Parsing and Speech Research at Brown University

Mark Johnson

Brown University

The University of Tokyo, March 2004

Joint work with Eugene Charniak, Michelle Gregory and Keith Hall

Supported by NSF grants LIS 9720368 and IIS0095940

Talk outline

- Language models for speech recognition
  - Dynamic programming for language modeling
- Prosody and parsing
- Disfluencies and parsing
  - Do disfluencies help parsing?
  - Recognizing and correcting speech repairs
- Conclusions and future work

Applications of (statistical) parsers

Two different ways of using statistical parsers:

1. Applications that use syntactic *parse trees*
   - information extraction
   - (short answer) question answering
   - summarization
   - machine translation
2. Applications that use the *probability distribution* over strings or trees (parser-based language models)
   - *speech recognition and related applications*
   - machine translation

Language modeling with parsers

The *noisy channel model* consists of two parts:

The language model: $\mathrm{P}(x)$, where x is a sentence

The acoustic model: $\mathrm{P}(y|x)$, where y is the acoustic signal

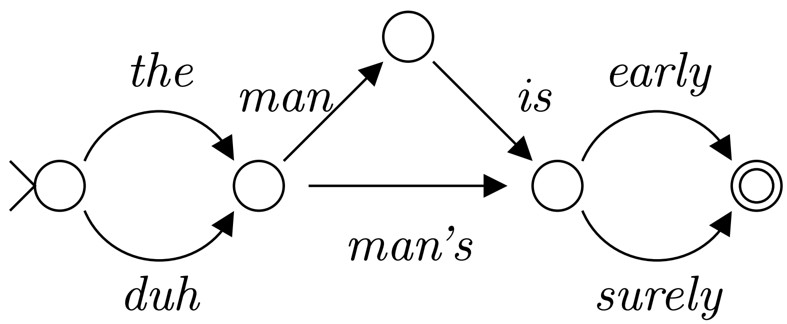
$$\begin{aligned} \mathrm{P}(x|y) &= \frac{\mathrm{P}(y|x)\mathrm{P}(x)}{\mathrm{P}(y)} \qquad \textit{(Bayes Rule)} \\ x^\star(y) &= \operatorname*{argmax}_x \mathrm{P}(x|y) = \operatorname*{argmax}_x \mathrm{P}(y|x)\mathrm{P}(x) \end{aligned}$$

Syntactic parsing models now provide state-of-the-art performance in language modeling $\mathrm{P}(x)$ (Chelba, Roark, Charniak).

Parsing vs language modeling

- A language model models the *marginal distribution* $\mathrm{P}(X)$ over strings X
- A parser models the *conditional distribution* $\mathrm{P}(Y|X)$ of parses Y given a string X
- Different kinds of features seem to be useful for these tasks (Charniak 01)
  - Tri-head features (the syntactic analog of trigrams) are useful for language modeling, but not for parsing
  - EM(-like) training on unparsed data helps language modeling, but not parsing

n-best list approaches



1. the man is early

2. duh man is early

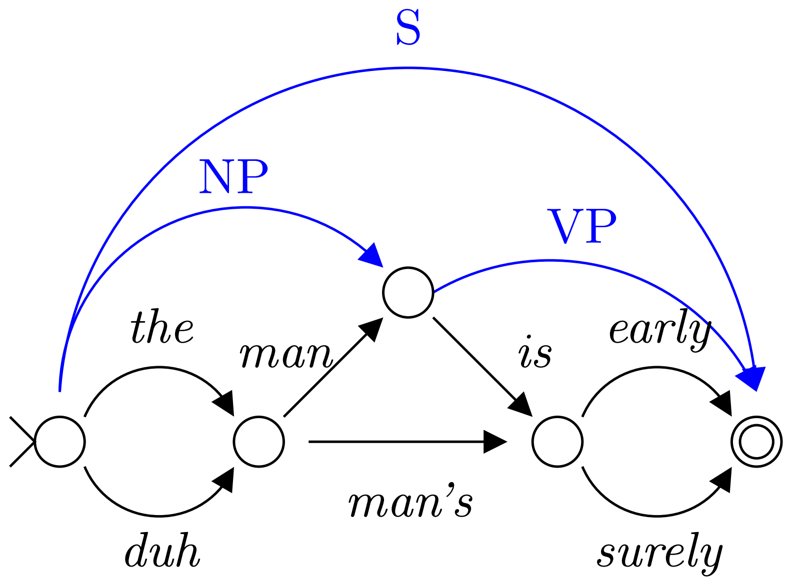
3. the man's early

4. the man is surely

$\ldots$

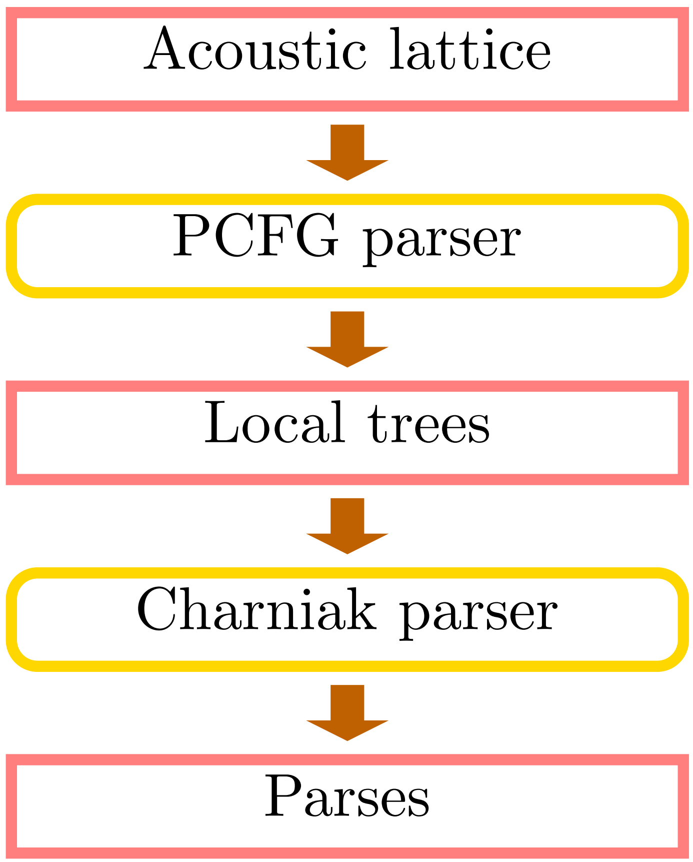
- Roark (p.c.) reports WER improvements with 1,000-best lists
- Can we improve search efficiency and WER by parsing from the lattice? (Chelba, Roark)

Lattices and Charts (IEEE ASRU '03)



- Lattices and charts are the same *dynamic programming* data structure
- Best-first chart parsing works well on strings
- Can we adapt *best-first coarse-to-fine chart-parsing techniques* to lattices?

Coarse to fine architecture



- Use a “coarse-grained” analysis to identify where a “fine-grained” analysis should be applied

Coarse to fine parsing

- Parsing with the full "fine-grained" grammar is slow and takes a lot of memory (Charniak 2001 parser)
- Use a "coarse-grained" grammar to indicate location of likely constituents (PCFG)
- Fine-grained grammar splits each coarse constituent into many fine constituents
- Works well for string parsing:
  - Posits $\approx$ 100 edges to first parse
  - A very good parse is included in $10\times$ overparsing
- Will it work on speech lattices?

