T-61.184
Automatic Speech Recognition: From Theory to Practice

http://www.cis.hut.fi/Opinnot/T-61.184/
November 1, 2004

Prof. Bryan Pellom
Department of Computer Science
Center for Spoken Language Research
University of Colorado

pellom@cslr.colorado.edu

Course Announcements

- Exercise #5 has been posted on the website. If you have successfully completed Exercises 1-4 with full credit, then this is Exercise is optional. Send me email if you have any questions if this is optional or not for you.
- If you need 0.5 points to complete 4/5 excercises, then you can solve 2 out of the 4 problems on Exercise #5.
- Course presentations will be 10 minutes per project group. I need 4 volunteers for November 22nd. The remaining 8 projects will be presented on November 29th.
- The course schedule has been updated on the web page.

References for Today's Material

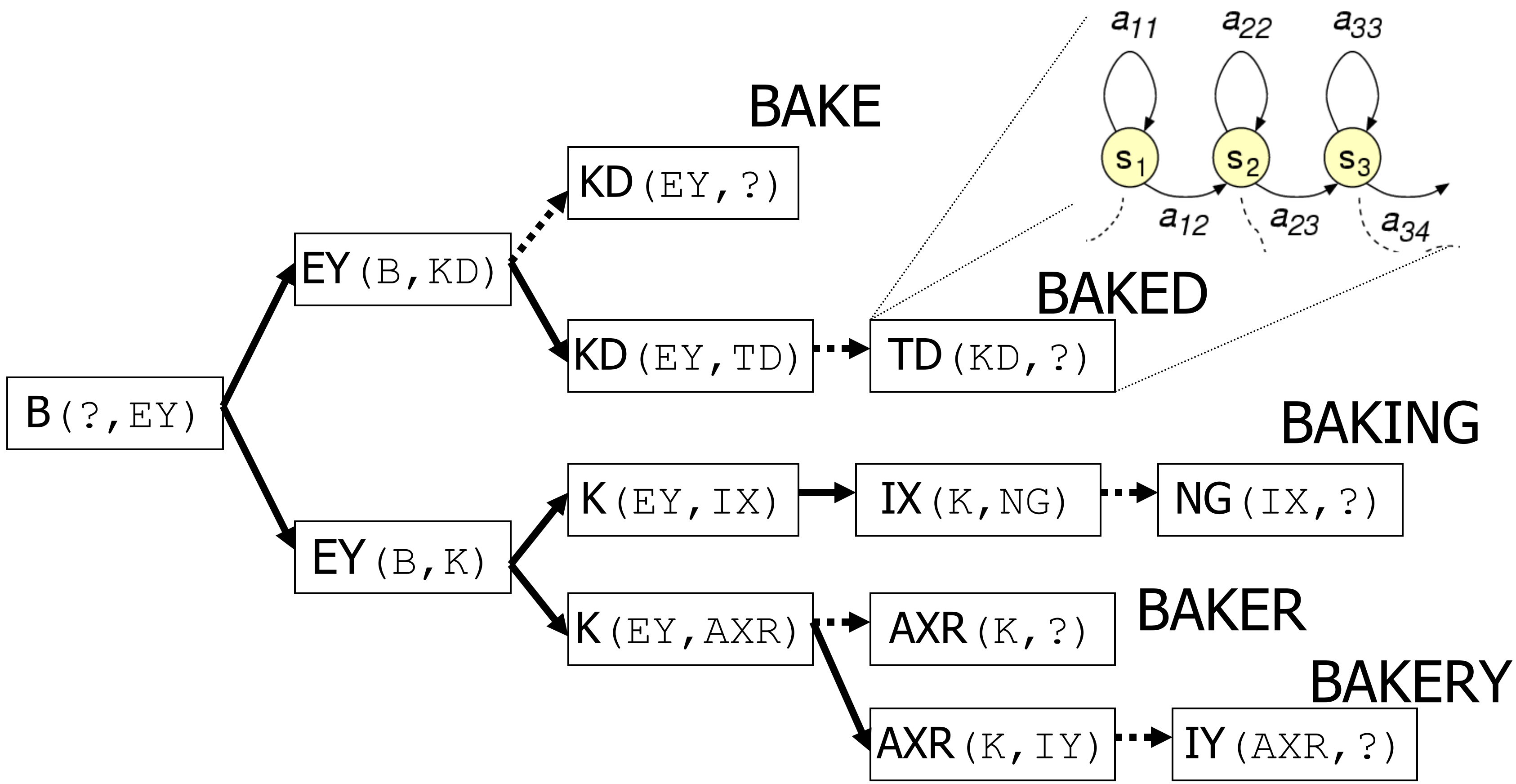
- M. Ravishankar, "Efficient Algorithms for Speech Recognition," Ph.D. thesis, Carnegie Mellon University, 1996.
- W. Daelemans, A. van der Bosch, "Language-Independent Data-Oriented Grapheme to Phoneme Conversion," In Progress in Speech Synthesis, Ed. J. Van Santen, R. Sproat, J. Olive, J. Hirschberg, pp. 77-88, 1997.
- Black, Lenzo, and Pagel, "Issues in Building General Letter to Sound Rules," for the 1998 ESCA Speech Synthesis Workshop, Jenolan Caves, Blue Mountains, Australia.
- R. Damper, Y. Marchard, M. Adamson, K. Gustafson, "Comparative Evaluation of Letter-to-Sound Conversion Techniques for English Text-to-Speech Synthesis," *Proc. The 3rd ESCA / COCOSDA Workshop on Speech Synthesis*, 1998.

(Review from Last Time)
Lexical Prefix Tree Search

Lexical Prefix Tree Search

- **As vocabulary size increases:**
  - Number of states needed to represent the flat search network increases linearly
  - Number of cross-word transitions increases rapidly
  - Number of language model calculations (required at word boundaries) increases rapidly
- **Solution: Convert Linear Search Network into a Prefix Tree.**

