Transducers as a Substrate for Natural Language Processing

Stuart M. Shieber
Provenance

These slides were developed for the course

Transducers

taught at the 15th European Summer School in Logic Language and Information, in Vienna, Austria, August 18-22, 2003.

Please do not cite, quote, copy or redistribute.

Motivation


*Blade Runner, 1982*
Motivation

![Dragon Naturally Speaking Command Browser](image)
Voice to Commands

Voice ➔ Command
Voice to Commands

Voice → Words → Command
Voice to Commands

Hidden Markov Models
phonotactic models
dictionaries
language models
Voice to Commands

Voice → Words → Command

Samples → Phones → Triphones → Words

Hidden Markov Models

Weighted Finite State Transducers
dictionaries

phonotactic models

language models
Voice to Commands

String → WFST → String → Command
Voice to Commands

String ➔ \textit{WFST} ➔ String ➔ \textit{Parser} ➔ Tree ➔ ??? ➔ Command
The Universal NL Pipeline

Strings is to WFST as trees is to ???
Ubiquity of Tree Transformation

Command interpretation
  - Database query construction

Semantic interpretation
  - Semantic disambiguation

Machine translation transfer

Parsed corpus manipulation and normalization

Natural language generation
  - Logical form canonicalization
Overview

Finite state automata and transducers
Weighted automata and transducers
Tree transducers and their insufficiency
Extensions via bimorphism generalization
Other extensions
Plan

Review finite state automata

- regular languages
- automata
- Thompson’s construction
- degrees of freedom
  - left/right reversal
  - granularity
  - epsilon removal
  - determinization
  - minimization
Plan

Review finite state automata

Finite state transducers

- regular relations
- degrees of freedom
  - composition
  - inversion
- application
Regular Languages and Finite State Automata
Regular Languages

A language is a set of strings.

Regular languages is the smallest class of languages including

- the empty language
- singleton languages
- closure under
  - union \( (L_1 \cup L_2) \)
  - concatenation \( (L_1 \cdot L_2) \)
  - iteration closure \( (L^*) \)
Regular Expressions

Notation for regular languages:

- Empty string: $\epsilon$
- Singleton language: $a$
- Union: $x \mid y$
- Concatenation: $xy$
- Iteration: $x^*$

Example alphabet $= a \ldots z:$

$$(((be \mid it \mid let) \_)^* \ni let \_it \_be \_)$$
Finite State Automata

![Finite State Automata Diagram]

- Initial state: 0
- Transition: a
- Final state: 2
- Input: a, bb, a

Division of Engineering and Applied Sciences
Harvard University
Finite State Automata

A derivation:

\[ aababa\# \rightarrow aababa\# \rightarrow abba\# \rightarrow bba\# \]
\[ \rightarrow bba\# \rightarrow a\# \rightarrow 2a\# \rightarrow 2\# \rightarrow \# \]

Definition of recognition:

\[ \text{accept } w \text{ if and only if } \circ w\# \rightarrow^* \# \]
FSAs from Regular Expressions

Thompson’s Construction
Sample FSA

\[((is \mid it \mid let) \_\_\_)^*\]
Degrees of Freedom

Lots of things make no difference:

- Granularity
- Epsilon removal
- Left right reversal
- Determinization
- Minimization
Granularity of Input

\[((is \mid it \mid let))\]
Granularity of Input

\(((\text{is} \mid \text{it} \mid \text{let})_{\leq})^*\)
Epsilon Removal

\[
is \rightarrow \text{it} \rightarrow \text{let}
\]
Epsilon Removal
Epsilon Removal
Epsilon Removal
Left Right Reversal
Left Right Reversal
Determinization
Determinization
Determinization
Determinization
Determinization
Minimization

States correspond to equivalence classes of suffixes. States corresponding to identical equivalence classes can be merged.
Minimization

States correspond to equivalence classes of suffixes. States corresponding to identical equivalence classes can be merged.
Minimization

States correspond to equivalence classes of suffixes. States corresponding to identical equivalence classes can be merged.
Minimization

States correspond to equivalence classes of suffixes. States corresponding to identical equivalence classes can be merged.
Minimization

States correspond to equivalence classes of suffixes. States corresponding to identical equivalence classes can be merged.
Minimization

States correspond to equivalence classes of suffixes. States corresponding to identical equivalence classes can be merged.
Finite State Transducers
Regular Relations

A string relation is a set of *pairs of strings.*

\[ \text{input} : \text{output} \]

Regular relations is the smallest class of relations including:

- the empty language
- singleton languages, e.g., \{a : \epsilon\}, \{\epsilon : a\}, \ldots
- closure under
  - union
  - concatenation
  - iteration closure
Sample FST
Sample FST
Sample FST

A derivation:
\[ \varepsilon a b b a \# \rightarrow \varepsilon a b b a \# \rightarrow \varepsilon a b b a \# \rightarrow \varepsilon b b a \# \rightarrow \varepsilon b b a \# \rightarrow \varepsilon c a \# \rightarrow \varepsilon c a \# \rightarrow \varepsilon c \# \]

Definition of recognition:
- accept \( s : t \) if and only if \( \varepsilon s \# \rightarrow^* t \# \)
Degrees of Freedom

granularity of input and output
left right reversal
epsilon removal *

pushing
determinization *
minimization *

inversio#
compositio#

* sometimes
Granularity of Output

A Spelling Dictionary
Granularity of Output
Left Right Reversal

...by treating FST as FSA over cross product vocabulary.
Left Right Reversal

...by treating FST as FSA over cross product vocabulary.
Inversion
Inversion

\[ i : \langle \text{is} \rangle \quad s : \epsilon \]

\[ i : \langle \text{it} \rangle \quad t : \epsilon \]

\[ l : \langle \text{let} \rangle \quad e : \epsilon \quad t : \epsilon \]
Pushing

Movement of output symbols along a path.
Pushing
Pushing
Pushing
Determinization
Determinization

No more determinization possible...
Determinization

No determinization possible without pushing ...
The transducer determinization algorithm performs forward pushing implicitly.
Pushing

Forward pushing: When all in edges end in \(x\),
- remove \(x\) from end of all in edges
- add \(x\) to start of all out edges.
Pushing

Forward pushing: When all in edges end in $x$,

- remove $x$ from end of all in edges
- add $x$ to start of all out edges.
Pushing

Forward pushing: When all in edges end in $x$,

- remove $x$ from end of all in edges
- add $x$ to start of all out edges.

```
\begin{align*}
&: w_1 x & : y_1 \\
&: w_2 x & : y_2 \\
&: w_3 x
\end{align*}
```
Pushing

Forward pushing: When all in edges end in $x$,
- remove $x$ from end of all in edges
- add $x$ to start of all out edges.

