MISHEARD ME ORONYM TREE: USING ORONYM TREES TO VALIDATE THE CORRECTNESS OF FREQUENCY DICTIONARIES

A Thesis

Presented to

the Faculty of California Polytechnic State University

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Jennifer "Jenee" Gayle Hughes

June 2012

COMMITTEE MEMBERSHIP

TITLE:	Misheard Me Oronym Tree: Using Oronym Trees to Validate the Correctness of Frequency Dictionaries
AUTHOR:	Jennifer "Jenee" Gayle Hughes
DATE SUBMITTED:	June 2012
COMMITTEE CHAIR:	Zoë Wood, Ph.D.
COMMITTEE MEMBER:	Franz Kurfess, Ph.D.
COMMITTEE MEMBER:	John Clements, Ph.D.

Abstract

Misheard Me Oronym Tree: Using Oronym Trees to Validate the Correctness of Frequency Dictionaries

Jennifer "Jenee" Gayle Hughes

In the field of speech recognition, an algorithm must learn to tell the difference between "a nice rock" and "a gneiss rock". These identical-sounding phrases are called oronyms. Word frequency dictionaries are often used by speech recognition systems to help resolve phonetic sequences with more than one possible orthographic phrase interpretation, by looking up which oronym of the root phonetic sequence contains the most common words. However, this approach is highly dependent upon the manner in which the frequency values in a word frequency dictionary are obtained.

Our paper demonstrates a technique used to validate word frequency dictionary values. We use oronym trees to compare phrase frequency values from dictionaries, to the frequency with which our human test subjects heard different variations of the root phrase. We chose to use frequency values from the UNISYN dictionary, which uses tallies each word occurance in a propriatary text corpus (***Glossary term***?).

Given any valid English phrase, herein referred to as the root phrase, our system will first generate all possible correct phonetic sequences for a General American accent. Then, it parses through these phonetic transcriptions depth-first, looking for valid orthographic words for each subsequent phonetic subsequence, generating full and partial phrases from these words. In the event that the entire phonetic sequence branch can be parsed into a valid orthographic phrase, we save this orthographic phrase as an oronym of the root phrase.

We also developed a visual representation of the oronym trees, to allow for visualizing phonetic dead-ends. In the event that a branch's phonetic "tail" is not orthographically interpretable, we visually "dead-end" the branch by drawing a red sphere. A particularly strong orthographic partial phrase before a phonetic dead-end can mislead a listener, causing them to lose track of the words in the rest of the phrase. In the event that the entire phonetic sequence can be parsed into a valid orthographic phrase, we indicate this successfully-found oronym with a green sphere.

Using the oronyms generated from our oronym tree, we then conducted a user study. Our multi-phase user study, incorporated over 851 data points from 208 test subjects. In it, we tested the validity of our oronym generation by having participants record themselves reading an oronym phrase. Then, a second set of subjects transcribed the recordings.

In the first phase, we generated oronym strings for the phrase *"a nice cold hour"*, and had over a dozen people make 72 recordings of the most common oronyms for that phrase. We then compared their pronunciations to the pronunciations we were expecting, and found that in 71 cases, the recorded phrase's phonemics matched our expectation. This indicates that, while not exhaustive, our pronunciation dictionary is a good match for actual American-English pronunciations. In the second phase, we selected 15 of the phase one recordings, and had 30 to 60 different people transcribe each one.

If the frequency dictionary values for our test phrases accurately reflect the real-world expectations of actual listeners, we would expect that the most commonly transcribed phrases in our user study would roughly correspond with our metric for the most likely oronym interpretation of the root phrase.

The best possible use case to show this is the case of *"a nice cold hour"*, whose commonly-misheard **********(find citation)********** oronym is *"an ice cold hour"*. The words "a" and "an" are identical in function, but "an" is only used in the case that the following word starts with a vowel sound. As there are more consonant sounds than vowel sounds, "an" is used far less often than "a" is. The UNISYN dictionary has a frequency value of 7,536,297 for "a", and of 794,169 for "an", for an approximate ratio of 10 to 1, where "a" accounts for 90.46% (p = 0.9047) of the combined count.

In this case, we'd expect that the phrase *"*a *nice cold hour"* would be transcribed nearly ten times as often as *"*an *ice cold hour"*.

In the event that this was not the case, we can conclude that tally-per-occurance frequency dictionary values does not apply well to an average audience's auditory expectations.

During the course of our study, we found that the presence of excessively-common words (such as "the", "is", and "a") threw off our frequency metric when we used per-occurance frequency value. These super-common words have such high per-occurance tallies that it overpowered the effect that any regular word had on a frequency metric. However, when we used document-count frequency values, we found that this effected was mitigated.

The frequency dictionary from the Corpus of Contemporary American English[2] tallies the number of documents that a word is found in, instead of tallying the total number occurances of that over all documents. In this dictionary, "a" has a document-count frequency value of 168619 , and "an" has a frequency value of of 159720 , for a ratio of 1.055 to 1, where "a" accounts for 51.35% (p = 0.5135) of the combined count.

In our user study, we found that 125 people transcribed “a nice cold hour , and 191 people transcribed “an ice cold hour , for a ratio of 0.65 to 1, where “a nice cold hour accounts for 39.56% (p = 0.3956) of the combined count. We did a statistical test with an alpha of .01, and got a value that was so low we can’t find a calculator that has enough decimal places to show it without rounding it to zero. In short, our per-occurance frequency metric predictions don’t even remotely match the projected data.

Our COCAE-derived document-count frequency metric preditions more closely matched our actual findings. We calculated that “a nice cold hour had a COCA-derived frequency metric value of 247719 , and “an ice cold hour had a value of 227405 , giving us a ratio of 1.08 to 1, where “a nice cold hour accounts for 52.17% (p = 0.5217) of the combined count. When compared to our actual results using a one-sample proportion z test, we got a p-value of ***COMPUTE LATER****, which is slightly better, but not great.

We found that using per-occurance frequency values when computing our overall-phrase-frequency metric caused the thrown off by excessively common words, such as “the”, “is”, and “a”. These super-common words have such high per-occurance tallies that it overpowered the effect that any regular word had on a frequency metric. However, when we tally on a document-count basis, instead of a by-occurance basis, we found that this effect was mitigated.

I need help with statistics here. To facilitate comparison, we created two rankings for the actual result phrases: one list ranked by expected frequency, and one ranked by number of actual transcriptions by our test subjects. In our phase two results, we found that out of the 578 transcriptions acquired for 53 unique phrases, only 11 had less than a difference of 5 ranks between the actual and expected occurances. The top 10 unique results, accounting for 88.00% of

total transcriptions, were on average 25 ranks more common than the frequency metric ranking predicted they'd be, with over half of them more than 39 ranks higher (out of 53 total ranks). From this, we can conclude that the frequency dictionary that we used is flawed.