Lexical Prefix Tree



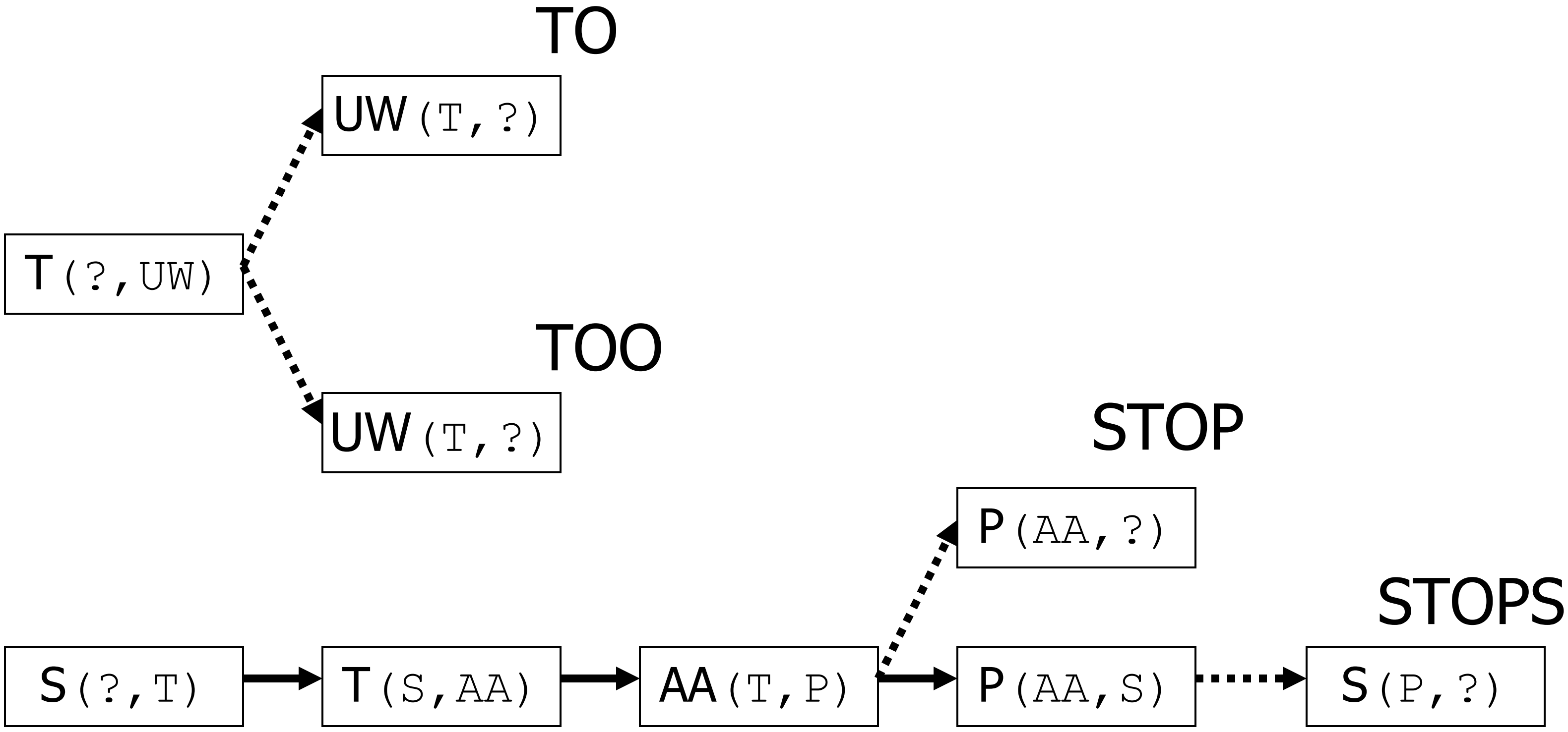
** Figure adapted from Huang et al., Spoken Language Processing, Prentice Hall*

T-61.184

Leaf Node Construction

- **Leaf Nodes ideally should have unique word identity**
- **Allows for efficient application of language model**
- **Handles instances such as,**
  - When word is the prefix of another word [“stop”, “stops”].
  - Homophones like “two” and “to”.

Leaf Node Construction



Advantages of Lexical Tree Search

- High degree of sharing at the root nodes reduces the number of word-initial HMMs needed to be evaluated in each frame
- Reduces the number of cross-word transitions
- Number of active HMM states and cross-word transitions grow more slowly with increasing vocabulary size

Advantages of Lexical Tree Search

- Savings in the number of nodes in the search space [e.g., 12k vocabulary, 2.5x less nodes].
- Memory savings; fewer paths searched
- Search effort reduced by a factor of 5-7 over linear lexicon [since most effort is spent searching the first or second phone of each word due to ambiguities at word boundaries].

Comparing Flat Network and Tree Network in terms of # of HMM states

	58K		
Level	Tree	Flat	Ratio
1	851	61657	1.4%
2	5782	61007	9.5%
3	18670	57219	32.6%
4	26382	49390	53.4%
5	24833	38254	64.9%
6	18918	26642	71.0%
7	13113	17284	75.9%
8	8129	10255	79.3%

Speed Comparison between Flat and Tree Search

Task	Dev93	Dev94	Eval94	Mean
20K	4.8	4.7	4.7	4.7
58K	5.2	4.8	4.5	4.9

- **CMU Sphinx-II : Speed Improvements of tree search compared to flat search for 20k and 58k word vocabularies [speed is about 4-5x faster!]**
- **Accuracy is about 20% relative worse for tree search.**

Disadvantages of Lexical Tree

- Root nodes model the beginnings of several words which have similar phonetic sequences

- Identity of word not known at the root of the tree

→ Can not apply language model until tree represents a unique word identity. "Delayed Language Modeling"

→ Delayed Language Modeling implies that pruning early on is based on acoustics-alone. This generally leads to increased pruning errors and <u>loss in accuracy</u>

Multi-Pass Search, N-Best Lists, Word-Lattices & Graphs

First-Pass Recognition Output

- **N-best List**
  - List of N most probable word sequences
- **Word Lattice**
  - Representation in which each word is represented by a score and a time-interval
- **Word Graph**
  - Finite state automata in which arcs are labeled with words