\[
\begin{align*}
&: w_1 \\
&: w_2 \\
&: xw_3 \\
&: xy_1 \\
&: xy_2
\end{align*}
\]
Limitations on Determinization

Phrasal verb marking:

1. John \textit{called} the teacher \textit{up}.
2. John \textit{called} the dogs \textit{off}.

Idealized transduction problem:

- \textit{call} \(x^*\) \textit{up} \rightarrow \textit{call}1 \(x^*\) \textit{up}
- \textit{call} \(x^*\) \textit{off} \rightarrow \textit{call}2 \(x^*\) \textit{off}

based on Roche and Schabes 1996
Limitations on Determinization

Idealized transduction problem:

- \textit{call} \, x^* \, \textit{up} \rightarrow \textit{call1} \, x^* \, \textit{up}
- \textit{call} \, x^* \, \textit{off} \rightarrow \textit{call2} \, x^* \, \textit{off}
Limitations on Determinization

Idealized transduction problem:

- call \( x \ x \ \text{up} \rightarrow \text{call}1 \ x \ x \ \text{up} \)
- call \( x \ x \ \text{off} \rightarrow \text{call}2 \ x \ x \ \text{off} \)
Limitations on Determinization

Idealized transduction problem:

- \( \text{call } x \ x \ \text{up} \rightarrow \text{call1 } x \ x \ \text{up} \)
- \( \text{call } x \ x \ \text{off} \rightarrow \text{call2 } x \ x \ \text{off} \)
Limitations on Determinization

Idealized transduction problem:

- call \( x \times x \uparrow \rightarrow \text{call1} \times x \uparrow \)
- call \( x \times x \downarrow \rightarrow \text{call2} \times x \downarrow \)
Limitations on Epsilon Removal

No state \( q \) and string \( w \neq \epsilon \) such that \( q \rightarrow^* w q \)
Composition

Given epsilon-free letter transducers

\[ T_1 = \langle Q, \Sigma, \Sigma', \Delta, q_0, F \rangle \]
\[ T_2 = \langle Q', \Sigma', \Sigma'', \Delta', q'_0, F' \rangle \]

the composition

\[ R(T_1 \circ T_2) = \{(s, t) \mid (s, u) \in R(T_1), (u, t) \in R(T_2)\} \]

The composition is constructed as

\[ T_1 \circ T_2 = \langle Q \times Q', \Sigma, \Sigma'', \Delta'', \langle q_0, q'_0 \rangle \rangle, F \times F' \]

where

\[ \delta'' = \{ \langle q_s, q'_s \rangle, a, b, \langle q_d, q'_d \rangle \mid \exists c \in \Sigma', \langle q_s, a, c, q_d \rangle, \langle q'_s, c, b, q'_d \rangle \} \]

The construction is easily extended to arbitrary letter transducers and arbitrary transducers.
Application: Morphological Parser

Overview

%% Language model: nouns with optional plural marker
%% separated by word boundaries
macro(lm, [nouns, option('<+s>'), '<wb>']*).

%% Nouns
macro(nouns, id({'<boy>',
    '<child>',
    '<sky>',
    '<box>'
})).
Application: Morphological Parser

Overview

%% Replace irregular inflected forms with their spelling
macro(spellirreg,
    replace(['<child>', '<+s>'] x word(children), [], [])).

%% Replace regular forms with their spelling
macro(spellreg, {'<boy>':word('boy'),
    '<child>':word('child'),
    '<sky>':word('sky'),
    '<box>':word('box'),
    '<+s>':word('+s'),
    '<wb>':'
    id(a..z)
})*).
Application: Morphological Parser

Overview

%%% Consonants
macro(consonant, {b, c, d, f, g, h, j, k, l, m, n,
                   p, q, r, s, t, v, w, x, y, z}).

%%% Orthographic rules for pluralization
macro(ortho,
      replace(word('y+'):word('ie'), consonant, s)
      o replace('+':e, {s, z, x, word(ch), word(sh)}, s)
      o replace('+':[], [], s)
    ).

%%% Morphological parser that inverts orthography
macro(parse, invert(lm o spellirreg o spellreg o ortho)).
Application: Morphological Parser

Demo
Weighting
Why Weights?

FSAs have multiple paths
How to adjudicate?
  • Notion of best path

Other applications:
  • Numeric functions over strings
    • perfect hashing
Semirings

A set $K$ along with operations of sum and product obeying the following algebraic laws:

- **Associativity of $+$**: $(x + y) + z = x + (y + z)$
- **Commutativity of $+$**: $x + y = y + x$
- **Associativity of $\times$**: $(x \times y) \times z = x \times (y \times z)$
- **Identity for $+$**: $x + 0 = 0 + x = x$
- **Identity for $\times$**: $x \times 1 = 1 \times x = x$
- **Zero idempotence**: $x \times 0 = 0 \times x = 0$
Semiring Operations on Automata

Product $\otimes$: along paths

Sum $\oplus$: among paths

$1 \otimes x_a \otimes x_b \otimes x_f$

$1 \otimes x_a \otimes x_b \otimes x_f \oplus 1 \otimes y_a \otimes y_b \otimes y_f$

$1 \otimes y_a \otimes y_b \otimes y_f$
Example Semiring: Strings

String sets form a semiring with:

- **Sum**: union
- **Product**: concatenation
- **0**: the empty language
- **1**: the language containing the empty string

Transducers can generate their output in any semiring.
String Semiring = FST

Product \cdot : along paths
Sum \cup : among paths

\[ \epsilon \cdot c \cdot d \cdot c = cdc \]
\[ \epsilon \cdot dd \cdot \epsilon \cdot cc = ddcc \]
\[ cdc \cup ddcc = \{ cdc, ddcc \} \]
Example Semiring: Probabilities

Probabilities (0...1) form a semiring with

- Sum: +
- Product: ×
- 0: 0
- 1: 1

Weights place *relative values* on transitions, hence paths as product of transitions, hence inputs as sum over paths.
Probability Semiring = WFSA

Product $\times$: along paths

Sum $+$: among paths

$1 \times .7 \times 1 \times 1 = .7$

$.7 + .3 = 1$

$1 \times .3 \times 1 \times 1 = .3$
Weighted Finite State Automaton

Defines probability distribution over strings:

- $a$
- $bb$
- $ab$
- $1.0$

![Automaton diagram]

- $a: 0.1$
- $b: 0.5$
- $a: 0.4$
- $b: 0.8$
- $b: 1.0$
Weighted Finite State Automaton

Defines probability distribution over strings:

