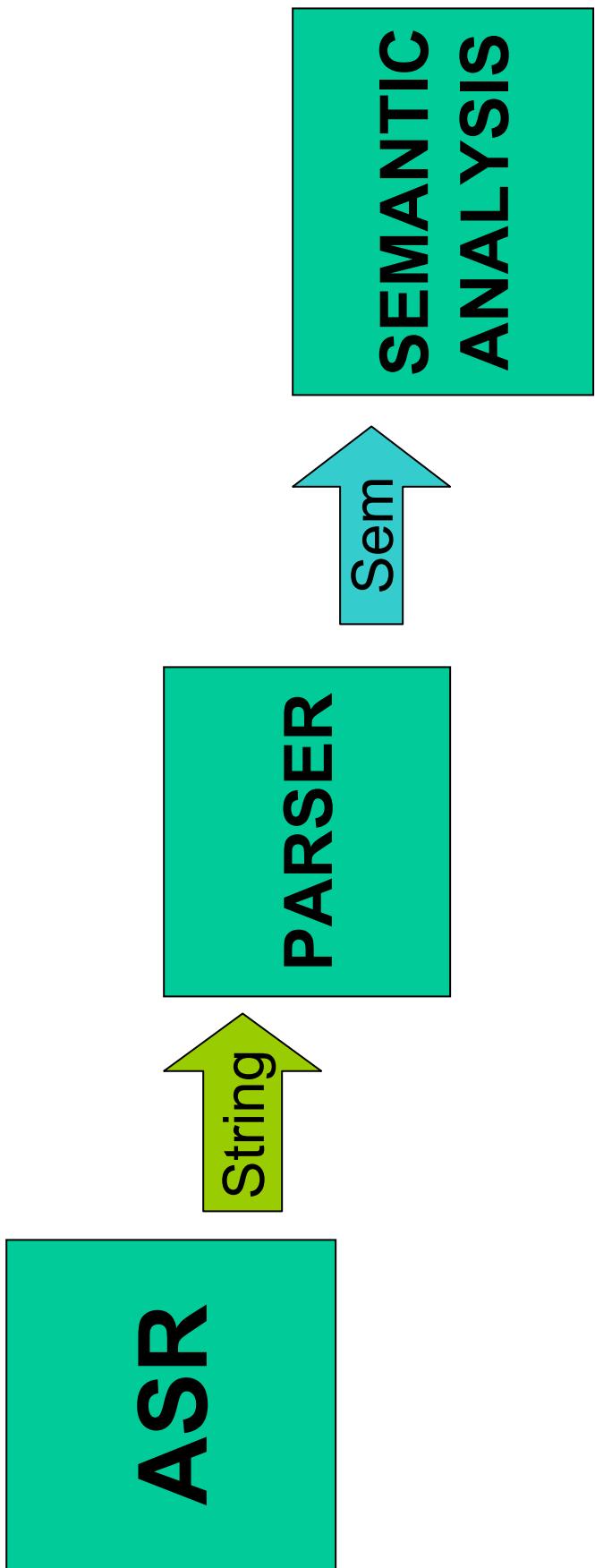


ASR

Go too the kitchen
Go to a kitchen
Oh to the kitchen
Go it at
Go and take it

- ASR output is a lattice or a set of strings
- Many non-grammatical productions
- Use parser to select string and produce logical form for interpretation

The basic pipeline for natural language understanding in speech applications



Automatic Speech Recognition

- The words an ASR can recognize are limited and mostly tuned to a particular application
- Build a speech recognition package:
 - pronunciations of the words
 - acoustic model
 - language model
 - Grammar-based
 - Statistical model

Language Models

- Statistical Language Models (bigrams)
 - Bad: need a large corpus
 - Bad: non-grammatical output possible
 - Good: relatively high accuracy (low WER)
- Grammar-based Language Models
 - Good: no large corpus required
 - Good: output always grammatical
 - Bad: lack of robustness
- In this talk we will explore grammar-based approaches

An Example: NUANCE

- The NUANCE speech recognizer supports the Grammar Specification Language (GSL)
 - lowercase symbols: terminals
 - uppercase symbols: non-terminals
 - $[X \dots Y]$: disjunction
 - $(X \dots Y)$: conjunction
- Suppose we want to cover the following kind of expressions
 - Go to the kitchen/hallway/bedroom
 - Turn left/right
 - Enter the first/second door on your left/right