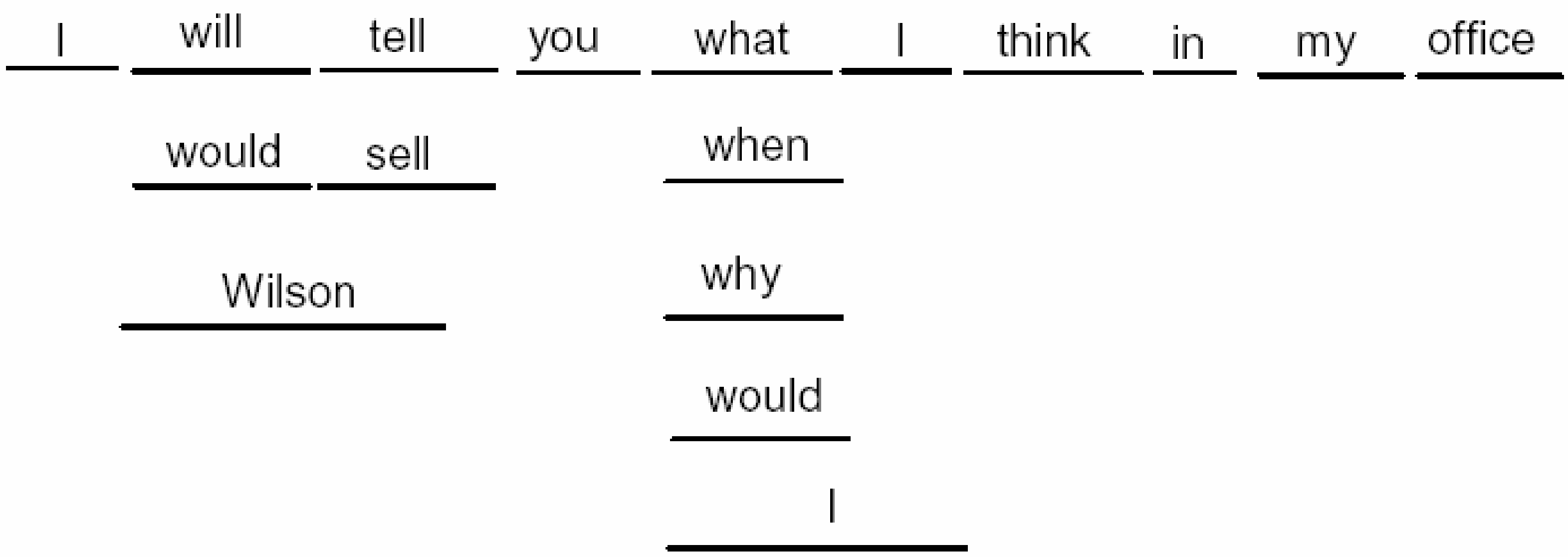
Example N-best List

1. I will tell you would I think in my office
2. I will tell you what I think in my office
3. I will tell you when I think in my office
4. I would sell you would I think in my office
5. I would sell you what I think in my office
6. I would sell you when I think in my office
7. I will tell you would I think in my office
8. I will tell you why I think in my office
9. I will tell you what I think on my office
10. I Wilson you I think on my office

Word Lattice Representation

- **More compact compared to N-best lists**
- **Minimally Encodes:**
  - Word Identity
  - Time-interval for word
  - acoustic score the word
  - (sometimes) total path score

Word Lattice Representation



Lattices and N-Best Lists

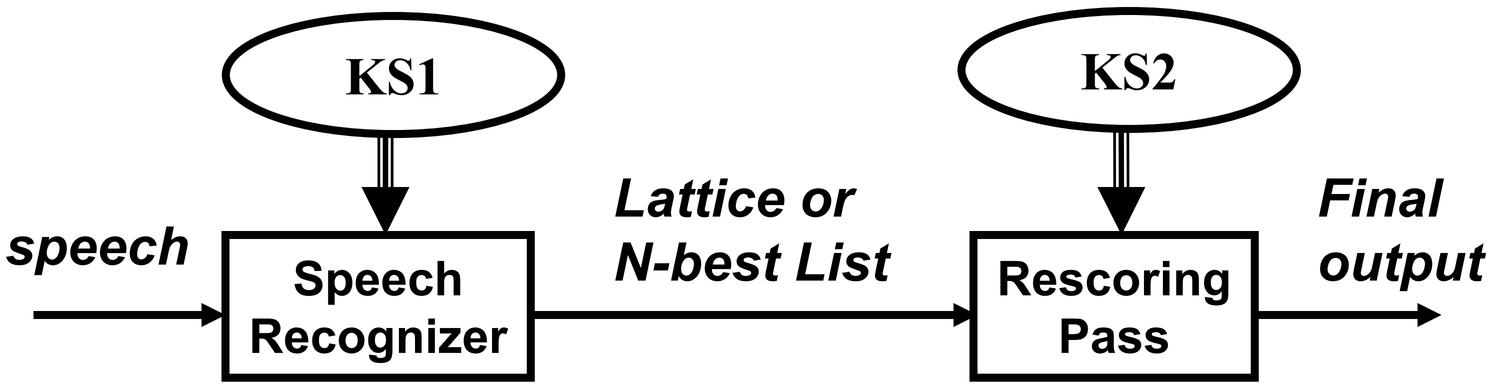
- **Provide a lower-bound on word-error rate.**
  - Given the anticipated correct word string we can compute a “lattice-error rate” or “n-best list error rate”.
  - Lowest error rate which can be possibly obtained with the knowledge source.
- **Density**
  - We often talk of “lattice-density”: number of hypotheses or word-arcs per uttered word

Multi-pass Search Methods

- **Some knowledge sources increase the complexity of search,**
  - ❑ Higher order n-gram language models (N > 3)
  - ❑ Cross-word acoustic models [remember fan-out issue]
  - ❑ Longer-context acoustic models [beyond triphone]
  - ❑ Pronunciation Models
- **Multi-pass methods reduce search space by first using "simple-to-compute" acoustic or language models and then later "rescore" remaining hypotheses with more complex knowledge sources**

Multi-Pass Search

- **Step 1: Use Knowledge Source (KS) #1 to generate a reduced hypothesis space**
- **Step 2: Rescore resulting hypothesis space with Knowledge Source #2.**