- $a : 0.02$
- $bb$
- $ab$
- $I.O$

![Diagram of a weighted finite state automaton with transitions and weights labeled with probabilities.](image-url)
Weighted Finite State Automaton

Defines probability distribution over strings:

- $a : .02$
- $bb : .5$
- $ab$
- $1.0$

![Automaton Diagram]

- $a : .1$
- $b : .5$
- $a : .4$
- $b : .1$
- $b : .8$
- $1$
Weighted Finite State Automaton

Defines probability distribution over strings:

- \( a : 0.02 \)
- \( bb : 0.5 \)
- \( ab : 0.48 \)
- \( 1.0 \)

\[ a : 0.1 \]
\[ b : 0.8 \]
\[ a : 0.4 \]
\[ b : 0.5 \]
Best Path

Best path through a weighted automaton

- Viterbi decoding
- Dijkstra’s algorithm
- dynamic programming
  - computes score of best path from start state to each state
  
\[
\delta_q(0) = \begin{cases} 
1 & \text{if } q = q_0 \\
0 & \text{otherwise} 
\end{cases}
\]

\[
\delta_q(t + 1) = \max_{\langle q', a : p, q \rangle \in \Delta} \delta_{q'}(t) \cdot p
\]

- for a given input, just intersect
Best Path Computation

Example

\[
\begin{align*}
1 & \quad a : 0.1 \quad b : 0.5 \\
 & \quad a : 0.4
\end{align*}
\]

\[
\begin{align*}
 & \quad b : 0.8 \\
 & \quad b : 1 \\
 & \quad 1
\end{align*}
\]
Best Path Computation

Example
Best Path Computation

Example
Best Path Computation

Example

![Diagram showing a network with nodes and edges labeled with probabilities.](image)
Best Path Computation

Example
Weights Over Cycles

\[a : 1\]
\[b : 0.2\]
\[c : 0.2\]
\[a : 0.8\]

Diagram:

- Two circles connected by arrows labeled with weights.
- The first circle transitions to the second with weight 1.
- The second circle transitions back to the first with weight 0.2.
- The second circle also transitions to a third circle with weight 0.8.
- The third circle transitions back to the second circle with weight 0.2.
Weights Over Cycles

\[ P(a(bc)^*) = \sum_{n=0}^{\infty} (.2 \cdot .2)^n \cdot .8 = .8 \frac{1}{1 - .04} = .83 \]
Weights Over Cycles

\[ P(a(bc)^*) = \sum_{n=0}^{\infty} (0.2 \cdot 0.2)^n \cdot 0.8 = 0.8 \frac{1}{1 - 0.04} = 0.83 \]

\[ P(a(bc)^*ba) = \sum_{n=0}^{\infty} (0.2 \cdot 0.2)^n \cdot 0.2 \cdot 0.8 = 0.16 \frac{1}{1 - 0.04} = 0.16 \]
Application:

**Language Modeling**

Goal: describe probability of strings of a language based on a sample training corpus

\[
P(w_1 \cdot \cdot \cdot w_c) = \prod_{i=1}^{k} P(w_i \mid w_1 \cdot \cdot \cdot w_{i-1})
\]

Approximate under a Markovian assumption that words depend only on the previous \(N+1\)

\[
P(w_1 \cdot \cdot \cdot w_c) = \prod_{i=1}^{k} P(w_i \mid w_{i-N+1} \cdot \cdot \cdot w_{i-1})
\]
Application: Language Modeling

Training

$N$ gram approximation:

$$P(w_1 \cdots w_c) = \prod_{i=1}^{k} P(w_i \mid w_{i-N+1} \cdots w_{i-1})$$

Maximum likelihood estimates of component $N$ gram probabilities:

$$P(w_i \mid w_{i-N+1} \cdots w_{i-1}) \approx \frac{c(w_{i-N+1} \cdots w_i)}{c(w_{i-N+1} \cdots w_{i-1})}$$

To start:

$$P(w_1 \mid w_{-N} \cdots w_0) = P(w_1 \mid \underbrace{\cdots \cdots}_{N-1 \text{ times}})$$
Let it be when it is mine to be sure
let it be when it is mine when it is
mine let it be to be sure when it is
mine to be sure let it be let it be let
it be to be sure let it be to be sure
when it is mine to be sure let it to
be sure when it is mine let it be to
be sure let it be to be sure to be
sure let it be to be sure let it be to
be sure let it be mine to be sure let
it be to be sure to be mine to be
sure to be mine to be sure to be
mine let it be to be mine let it be
to be sure to be mine to be sure let
it be to be mine let it be to be sure
let it be to be sure to be sure let it
to be sure mine to be sure let it be
mine to let it be to be sure to let it
be mine when to be sure when to
be sure to let it to be sure to be
mine.

Gertrude Stein,
An Acquaintance With Description, 1929

Application: Language Modeling

Example

<table>
<thead>
<tr>
<th>Word Count</th>
<th>1-gram MLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>be</td>
<td>62 .276</td>
</tr>
<tr>
<td>to</td>
<td>41 .182</td>
</tr>
<tr>
<td>it</td>
<td>33 .147</td>
</tr>
<tr>
<td>sure</td>
<td>31 .138</td>
</tr>
<tr>
<td>let</td>
<td>27 .120</td>
</tr>
<tr>
<td>mine</td>
<td>17 .076</td>
</tr>
<tr>
<td>when</td>
<td>8 .036</td>
</tr>
<tr>
<td>is</td>
<td>6 .027</td>
</tr>
<tr>
<td>total</td>
<td>225</td>
</tr>
</tbody>
</table>

Division of Engineering and Applied Sciences
Harvard University
Let it be when it is mine to be sure
let it be when it is mine when it is
mine let it be to be sure when it is
mine to be sure let it be let it be let
it be to be sure let it be to be sure
when it is mine to be sure let it to
be sure when it is mine let it be to
be sure let it be to be sure to be
sure let it be to be sure let it be to
be sure let it be mine to be sure let
it be to be sure to be mine to be
sure to be mine to be sure to be
mine let it be to be mine let it be
to be sure to be mine to be sure let
it be to be mine let it be to be sure
let it be to be sure to be sure let it
to be sure mine to be sure let it be
mine to let it be to be sure to let it
be mine when to be sure when to
be sure to let it to be sure to be
mine.