Coarse to fine on speech lattices

- PCFG and Charniak Language Model WER:

	WER
trigram (40million words)	13.7
Roark01 (n-best list)	12.7
Chelba02	12.3
Charniak (n-best list)	11.8
100x overparsing on n-best lattices	12.0
100x overparsing on acoustic lattices	13.0

Summary and current work

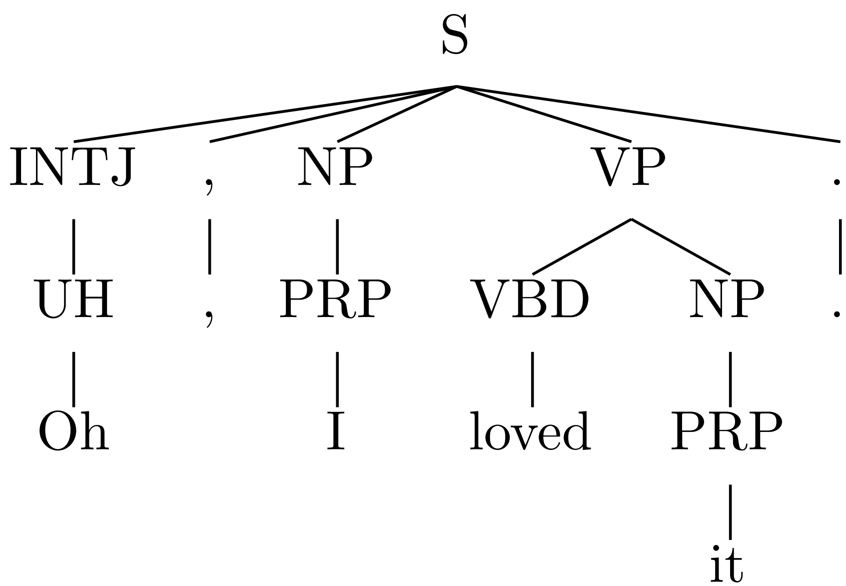
- The coarse-grained model doesn't seem to include enough good parts of the lattice
- If we open the beam further, the *fine-grained model runs out of memory*
- Current difficulties probably stem from *defective nature* of coarse-grained PCFG model

$\Rightarrow$ improve coarse-grained model

$\Rightarrow$ *lexicalization* will probably be necessary
(we are competing with trigrams, which are lexicalized)

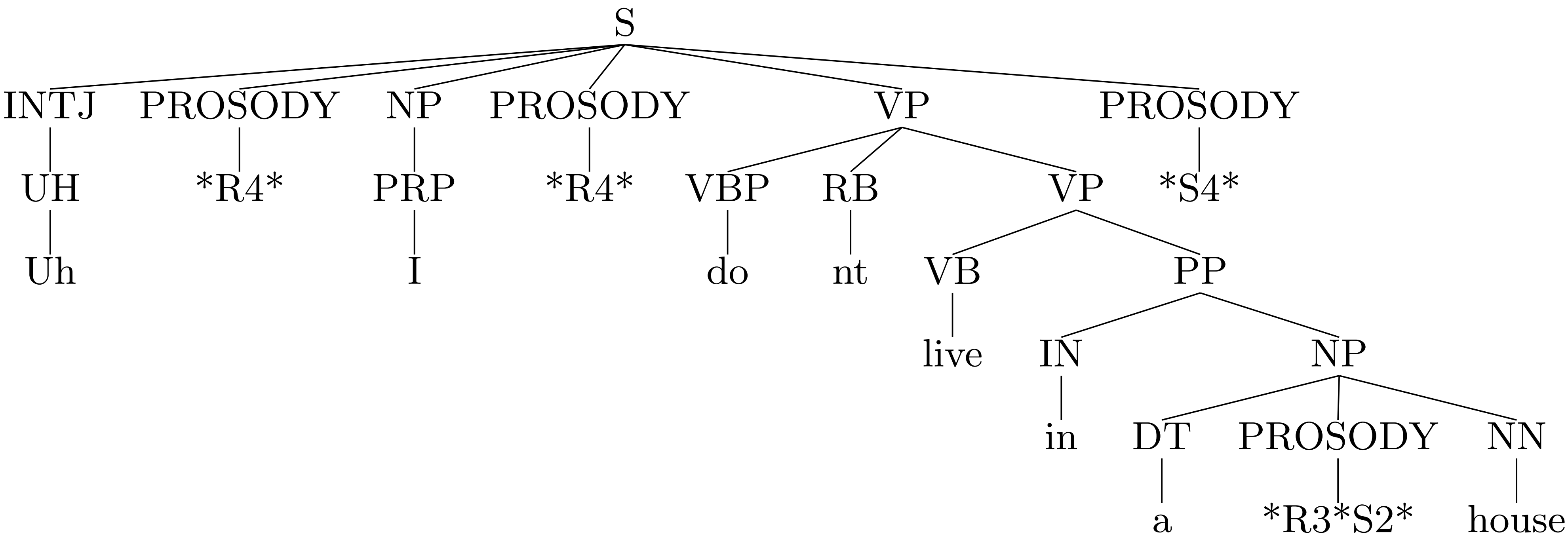
- Can we parse efficiently from a lattice with a lexicalized PCFG?
- Will a three-stage model work better?

Prosody and parsing (NAACL'04)



- Selectively removing punctuation from the WSJ significantly decreases parsing performance
- When parsing speech transcripts, would prosody enhance parsing performance also?

Prosody as punctuation



- Extract prosodic features from acoustic signal (Ferrer 02)
- Use a *forced aligner* to align Switchboard transcript with acoustic signal
- Extract prosodic features from acoustic signal and associate them with a word in transcript
- Bin prosodic features, and attach them in syntactic tree much as punctuation is

Prosodic features we tried

PAU_DUR_N: pause duration normalized by the speaker's mean sentence-internal pause duration,

NORM_LAST_RHYME_DUR: duration of the phone minus the mean phone duration normalized by the standard deviation of the phone duration for each phone in the rhyme,

FOK_WRD_DIFF_MNMN_NG: log of the mean f0 of the current word, divided by the log mean f0 of the following word, normalized by the speakers mean range,

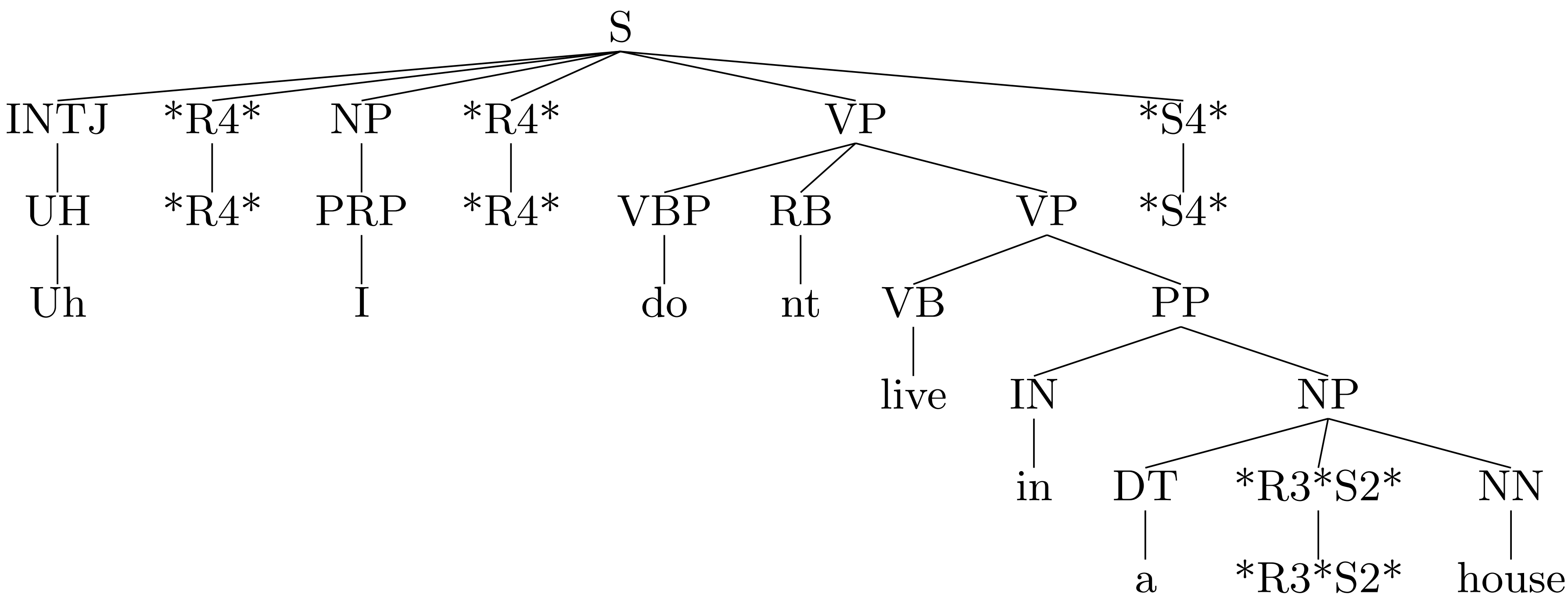
FOK_LR_MEAN_KBASELN: log of the mean f0 of the word normalized by speaker's baseline, and

SLOPE_MEAN_DIFF_N: difference in the f0 slope normalized by the speaker's mean f0 slope.