Example GSL Grammar

Command

```
[ (go to the Location)
  (turn Direction)
  (enter the Ordinal door on your Direction) ]
```

Location

```
[ kitchen hallway (dining room) ]
```

Direction

```
[ left right ]
```

Natural Language Understanding

- We don't just want a string of words from the recogniser!
- It would be nice if we could associate a semantic interpretation to a string
- Preferably a logical form of some kind
- Nuance GSL offers slot-filling
- Other methods (post-processing) are of course also possible

Interpretation: adding slots

Command

```
[ (go to the Location:a) {<destination $a>}  
  (turn Direction:b) {<rotate $b>}  
  (enter the Ordinal:c door on your  
   Direction:d) {<door $c> <position $d>} ]
```

Location

```
[ kitchen {return (kitchen)}  
  hallway {return (hallway)}  
  (dining room) {return (diningroom)} ]
```

GSL & NUANCE

- Good:
 - allows tuning to a particular application in a convenient way
- Bad:
 - Tedious to build for serious applications and difficult to maintain
 - Limited expressive power
 - Slot-filling not a serious semantics (compositional semantics preferred)

How to improve on this...

- Use a **linguistic grammar** as starting point (what's the idea behind this?)
- We will use a **unification grammar** which works with phrase structure rules
- Use a **generic semantics** in the UG
- Compile UG into GSL,
- and Bob is your uncle!



What I mean by '**Compositional Semantics**'

- Semantic operations based on lambda calculus, e.g.:
 - $S \rightarrow NP\ VP$ (without semantics)
 - $S:\alpha(\beta) \rightarrow NP:\alpha\ VP:\beta$ (with semantics)
- Functional application and beta-conversion (no unification)
- Independent of syntactic formalism

Example of a Unification Grammar we work with

```
[cat : np, case : _, num : N, per : 3, refl : no, sem : apply(X, Y)] --> [cat : det, count : C, num : N, sem : X],  
[cat : noun, count : C, num : N, sem : Y].  
  
[cat : noun, count : yes, num : sg, sem : lambda(X, radio(X))] --> [lex : radio].  
  
[cat : noun, count : yes, num : sg, sem : lambda(X, car(X))] --> [lex : car].
```

- Mostly atomic feature values, untyped
- Range of values extensionally determined
- Complex features for traces
- Feature sem to hold semantic representation
- Semantic representations are expressed as Prolog terms

Idea: compile UG to GSL

- Create a context-free backbone of the UG
- Use syntactic features in the translation to non-terminal symbols in GSL
- Previous Work:
 - Rayner et al. 2000, 2001
 - Dowding et al. 2001 (typed unification grammar)
 - Kiefer & Krieger 2000 (HPSG)
 - Moore (2000)
- Previous work does not concern semantics
- UNIANCE compiler (Sicstus Prolog)

Compilation Steps (UNIANCE)

- *Input:* UG rules and lexicon
- Feature Instantiation
- Redundancy Elimination
- Packing and Compression
- Left Recursion Elimination
- Incorporating Compositional Semantics
- *Output:* rules in GSL format



Feature Instantiation

- Create a context-free backbone of the unification grammar
- Collect range of feature values by traversing grammar and lexical rules (for features with a finite number of possible values)
- Disregard Feature **SEM**
- Result is set of rules of the form $C_0 \rightarrow C_1 \dots C_n$ where C_i has structure $\text{cat}(A, F, X)$ with
 - A** a category symbol,
 - F** a set of instantiated feature value pairs,
 - X** the semantic representation

Eliminating Redundant Rules

- Rules might be redundant with respect to application domain
 - (or grammar might be ill-formed)
- Two reasons for a production to be redundant:
 - A non-terminal member of a RHS does not appear in a production as LHS
 - A LHS category (not the beginner) does not appear as RHS member
- Remove such rules until fixed point is reached

Packing and Compression

- Pack together rules that share LHSs
- Compress productions by replacing a set of rules with the same RHS by a single production:
 - Replace pair $C_i \rightarrow C$ and $C_j \rightarrow C (i \neq j)$ by $C_k \rightarrow C (C_k \text{ a new category})$
 - Substitute C_k for all occurrences of C_i and C_j in the grammar