—Gertrude Stein,
An Acquaintance With Description, 1929

<table>
<thead>
<tr>
<th>Trigram</th>
<th>MLE Prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>it be be</td>
<td>0</td>
</tr>
<tr>
<td>it be is</td>
<td>0</td>
</tr>
<tr>
<td>it be it</td>
<td>0</td>
</tr>
<tr>
<td>it be let</td>
<td>0.083</td>
</tr>
<tr>
<td>it be mine</td>
<td>0.125</td>
</tr>
<tr>
<td>it be sure</td>
<td>0</td>
</tr>
<tr>
<td>it be to</td>
<td>0.708</td>
</tr>
<tr>
<td>it be when</td>
<td>0.083</td>
</tr>
</tbody>
</table>
Application: Language Modeling

WFSA for N gram Models

One state per \( N+1 \) gram conditioning context

Transitions among states to change context

Start state is \( \langle \vdash^{N-1} \rangle \)

Unigra

Bigra
Application: Language Modeling

WFSA for N-gram Models

Trigra
Application: Language Modeling

WFSA for N gram Models

\[ P(abbb) = P(a \mid \Downarrow \Downarrow) \cdot P(b \mid \Downarrow a) \cdot P(b \mid ab) \cdot P(b \mid bb) \]
Application: Language Modeling

**Smoothing**

**Training corpus:**
- \textit{abaab}
- \textit{babb}

**Test corpus:**
- \textit{aab}
Application: Language Modeling

Why Smoothing?

<table>
<thead>
<tr>
<th>Possible</th>
<th>Attested</th>
<th>Sparsity</th>
</tr>
</thead>
<tbody>
<tr>
<td>trigrams</td>
<td>584</td>
<td>39</td>
</tr>
<tr>
<td>bigrams</td>
<td>72</td>
<td>22</td>
</tr>
<tr>
<td>unigrams</td>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>

Let it be when it is mine to be sure let it be when it is mine when it is mine let it be to be sure when it is mine to be sure let it be let it be let it be to be sure let it be to be sure when it is mine to be sure let it to be sure when it is mine let it be to be sure let it be to be sure to be sure let it be to be sure let it be to be sure let it be to be sure let it be to be sure let it be to be sure let it be to be sure let it be to be sure let it be to be sure let it be to be sure let it be to be sure let it be to be sure let it be to be sure let it be to be sure let it be to be sure let it be to be sure when to be sure when to be sure to let it to be sure to be mine.

— Gertrude Stein,
*An Acquaintance With Description*, 1929
Application: Language Modeling

Smoothing

Smoothing involves redistributing probability from high probability events to low probability ones.

First, reserve some probability mass
  • Add delta smoothing

Then, redistribute held out mass to unseen events
  • Katz back off
Application: Language Modeling

Effect of Smoothing

<table>
<thead>
<tr>
<th>Trigram</th>
<th>Unsmoothed</th>
<th>Smoothed</th>
</tr>
</thead>
<tbody>
<tr>
<td>it be be</td>
<td>0</td>
<td>0.0066</td>
</tr>
<tr>
<td>it be is</td>
<td>0</td>
<td>0.0006</td>
</tr>
<tr>
<td>it be it</td>
<td>0</td>
<td>0.0035</td>
</tr>
<tr>
<td>it be let</td>
<td>0.083</td>
<td>0.0938</td>
</tr>
<tr>
<td>it be mine</td>
<td>0.125</td>
<td>0.1250</td>
</tr>
<tr>
<td>it be sure</td>
<td>0</td>
<td>0.1143</td>
</tr>
<tr>
<td>it be to</td>
<td>0.708</td>
<td>0.5625</td>
</tr>
<tr>
<td>it be when</td>
<td>0.083</td>
<td>0.0938</td>
</tr>
</tbody>
</table>
Application: Language Modeling

Reserving Probability

Add delta smoothing

- add a fictitious “count” of $\delta$ to each $N$-gram

$$P(w_N \mid w_1 \cdots w_{N-1}) \approx \frac{c(w_1 \cdots w_N)}{c(w_1 \cdots w_{N-1})}$$
Application: Language Modeling

Reserving Probability

Add delta smoothing

- add a fictitious “count” of $\delta$ to each $N$-gram

\[
\tilde{P}(w_N \mid w_1 \cdots w_{N-1}) = \frac{c(w_1 \cdots w_N) + \delta}{c(w_1 \cdots w_{N-1}) + \delta V}
\]
Application: Language Modeling

Reserving Probability

Add delta smoothing

- add a fictitious “count” of $\delta$ to each $N$ gram

$$\tilde{P}(w_N \mid w_1 \cdots w_{N-1}) = \frac{c(w_1 \cdots w_N) + \delta}{c(w_1 \cdots w_{N-1}) + \delta V}$$

- total held out probability

$$\tilde{P}_{\text{held}}(w_1 \cdots w_{N-1}) = 1 - \sum_{w_N : c(w_1 \cdots w_N) > 0} \tilde{P}(w_N \mid w_1 \cdots w_{N-1})$$
### Application: Language Modeling

**Reserving Probability**

<table>
<thead>
<tr>
<th>trigram</th>
<th>count</th>
<th>unsmoothed</th>
<th>$\tilde{P}_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>it be sure</td>
<td>0</td>
<td>0.0000</td>
<td></td>
</tr>
<tr>
<td>it be when</td>
<td>2</td>
<td>0.0833</td>
<td>0.0938</td>
</tr>
<tr>
<td>it be is</td>
<td>0</td>
<td>0.0000</td>
<td></td>
</tr>
<tr>
<td>it be it</td>
<td>0</td>
<td>0.0000</td>
<td></td>
</tr>
<tr>
<td>it be let</td>
<td>2</td>
<td>0.0833</td>
<td>0.0938</td>
</tr>
<tr>
<td>it be mine</td>
<td>3</td>
<td>0.1250</td>
<td>0.1250</td>
</tr>
<tr>
<td>it be to</td>
<td>17</td>
<td>0.7083</td>
<td>0.5625</td>
</tr>
<tr>
<td>it be be</td>
<td>0</td>
<td>0.0000</td>
<td></td>
</tr>
<tr>
<td>$\tilde{P}_{\text{held}}$</td>
<td></td>
<td></td>
<td>0.1250</td>
</tr>
<tr>
<td>$\tilde{P}_{\text{back}}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>total</td>
<td>24</td>
<td>1.0000</td>
<td>1.0000</td>
</tr>
</tbody>
</table>
Application: Language Modeling

Distributing Probability

Katz backoff

• probability of unseen $N$ gram is proportional to the probability for its suffix $N+1$ gram

\[
\hat{P}(w_N | w_1 \cdots w_{N-1}) = \begin{cases} 
\hat{P}(w_N | w_1 \cdots w_{N-1}) & \text{if } c(w_1 \cdots w_N) \neq 0 \\
\alpha(w_1 \cdots w_{N-1}) \hat{P}(w_N | w_2 \cdots w_{N-1}) & \text{otherwise}
\end{cases}
\]
Application: Language Modeling