If these frequency dictionary values were correct for the phrase words, we would expect that the most commonly transcribed phrases in our user study would roughly correspond with our metric for the most likely oronym interpretation of the root phrase. In the event that this was not the case, we could conclude that the frequency dictionary values were in error for that phrase. To facilitate comparison, we created two rankings for the actual result phrases: one list ranked by expected frequency, and one ranked by number of actual transcriptions by our test subjects. In our phase two results, we found that out of the 578 transcriptions acquired for 53 unique phrases, only 11 had less than a difference of 5 ranks between the actual and expected occurances. The top 10 unique results, accounting for 88.00% of total transcriptions, were on average 25 ranks more common than the frequency metric ranking predicted they'd be, with over half of them more than 39 ranks higher (out of 53 total ranks). From this, we can conclude that the frequency dictionary that we used is flawed.

Contents

List of Tables

List of Figures

Chapter 1

Preliminary Vocabulary

Before we start, there are a few uncommon terms we will use fairly often in this paper. We have briefly defined them here.

1.1 Mondegreens

A mondegreen is a word or phrase resulting from a misinterpretation of a word or phrase that has been heard[8]. The word was coined by American author Sylvia Wright in her article, "The Death of Lady Mondegreen", published in a 1954 issue of Harper's Bazaar. In it, she describes the origin of the word:

> When I was a child, my mother used to read aloud to me from Percy's Reliques, and one of my favorite poems began, as I remember:
>
> > Ye Highlands and ye Lowlands,
> > Oh, where hae ye been?
> > They hae slain the Earl O' Moray,
> > And Lady *Mondegreen.*

The fourth line of the quote is actually "and laid him on the green"[23].

Additional commonly-cited mondegreens include[4][19][21]:

Gladly the Cross-Eyed Bear	Gladly the Cross I'd Bear
Scuse me while I kiss this guy	Scuse me while I kiss the sky
There's a bathroom on the right	There's a bad moon on the rise

1.2 Oronyms

Oronyms are phrases that may differ in meaning or spelling, but sound near-identical when spoken. They are similar to mondegreens, and the terms are often used interchangeably. The difference, however, lies in the context. The label "mondegreen" is used more often in regards to music lyrics, where pronunciation can be affected by the addition of music and tone to the phrase. Oronyms, on the other hand, refer to spoken words, not sung lyrics.[9]

Common oronyms include:

i scream	ice cream
an ice cold hour	a nice cold hour
grape ants	gray pants
real eyes	realize

1.3 Orthography

The word 'orthographic' comes from the Latin *orthographia*, meaning *correct* writing. Orthography itself is the part of language study concerned with letters and spelling. More specifically, it's the standardized system of writing down words in a specific language, using a commonly-accepted set of letters according to accepted usage. [10]

The orthographic symbol set for a language is the commonly-accepted set of letters used to spell words in that language. In English, our orthographic symbol set is the Latin alphabet.

In this paper, "orthographic phrase", refers to a sequence of regularly-spelled words found in an English dictionary.

Example: "This is a orthographic phrase."

1.4 Phonetics and Phonology

To discover oronyms for a phrase, we must first to translate the root orthographic phrase to a representation that allows us to unambiguously measure pronunciation. Phonology and phonetics are branches of linguistics that deal with pronunciation.

1.4.1 Phonetics

Phonetics is a branch of *descriptive* linguistics, and refers to the study of the actual, uttered sound of human speech. It deals with describing the physical phenomena of how these sounds are produced from the vocal tract, how they are transmitted once spoken, and how they are recieved by audiences. The building blocks of phonetics are *phones*, which represent atomic sounds.

1.4.2 Phonology (aka phonemics)

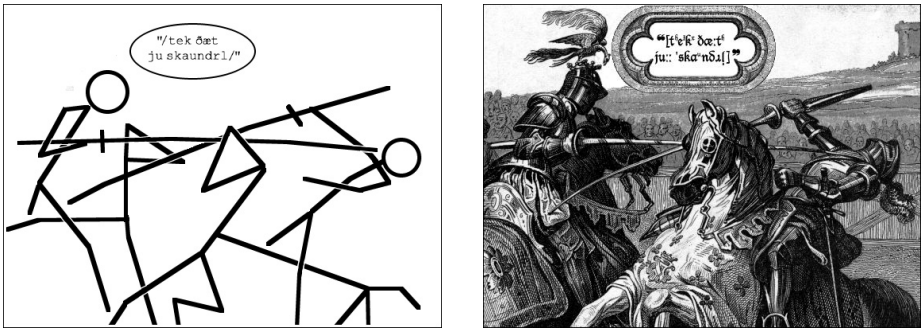
Phonology is a branch of *theoretical* linguistics, and as such, is primarily concered with the abstract grammartical characterization of sounds. It describes

the way that sounds function within a language and give meaning to words. The basis of phonological analysis is the grouping of sounds (*phones*) into distinct units within a languages. These distinct units are called *phonemes*.

These phonemes may contain different phones, depending on the accent of the speaker. For example, native speakers of General American English only generally recognize one 'L' sound phoneme. However, there are two different ways that that phoneme manifests itself: the 'l' in "male", and the 'l' in late. This difference is not noticable to a native speaker of American English, because that particular accent will parse any 'L' phone as the same 'L' phoneme.

1.4.3 Phonetics Vs Phonology

As we said previously, though the terms are sometimes used interchangably, the words 'phonemic' and 'phonetic' (and their corresponding sound building blocks, 'phone' and 'phoneme') indicate a different stages of sound parsing. *Phonemes* are idealized sounds; *phones* are the actual sounds that come out of a person's mouth. Figure 1.1 provides a final, illustrative metaphor of the difference.



(a) Phonology[5] (b) Phonetics[6]

Figure 1.1: The difference between phonetics and phonology

1.5 Phonemic/Phonetic Alphabets

As we stated in section 1.4.2, phonemes are the atomic building blocks of words. In a phonemic alphabet, every meaningful sound has its own letter. The way that we interact with phonemes in a concrete way is by using phonetic alphabets and phonetic dictionaries.

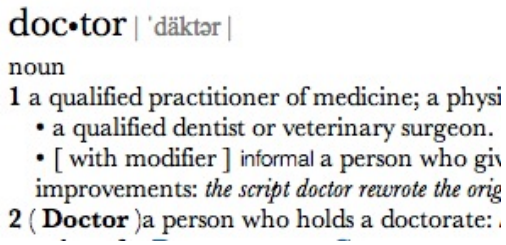

doc•tor | ˈdäktər |
noun
1 a qualified practitioner of medicine; a physi
• a qualified dentist or veterinary surgeon.
• [with modifier] informal a person who giv
improvements: *the script doctor rewrote the orig*
2 (**Doctor**)a person who holds a doctorate:

Figure 1.2: The characters to the right of the large bold word "doctor" are IPA symbols.