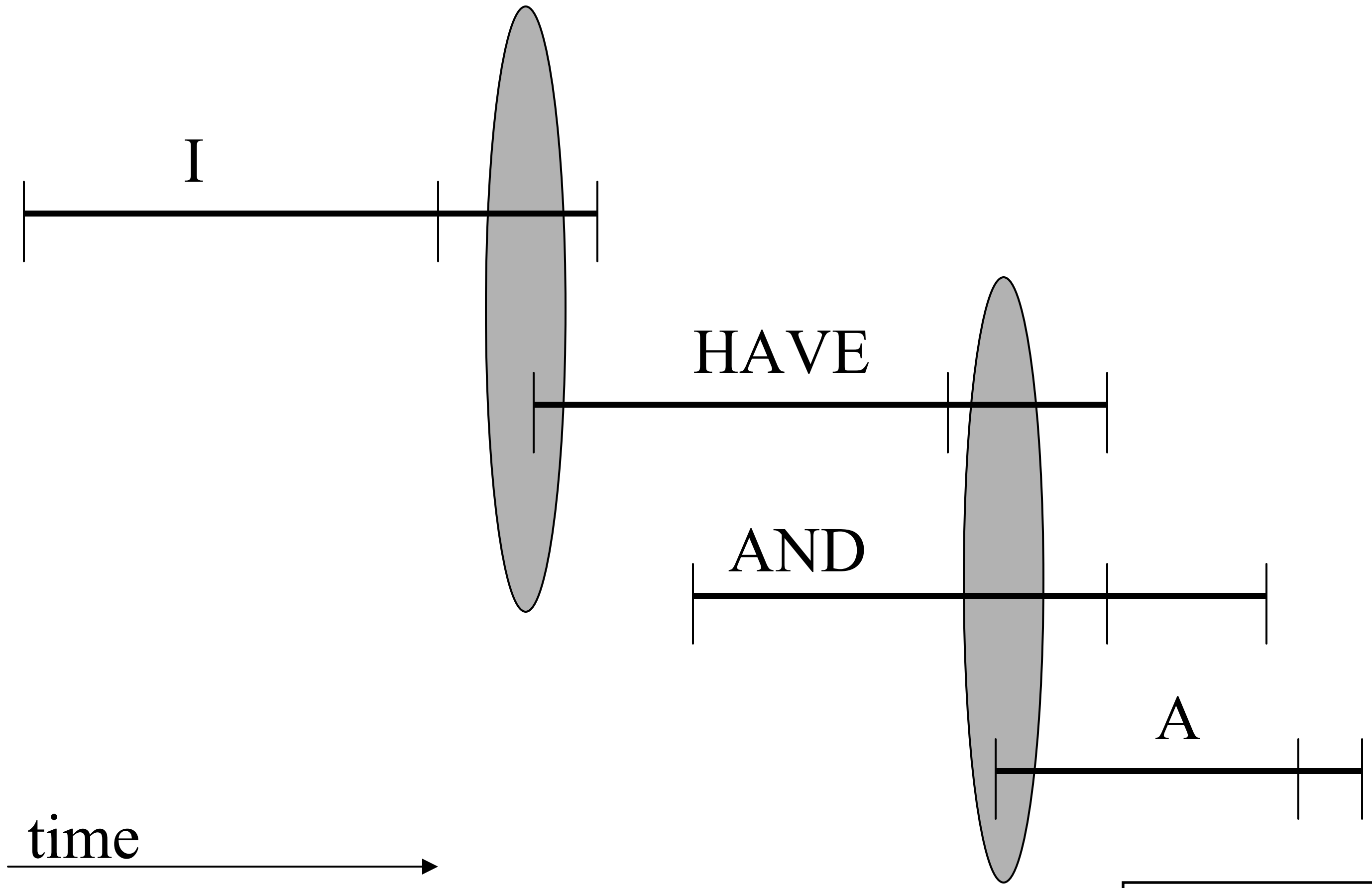
One Method for Word Graph Generation

- **Each word instance in the word lattice is represented by a pair (w,t)**
  - w is the word-id
  - t is the *begin frame* of the word

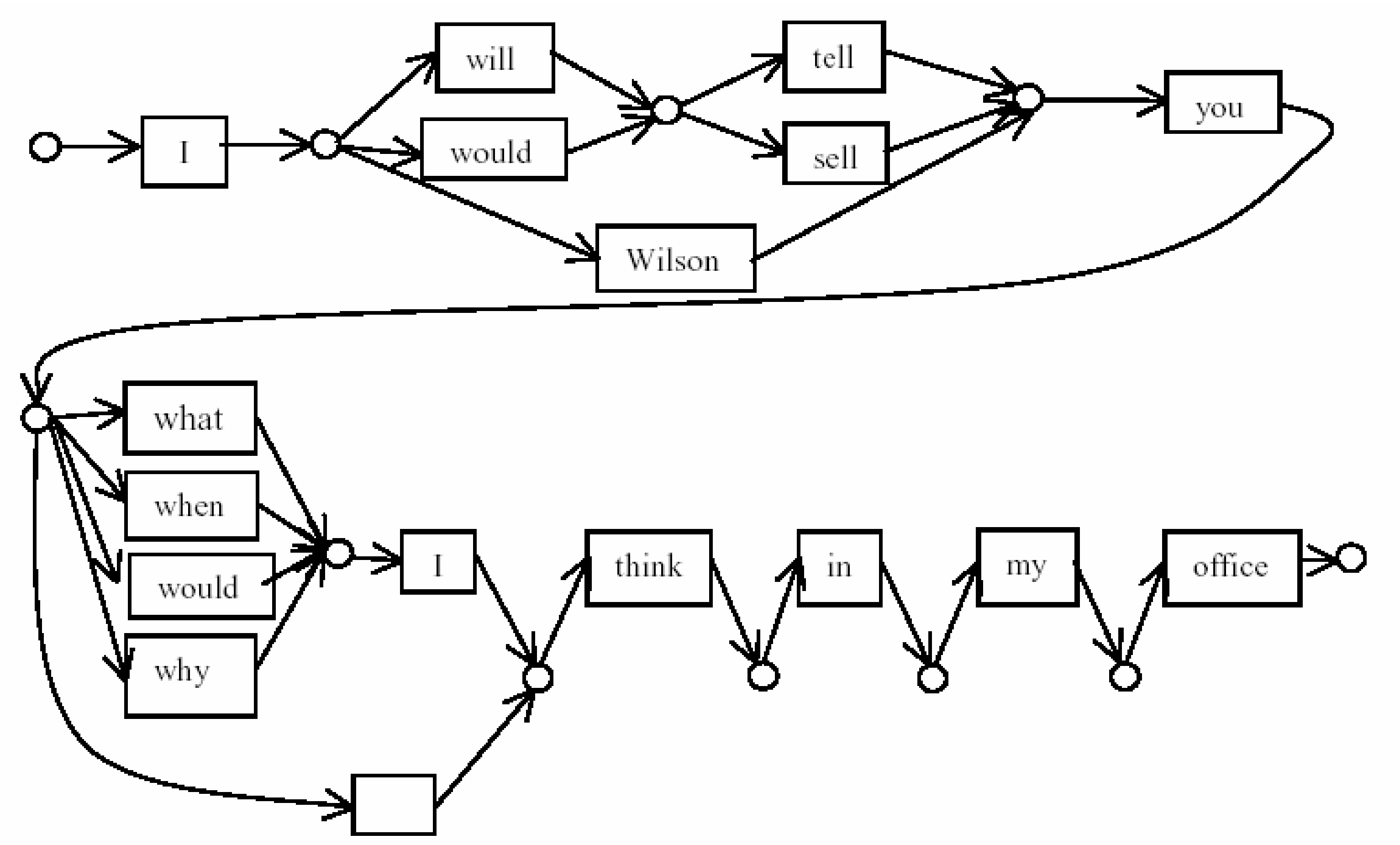
- **Each word can have a series of possible end-frames for each single begin time**

- **Create an edge from (w_i,t_i) to (w_j,t_j) iff t_j-1 is one of the possible end-times of (w_i,t_i)**

Word Graph Generation from Word Lattice



Word Graph Example



N-best List Rescoring

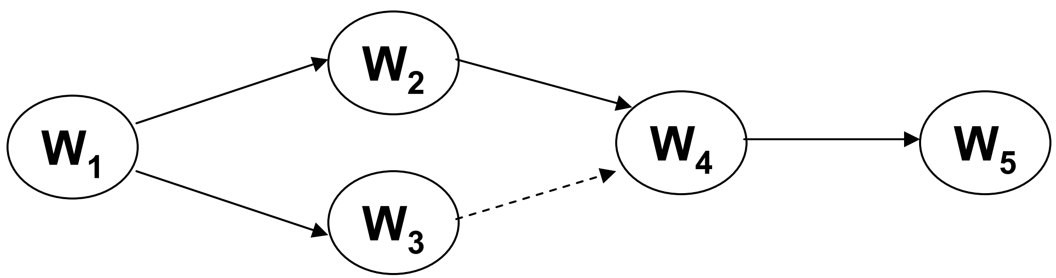
- **Use standard token-passing using a 2-gram language model & word-internal acoustic models**
- **Compute N-best list (10 < N < 500)**
- **Resort N-best list**
  - Recompute sentence probability using cross-word acoustic models and 3-gram language model
- **Pick top sentence as final hypothesis**

Word Graph Rescoring

- Use standard token-passing using a 2-gram language model & cross-word acoustic models
- Convert word lattice into a word-graph
- Rescore the elements in the word graph using 3-gram language models. [replace 2-gram LM scores with 3-gram LM scores].
- Find new best-path word string through graph

Why Does N-best List and Word-Graph Rescoring Work?