Distributing Probability

\[
\hat{P}(w_N \mid w_1 \cdots w_{N-1}) = \begin{cases} 
\hat{P}(w_N \mid w_1 \cdots w_{N-1}) & \text{if } c(w_1 \cdots w_N) \neq 0 \\
\alpha(w_1 \cdots w_{N-1})\hat{P}(w_N \mid w_{2} \cdots w_{N-1}) & \text{otherwise}
\end{cases}
\]

\[
\tilde{P}_{\text{held}}(w_1 \cdots w_{N-1}) = 1 - \sum_{w_N: c(w_1 \cdots w_N) > 0} \hat{P}(w_N \mid w_1 \cdots w_{N-1})
\]

\[
\tilde{P}_{\text{back}}(w_1 \cdots w_{N-1}) = 1 - \sum_{w_N: c(w_1 \cdots w_N) > 0} \hat{P}(w_N \mid w_2 \cdots w_{N-1})
\]

\[
\alpha(w_1 \cdots w_{N-1}) = \frac{\tilde{P}_{\text{held}}(w_1 \cdots w_{N-1})}{\tilde{P}_{\text{back}}(w_1 \cdots w_{N-1})}
\]
Application: Language Modeling

Backoff WFSA

Unsmoothed WFSA
Application: Language Modeling

Backoff WFSA

Adding backoff
Application: Language Modeling

Backoff WFSA

Redundant paths
Application: Language Modeling
Backoff WFSA

Eliminating redundancies
Renormalizing backoff

Application:
Language Modeling

Backoff WFSA

\[
\tilde{P}_{\text{back}}(w_1 \cdots w_{N-1}) = 
1 - \sum_{w_N : c(w_1 \cdots w_N) > 0} \tilde{P}(w_N \mid w_2 \cdots w_{N-1})
\]

Renormalized bigram probs

trigram probs
Application: Language Modeling

Backoff WFSA

Incorporating bigram model

Division of Engineering and Applied Sciences
Harvard University
Application: Language Modeling

Backoff WFSA

trigram probs

let: $P_3(\text{let} \mid \text{it be})$

be let

be

mine

be to

be when

be be

be is

is

be it

be sure

Pushing forward

fully renormalized bigram probs

Fully renormalized bigram probs

be: $P \frac{P_{\text{hed}}(\text{it be}) P_2(\text{be} \mid \text{be})}{P_{\text{back}}(\text{it be})}$
Application: Language Modeling

Backoff WFSA

trigram probs

fully renormalized bigram probs

Epsilon removal
Application: Language Modeling

Distributing Probability

\[
\hat{P}(w_N | w_1 \cdots w_{N-1}) = \begin{cases} 
\hat{P}(w_N | w_1 \cdots w_{N-1}) & \text{if } c(w_1 \cdots w_N) \neq 0 \\
\alpha(w_1 \cdots w_{N-1}) \hat{P}(w_N | w_2 \cdots w_{N-1}) & \text{otherwise}
\end{cases}
\]

\[
\hat{P}_{\text{held}}(w_1 \cdots w_{N-1}) = 1 - \sum_{w_N : c(w_1 \cdots w_N) > 0} \hat{P}(w_N | w_1 \cdots w_{N-1})
\]

\[
\hat{P}_{\text{back}}(w_1 \cdots w_{N-1}) = 1 - \sum_{w_N : c(w_1 \cdots w_N) > 0} \hat{P}(w_N | w_2 \cdots w_{N-1})
\]

\[
\alpha(w_1 \cdots w_{N-1}) = \frac{\hat{P}_{\text{held}}(w_1 \cdots w_{N-1})}{\hat{P}_{\text{back}}(w_1 \cdots w_{N-1})}
\]
## Application: Language Modeling

### Sample Smoothing

<table>
<thead>
<tr>
<th>trigram</th>
<th>count</th>
<th>unsmoothed</th>
<th>$\hat{P}_3$</th>
<th>$\hat{P}_2$</th>
<th>$\hat{P}_2$ renorm</th>
<th>smoothed</th>
</tr>
</thead>
<tbody>
<tr>
<td>it be sure</td>
<td>0</td>
<td>0.0000</td>
<td>0.4571</td>
<td>0.4571</td>
<td>0.9138</td>
<td>0.1142</td>
</tr>
<tr>
<td>it be when</td>
<td>2</td>
<td>0.0833</td>
<td>0.0938</td>
<td>0.0429</td>
<td></td>
<td>0.0938</td>
</tr>
<tr>
<td>it be is</td>
<td>0</td>
<td>0.0000</td>
<td>0.0026</td>
<td>0.0026</td>
<td>0.0051</td>
<td>0.0006</td>
</tr>
<tr>
<td>it be it</td>
<td>0</td>
<td>0.0000</td>
<td>0.0141</td>
<td>0.0141</td>
<td>0.0282</td>
<td>0.0035</td>
</tr>
<tr>
<td>it be let</td>
<td>2</td>
<td>0.0833</td>
<td>0.0938</td>
<td>0.0429</td>
<td></td>
<td>0.0938</td>
</tr>
<tr>
<td>it be mine</td>
<td>3</td>
<td>0.1250</td>
<td>0.1250</td>
<td>0.1571</td>
<td></td>
<td>0.1250</td>
</tr>
<tr>
<td>it be to</td>
<td>17</td>
<td>0.7083</td>
<td>0.5625</td>
<td>0.2571</td>
<td></td>
<td>0.5625</td>
</tr>
<tr>
<td>it be be</td>
<td>0</td>
<td>0.0000</td>
<td>0.0265</td>
<td>0.0265</td>
<td>0.0529</td>
<td>0.0066</td>
</tr>
<tr>
<td>$\hat{P}_{\text{held}}$</td>
<td></td>
<td></td>
<td>0.1250</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\hat{P}_{\text{back}}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.5002</td>
</tr>
<tr>
<td>total</td>
<td>24</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0002</td>
<td>1.0000</td>
<td>1.0000</td>
</tr>
</tbody>
</table>

Division of Engineering and Applied Sciences
Harvard University
Weighted Transducers

Combining output and weighting

Examples:

• typing models
Weighted Transducers

Combining output and weighting

Examples:

- typing models
Weighted Transducers

Combining output and weighting

Examples:

• typing models
• abbreviation models

Drop all vowels after the first character
Drop all but one repeated consonants

if y cn rd ths, y cn gt a gd jb
Abbreviation Decoding

Language Model

<an> <example> of <NUM> words

Spelling Model

an_example_of_<NUM>_words

Compression Model

an_exmpl_of_<NUM>_wrds

Unknowns Model

an_exmpl_of_5_wrds
Demos
Summary

Weighted finite state transducers
• provide an elegant, uniform, formalism
• cover a vast range of low level natural language processing tasks
  • characterizable as string to string transformations
Generality based on properties such as
• composability closure under composition
• efficiency determinizability and minimizability, enabled by pushing
• weighting for choice
Tree Automata
Trees as Terms

Trees can be thought of as terms over a ranked alphabet $\mathcal{F}$, notated $T(\mathcal{F})$.