The most common phonetic alphabet is the IPA (Internation Phonetic Alphabet). It contains representations of every sound in every known language globally, and allows for cross-cultural pronunciation guidelines. As shown in figure 1.2, IPA representations of orthographic words are found in traditional dictionaries to aid pronunciation.

1.5.1 SAMPA

SAMPA (Speech Assessment Methods Phonetic Alphabet) is a computer-readable phonetic alphabet, based upon the symbols found in the more-standard-but-not-easily-computer-readable IPA (International Phonetic Alphabet). It uses "letters" consisting of 1-2 ASCII characters to represent each phoneme. The ASCII sequences for the SAMPA letter are designed so that any SAMPA sequence is deterministically parsible.

We chose to use SAMPA instead of IPA because its ASCII-compliance makes it easy to integrate into other systems.

See table **??** for a full table of each SAMPA phoneme, its description, and its sub-parts.

For some brief context, the SAMPA spelling of the name 'Jenee Hughes' is *dZEni hjuz*. 'Dr Zoe Wood' becomes *dAkt@`r zoui wUd*. 'Dr John Clements' becomes *dAkt@`r dZAn klEm@nts*. 'Dr Franz Kurfess' becomes *dAk@`r fr{nz k3`rfEs*.

Chapter 2

Introduction

Human brains are built to come to single conclusions about things that have more than one interpretation. The way that you come to this end conclusion is dependent upon your experiences, cutural immersion, and language familiarity [22]. When attempting to write English phrases that will be read aloud and heard by people with other linguistic biases than you, it's important to make your prose as deterministically understandable as possible. The first step towards this is understanding and identifying how many ways a particular textual phrase be misheard, and why.

2.1 You and me...and Leslie?

In the song *"Groovin' (on a Sunday Afternoon)"*, by the Young Rascals, there's a part in the bridge that many people hear as *"Life would be ecstasy, you an' me an' Leslie"*. In fact, the line is *"Life would be ecstasy, you and me endlessly"*. The confusion lies with the last three syllables of the phrase. The pronunciation of each version, if spoken normally, is as follows:

Orthographic:	and Les- lie	end- less- ly
SAMPA:	@nd “lEs li	“End l@s li

In the song, the singer is doing what many singers are taught to do, to make it easier to sustain the singing of words that end with difficult-to-sing consonants: the unsingable consonant is displaced onto the front of the next word. In this case, the consonant “d” is not singable, so he displaces it onto the next syllable, when he can: “and ME” becomes “an dME”, and “end LESS” becomes “en dLESS”.

Basically, singers are *born* to ignore syllable boundries. So, our singer can effectively think of the sung phrase as:

YOU an dME en dLESS lee

This does not cause confusion for listeners, because they are used to hearing it. This does mean, however, that lyric placement does not provide an accurate barometer to a listener of where a word actually ends.

In addition, the singer is singing fudging his vowels, like singers are taught to do, so “and” and “end” sound almost indistinguishable. So, really, what listeners are hearing is this:

YOU en dME en dLESS lee

Now, the listener’s brain has to take this syllabic gobbledy-gook, and parse it into something useful. They’ve currently got this mess to deal with (represented in SAMPA syllables):

ju *En* ***dmi*** *En* ***dl@s*** *li*

They parse the first part just fine, because the emphases match:

you and **me** *En* ***dl@s*** *li*

But no one says endLESSly. People say ENDlessly. So, the listeners don't recognize it. They have to work with what they have. They already turned one "En d" into an "and", so they do it again:

you and **me** and ***l@s*** *li*

Now, they're just left with LESS lee. And that fits Leslie, a proper noun that fits in context and in emphasis placement. So, the final heard lyric is:

you and **me** and **Les-** lie

The misunderstanding can be traced back to improper emphasis placement. The songwriter probably didn't even think of that, and now he's stuck: a one-hit-wonder with a misunderstood song. We bet that in interview after interview, someone asks him who Leslie is. It's probably very frustrating — especially since he could have just moved the word an eight note later, and it would have been understood perfectly.

That's the sort of situation this program is going to help avoid.

2.2 Why it breaks down

There are two points at which the author's intendeded phrasing can be muddled : First, when the author's orthographic text becomes an orator's spoken (phonetic) interpretation, and second, when the orator's phonetic interpretation

is translated phonetically by an audience into a perceived orthographic phrase. Both of these interpretations must be made succesfully in order for the author's intended meaning to be conveyed.

The phrase "iced ink" undisputedly succeeds in the first translation, but fails on the second. Iced ink can only be pronouced one way, but it can be heard multiple ways–the most notable of which is "I stink", not "iced ink".

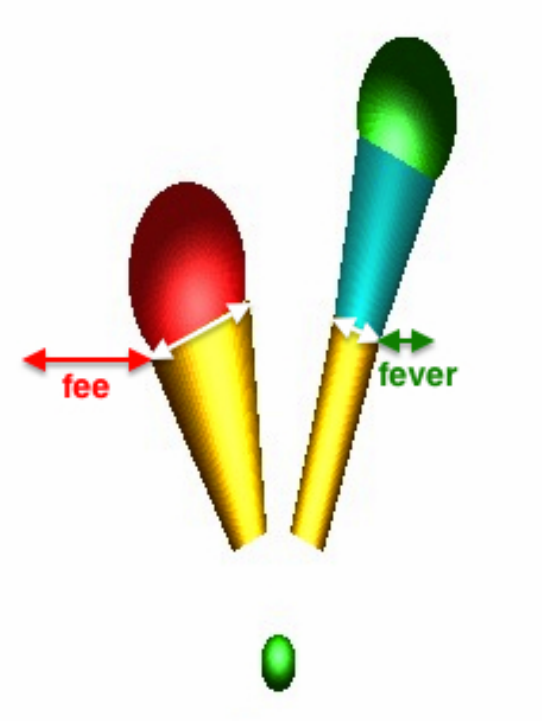
The phrase "a nice cold hour" can fail on both parts. First, the orator could have accidentally-capitalized the word Nice in their head, and made it sound like Nice, the city in France. An audience would likely hear this as "niece", and would be confused, at best. Even if the orator pronouces the phrase as the author intended, the audience could hear multiple orthographic phrases in the same phonetic sequence: "a nice cold hour", "an ice cold hour", or even "a nigh scold our".

A third, more rare and nefarious type of audience misunderstanding can be caused by parse-tree misdirection, where an audience member is absolutely sure they're hearing one phrase, only to get lost halfway through the lyric because they though they were interpreting a phonetic sequence in a way that resulted in an orthographic dead end. This happens due to the relative frequency of the possible lyrics heard.