- Has to do with the ***sub-optimality*** of the Viterbi search with n-gram LMs (following Ravishankar (1996)):



- Initially, P(w4|w2,w1) is much greater than P(w4|w3,w1). So, the path from w3,w4 may be pruned away.
- As the search proceeds, we might discover that P(w5|w4,w3) is much more likely than P(w5|w4,w2). Although this better path has been pruned away!

Multi-pass Search Criticisms

- **Not Suitable for real-time applications**
  - ❑ Second pass search can not start until the user stops speaking
- **Argument:**
  - ❑ Second pass operations tend to be extremely fast since search space is minimized.
  - ❑ Minimal delay experienced by the user.

Multi-pass Search Criticisms

- **Introduces Inadmissible Pruning**
  - Decisions in early search pass made using simple acoustic and language models
  - Correct hypothesis can be accidentally pruned early-on

- **Argument:**
  - Even a problem in one-pass methods
  - Since one-pass methods use beam search which is a form of inadmissible search
  - Search errors can be minimized by careful choice of pruning thresholds.

Arguments for Multi-Pass Search

- Incorporation of higher order knowledge sources
- Search space reduction for very large vocabularies
- Spoken Language Understanding
- Offline Development of ASR modules (rescoring is quick and convenient way to test new ideas)

Measuring ASR Performance & Practical Optimization Issues

Measuring ASR Performance

- **Substitution Errors**
  - Recognizer confuses word ‘a’ for word ‘b’
- **Deletion Errors**
  - Recognizer does not output an expected word
- **Insertion Errors**
  - Recognizer outputs an extra word not spoken

NIST sctk-1.2 scoring software

http://www.nist.gov/speech/tools/

```
Alignment# 84 for speaker sls
id: (sls-20000629-006-001)
Scores: (#C #S #D #I) 8 0 2 0
REF:  i'd like to go TO st louis on september SEVENTH
HYP:  i'd like to go ** st louis on september *******
Eval:                D                        D
```

```
Alignment# 90 for speaker sls
id: (sls-20000629-006-007)
Scores: (#C #S #D #I) 2 1 0 1
REF:  ** no THAT'S incorrect
HYP:  NO no NO     incorrect
Eval: I     S
```

Typical Outputs for Scoring Software

$$\text{Word Error Rate} = 100\% \times \frac{\text{No. Subs} + \text{Dels} + \text{Ins}}{\text{No. words in Correct Sentence}}$$

$$\%\ \text{Correct} = 100\% \times \frac{\text{No. Correct words}}{\text{No. words in Correct Sentence}}$$

$$\%\ \text{Subs} = 100\% \times \frac{\text{No. Substitutions}}{\text{No. words in Correct Sentence}}$$

$$\%\ \text{Ins} = 100\% \times \frac{\text{No. Insertions}}{\text{No. words in Correct Sentence}}$$

Example output from Sclite (sctk-1.2)

- Input (1): Reference transcription with key identifier surrounded by parentheses:

```
START OVER (sls-20000621-006-009)
TO GO TO LOS ANGELES (sls-20000628-003-001)
PITTSBURGH (sls-20000628-003-002)
OCTOBER TWENTY THIRD (sls-20000628-003-003)
LATE MORNING AFTER NINE (sls-20000628-003-004)
```

- Input (2): Hypothesis from recognizer with key identifier for each sentence [like reference]

Example Output from Sclite (sctk-1.2)

- **Scoring:** `sclite -i wsj -r ref.txt -h hyp.txt`

```
| SPKR    | # Snt  # Wrd | Corr    Sub    Del    Ins    Err  S.Err |
|---------+--------------+-----------------------------------------|
| sls     |  789    2009 | 88.8    7.4    3.9    1.6   12.8   15.2 |
|=================================================================|
| Sum/Avg|  789    2009 | 88.8    7.4    3.9    1.6   12.8   15.2 |
|=================================================================|
|  Mean   |789.0  2009.0 | 88.8    7.4    3.9    1.6   12.8   15.2 |
|  S.D.   |  0.0     0.0 |  0.0    0.0    0.0    0.0    0.0    0.0 |
| Median  |789.0  2009.0 | 88.8    7.4    3.9    1.6   12.8   15.2 |
`-----------------------------------------------------------------'
```

Speaker Name

Overall Word Error Rate

How to Calculate Word Error Rates?

- **An algorithm is shown on page 421 of the book "Spoken Language Processing" by Acero et al.**
- **Algorithm based on dynamic programming**
- **Define correct word string as,**

$$W_1 W_2 \cdots W_n$$

- **Define hypothesized word string as,**

$$\hat{W}_1 \hat{W}_2 \cdots \hat{W}_m$$

How to Calculate Word Error Rates

- **Define R[i,j] as the minimum error of aligning the two substrings:**

$$w_1 w_2 \cdots w_n \qquad\qquad \hat{w}_1 \hat{w}_2 \cdots \hat{w}_m$$

- **B[i,j] is a back pointer used to recover error types**

- **Initialization:**

$$R[0,0] = 0;$$
$$R[\mathrm{i},\mathrm{j}] = \infty \ \text{ if } (\mathrm{i} < 0) \text{ or } (\mathrm{j} < 0)$$

How to Calculate Word Error Rates

$$\text{for } i = 1, \ldots, n \ \{$$

$$\text{for } j = 1, \ldots, m \ \{$$

$$R[i,j] = \min \begin{bmatrix} R[i-1,j]+1 & \text{(deletion)} \\ R[i-1,j-1] & \text{(match)} \\ R[i-1,j-1]+1 & \text{(substitution)} \\ R[i,j-1]+1 & \text{(insertion)} \end{bmatrix}$$

$$B[i,j] = \begin{bmatrix} 1 \ \text{(deletion)} \\ 2 \ \text{(insertion)} \\ 3 \ \text{(match)} \\ 4 \ \text{(substitution)} \end{bmatrix}$$

R[i,j] Computation Example

Hypothesis

Reference

	NO	NO	NO	INCORRECT
NO	R[1,1]= **0**	R[1,2]= **1**	R[1,3]= **2**	R[1,4]= **3**
THAT'S	R[2,1]= **1**	R[2,2]= **1**	R[2,3]= **2**	R[2,4]= **3**
INCORRECT	R[3,1]= **2**	R[3,2]= **2**	R[3,3]= **2**	R[3,4]= **2**

B[i,j] Computation Example

Hypothesis

Reference

	NO	NO	NO	INCORRECT
NO	M=3	I=2	I=2	I=2
THAT'S	D=1	S=4	S=4	S=4
INCORRECT	D=1	S=4	S=4	M=3

Calculating Word Error Rates

- **Error sequence (correct, subs, del, insertions) can be recovered by back-tracing through B[i,j] terms.**

$$\text{word} \quad \text{error} \quad \text{rate} = 100\,\% \times \frac{R[n,m]}{n}$$