$\mathcal{F} = \{ S_2, NP_1, VP_2, V_1, Pat_0, Kim_0, saw_0 \}$

$$S(NP(Kim), VP(V(saw), NP(Pat)))$$
Incomplete Trees

Trees can be thought of as terms over a ranked alphabet $\mathcal{F}$, notated $T(\mathcal{F})$.

$\mathcal{F} = \{ S_2, NP_1, VP_2, V_1, Pat_0, Kim_0, saw_0 \}$

To express incomplete trees (with “holes”) we allow variables $\mathcal{X}$ at the leaves, notated $T(\mathcal{F}, \mathcal{X})$.

\[ S(x, VP(V(saw), y)) \]
Tree Definitions

The set of trees over a ranked alphabet \( \mathcal{F} \) and variables \( \mathcal{X} \), notated \( \mathcal{T}(\mathcal{F}, \mathcal{X}) \), is the smallest set such that

- **Nullary symbols at leaves:**
  
  \[ f \in \mathcal{T}(\mathcal{F}, \mathcal{X}) \text{ for all } f \in \mathcal{F} \text{ such that } \text{arity}(f) = 0; \]

- **Variables at leaves:**
  
  \[ x \in \mathcal{T}(\mathcal{F}, \mathcal{X}) \text{ for all } x \in \mathcal{X}; \]

- **Internal nodes:**
  
  \[ f(t_1, \ldots, t_p) \in \mathcal{T}(\mathcal{F}, \mathcal{X}) \text{ for all } p \geq 1 \text{ and } t_1, \ldots, t_p \in \mathcal{T}(\mathcal{F}, \mathcal{X}). \]

Ground trees:

\[
\mathcal{T}(\mathcal{F}) = \mathcal{T}(\mathcal{F}, \emptyset)
\]

Alphabet implicit:

\[
\mathcal{T}(\mathcal{X})
\]

\[ n \text{ numerically ordered variables: } \mathcal{X}_n = \{x_1, \ldots, x_n\} \]
Example:

**Propositional Formulae**

Propositional formulae: $\mathcal{T}(\mathcal{F}_{prop})$

$\mathcal{F}_{prop} = \{\land_2, \lor_2, \neg_1, \text{TRUE}_0, \text{FALSE}_0\}$

(Arities are given in the subscripts.)
Tree Definitions

Substitution

For a context $C \in \mathcal{T}(\mathcal{F}, \mathcal{X}_n)$ and a sequence of $n$ trees $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F})$, the substitution of $t_1, \ldots, t_n$ into $C$, notated $C[t_1, \ldots, t_n]$, is defined as follows:

$$(f_m(u_1, \ldots, u_m))[t_1, \ldots, t_n] = f_m(u_1[t_1, \ldots, t_n], \ldots, u_m[t_1, \ldots, t_n])$$

$$x_i[t_1, \ldots, t_n] = t_i$$
Strings as Trees

Ranked alphabet

- Vocabulary as unary symbols
- End marker # as sole nullary symbol

\[ \text{aaabbaa} \]

\[ a(a(a(b(b(a(#)))))) = \]
Finite State Derivations

A derivation:

\[
0 \text{aaabba}\# \rightarrow 0 \text{aabba}\#
\rightarrow 0 \text{bbab}\#
\rightarrow 1 \text{bbab}\#
\rightarrow c1\text{a}\#
\rightarrow c2\text{a}\#
\rightarrow c2\#
\rightarrow c\#
\]

Definition of recognition:
accept \(s : t\) if and only if 
\[
0s\# \rightarrow^* t\#
\]
Strings as Trees

A derivation:

$$0(a(a(b(b(a#)))))) \rightarrow 0(a(a(b(b(a#))))))$$
$$\rightarrow 0(a(b(b(a#))))$$
$$\rightarrow 0(b(b(a#))))$$
$$\rightarrow 1((b(b(a#))))$$
$$\rightarrow c(1(a#))$$
$$\rightarrow c(2(a#))$$
$$\rightarrow c(2(#))$$
$$\rightarrow c(#)

Definition of recognition:
accept $s : t$ if and only if
$$0(s) \rightarrow^* t$$
States as Unary Symbols

The set of upper trees over a ranked alphabet \( \mathcal{F} \), states \( \mathcal{Q} \), and variables \( \mathcal{X} \), notated \( \mathcal{T}_Q^U(\mathcal{F}, \mathcal{X}) \), is the set of trees \( q(t) \) where \( q \in \mathcal{Q} \) and \( t \in \mathcal{T}(\mathcal{F}, \mathcal{X}) \).

The set of lower trees over a ranked alphabet \( \mathcal{F} \), states \( \mathcal{Q} \), and variables \( \mathcal{X} \), notated \( \mathcal{T}_Q^L(\mathcal{F}, \mathcal{X}) \), is the smallest set of trees such that

**Nullary symbols at leaves:** \( f \in \mathcal{T}_Q^L(\mathcal{F}, \mathcal{X}) \) for all \( f \in \mathcal{F} \) such that \( \text{arity}(f) = 0 \);

**States over variables at leaves:** \( q(x) \in \mathcal{T}_Q^L(\mathcal{F}, \mathcal{X}) \) for all \( x \in \mathcal{X} \) and \( q \in \mathcal{Q} \);

**Internal nodes:** \( f(t_1, \ldots, t_p) \in \mathcal{T}_Q^L(\mathcal{F}, \mathcal{X}) \) for all \( p \geq 1 \) and \( t_1, \ldots, t_p \in \mathcal{T}_Q^L(\mathcal{F}, \mathcal{X}) \).
Examples

\[ \mathcal{F}_{fg} = \{ f_2, g_2, a_0, b_0, c_0 \} \]
\[ Q = \{ q_0, q_1 \} \]

\[ T^Q(\mathcal{F}_{fg}, X_3) \supset \{ q_0(x_3), q_1(f(g(x_1, x_2), x_3)), q_0(a) \} \]

\[ T_Q(\mathcal{F}_{fg}, X_3) \supset \{ q_0(x_3), f(g(q_0(x_1), q_0(x_2)), q_1(x_3)), a \} \]
Tree Linearity and Height

A tree $t \in T(\mathcal{X})$ is linear if and only if no variable in $\mathcal{X}$ occurs more than once in $t$.