Binning the prosodic features

- Modern statistic parsers take categorical input, our prosodic features are continuous
- We experimented with many ways of *binning the prosodic feature values*:
  - construct a histogram for all features used
  - divide feature values into 2/5/10 equal sized bins
  - only introduce pseudo-punctuation for the most extreme 40% of bins
  - conjoin binned features
- When *all features* are used:
  - 89 distinct types of pseudo-punctuation symbols
  - 54% of words are followed by pseudo-punctuation

Prosody as punctuation



- Different types of punctuation have different POS tags in WSJ
- POS tags and lexical items are used in different ways in Charniak parsing model

⇒ Also evaluate with "raised" prosodic features

Prosodic parsing results

Annotation	unraised	raised
punctuation	88.212	
none	86.891	
L	85.632	85.361
NP	86.633	86.633
P	86.754	86.594
R	86.407	86.288
S	86.424	85.75
W	86.031	85.681
P R	86.405	86.282
P W	86.175	85.713
P S	86.328	85.922
P R S	85.64	84.832

- Punctuation improves parsing accuracy
- All combinations of prosodic features decrease parsing accuracy
- The more features we used, the more accuracy decreased

Discussion

- Wrong features? Wrong model? (But why does the "wrong model" work so well with punctuation?)
- Why did performance go down?
  - Charniak parser backs off to a bigram model
  - Prosodic punctuation pushes preceding word out of window
  - A manually identified word is probably more useful than an automatically extracted prosodic feature
- *Punctuation is annotated by humans* (who presumably understood each sentence)
- Prosody was annotated by machine (which presumably did not understand)
- *Prosody may prove more useful when parsing from speech lattices*

A TAG-based noisy channel model of speech repairs

- Goal: Apply parsing technology and "deeper" linguistic analysis to (transcribed) speech
- Identifying and correcting speech errors
  - Types of speech errors
  - Speech repairs and "rough copies"
  - Noisy channel model

Speech errors in (transcribed) speech

- Filled pauses

  I think it's, *uh*, refreshing to see the, *uh*, support . . .

- Frequent use of parentheticals

  But, *you know*, I was reading the other day . . .

- Speech repairs

  Why didn't he, why didn't she stay at home?

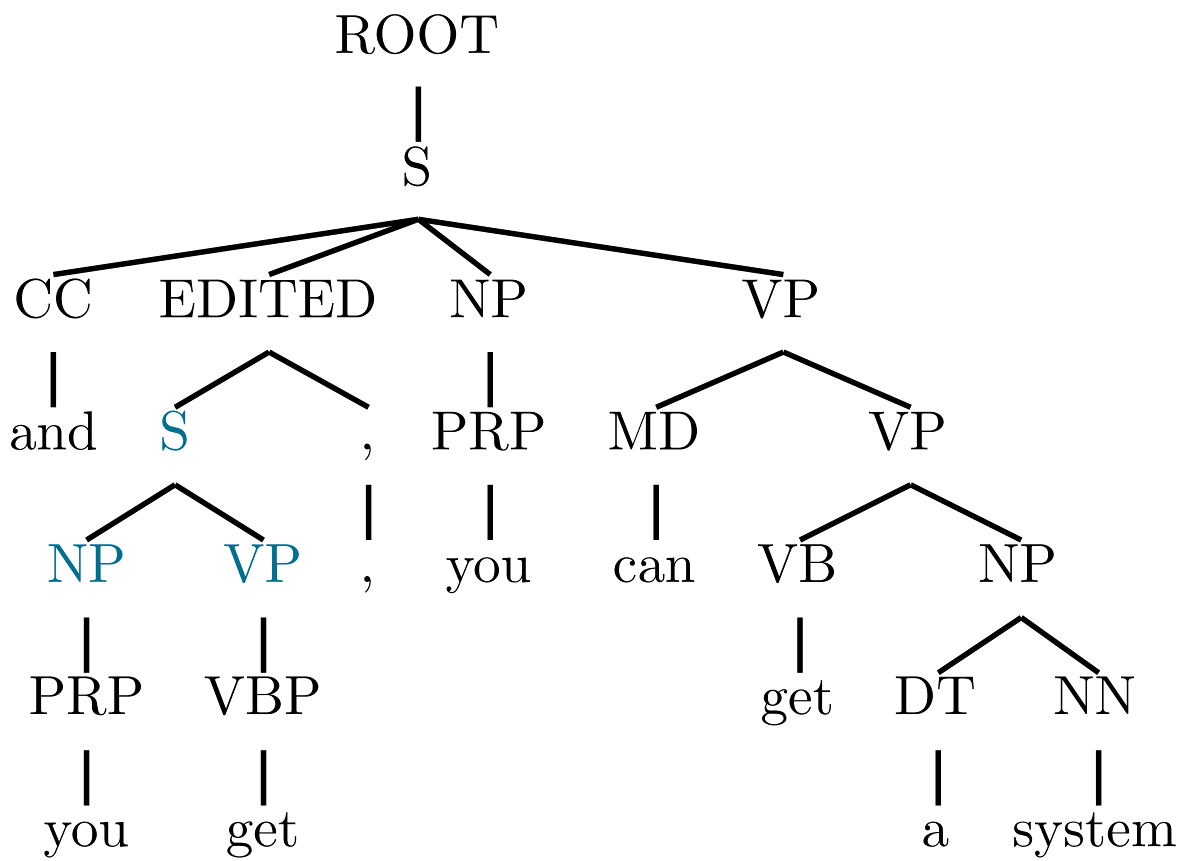
- Ungrammatical constructions

Bear, Dowding and Schriberg (1992), Charniak and Johnson (2001), Heeman and Allen (1997, 1999), Nakatani and Hirschberg (1994), Stolcke and Schriberg (1996)

Special treatment of speech repairs

- *Filled pauses* are easy to recognize (in transcripts)
- *Parentheticals* appear in WSJ, and current parsers identify them fairly well
- *Filled pauses* and *parentheticals* are useful for identifying constituent boundaries (just as punctuation is)
  - Charniak's parser performs slightly better with parentheticals and filled pauses than with them removed
- *Ungrammatical constructions* aren't necessarily fatal
  - Statistical parsers learn mapping of sentences to parses in training corpus
- …but *speech repairs* warrant special treatment, since Charniak's parser doesn't recognize them …