- **There can be multiple back traces with equal error rate!!! All give same number of errors, but different string alignments!**
- **It's not as easy as it may seem! Use *standard* tools like NIST's `sclite` when possible.**

Alternative Computation of R[i,j]

$$R[i,j] = \min \begin{bmatrix} R[i-1,j] + \mathbf{3} & \text{(deletion)} \\ R[i-1,j-1] & \text{(match)} \\ R[i-1,j-1] + \mathbf{4} & \text{(substitution)} \\ R[i,j-1] + \mathbf{3} & \text{(insertion)} \end{bmatrix}$$

- **Use this scale to update R[i,j] and B[i,j]. Use B[i,j] to decode the insertions, subs, deletions.**
- **Count the number of errors and divide by the number of words in the correct sentence.**
- **This cost function is used by the NIST Sclite scoring package.**

Optimization of Recognizers

- **Requires deep understanding of how the recognizer operates, some intuition as well.**
- **Several parameters, each influence each other**
- **Difficult to exhaustively search for the best settings (LM scale factor, insertion penalty, beams).**
- **Requires optimization on a *development* test set. DO NOT USE YOUR FINAL TEST SET!**

Remember…

- **Viterbi Path contains acoustic and scaled language model score with a word-transition penalty,**

$$\hat{\mathrm{W}} = \arg\max_{\mathrm{W}} \left\{ \log\left(P(\mathrm{O} \mid \mathrm{W}) P(W) \right) \right\}$$

$$\hat{\mathrm{W}} = \arg\max_{W} \left\{ \underbrace{\log\left(P(\mathrm{O} \mid \mathrm{W}) \right)}_{\text{acoustic model}} + \underbrace{\mathrm{s} \cdot \log\left(\mathrm{P}(\mathrm{W}) \right) + \mathrm{p}}_{\text{language model}} \right\}$$

- **Path scores are maintained by tokens (token-pass search). At each frame, tokens are pruned by comparing the token's path score to the best token's score (minus a search beam).**

Viterbi Beam Search Settings

- **Wide beam:**
  - Prunes fewer competing hypotheses
  - Slower search since more paths explored
  - (sometimes) Fewer deletion errors; more insertion errors
- **Narrow beam:**
  - Faster search since fewer paths are explored
  - (sometimes) More deletion errors; fewer insertion errors
  - (sometimes) More substitution errors as correct path may be pruned away during search

Language Model Scaling Factor

- **Increasing the language model scale factor reduces the influence of the acoustic models in the selection of the final word sequence.**

- **As LM scale factor is increased:**
  - More deletion errors (since there is an increased penalty for transitioning between words)
  - Fewer insertion errors
  - Need wider beams! (since path scores will become larger)
  - Less influence of acoustic model observation probabilities

Word Insertion Penalty

- **Can be used to control the trade-off between insertion and deletion errors**
- **As penalty becomes larger (more negative),**
  - More deletion errors
  - Fewer insertion errors
- **Some recognizers include a “short-word transition penalty” for words which contain few phonemes.**
- **Positive values of this type of transition penalty are used to reduce deletions of short words**

Speed Optimization

- ~80% of the state hypotheses being searched are in the first phoneme of words in a medium-sized vocabulary recognition task.
- Word initial positions are searched *quite often* due to the ambiguities at word boundaries
- Efforts to reduce word-initial state explorations will improve the speed of a speech recognizer.

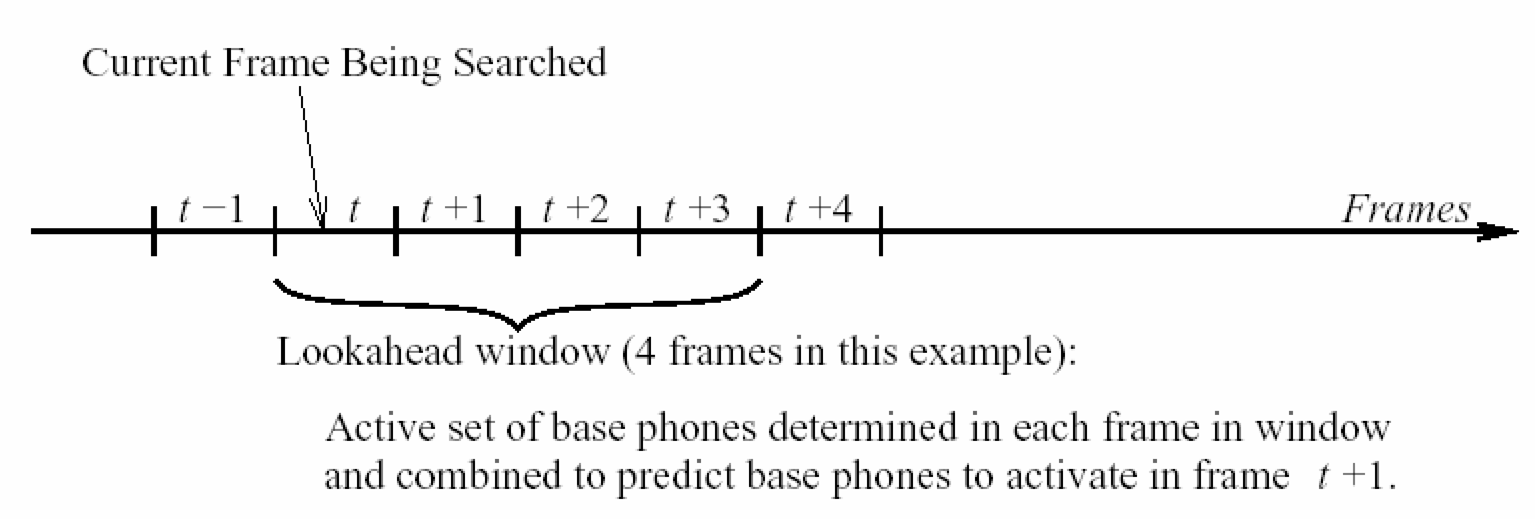
Phonetic Fast Match

- Look ahead "N" frames and determine which phonemes are currently "active".
- Do not search states involving those "in-active" phonemes.
- How to determine which phonemes are <u>active</u> in the upcoming frames?

```
Frame#   Base phones active in each frame, ranked by score
------ ---------------------------------------------------
[ 331] dh  b  p th  d ax ix  k td
[ 332] dh  b th  p ix ax ih eh  k ae ey  d  g td
[ 333] dh  b th  p ax ix ey ih eh
[ 334] dh th ey ih ax eh  b ix  p ae  t  d
[ 335] ih ax ey  t dh ix eh  d ae th  b  p
[ 336] ih ix ey ax ae eh  t  d
[ 337] ih ix ax ey ae eh er
[ 338] ix ih ax er ey eh
[ 339] ih ix ax ey eh er uw ah
[ 340] ih ix ax ey uw
[ 341] ix ih ax  z
[ 342]  z ix  s ax
[ 343]  z  s
[ 344]  z  s ts
[ 345]  s  z ts
[ 346]  s  z ts
[ 347]  s  z ts td
[ 348]  s td  z ts  t dd
```

"THIS" = DH IH S

Phonetic Fast Match (CMU Sphinx-II Approach)



- **Frame likelihoods are computed for each phoneme from context-independent HMM states**
- **Cross-HMM and Cross-Word transitions are only allowed for phonemes labeled as active.**

T-61.184

Expected Gains from Fast-Match

- **Ravishankar (1996)**
  - CMU Sphinx-II recognizer.
  - 3-frame look-ahead
  - 20k and 58k word vocabulary system
  - 45% reduction in execution time
  - 2% relative increase in word error

- **Gopolakrishan (ICASSP, 1994)**
  - IBM recognizer
  - Almost 50% reduction in execution time
  - 10% relative increase in word error rate.