For example, when asked to sing along with the Adele song, Rolling in the Deep, people who were starting to sing enthusiastically dropped out around the line "reaching a fever pitch"[16]. Let us consider the phrase "fever pitch". This phrase has no exact oronyms, but it does have a potential dead end– a listener could hear the first syllable of the phrase as the word "fee", which has a frequency of 7265. That's more than double the frequency of the word "fever", which is

3095.

Looking at the oronym parse tree for the phrase "fever pitch" in figure 2.1, we can see that the branch for "fever" ends in a much smaller radius than the branch on the left for the word "fee". As you can see by the relative size of the end spheres of the branches, the word "fee" even outweighs the last word in the other branch as well (which is "pitch" with a frequency of 5104). Since the human brain is pre-disposed to parse more-familiar words, having that heavily-weighted dead-end branch is likely the cause of the casual listener not being able to memorize the lyrics.



[h]

Figure 2.1: Annotated Oronym Parse tree generated for the phrase "fever pitch"

Chapter 3

Implementation

We present a computer program which takes in a textual phrase in English, determines all oronyms for that phrase, and then visualizes those oronyms in tree form, with branch width scaled by word frequency metrics to indicate the likelihood of interpretation.

To accomplish this, the program has three major functional parts: a custom phonetic dictionary, a command-line oronym generator, and a OpenGL oronym-parse-tree visualization generator.

3.1 Customized Phonetic Dictionary

In order to discover oronyms for each phrase, we first needed to determine how each phrase is pronounced. Pronunciation can vary depending on the speaker's accent, so it was important for us to (1) chose an accent that we could easily replicate and (2) find a dictionary that supported that accent.

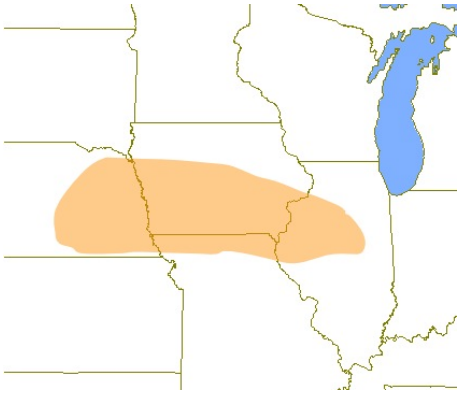


Figure 3.1: This is the geographic area whose accent most closely resembles the General American Accent [3]

3.1.1 Accent Choice

We decided to utilize a General American accent, due to its ubiquity in media and news sources. The General American accent, also known as the "Standard American English" dialect, is not spoken by most Americans, but is used as an "average accent". It most closely resembles the Midwestern accent used in the area in Figure 3.1, but is more commonly recognized as "the newscaster accent". Newscasters learn this accent for use on national TV, because it is the "least-accented" of the American accents[15].

The downside of using the General American accent is that, while it does give a good approximation of most American's speaking accents, it does not perfectly reflect a "singing accent". Singers tend to elongate syllables, changing emphasis placement in words, and vowels tend to be sung in a more "round" matter[17].For example, though the dictionary pronunciation of the word "baby" is `"be$bi` (bay-bee), in songs, you commonly hear the pronunciation `"be$be` (bay-bay). The `e` sound is easier to sing than the `i` (ee) sound, because the latter requires the the

```
ABBREVIATE  AH0 B R IY1 V IY0 EY2 T
```

Figure 3.2: Here is the CMU dictionary entry for the word "abbreviate"

singer move their mouth and vocal cord position further from neutral than the former does[12].

However, different singers will change pronunication based upon which vowels are easiest for them to sing, so using the General American accent still gives us a fairly good approximation[14].

3.1.2 Dictionary Options

We considered using three different phonetic dictionaries: the CMU dictionary, LC-STAR dictionary and UNISYN dictionary[7] [1] [13]. We started out by looking at the LC-STAR dictionary, but quickly decided that it wasnt going to be as useful to us, because the LC-Star project is relatively focused on Speech-to-Speech or Text-to-Speech tech. In addition, the dictionary is not well-maintained.

The CMU dictionary showed promise, but had a few shortcomings. It had a very simple way of encoding words: first the word, then the identifier number in parentheses (if needed), then a space, then a one-to-two char code for each sound in the word, with the numbers 0, 1, 2 appended to indicate emphasis (if needed), separated by spaces. An example of a CMU dictionary entry can be seen in Figure 3.2.

The problem that arose with this format, was that there was no explicit definition of where to hyphenate the word when splitting it up. This causes problems for words in song lyrics, where each note has its own syllable underneath it, and each syllable might have many different sounds. In addition, it used non-

standard symbols for its phonetic alphabet, which would complicate matters if, in the future, we chose to combine data from other dictionaries with our existing dictionary.

However, unlike some other dictionaries we considered, the CMU dictionary (1) was actively maintained, (2) included proper nouns, which are often found in lyrics, but not in dictionaries, and (3) was ridiculously easy to read.

With the downsides and benefits in mind, the CMU dictionary could not be used in isolation, especially if we wanted to incorporate contextual data from other sources.

The UNISYN dictionary is used primarily to phonetically translate words into multiple accents. It has its own formated dictionary, with a bunch of wild-cards representing different phones. UNISYN provides some semi-functioning perl scripts that allow you to specify a dialect youd like to use (For example, a Californian would say "cooking" differently than someone from the Deep South, and both would say it differently than someone from London. However, they are all speaking English. The UNISYN dictionary facilitates this translation).

It had all the information we needed, and then some. However, it was case-insensitve, meaning that it didn't make it easy to differenciate pronunciations for some words. For example, the word "nice" is pronounced differently from the city "Nice", but they were both stored as "nice" in the orthography of UNISYN. The CMU dictionary did keep track of capitalization. The obvious conclusion, then, was to grab the capitalizations from the CMU dictionary and put them in the UNISYN dictionary, aligning them by pronunciation and part of speech.

However, we ran into a setback, mentioned in the very first article we found references to both dictionaries in: the dictionaries were inconsistent[20]. They

Example:

```
transfer : 2 : VB/VBP : tr{ns"f3‘r : tr{nsf3‘r : {trans==fer}
                                : 7184
```

Figure 3.3: Here is an example an entry in our custom phonetic dictionary, using the word “transfer”

didnt always put stresses in the same place, nor did they always have the same pronunciation. Because of this, it was difficult to match words, especially words that were homographic heteronyms[1]. Because of this, we decided to use the UNISYN dictionary exclusively.

3.1.3 Custom dictionary fields

Here is the format for the fields in an entry in our custom phonetic dictionary, after we were done with fixing the UNISYN output:

<ortho> : <uniqueID> : <partOfSpeech> : <SAMPAspelling> : <SAMPAnoEmph> : <extendedOrtho> : <freq>

<ortho> is the regular, orthographic spelling of the word.