Representation of repairs in Switchboard treebank



- Speech repairs are indicated by EDITED nodes in corpus

Speech repairs and interpretation

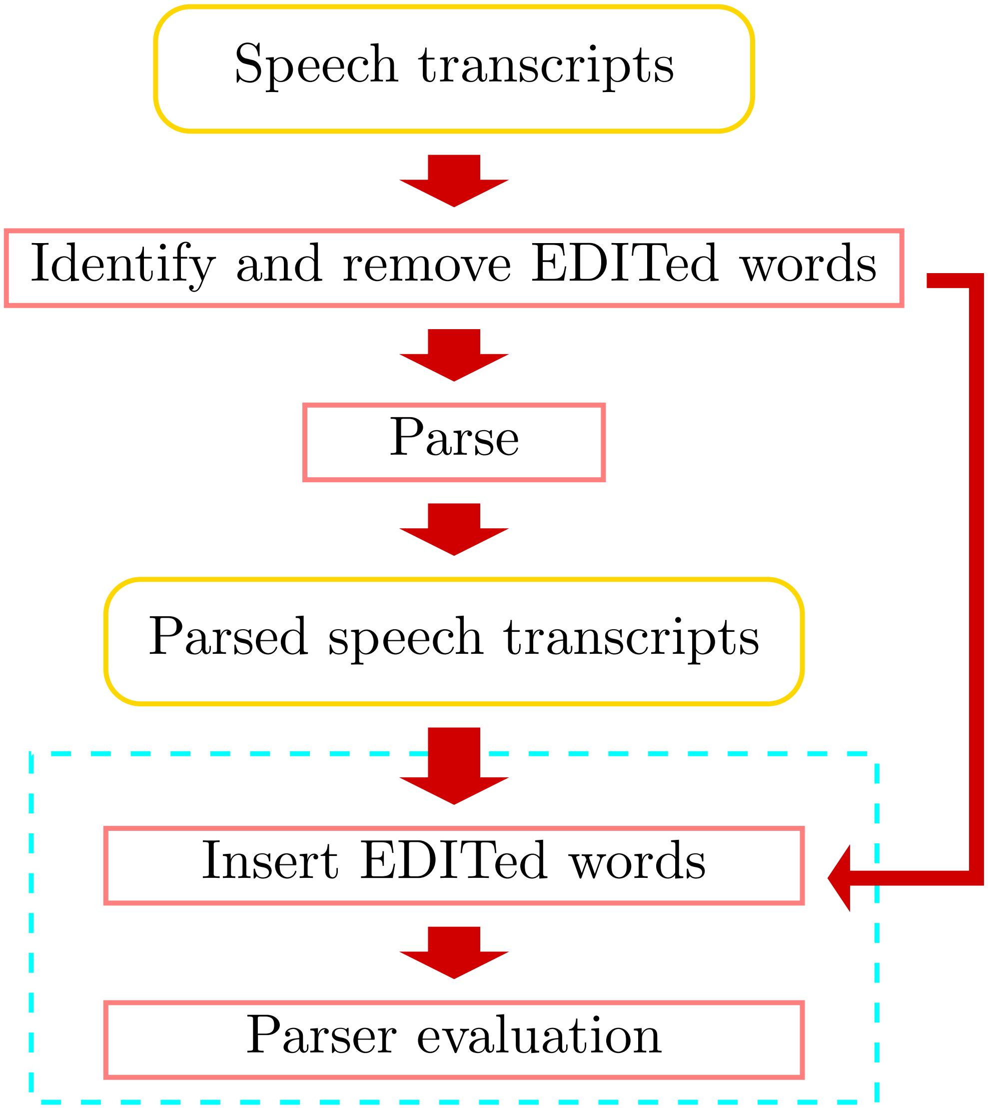
- Speech repairs are indicated by EDITED nodes in corpus
- The unadorned parser does not posit any EDITED nodes even though the training corpus contains them
  - Parser is based on context-free headed trees and head-to-argument dependencies
  - Repairs involve context-sensitive "rough copy" dependencies that cross constituent boundaries

Why didn't he, uh, why didn't she stay at home?

- The interpretation of a sentence with a speech repair is (usually) the same as with the repair excised

⇒ Identify and remove EDITED words (Charniak and Johnson, 2001)

Parser architecture



The noisy channel model

Source model $\mathrm{P}(X)$
Bigram/Parsing LM

Source signal x
a flight to Denver on Friday

Noisy channel $\mathrm{P}(U|X)$
TAG transducer

Noisy signal u
a flight to Boston uh I mean to Denver on Friday

$$\mathrm{P}(x|u) = \frac{\mathrm{P}(u|x)\mathrm{P}(x)}{\mathrm{P}(u)} \qquad \textit{(Bayes Rule)}$$

$$\operatorname*{argmax}_x \mathrm{P}(x|u) = \operatorname*{argmax}_x \mathrm{P}(u|x)\mathrm{P}(x)$$

The structure of a repair

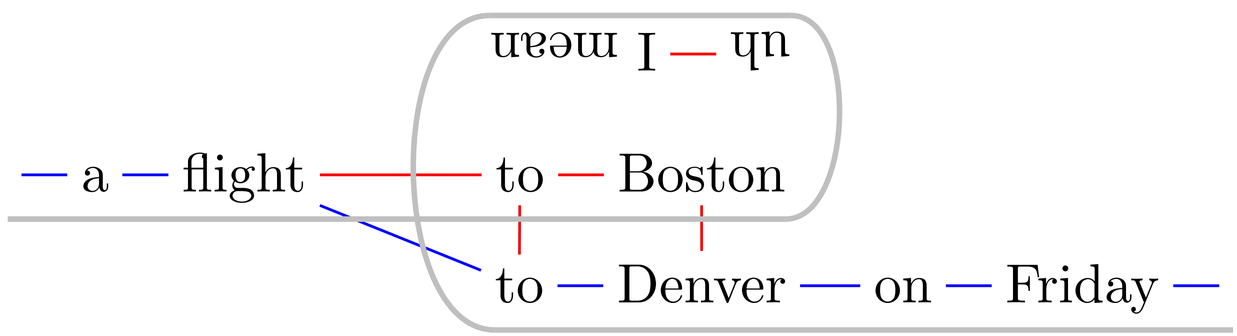
$$\ldots\text{a flight }\underbrace{\text{to Boston,}}_{\text{Reparandum}}\ \underbrace{\text{uh, I mean,}}_{\text{Interregnum}}\ \underbrace{\text{to Denver}}_{\text{Repair}}\text{ on Friday}\ldots$$

- The Interregnum is usually lexically (and prosodically marked), but can be empty
- The Repair is often "roughly" a copy of the Reparandum
  - Finite state and context free grammars cannot generate ww "copy languages" *but Tree Adjoining Grammars can*
  - Repairs are typically short
  - Repairs are not always copies

Shriberg 1994 "Preliminaries to a Theory of Speech Disfluencies"

“Helical structure” of speech repairs

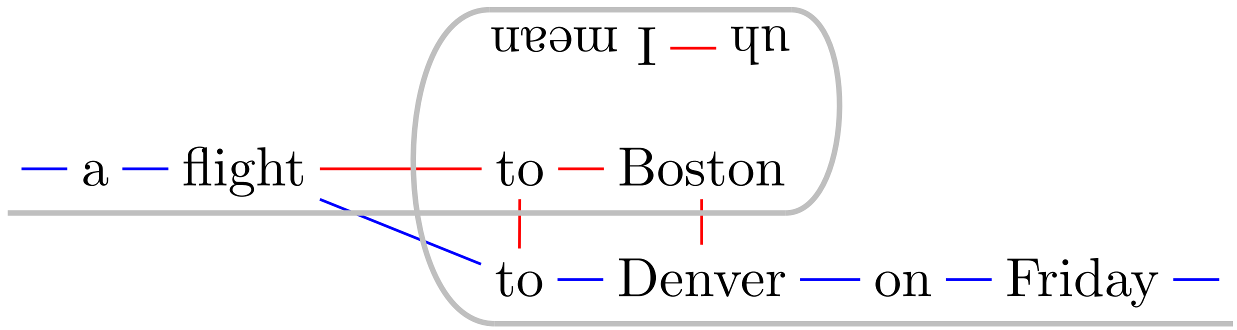
$$\ldots \text{a flight } \underbrace{\text{to Boston,}}_{\text{Reparandum}} \ \underbrace{\text{uh, I mean,}}_{\text{Interregnum}} \ \underbrace{\text{to Denver}}_{\text{Repair}} \text{ on Friday} \ldots$$