The height of a tree $t$, notated $\text{height}(t)$, is defined as follows:

$$
\text{height}(x) = 0 \text{ for all } x \in \mathcal{X}
$$

$$
\text{height}(f) = 1 \text{ for all } f \in \mathcal{F} \text{ where } \text{arity}(f) = 0
$$

$$
\text{height}(f(t_1, \ldots, t_n)) = 1 + \max_{i=1}^{n} \text{height}(t_i)
$$

for all $f \in \mathcal{F}$ where $\text{arity}(f) = n \geq 1$

If the trees include states in $Q$, then we further define that $\text{height}(q(t)) = \text{height}(t)$ for all $q \in Q$ and all trees $t$. 

Simple Subclasses of Trees

The consecutively numbered linear upper trees of height 1

\[ T^Q \]

are of the form

\[ q(f_n(x_1, \ldots, x_n)) \]

The consecutively numbered linear lower trees of height 1

\[ T_Q \]

are of the form

\[ f_n(q(x_1), \ldots, q(x_n)) \]

Used as patterns matching a parent and its immediate children in the most general way.

\[ \begin{align*}
0(a(x)) & \rightarrow 0(x) \\
0(x) & \rightarrow 1(x) \\
1(b(b(x))) & \rightarrow c(1(x)) \\
1(x) & \rightarrow 2(x) \\
2(a(x)) & \rightarrow 2(x) \\
2(#) & \rightarrow #
\end{align*} \]
Tree Automata

A nondeterministic top-down tree automaton (NTTA) is a tuple \( \langle Q, \mathcal{F}, \Delta, q_0 \rangle \) where

- \( Q \) is a finite set of states;
- \( \mathcal{F} \) is a ranked alphabet;
- \( \Delta \in T_Q(\mathcal{F}, X_n) \times T_Q(\mathcal{F}, X_n) \) is a set of transitions;
- \( q_0 \in Q \) is a distinguished initial state.

We notate transitions

\[
q(f_n(x_1, \ldots, x_n)) \rightarrow f_n(q_1(x_1), \ldots, q_n(x_n))
\]
Tree Automaton Derivation

Given an NTTA $\langle Q, \mathcal{F}, \Delta, q_0 \rangle$ and two trees $t, t' \in \mathcal{T}(\mathcal{F})$, tree $t$ derives $t'$ in one step, notated $t \rightarrow t'$ if and only if there is a transition $u \rightarrow u' \in \Delta$ with $u, u' \in \mathcal{T}(\mathcal{F}, \mathcal{X}_n)$ and trees $C \in \mathcal{T}(\mathcal{F}, \mathcal{X}_1)$ and $u_1, \ldots, u_n \in \mathcal{T}(\mathcal{F})$, such that
\[ t = C[u[u_1, \ldots, u_n]] \]
and
\[ t' = C[u'[u_1, \ldots, u_m]] \].

A tree $t \in \mathcal{T}(\mathcal{F})$ is accepted by an NTTA just in case
\[ q_0(t) \rightarrow^* t \].

The tree language of an NTTA is the set of trees accepted by the NTTA.
Tree Automaton Derivation

\[ q(f_n(x_1, \ldots, x_n)) \rightarrow f_n(q_1(x_1), \ldots, q_n(x_n)) \]
Example NTTA

\[ q_0(f(x_1, x_2)) \rightarrow f(q_0(x_1), q_0(x_2)) \]
\[ q_0(a) \rightarrow a \]
\[ q_0(b) \rightarrow b \]
Example NTTA

\[ q_0(f(x_1, x_2)) \rightarrow f(q_0(x_1), q_0(x_2)) \]

\[ q_0(a) \rightarrow a \]
\[ q_0(b) \rightarrow b \]
Example NTTA

\[ q_0(f(x_1, x_2)) \rightarrow f(q_0(x_1), q_0(x_2)) \]

\[ q_0(a) \rightarrow a \]
\[ q_0(b) \rightarrow b \]
Example NTTA

\[ q_0(f(x_1, x_2)) \rightarrow f(q_0(x_1), q_0(x_2)) \]
\[ q_0(a) \rightarrow a \]
\[ q_0(b) \rightarrow b \]
Example:

**Recognizing True Formulae**

- \( q_t(\text{true}) \rightarrow \text{true} \)
- \( q_f(\text{false}) \rightarrow \text{false} \)
- \( q_t(\neg x_1) \rightarrow q_f(x_1) \)
- \( q_f(\neg x_1) \rightarrow q_t(x_1) \)
- \( q_t(x_1 \land x_2) \rightarrow q_t(x_1) \land q_t(x_2) \)
- \( q_f(x_1 \land x_2) \rightarrow q_t(x_1) \land q_f(x_2) \)
- \( q_f(x_1 \land x_2) \rightarrow q_f(x_1) \land q_t(x_2) \)
- \( q_f(x_1 \land x_2) \rightarrow q_f(x_1) \land q_f(x_2) \)
- \( q_t(x_1 \lor x_2) \rightarrow q_t(x_1) \lor q_t(x_2) \)
- \( q_t(x_1 \lor x_2) \rightarrow q_f(x_1) \lor q_f(x_2) \)
- \( q_t(x_1 \lor x_2) \rightarrow q_f(x_1) \lor q_t(x_2) \)
- \( q_f(x_1 \lor x_2) \rightarrow q_f(x_1) \lor q_f(x_2) \)

\[
q_t(\text{false} \lor \neg \text{false}) \\
\rightarrow q_f(\text{false}) \lor q_t(\neg \text{false}) \\
\rightarrow \text{false} \lor q_t(\neg \text{false}) \\
\rightarrow \text{false} \lor \neg q_f(\text{false}) \\
\rightarrow \text{false} \lor \neg \text{false}
\]
Top Down Tree Automata

A nondeterministic top-down tree automaton (NTTA) is a tuple $\langle Q, \mathcal{F}, \Delta, q_0 \rangle$ where

- $Q$ is a finite set of states;
- $\mathcal{F}$ is a ranked alphabet;
- $\Delta \in T^Q(\mathcal{F}, X_n) \times T^Q(\mathcal{F}, X_n)$ is a set of transitions;
- $q_0 \in Q$ is a distinguished initial state.

A tree $t \in T(\mathcal{F})$ is accepted by an NTTA just in case

$$q_0(t) \rightarrow^* t$$

The tree language of an NTTA is the set of trees accepted by the NTTA.
Bottom Up Tree Automata

A nondeterministic bottom-up tree automaton (NBTA) is a tuple \( \langle Q, F, \Delta, Q_f \rangle \) where

- \( Q \) is a finite set of states;
- \( F \) is a ranked alphabet;
- \( \Delta \in T_Q(F, X_n) \times T_Q(F, X_n) \) is a set of transitions;
- \( Q_f \subseteq Q \) is a distinguished set of final states.