Eliminating Left Recursion

- Left-recursive rules are common in linguistically motivated grammars
- GSL does not allow LR
- Standard way of eliminating LR
 - Aho et al. 1996, Greibach Normal Form
 - Here we only consider immediate left-recursion
- Replace pairs of $A \rightarrow AB$, $A \rightarrow C$ by $A \rightarrow CA'$, $A' \rightarrow BA'$ and $A' \rightarrow \epsilon$
- Put differently:
 - by $A \rightarrow CA'$, $A' \rightarrow BA'$, $A \rightarrow C$ and $A' \rightarrow B$

LR Elimination Rule (1)

For each left-recursive category with non-recursive grammar productions of the form

$$\text{cat}(A, F, X) \rightarrow C_1 \dots C_n$$

All dependencies in C_i on X are preserved

extend the grammar with productions

$$\text{cat}(A, F, Z(X)) \rightarrow C_1 \dots C_n \text{ cat}(A', F, Z)$$

LR Elimination Rule (2)

For each left-recursive category, replace all productions of the form

$$\text{cat}(A, F, X) \rightarrow \text{cat}(A, F, Y) C_1 \dots C_n$$

by the following two productions

$$\text{cat}(A', F, \lambda Y . Z(X)) \rightarrow C_1 \dots C_n \text{ cat}(A', F, Z)$$

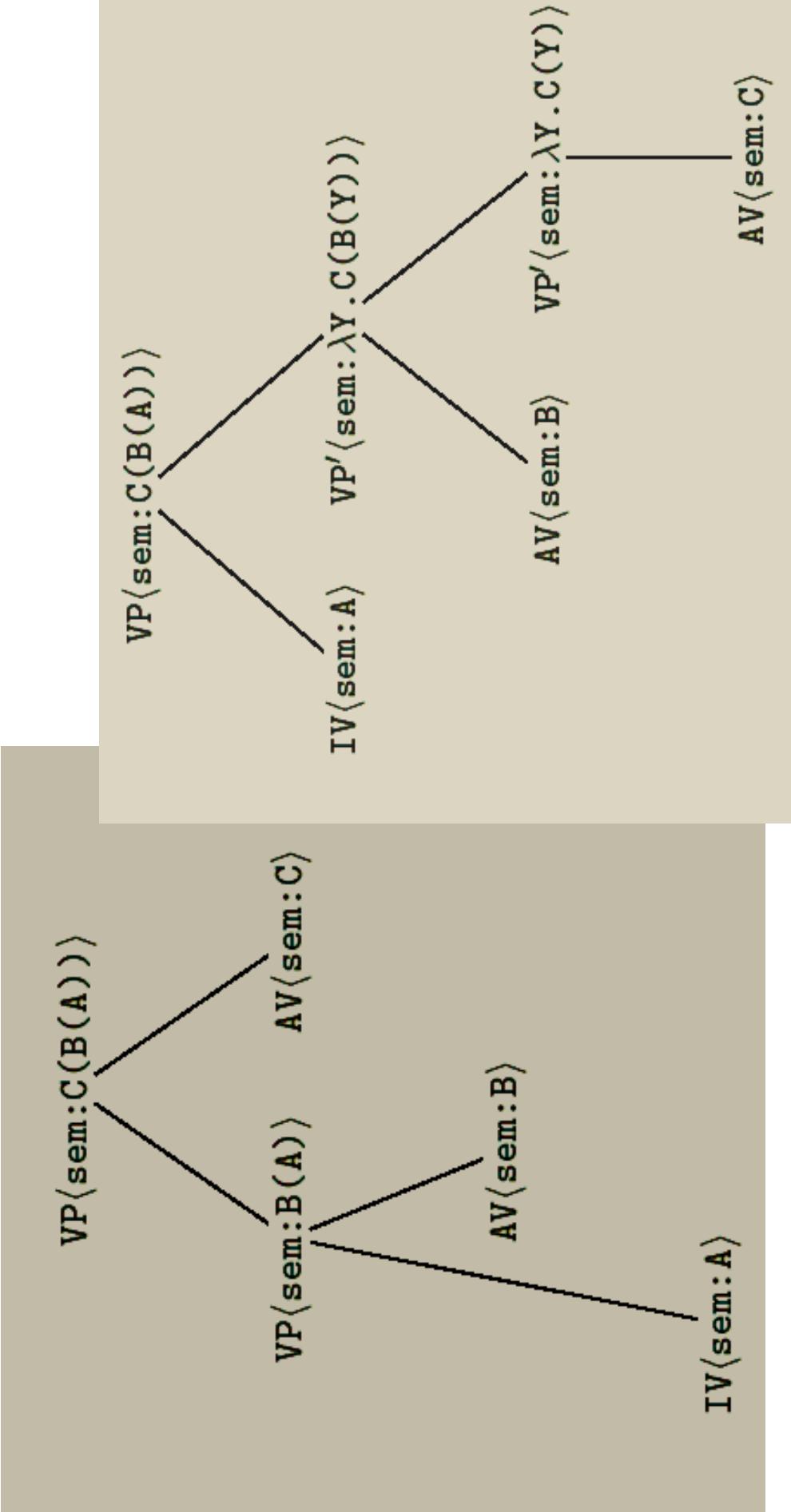
$$\text{cat}(A', F, \lambda Y . X) \rightarrow C_1 \dots C_n$$