- *Language model* generates *repaired string*
- *TAG transducer* generates *reparandum* from repair
- *Interregnum* is generated by specialized finite state grammar in TAG transducer

Joshi (2002), ACL Lifetime achievement award talk

TAG transducer models speech repairs



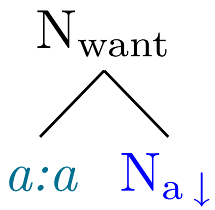
- Source (bigram) language model: *a flight to Denver on Friday*
- TAG generates string of $u{:}x$ pairs, where u is a speech stream word and x is either $\emptyset$ or a source word:
  a:a flight:flight to:$\emptyset$ *Boston:*$\emptyset$ *uh:*$\emptyset$ *I:*$\emptyset$ *mean:*$\emptyset$ *to:to Denver:Denver on:on Friday:Friday*
  - TAG does not reflect grammatical structure (but LM can)
  - right branching finite state model of non-repairs and interregnum
  - adjunction used to describe copy dependencies in repair

Sample TAG derivation (simplified)

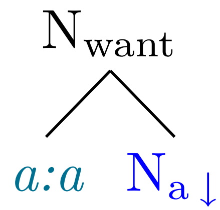
(I want) a flight to Boston uh I mean a flight to Denver on Friday ...

Start state: $\mathrm{N}_{\mathrm{want}\downarrow}$

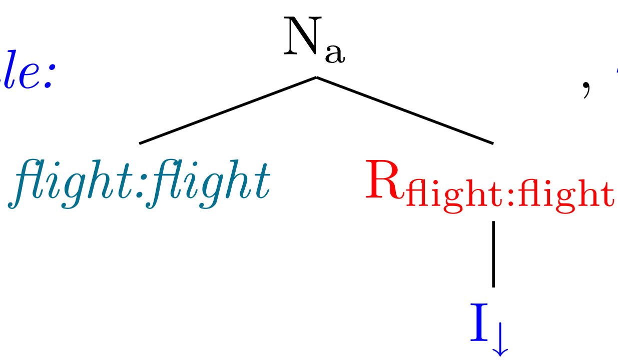
TAG rule:



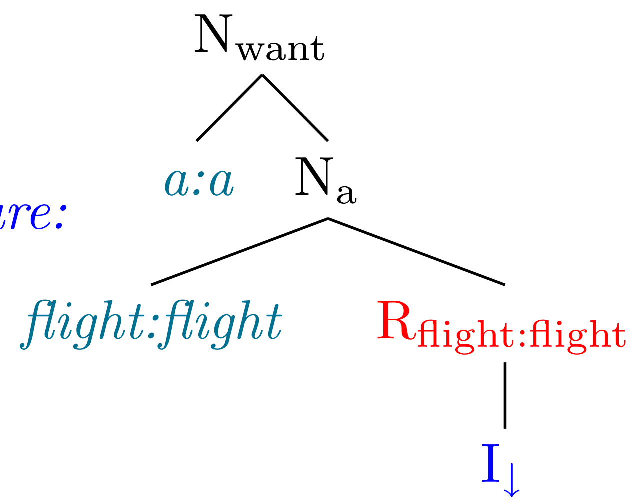
, *resulting structure:*



TAG rule:

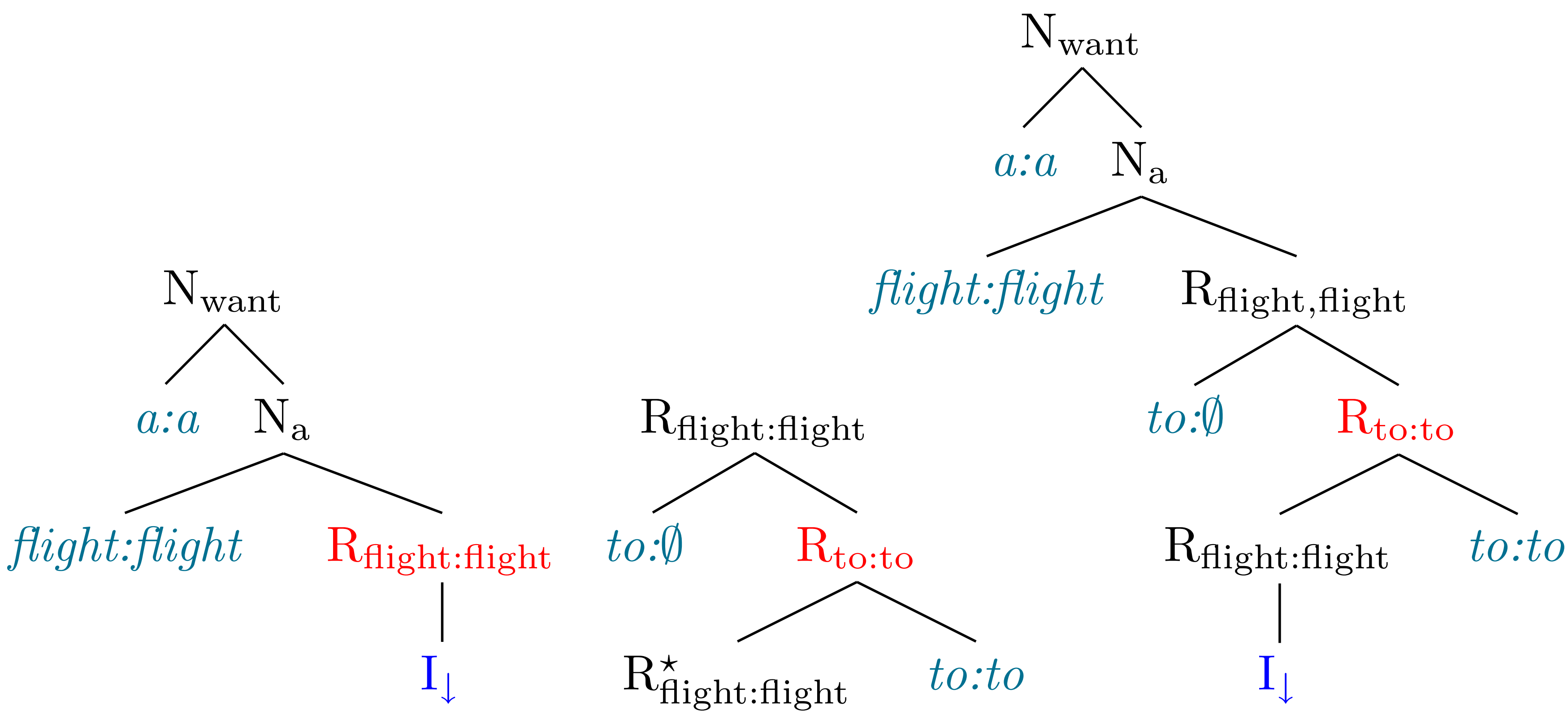


, *resulting structure:*



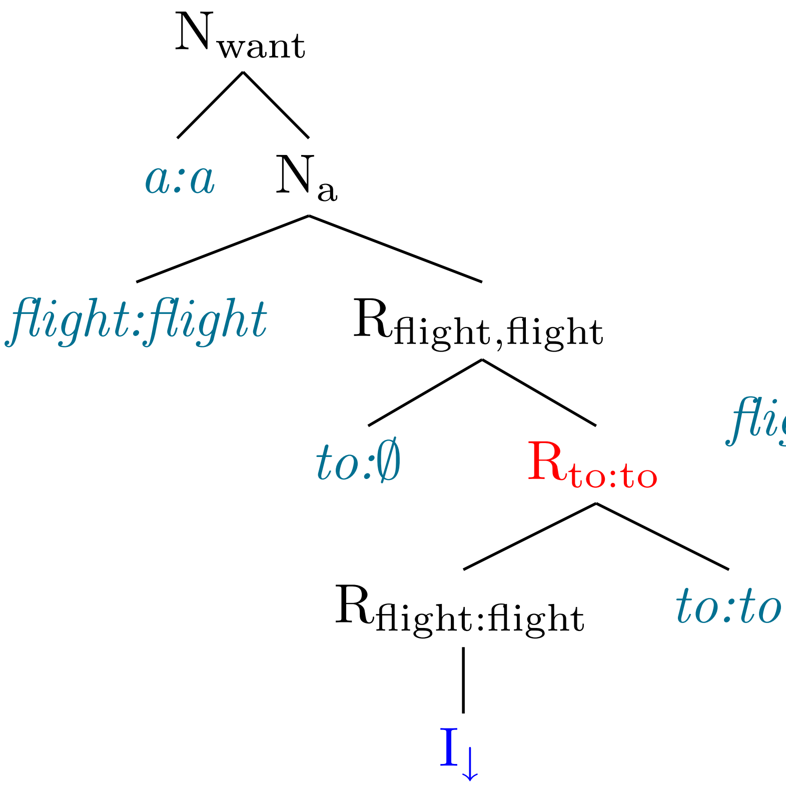
Sample TAG derivation (cont)