<uniqueID> is a number (and optional string) used to differentiate homographs[2].

<partOfSpeech> is used to identify the specific part of speech for the word.

<SAMPAspelling> is the breakdown of the word, phonetically. It uses the SAMPA alphabet, and separators to show where breaks in the word are, and

[1] Homographic heteronyms are words that are spelled identically by pronouced differently, such as “Do you know what a buck *does* to *does*?”

[2] A homograph shares the same written form as another word but has a different meaning; For example, a farmer would **sow** *(verb)* seeds in a field, but could also raise a **sow** *(noun)* for bacon.

how they're emphasized. If a separator is `$`, the subsequent phones (until the next separator) are not emphasized. If it's `%`, then they are pronounced using secondary emphasis. If it's `"`, then they are given the primary emphasis in the word.

<SAMPAnoEmph> is the same as *<SAMPASpelling>*, but with all emphasis separator characters stripped out. We chose to add this field so that we could more easily look up phonetic sequence matches.

<extendedOrtho> allows for stemming analysis of words, for possible use in future work.

<freq> is the frequency at which the word occurs in language, according to UNISYN. The frequency count is "taken from a composite of a number of on-line sources of word-frequency. It includes frequencies from the British National Corpus and Maptask, and frequencies derived from Time articles and online texts such as Gutenberg. They were weighted to give more importance to sources of spoken speech, and also to increase the numeric frequency of smaller corpuses"[18].

An example of a entry in our custom phonetic dictionary can be seen in **Figure 3.3**.

3.1.4 Transferring the dictionary to a sqlite database

Because there are several hundred thousand entries in our phonetic dictionary, it was necessary to have a database, rather than store them all in-program in a multi-dimensional array. We decided to use a SQLite database for this purpose.

To turn the colon-delimited dictionary file into a SQLite database, we decided

to use a program called the SQLite Database Browser, an open source, public domain, freeware visual tool to create, design, and edit SQLite3.x database files. We specifically used version 2.0b1 of the program, which was built with version 3.6.18 of the SQLite engine[11].

3.2 Oronym Generation

3.2.1 Step 1: Find all phonemic variations of an orthographic phrase

First, our program takes an orthographic phrase to find oronyms for (Figure ??).

'a nice cold hour'

We then tokenize this phrase into its component words, using whitespaces as a delimiter (Figure ??).

'a', 'nice', 'cold', 'hour'

For each word in the phrase, we query our phonetic dictionary for all possible SAMPA pronunciations (Figure 3.4). .

Now that we have the pronunciation of each of the words in the form of SAMPA strings, we can list all the possible phonetic permutations of the original phrase(Figure 3.5). .

The pseudocode for this process can be reviewed in figure 3.6.

‘a’ → `e, @, A`

‘nice’ → `naIs, nis`

‘cold’ → `kould`

‘hour’ → `aU`r`

Figure 3.4: In this and all subsequent diagrams, a ‘string in quotes’ indicates an orthographic word or phrase, and a `monospaced string` indicates that it is a SAMPA word or phrase.

`e naIs kould aU`r`

`@ naIs kould aU`r`

`A naIs kould aU`r`

`e nis kould aU`r`

`@ nis kould aU`r`

`A nis kould aU`r`

Figure 3.5:

3.2.2 Step 2: Finding all Orthographic phrases for a Phonemic Sequence

Then, for each phonemic phrase, we want to figure out all valid orthogrpahic interpretations. For this, we have to go back to our phonetic dictionary.

The ideal way to think about searching for words in a phonetic sequence is by picturing the phoenetic sequence in a tree form, similar to the tree pictured in abbreviated form in Figure 3.7. For example, if I had a phonetic tree with the entire dictionary in it, each phonetic tree node would have at least 45 child nodes: one for each phone. A node might also have “word” nodes, if the phones along the path to that node construct a valid orthographic word:

```
findAllPhoneSeqsForOrthoPhrase( orthoPhrase ) {
  allFullPhrasePhoneSeqs = empty list of list of phones
  orthoWords = split orthoPhrase on spaces

  origNumFullPhrases = 0
  for( orthoWord in orthoWords with index i ) {
    nextWordSampaPhoneSeqs = possible phone seqs following orthoWord

    if ( orthoWord is the first word in orthoPhrase ) {
      for( phoneSubSeq in nextWordSampaPhoneSeqs ) {
        append phoneSubSeq to allFullPhrasePhoneSeqs[i]
      }
    } else {
      origNumFullPhrases = allFullPhrasePhoneSeqs.size()
      if theres more than one vector <phone> in nextWordSampaPhoneSeqs
        then we need to create duplicates of all existing allFullPhrasePhoneSeqs
    }

    for( m = 0 to allPhrasePhoneSeqs.size() ) {
      phraseToAppendIndex = m / origNumFullPhrases
      phoneSeqToAppend = nextWordSampaPhoneSeqs[phraseToAppendIndex]
      append phoneSeqToAppend to allFullPhrasePhoneSeqs[m]
    }
  }

  return allFullPhrasePhoneSeqs
}
```

Figure 3.6: Algorithm to get all phonetic sequences for an orthographic phrase.

When there are multiple orthographic interpretations at a single phonetic node, the most likely interpretation can be determined by checking the frequency of use for each word. For example, the sequence `n aI s` is much more likely to be "nice" than "gneiss". Figure 3.7 shows a visual representation of traversing an entire dictionary's phonetic tree for nodes along the paths for the SAMPA sequences `aI s` and `n aI s`.

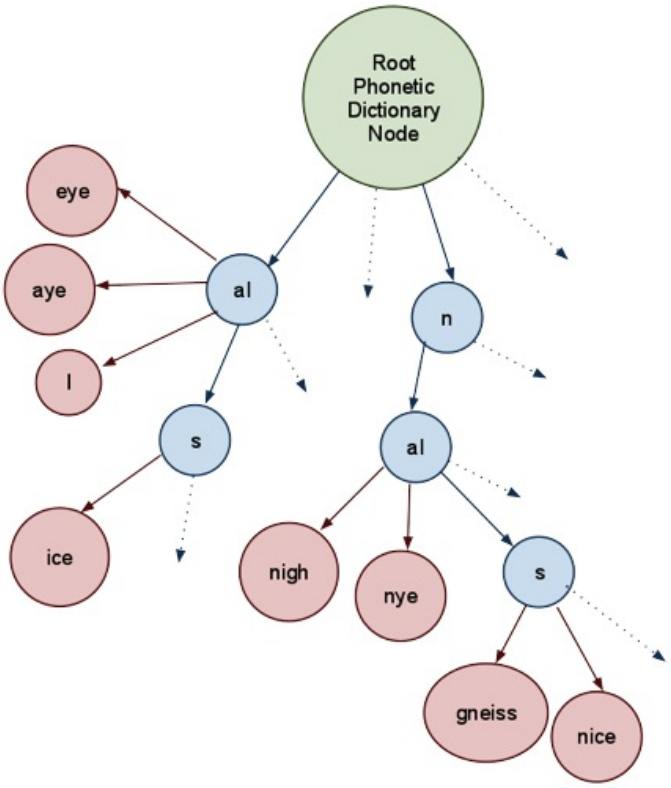


Figure 3.7:

We can use this dictionary tree method to discover all valid orthographic interpretations for any phonetic sequence of our root orthographic phrase, as shown in figure 3.8 for the phrase:

Once we have grabbed all the orthographic interpretations for each phonetic sequence, we combine them all into an orthographic oronym phrase list. This

process may leave us with some redundant oronyms, so we de-duplicate that list.