Grapheme-to-Phoneme Conversion Methods for Speech Recognition Lexicon Development

(A problem for researchers in text-to-speech synthesis and automatic speech recognition)

Lexicon Development for ASR

- **Quality of the pronunciations of words will directly impact the speech recognition error rate.**
- **Unlike Finnish, many languages there is a less-then-obvious mapping between letters and sounds**
- **In English we have many issues,**
  - Pronunciation of Proper Names (Streets, First/Last Names, Places)
  - Pronunciation of infrequent words, task-dependent vocabularies
  - CMU Pronouncing Dictionary for English (125,000 words)
- **What to do when word is not part of the dictionary?**

Problem Complexity

- In some languages (Spanish, Greek, Turkish, Finnish) the association of text to phonemes can be described by a very small set of rules
- English poses many problems:

S	P	E	E	C	H
S	P	IY	_	CH	_

Some Pronunciation Generation "Tricks" for English

- **Method 1: Morphological Analysis**
  - Ericson; McDonald; Ivanovich (-son, Mc-, -ovich)

- **Method 2: Pronunciation by Analogy**
  - Can be applied to names (e.g., *Trotsky* in dictionary but *Plotsky* is not)
  - Words that share the same final letter sequence are assumed to rhyme.

If all else fails?

- **Generate the pronunciation by hand**
  - Not possible for text-to-speech synthesis systems
  - Doable for speech recognition systems

- **Can consider automated methods**

- **Two basic "automated" approaches**
  - Rule-based (Letter-to-Sound Rules)
  - Data-driven (Letter-to-Sound Predictive Model)

Rule-Based Methods

- Develop a set of rules by hand to account for letter-to-sound conversion
- A[B]C → D
- Multiple rules tend to apply to same string
- Must apply them in a specific order (most specific at the top, most general at the bottom)

Comparison of Existing Methods (Damper et. al, 1998)

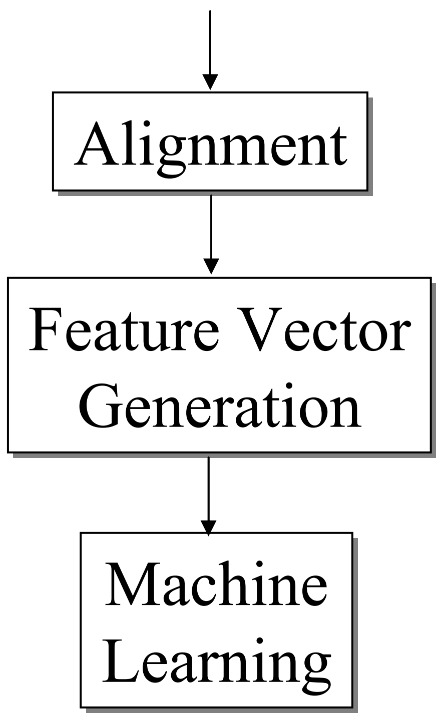
- **Phonological Rules**
  - Hand-crafted letter-to-sound rules
  - Elovitz, IEEE Trans. ASSP, Vol 24:446-459, 1976.
- **NetSpeak**
  - Neural Network; Coded letter context as input
  - 25 output features to represent the target phone
- **Nearest-Neighbor (1B1-IG)**
  - Feature weighting function used to provide a real-valued weight for feature values (letter positions)
  - Compute similarity between new instance and all stored instances; return the class label of the most similar instance.
- **Pronunciation by Analogy**
  - Pronunciation of an unknown is assembled by matching substrings of the input to substrings of known words

Comparison of Existing Methods (Damper et. al, 1998)

Phonological Rules	**25.7% correct**
NetSpeak	**46.0% correct**
1B1-IG (Nearest Neighbor)	**57.4% correct**
Pronun. by Analogy	**71.8% correct**

Hand-driven phonological rules always *under performed* the data driven methods *significantly*.

Automated Method



Lexicon

Map Letters to Phones

Feature Vector of Contexts

Decision Tree, Neural Net,
Support Vector Machine, etc.

Letter-to-Phone Alignment

- **Number of letters in a word and the number of phones is not a one-to-one match**
- **Generally, each letter might map to 0, 1, 2, and sometimes 3 phones.**
- **Less phones than letters in most cases with some exceptions:**
  - X → /k s/ in "extra"
  - O → /w uh/ in "one"

Example Candidate Alignments

- **Letter-to-Phone alignments become training data for our automatic classifier.**

SPEECH S P IY CH

S	P	E	E	C	H
S	P	IY	_	CH	_
S	P	IY	_	_	CH
S	P	_	IY	_	CH
S	P	_	IY	CH	_

T-61.184

Hand-Seeded Candidate Alignments

- **Improve candidate alignments by pre-determining which phonemes each letter can map onto.**

- **For example, the letter "c"**
  - _epsilon_ (e.g., "muscle")
  - K, CH, S, SH, T-S (e.g., "church")

- **Vowel letters can have a longer list of potential phonemes**

Algorithm Initialization

- **For each word in the dictionary, generate all possible alignments of letters to phones considering various epsilon placements**

- **Determine probability of aligning letter *l* with phone *p* for all *l* and *p*.**

$$P(phone_l \mid letter_l) = \frac{Count(letter_l \rightarrow phone_l)}{Count(letter_l)}$$

T-61.184

Determining the Best Alignment

- Input to algorithm is a word containing letters and the associated phonemic sequence
- Generate all possible candidate alignments including the epsilon symbol
- Score each possible alignment and choose the most probable alignment for each word,

$$S = \prod_{l=1}^{L} P(phone_l \mid letter_l)$$

Alignment Results in Letter-to-Phoneme Mapping

ABSORBED	AX B Z AO R BD _ DD
ABSORBENCY	AX B Z AO R B AX N S IY
ABSORBENT	AX B Z AO R B AX N TD
ABSORBER	AX B Z AO R B _ AXR
ABSORBERS	AX B Z AO R B _ AXR Z
ABSORBING	AX B Z AO R B IX _ NG
ABSORBS	AX B Z AO R B Z
ABSORPTION	AX B S AO R P _ SH AX N
ABSTAIN	AE B S T EY _ N
ABSTAINED	AE B S T EY _ N _ DD
ABSTAINING	AE B S T EY _ N IX _ NG
ABSTENTION	AE B S T EH N CH _ AX N
ABSTENTIONS	AE B S T EH N CH _ AX N Z