(I want) a flight to Boston uh I mean to Denver on Friday . . .

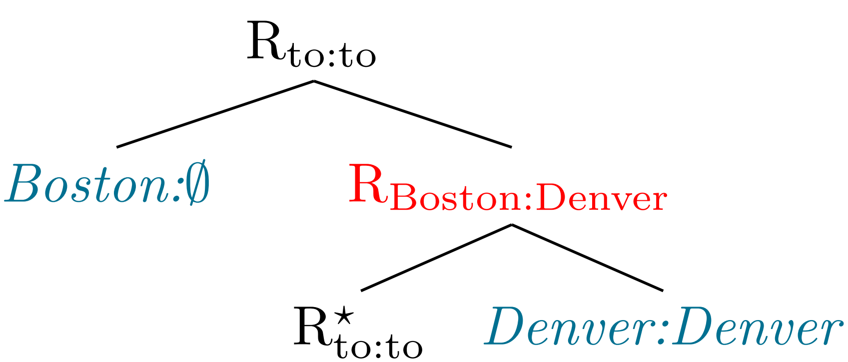


previous structure *TAG rule* *resulting structure*

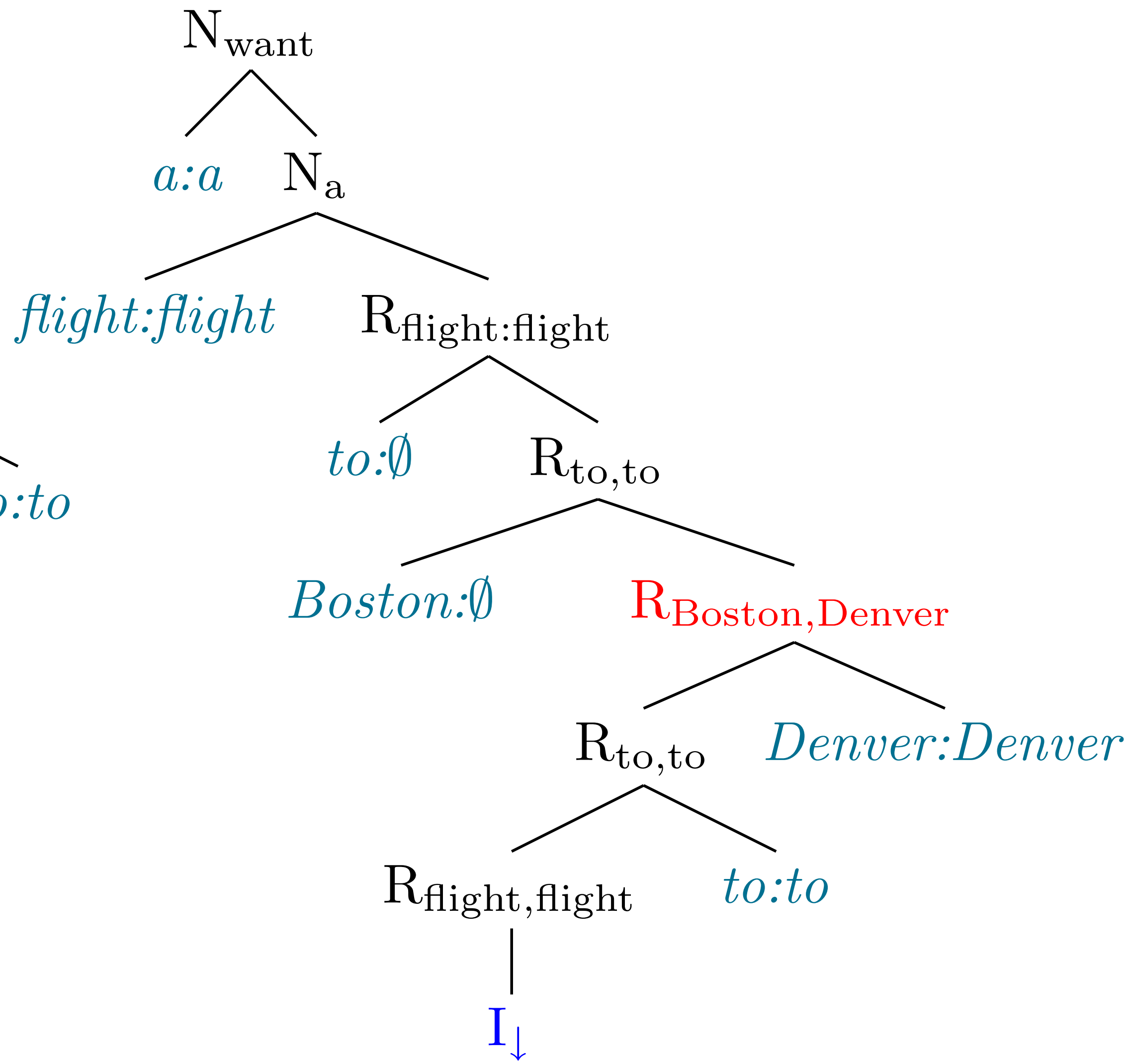
(I want) a flight to Boston uh I mean *to* Denver on Friday . . .