After this, we have a list of all unique and valid oronyms for the original root phrase.

In the case of “a nice cold hour”, this returns 290 oronyms, as seen in the first column of figure **??**.

The pseudocode for this process can be reviewed in figure 3.9

3.2.3 Word Frequency Evaluation

Next, we want to evaluate all our oronyms based on how common each oronym’s component words are. For example, “a nice cold hour” is much more likely to be heard “a gneiss cold hour”, even though both are phonetically identical.

To do this, we tokenize each oronym phrase into its component words, once again using non-newline whitespaces as a delimiter.

Then, we query our phonetic dictionary with each word for the word’s frequency value. We store each word’s value separately. When we have retrieved the frequencies for all the words in a phrase, we then add all the frequencies up to give a combined-frequency of the entire phrase.

You can see these frequency counts for the phrase “a nice cold hour” in figure **??**.

3.3 Visual Representation

We go about building the visual representation of the oronym parse tree in much the same way that we build the textual list of oronyms, with one important difference: our oronym parse trees may contain oronym fragments. To deal with these we've got to keep track of all our abandoned sub-phrases.

Our algorithm for doing this is recursive, called from a parent function that draws the tree's 'seed' sphere. This parent function is documented in figure 3.10

We start in the parent function by getting all the oronyms of our orthographic phrase, using the process in sections 3.2.1 3.2.2. However, instead of ignoring any incomplete orthographic interpretation of a phonetic sequence, as we do in section 3.2.2, we add them to the list of oronyms, keeping track of them by appending 'xxx' or 'fff' to the end of the incomplete oronym string. Then, we tokenize our phrases by whitespace, and look up the frequency of each word, keeping track of only the maximum and minumum values. We will later scale our branches' radiuses using these values.

Once we have all the partial and complete oronyms and the max and min word frequency values for them, we pass them into our recursive function, along with the radius of the seed sphere. That radius will be the beginning radius of each first-level branch.

Inside our recursive function, we pull the first word out of every orthographic phrases we were passed, and create a set of unique first words.

We then go through this set of unique first words iteratively.

For each word, we look up frequency in the phonetic dictionary. Then, we use the max and min frequencies that we found in our parent function, plus constants

for max and min radius size, to scale that frequency into a usable radius size.

Then, we check the contents of the word.

If the word is "xxx" or "fff", then it's not a word at all–just an indication of the dead end of a partial oronym. In this case, we draw a red sphere with the radius of the branch's ancestor, using the parameter past into our recursive function for 'lastRadius'.

If the word is "___SUCCESS!___", that is also not a real word. It indicates that a full oronym has been successfully found, and is terminating at that point. This time, we draw a green sphere using the 'lastRadius' parameter for size.

If the word is neither of these, then it must be a real word. We then draw a cylinder "branch" representing that word. The cylinder's bottom radius is equal to *lastRadius*, and the top radius is equal to the scaled radius that we got from the word's frequency.

After we draw the cylinder, we then go through the full list of phrases, and compile a list of all phrases that start with the word we just drew the cylinder for. Then, we remove the first word from each of those phrases, deduplicating the resulting list of "tail" phrases.

Then, we change our material color (so that different levels of branches will be different colors), and make a recursive call to our current function, passing as parameters the scaled radius and the list of tail phrase.

After this recursive call, we change our color material back to whatever it was before the call, and then continue on to the next unique first word in our set.

Once we have looped through all our unique first words, we know we're done drawing that set of branches, and we return.

This gives us the oronym parse tree seen in figure 3.12. As shown in figure 3.13 (the annotated version of figure 3.12) each branch on the tree represents a single orthographic word.

At the end of this process, we have generated a tree like the one in figure 3.12. Each branch represents a word, as can be seen in figure 3.13.