All dependencies of Y and
C_i on X are preserved

Example

- Original grammar (left recursive VP)
 - $\text{VP}[\text{sem}:X] \rightarrow \text{IV}[\text{sem}:X]$
 - $\text{VP}[\text{sem}:A(B)] \rightarrow \text{VP}[\text{sem}:B] \text{ AV}:[\text{sem}:A]$
- Transferred grammar (right recursive VP)
 - $\text{VP}[\text{sem}:X] \rightarrow \text{IV}[\text{sem}:X]$
 - $\text{VP}[\text{sem}:Z(X)] \rightarrow \text{IV}[\text{sem}:X] \text{ VP}'[\text{sem}:Z]$
 - $\text{VP}'[\text{sem}:\lambda B.A(B)] \rightarrow \text{AV}[\text{sem}:A]$
 - $\text{VP}'[\text{sem}:\lambda B.Z(A(B))] \rightarrow \text{AV}[\text{sem}:A] \text{ VP}'[\text{sem}:Z]$

Example (Picture)



Left-Recursive Derivation / Right-Recursive Derivation

Incorporating Compositional Semantics

- At this stage we have a set of rules of the form $\text{LHS} \rightarrow \mathbf{C}$, where \mathbf{C} is a set of ordered pairs of RHS categories and corresponding semantic values
- Convert LHS and RHS to GSL categories (straightforward)
- Bookkeeping required to associate semantic variables with GSL slots
 - Semantic operations are composed using the built-in `strcat / 2` function

Example (Input UG)