previous structure

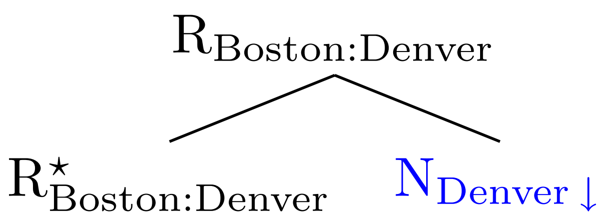


TAG rule

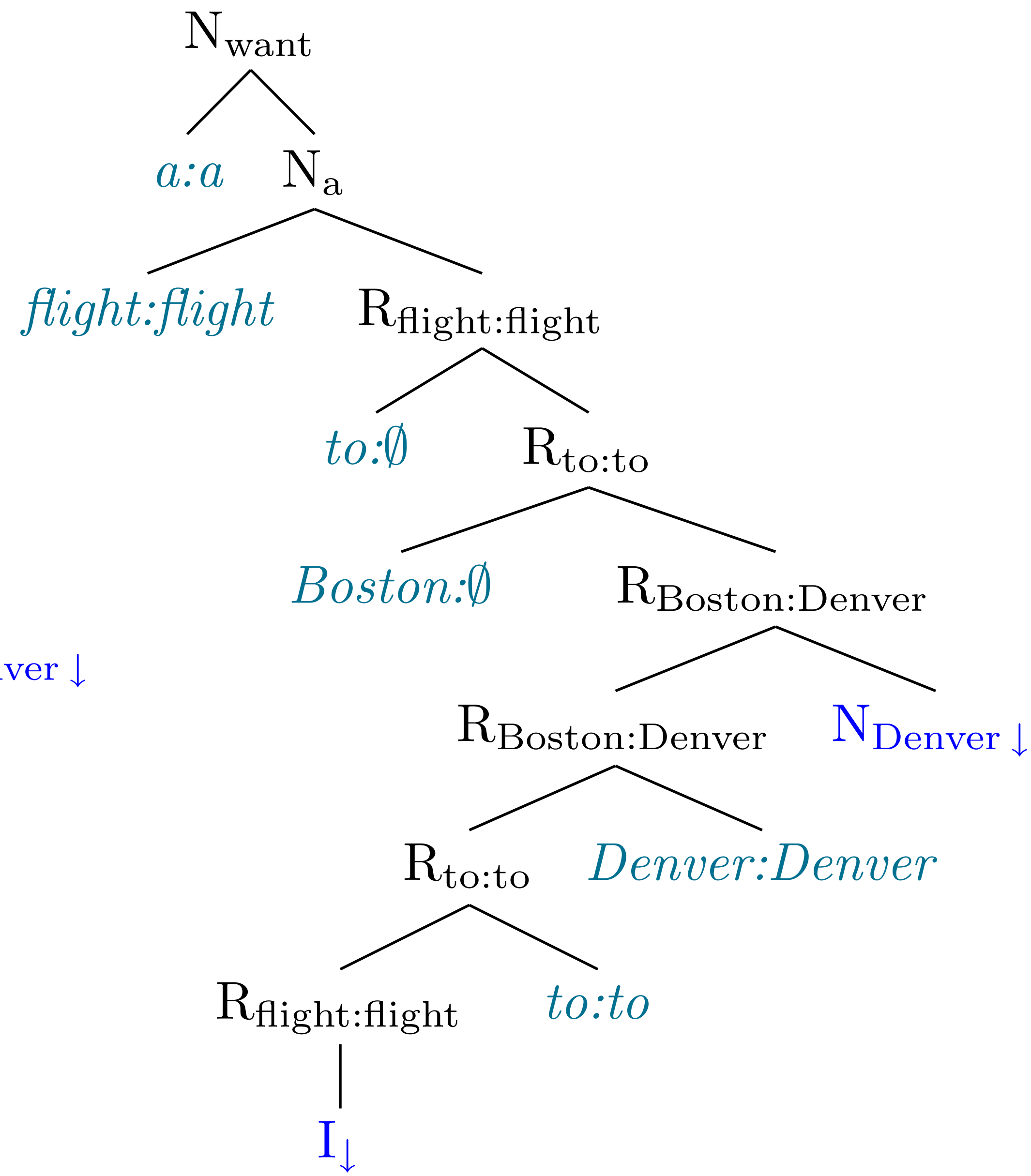


resulting structure

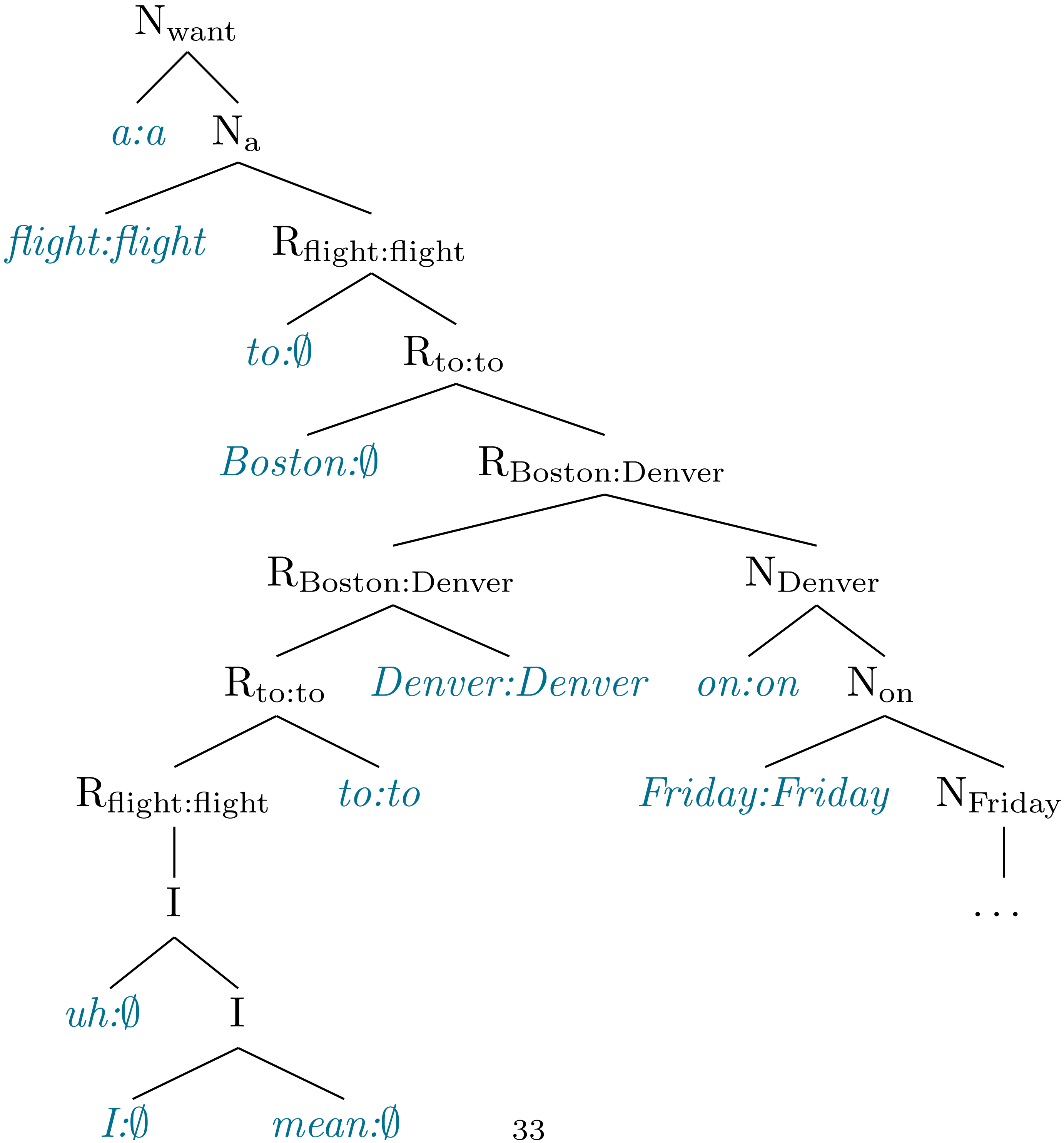
(I want) a flight to Boston uh I mean *to Denver* on Friday . . .



TAG rule



resulting structure


N_want
a:a
N_a
flight:flight
R_flight:flight
to:∅
R_to:to
Boston:∅
R_Boston:Denver
R_Boston:Denver
N_Denver
R_to:to
Denver:Denver
on:on
N_on
R_flight:flight
to:to
Friday:Friday
N_Friday
I
. . .
uh:∅
I
I:∅
mean:∅

Disfluencies in Switchboard

. . . a flight $\underbrace{\text{to Boston,}}_{\text{Reparandum}}$ $\underbrace{\text{uh, I mean,}}_{\text{Interregnum}}$ $\underbrace{\text{to Denver}}_{\text{Repair}}$ on Friday . . .