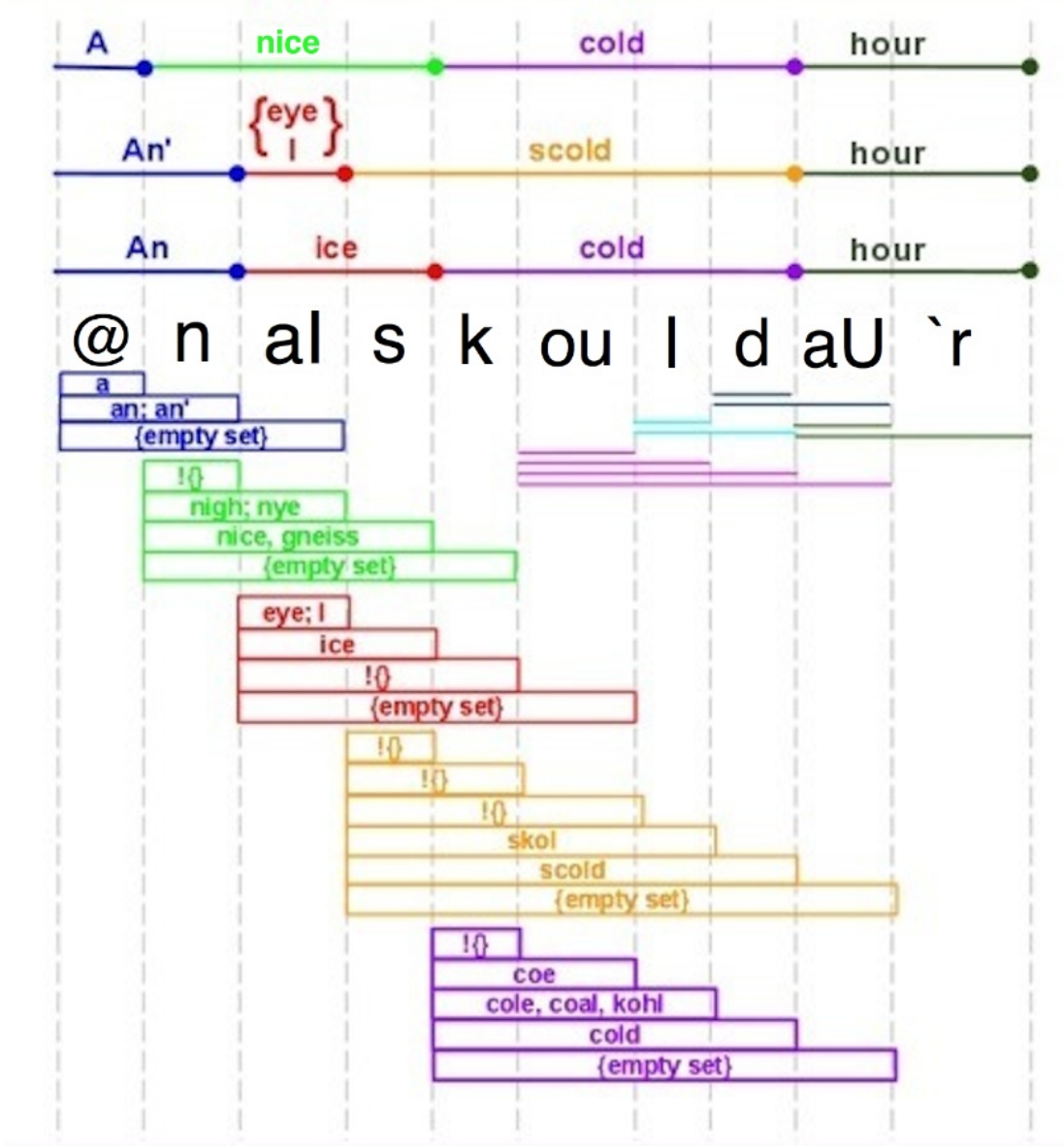

A
nice
cold
hour
eye
I
An'
scold
hour
An
ice
cold
hour
@ n aI s k ou l d aU `r
a
an; an'
{empty set}
!{}
nigh; nye
nice, gneiss
{empty set}
eye; I
ice
!{}
{empty set}
!{}
!{}
!{}
skol
scold
{empty set}
!{}
coe
cole, coal, kohl
cold
{empty set}

Figure 3.8:

```
discoverOronymsForPhrase( origOrthoPhrase, includeDeadends ) {
   orthoMisheardAsPhrases = empty list
   allPhoneSeqsOfOrigPhrase = origOrthoPhrase.findAllPhoneSeqs()

   for( curPhoneSeqWithEmph in allPhoneSeqsOfOrigPhrase ) {
      // Remove emphasis marking for easier lookups
      curPhoneSeq = curPhoneSeqWithEmph.stripEmphasis()

      altOrthoPhrases = findOrthoStrsForPhoneSeq( curPhoneSeq )

      for( altOrthoPhrase in altOrthoPhrases ) {
         // Ensure it contains valid ortho text in all cases, and if
         // includeDeadends=false, contains no deadEndDelims so we only add
         // fully valid strings
         if ( ( includeDeadends == true &&
                altOrthoPhrase != deadEndDelim1 &&
                altOrthoPhrase != deadEndDelim2 ) ||
              ( altOrthoPhrase.contains( deadEndDelim1 ) == false &&
                altOrthoPhrase.contains( deadEndDelim2 ) == false ) ) {
            append altOrthoPhrase to orthoMisheardAsPhrases
         }
      }
   }

   orthoMisheardAsPhrases.removeDuplicates()

   return orthoMisheardAsPhrases
}
```

Figure 3.9: Algorithm to get all oronyms for an orthographic phrase.

```
buildAndDrawFullTree( orthoPhrase ) {
   fullPhrases = orthoPhrase.discoverOronyms()
   (maxWordFreq, minWordFreq) = fullPhrases.getMaxAndMin()

   // Draw the tree's seed.
   glPushMatrix()
   {
      glTranslated(0.0, -1.0 * DEFAULT_BRANCH_LEN, 0.0)
      materials(GreenShiny)
      drawSphere(DEFAULT_RADIUS)
      materials(allMaterials.at( mat % allMaterials.size() ) )

      drawBranchesAtFork ( fullPhrases, DEFAULT_RADIUS )
   }
   glPopMatrix()
}
```

Figure 3.10: Given an orthographic phrase, this function prepares to draw the tree

```
drawBranchesAtFork( fullPhrases, lastRadius) {
   if( fullPhrases.size() == 0 ) {
      return
   }

   // Use a set to ensure no duplicates.
   firstWords = empty set

   for( phrase in fullPhrases ) {
      if( phrase.size() > 0 ) {
         firstWords.insert( phrase.firstWord() )
      }
   }

   // Calculate positioning variables for the spread of branches for firstWords.
   for ( curFirstWord in firstWords ) {
      firstWordFreq = curFirstWord.frequency()
      newAdditiveRadius = firstWordFreq.scaleToRadius()

      glPushMatrix()
      {
         // Translate and rotate into place
         if( curFirstWord == deadEndDelim1 || curFirstWord == deadEndDelim2 ) {
            // Draw a red sphere at the end of the last branch
         } else if ( curFirstWord == successDelim ) {
            // Draw a green sphere at the end of the last branch
         } else {
            // Draw a branch
            drawBranch( radiansToDegrees(tiltAngle), curXOffset, curYOffset,
                        newAdditiveRadius, lastRadius )

            // Find all phrases in fullPhrases that start with that firstWord
            tailsVect = fullPhrases.findAllWithPrefix(curFirstWord)

            // Change the colors for each branch level

            // Pass those phrases to drawBranchesAtFork
            drawBranchesAtFork( tailsVect, newAdditiveRadius, curXOffset, curYOffset )

            // Change the colors back to ensure consistency for each branch level
         }
      }
      glPopMatrix()
   }
}
```


Figure 3.11: This is the function that facilitates the in-time drawing of the tree as we parse though our oronyms possibilities