Feature Vector Generation

"Abbreviating → AX B _ R IY V IY EY DX IX _ NG"

L3	L2	L1	L	R1	R2	R3	
-	-	-	A	B	B	R	AX
-	-	A	B	B	R	E	B
-	A	B	B	R	E	V	_
A	B	B	R	E	V	I	R
B	B	R	E	V	I	A	IY
B	R	E	V	I	A	T	V
R	E	V	I	A	T	I	IY
E	V	I	A	T	I	N	EY
V	I	A	T	I	N	G	DX
I	A	T	I	N	G	-	IX
A	T	I	N	G	-	-	_
T	I	N	G	-	-	-	NG

6 Letter Context Window

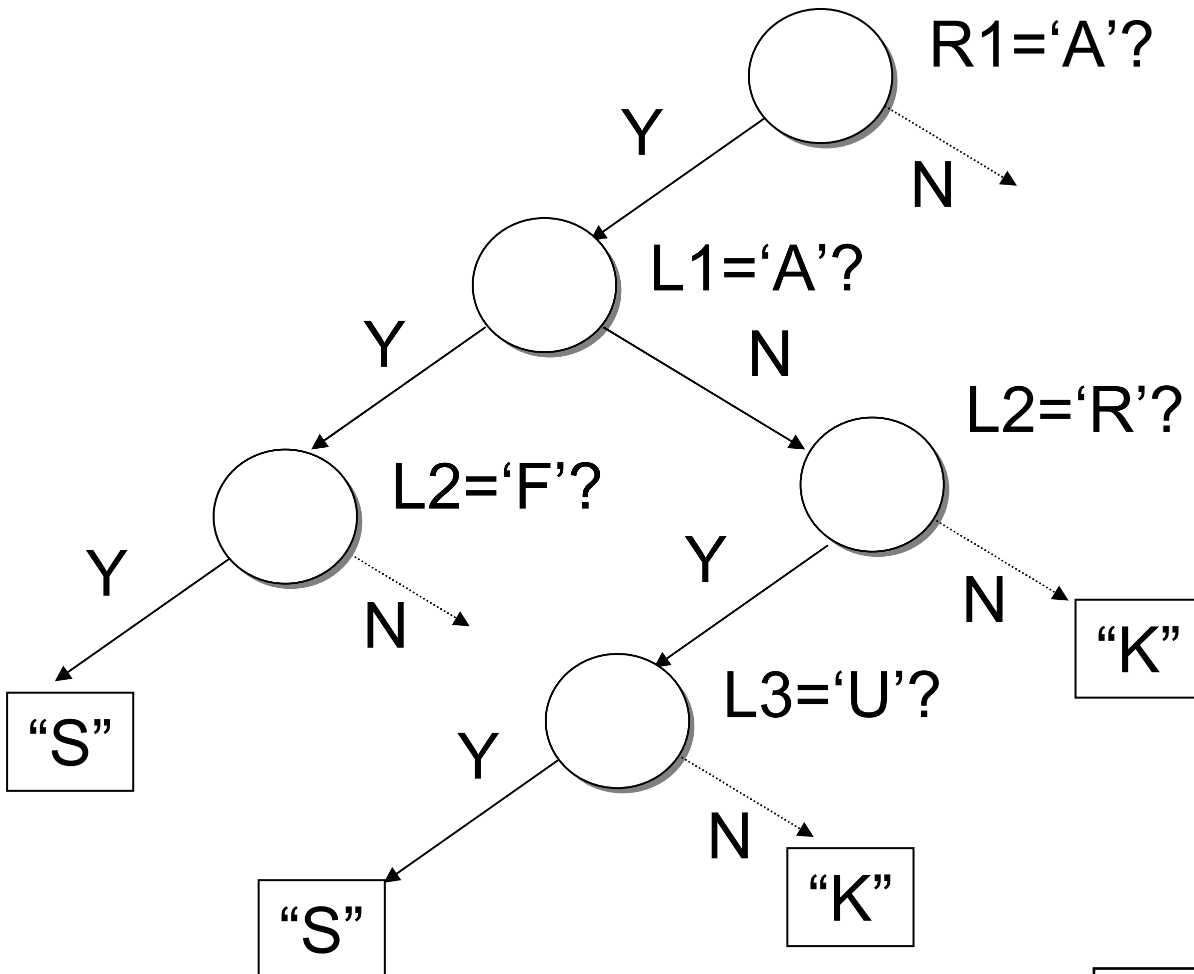
Output Target Phone

T-61.184

Potential Machine Learning Algorithms

- **Neural Network**
  - ❑ Input to NN : coded versions of letters (with 3-letter surrounding context)
  - ❑ Output of NN : vector of probabilities for selecting one of N phonemes
- **Support Vector Machine**
- **Decision Tree**
  - ❑ Input : letter context (center letter plus 3 letters to left and right)
  - ❑ Output : phoneme symbol prediction (possibly epsilon)

Resulting Decision Tree (L='C')



Evaluating the Decision Tree

Feature
Input Letters

L3	L2	L1	L	R1	R2	R3
?	F	A	C	A	?	?

Output Phoneme is “S”

Example Decision Tree Evaluation
for the Word “**Façade**”

Tree Size vs. % Correct

Minimum # of Examples	Letters Correct	Words Correct	Tree Size
8	92.9%	59.6%	9884
6	93.4%	61.6%	12782
5	93.7%	63.1%	14968
4	94.0%	65.2%	17948
3	94.4%	67.2%	22912
2	94.9%	69.4%	30368
1	95.8%	74.6%	39500

DT Performance for English, French, and German

Lexicon	Letters Correct	Words Correct
OALD (British English)	95.8%	74.6%
CMUDICT (American English)	92.0%	57.8%
BRULEX (French)	99.0%	93.0%
DE-CELEX (German)	98.8%	89.4%

** words with less than 4-letters were removed from test

Does it Really Work Well? Probably Not for English!

- **SONIC Speech Recognition System**
  - ❑ Wall Street Journal 20k-word vocabulary
  - ❑ DARPA Nov. 1992 test set

- **Compare word error rate for system designed with hand-driven pronunciations vs. one with <u>completely automatic pronunciations</u> (DT method).**

- **Word Error Rate with Hand-Driven Pronunciations**
  - ❑ 9.6% (8.5% after adaptation)

- **Word Error Rate with Decision-Tree Pronunciations**
  - ❑ 41.2% (38.2% after adaptation)

Next Time

- **Some discussion and comparison of available speech recognition systems**
- **Some discussion about tools used in the field and trends in development of open-source components for speech recognition**