S(sem:X(Y($\lambda e.\text{present}(e)$))) → NP(sem:X) VP(sem:Y)
NP(sem: $\lambda P.\exists x[\text{person}(x) \& P(x)]$) → somebody
NP(sem: $\lambda P.P(\text{hearer})$) → you
VP(sem: X(Y)) → VP(sem: Y) AV(sem: X)
VP(sem: X) → IV(sem: X)
IV(sem: $\lambda P.\lambda x.\exists e[\text{walk}(e) \& [\text{agent}(e, x) \& P(e)]]$) → walk
IV(sem: $\lambda P.\lambda x.\exists e[\text{walk}(e) \& [\text{agent}(e, x) \& P(e)]]$) → walks
AV(sem: $\lambda E.\lambda P.E(\lambda e.[\text{home}(e) \& P(e)])$) → home
AV(sem: $\lambda E.\lambda P.E(\lambda e.[\text{quick}(e) \& P(e)])$) → quickly

```

.u [S:sem {<sem strcat($sem ".")>}]

Av
[(" quickly ") {return( "l(A,l(B,a(A,l(C,quick(C),a(B,C))))))")]
 (" home ") {return( "l(D,l(E,a(D,l(F,home(F),a(E,F))))))")]
 (" walk ")
[(" walks ")
{return( "l(A,l(B,exists(C,walk(C),and(agent(C,B),a(A,C))))))")]
 (" you ")
{return( "l(A,a(A,hearer))")]

MpCasenomMumsMsgPer2
[(" somebody ") {return( "l(A,exists(B,exists(C,person(B),a(A,B))))")}]
 (" ")
[(" MpCasenomMumsMsgPer3:a VpNumsgPer3:b )
{return(strcat("a(" strcat(strcat($a strcat("$a strcat("a(" strcat(strcat(strcat($b strcat("$b strcat("l(C," strcat("present(C)" "
"))))) "))))"))
 (" ")
[(" MpCasenomMumsMsgPer2:a VpNumsgPer2:b )
{return(strcat("a(" strcat(strcat($a strcat("$a strcat("a(" strcat(strcat(strcat($b strcat("$b strcat("l(F," strcat("present(F)" "
"))))) "))))"))
 (" ")
[(" VpNumsgPer2
[(" IvnMsgPer2:a ) {return($a)}
 (" IvnMsgPer2:a RecNewCatVpNumsgPer2:b )
{return(strcat("a(" strcat(strcat($b strcat("$b strcat("a(" strcat(strcat("$b strcat("a(" strcat("$a ) "))))"))
 (" ")
[(" VpNumsgPer3
[(" IvnMsgPer3:a ) {return($a)}
 (" IvnMsgPer3:a RecNewCatVpNumsgPer3:b )
{return(strcat("a(" strcat(strcat("$b strcat("$b strcat("a(" strcat("$a ) "))))"))

```

Example (GSL Output)

Example (Nuance Output)

Epilepsia, Vol. 46, No. 2, 2005

Grammaticalization

Krakowian + Voll+

1000 1000

RESULTS

The Second

you turn the second

Rec Errors: 0 ins

Rec Total : 0 ins

Precision: 100,00

Bevacau 100.00

- 100 -

—Meschi et al.

NL RES. #0

卷之三

Passer (acc) (B, I)

(a), (b), (c), (d), (e)

H_{CO}₂-patient(H_{CO}₂)_pr

LCD memory [1]