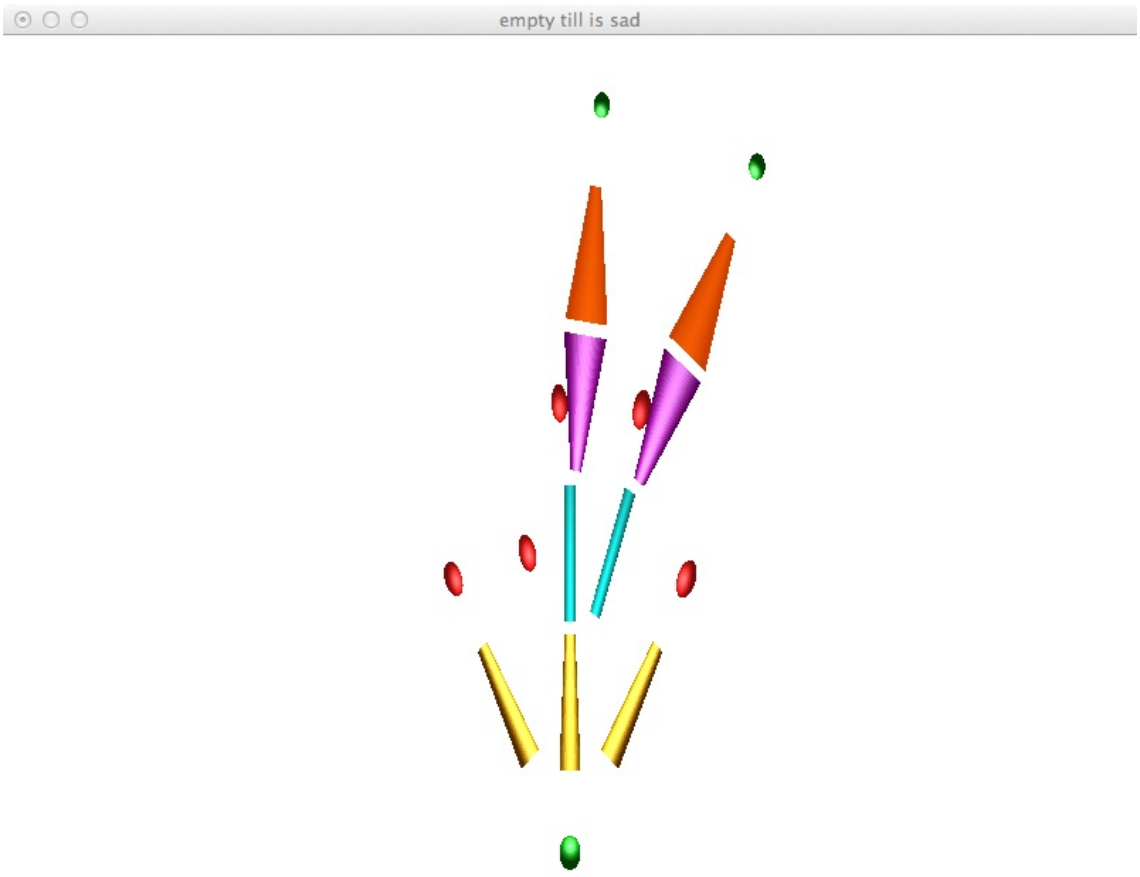


Figure 3.12: This is the parse tree for the phrase "empty till is sad"

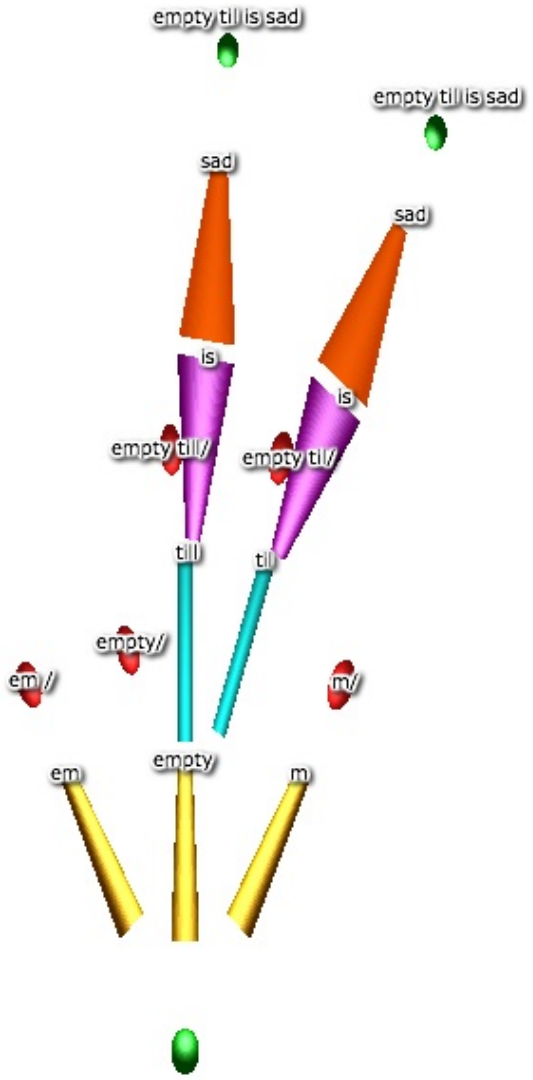


Figure 3.13: This is the annotated parse tree for the phrase "empty till is sad"

Bibliography

[1] The CMU pronouncing dictionary. http://www.speech.cs.cmu.edu/cgi-bin/cmudict.

[2] Corpus-based word frequency lists, collocates, and n-grams. http://www.wordfrequency.info/comparison.asp.

[3] File:General american.png - wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/File:General_American.png.

[4] Keep thou my way, hymnlyrics.org. http://www.hymnlyrics.org/newlyrics_k/keep_thou_my_way

[5] knights_emic.gif (GIF image, 552 407 pixels) - scaled (0%).

[6] knights_phonetic.jpg (JPEG image, 552 407 pixels) - scaled (0%).

[7] LCStar project web – schedule. http://www.lc-star.com/schedule.htm.

[8] Mondegreen | define mondegreen at dictionary.com. http://dictionary.reference.com/browse/mondegreen?s=t.

[9] oronym - definition and meaning. http://www.wordnik.com/words/oronym.

[10] Orthography | define orthography at dictionary.com. http://dictionary.reference.com/browse/orthography.

[11] SQLite database browser. http://sqlitebrowser.sourceforge.net/.

[12] Understanding how to sing the vowels - a technical description for classical singers. http://ezinearticles.com/?Understanding-How-to-Sing-the-Vowels—A-Technical-Description-For-Classical-Singers&id=1671860.

[13] Unisyn lexicon. http://www.cstr.ed.ac.uk/projects/unisyn/.

[14] Why standard american english really is "no accent". http://boards.straightdope.com/sdmb/archive/index.php/t-619668.html.

[15] 'At the tone' it will be jane barbe, america's answer to big ben. *People Magazine*, Aug. 1976.

[16] (1) "Rolling in the deep" cover, front porch band, Jan. 2012.

[17] D. Crystal. DCblog: on singing accents, Nov. 2009.

[18] S. Fitt. Documentation and user guide to UNISYN lexicon and post-lexical rules. *Center for Speech Technology Research, University of Edinburgh, Tech. Rep*, 2000.

[19] J. Hendrix. Purple haze, June 1967.

[20] T. Polyakova and A. Bonafonte. Fusion of dictionaries in voice creation and speech synthesis task. In *Proc. of SPECOM*, 2007.

[21] C. C. Revival. Bad moon rising, Apr. 1969.

[22] G. P. Smith. Music and mondegreens: extracting meaning from noise. *ELT Journal*, 57(2):113121, 2003.

[23] S. Wright. The death of lady mondegreen. *Harpers Magazine*, 209(1254):4851, 1954.