- Penn Switchboard corpus annotates reparandum, interregnum and repair
- Trained on the disfluency and POS tagged Switchboard files sw[23]*.dps (1.3M words)
- Tested on Switchboard files sw4[5-9]*.dps (65K words)
- Punctuation and partial words ignored
- 5.4% of words are in a reparandum
- 31K repairs, average repair length 1.6 words
- Number of training words: reparandum 50K (3.8%), interregnum 10K (0.8%), repair 53K (4%), unclassified 24K (1.8%)

Training data for the model

$$\ldots\text{a flight }\underbrace{\text{to Boston,}}_{\text{Reparandum}}\ \underbrace{\text{uh, I mean,}}_{\text{Interregnum}}\ \underbrace{\text{to Denver}}_{\text{Repair}}\text{ on Friday }\ldots$$

- Minimum edit distance aligner used to align reparandum and repair words
  - Prefers identity, POS identity, similar POS alignments
- Of the 57K alignments in the training data:
  - 35K (62%) are identities
  - 7K (12%) are insertions
  - 9K (16%) are deletions
  - 5.6K (10%) are substitutions
    - 2.9K (5%) are substitutions with same POS
    - 148 of the 352 substitutions (42%) in heldout data were not seen in training

Estimating the model from data

$$\ldots \text{a flight } \underbrace{\text{to Boston,}}_{\text{Reparandum}} \; \underbrace{\text{uh, I mean,}}_{\text{Interregnum}} \; \underbrace{\text{to Denver}}_{\text{Repair}} \text{ on Friday} \ldots$$

$P_n(\mathsf{repair}|\mathit{flight})$ The probability of a repair beginning after *flight*

$P(m|\mathit{Boston}, \mathit{Denver})$, where $m \in \{\mathsf{copy}, \mathsf{substitute}, \mathsf{insert}, \mathsf{delete}, \mathsf{nonrepair}\}$:
The probability of repair type m when the last reparandum word was *Boston* and the last repair word was *Denver*

$P_w(\mathit{tomorrow}|\mathit{Boston}, \mathit{Denver})$ The probability that the next reparandum word is *tomorrow* when the last reparandum word was *Boston* and last repair word was *Denver*

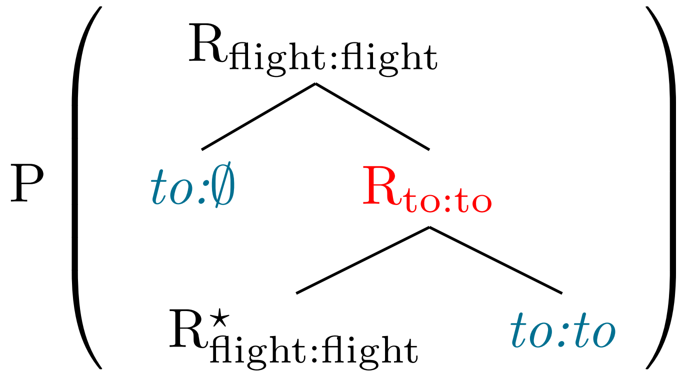
The TAG rules and their probabilities

$$P\left(\begin{array}{c}\mathrm{N}_{\mathrm{want}}\\ \diagup\;\diagdown\\ \color{teal}{a{:}a}\quad \color{blue}{\mathrm{N}_{\mathrm{a}}\downarrow}\end{array}\right) \quad = \quad (1 - \mathrm{P}_n(\mathsf{repair}|a))$$

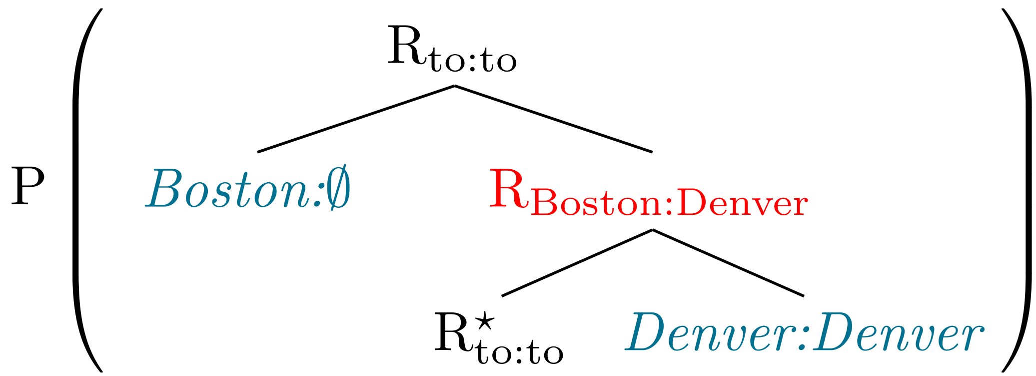
$$P\left(\begin{array}{c}\mathrm{N}_{\mathrm{a}}\\ \diagup\qquad\diagdown\\ \color{teal}{\mathit{flight{:}flight}}\qquad \color{red}{\mathrm{R}_{\mathrm{flight:flight}}}\\ \qquad\qquad\qquad |\\ \qquad\qquad\qquad \color{blue}{\mathrm{I}_\downarrow}\end{array}\right) \quad = \quad \mathrm{P}_n(\mathsf{repair}|\mathit{flight})$$

- These rules are just the TAG formulation of a HMM.

The TAG rules and their probabilities (cont.)



$$= \mathrm{P}_r(\mathsf{copy} \mid \mathit{flight}, \mathit{flight})$$



$$= \mathrm{P}_r(\mathsf{substitute} \mid to, to)$$
$$\mathrm{P}_w(\mathit{Boston} \mid to, to)$$

- Copies generally have higher probability than substitutions

The TAG rules and their probabilities (cont.)

$$\mathrm{P}\left(\begin{array}{ccc} & \mathrm{R}_{\mathrm{Boston,Denver}} & \\ \diagup & & \diagdown \\ \mathit{tomorrow}{:}\emptyset & & \mathrm{R}_{\mathrm{tomorrow,Denver}} \\ & & | \\ & & \mathrm{R}^{\star}_{\mathrm{Boston,Denver}} \end{array}\right) = \begin{array}{l} \mathrm{P}_r(\textsf{insert}|\mathit{Boston}, \mathit{Denver}) \\ \mathrm{P}_w(\mathit{tomorrow}|\mathit{Boston}, \mathit{Denver}) \end{array}$$

$$\mathrm{P}\left(\begin{array}{ccc} & \mathrm{R}_{\mathrm{Boston,Denver}} & \\ & | & \\ & \mathrm{R}_{\mathrm{Boston,tomorrow}} & \\ \diagup & & \diagdown \\ \mathrm{R}^{\star}_{\mathrm{Boston,Denver}} & & \mathit{tomorrow}{:}\mathit{tomorrow} \end{array}\right) = \mathrm{P}_r(\textsf{delete}|\mathit{Boston}, \mathit{Denver})$$

$$\mathrm{P}\left(\begin{array}{ccc} & \mathrm{R}_{\mathrm{Boston:Denver}} & \\ \diagup & & \diagdown \\ \mathrm{R}^{\star}_{\mathrm{Boston:Denver}} & & \mathrm{N}_{\mathrm{Denver}}\downarrow \end{array}\right) = \mathrm{P}_r(\textsf{nonrepair}|\mathit{Boston}, \mathit{Denver})$$

Decoding speech repairs

- We could find the most likely analysis of a sentence
- or alternatively:
  1. compute the probability that each triple of adjacent substrings can be analysed as a reparandum/interregnum/repair
  2. divide by the probability that the substrings do not contain a repair
  3. if the *odds* is greater than a fixed threshold, declare that there is a repair
- Advantages of the more complex approach:
  - Doesn't require parsing the whole sentence (rather, only look for repairs up to some maximum size)
  - Adjusting the odds threshold trades precision for recall
  - Handles *overlapping repairs* (where the repair is itself repaired)

[[What did + what does he] + what does she] want?

Empirical results

- Training and testing data has *partial words and punctuation removed*
- CJ01′ is the Charniak and Johnson 2001 word-by-word classifier trained on new training and testing data

	CJ01′	Bigram	Trigram	Parser
Precision	0.951	0.776	0.774	0.820
Recall	0.631	0.736	0.763	0.778
F-score	0.759	0.756	0.768	0.797

Conclusion and future work

- There are lots of interesting ways of combining speech and parsing
- Some of them don't work better than existing techniques (yet)
- *Syntactic parsers make very good language models*
- (Discriminative models might also be a good thing to try).