卷之三

卷之三

卷之三

卷之三

Glossary

卷之三

卷之三

100-1

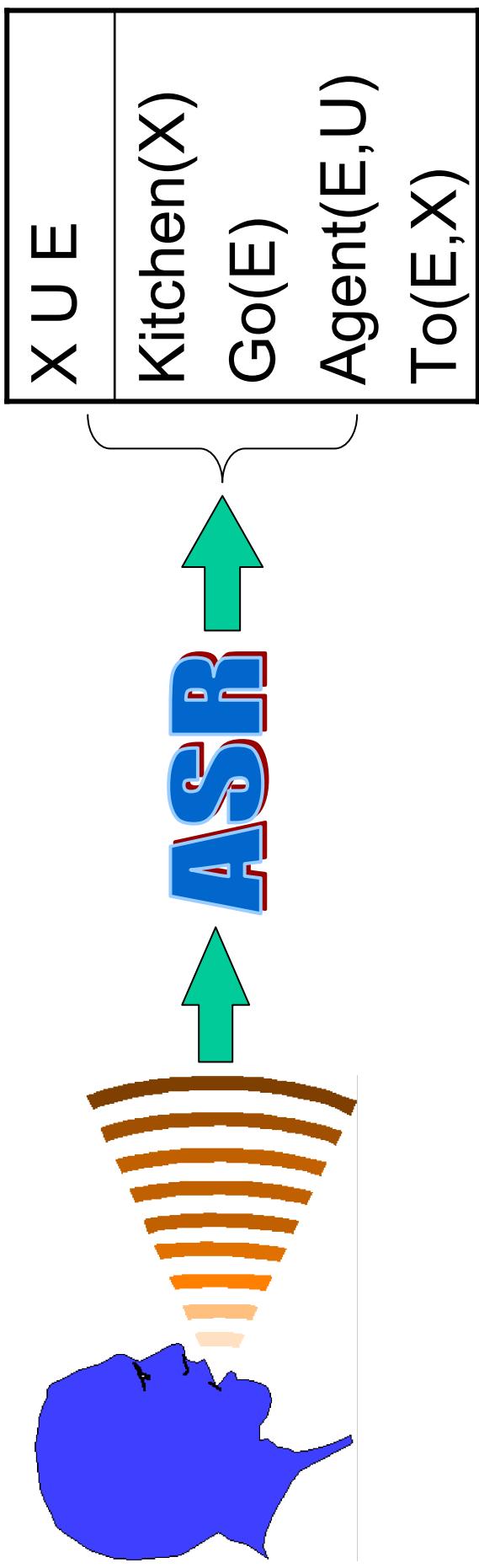
卷之三

卷之三

卷之三

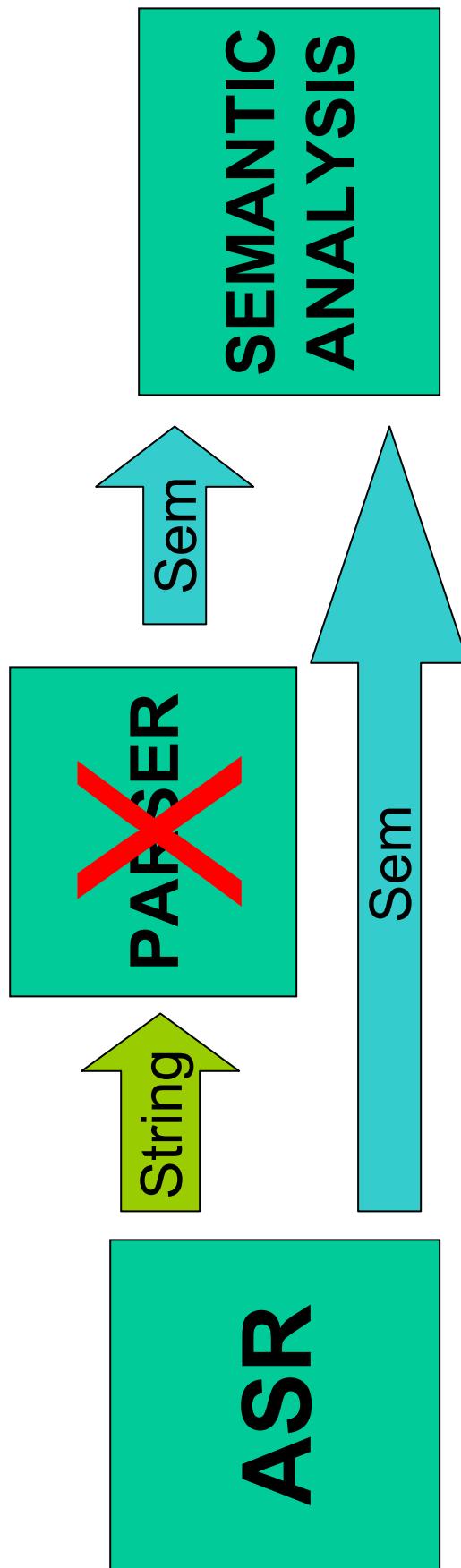
לען, לערן, לערן

Automatic speech recognition with our new approach



- Put compositional semantics in language models
- ASR output comprises logical forms (e.g., a DRS)
- No need for subsequent parsing

This is nice because it makes
the parser redundant



Practical Results

- Generate two GSL grammars:
 - One *without* compositional semantics
 - One *with* compositional semantics
- Results:

	Without semantics	With semantics
Average speech processing time	4.74 secs	4.97 secs
× Real Time	1.028xRT	1.077xRT

Further improvements: Adding Probabilities to GSL

- Include probabilities to increase recognition accuracy
- Done by bootstrapping GSL grammar:
 - Use first version of GSL to parse a domain specific corpus
 - Create table with syntactic constructions and frequencies
 - Choose closest attachment in case of structural ambiguities
 - Add obtained probabilities to original GSL grammar



Conclusions (Possibly Bad News)

- No means for robust processing
- Integration with Statistical Models non-trivial
- Only cases of immediate left-recursion are covered
 - Moore (2000) uses Greibach Normal Form with regular expressions in GSL
 - Unclear how to integrate compositional semantics



Conclusions (Good News)



- The Grammar-based approach to Speech Recognition: Post-processing of speech output restricted to beta-conversion
- No computational overhead
- Empirical evidence that such language models are useful in applications
 - Only small corpus required