A tree \( t \in \mathcal{T}(F) \) is accepted by an NBTA just in case

\[
t \rightarrow^* q(t)
\]

for some \( q \in Q_f \).

The tree language of an NBTA is the set of trees accepted by the NBTA.
Properties of Tree Automata

Characterize context free trees

Closure under

- left right reversal
- top down+bottom up reversal
- union
- substitution
- iterative substitution
- granularity
- epsilon removal: \( q(x) \rightarrow q'(x) \)
- determinization: bottom up only
Properties of Tree Automata

Determinization

- an automaton is deterministic if no two transitions share left hand side
- bottom up automata are determinizable
- top down automata are not

\[
q(f(x, x)) \rightarrow f(q_a(x), q_a(x)) \\
q(f(x, x)) \rightarrow f(q_b(x), q_b(x)) \\
q_a(a) \rightarrow a \\
q_b(b) \rightarrow b \\
\{f(a, a), f(b, b)\}
\]
Tree Transducers
Tree Transducers

Regarding tree transformations, results do not flow so easily. Several definitions are candidate for the label ‘tree transductions’, with (unrelated) properties. People will keep in mind how gracefully behaved rational word transductions (are).

, Raoult, 1992
Top Down Tree Automata

A nondeterministic top-down tree automaton (NTTA) is a tuple \( \langle Q, \mathcal{F}, \Delta, q_0 \rangle \) where

- \( Q \) is a finite set of states;
- \( \mathcal{F} \) is a ranked alphabet;
- \( \Delta \in T^Q(\mathcal{F}, \mathcal{X}_n) \times T^Q(\mathcal{F}, \mathcal{X}_n) \) is a set of transitions;
- \( q_0 \in Q \) is a distinguished initial state.

A tree \( t \in \mathcal{T}(\mathcal{F}) \) is accepted by an NTTA just in case

\[
q_0(t) \rightarrow^* t
\]

The tree language of an NTTA is the set of trees accepted by the NTTA.
A nondeterministic top-down tree transducer (NTTT) is a tuple \( \langle Q, F, \Delta, q_0 \rangle \) where

- \( Q \) is a finite set of states;
- \( F \) is a ranked alphabet;
- \( \Delta \in T^Q(F, X_n) \times T_Q(F, X_n) \) is a set of transitions;
- \( q_0 \in Q \) is a distinguished initial state.

The tree relation defined by an NTTT \( \langle Q, F, \Delta, q_0 \rangle \) is the set of all tree pairs \( \langle s, t \rangle \) such that

\[
q_0(s) \rightarrow^* t
\]
Example NTTT FST

A derivation:

\[
0(a(a(a(b(b(a(\#))))))) \rightarrow 0(a(a(b(b(a(\#)))))) \\
\rightarrow 0(a(b(b(a(\#)))))) \\
\rightarrow 0(b(b(a(\#)))) \\
\rightarrow 1((b(b(a(\#)))))) \\
\rightarrow c(1(a(\#)))) \\
\rightarrow c(2(a(\#)))) \\
\rightarrow c(2(\#)) \\
\rightarrow c(\#)
\]

Definition of recognition:

accept \( s : t \) if and only if

\[ 0(s) \rightarrow^* t \]
Example:

Concrete Syntax of Formulae

\[
\begin{align*}
q_0(\neg x) & \rightarrow un(\neg, q_{\land\lor}(x)) \\
q_0(x \land y) & \rightarrow bin(q_{\land\lor}(x), \land, q_{\lor}(y)) \\
q_0(x \lor y) & \rightarrow bin(q_{\lor}(x), \lor, q_0(y)) \\
q_0(\text{true}) & \rightarrow \text{true} \\
q_0(\text{false}) & \rightarrow \text{false} \\
q_{\land\lor}(\neg x) & \rightarrow un(\neg, q_{\land\lor}(x)) \\
q_{\land\lor}(x \land y) & \rightarrow par([, bin(q_{\land\lor}(x), \land, q_{\lor}(y)), ]) \\
q_{\land\lor}(x \lor y) & \rightarrow par([, bin(q_{\lor}(x), \lor, q_0(y)), ]) \\
q_{\land\lor}(\text{true}) & \rightarrow \text{true} \\
q_{\land\lor}(\text{false}) & \rightarrow \text{false} \\
q_{\lor}(\neg x) & \rightarrow un(\neg, q_{\land\lor}(x)) \\
q_{\lor}(x \land y) & \rightarrow bin(q_{\land\lor}(x), \land, q_{\lor}(y)) \\
q_{\lor}(x \lor y) & \rightarrow par([, bin(q_{\lor}(x), \lor, q_0(y)), ]) \\
q_{\lor}(\text{true}) & \rightarrow \text{true} \\
q_{\lor}(\text{false}) & \rightarrow \text{false}
\end{align*}
\]

Diagram:

```
un
  / \     \\
 par  bin
    / \   / \    \\
false / \ true / \  \\
```

Division of Engineering and Applied Sciences
Harvard University
Example:

Concrete Syntax of Formulae

\[
\begin{align*}
q_0(\neg x) & \quad \rightarrow \text{un}(\neg, q_\land(x)) \\
q_0(x \land y) & \quad \rightarrow \text{bin}(q_\land(x), \land, q_\lor(y)) \\
q_0(x \lor y) & \quad \rightarrow \text{bin}(q_\lor(x), \lor, q_0(y)) \\
q_0(\text{true}) & \quad \rightarrow \text{true} \\
q_0(\text{false}) & \quad \rightarrow \text{false} \\
q_\land(\neg x) & \quad \rightarrow \text{un}(\neg, q_\land(x)) \\
q_\land(x \land y) & \quad \rightarrow \text{par}([, \text{bin}(q_\land(x), \land, q_\lor(y)), ]) \\
q_\land(x \lor y) & \quad \rightarrow \text{par}([, \text{bin}(q_\lor(x), \lor, q_0(y)), ]) \\
q_\land(\text{true}) & \quad \rightarrow \text{true} \\
q_\land(\text{false}) & \quad \rightarrow \text{false} \\
q_\lor(\neg x) & \quad \rightarrow \text{un}(\neg, q_\land(x)) \\
q_\lor(x \land y) & \quad \rightarrow \text{bin}(q_\land(x) \text{ fringe}(T(\neg(\land(\text{true, false})))) \\
q_\lor(x \lor y) & \quad \rightarrow \text{par}([, \text{bin}(q_\lor(x), \lor, q_0(y)), ]) \\
q_\lor(\text{true}) & \quad \rightarrow \text{true} \\
q_\lor(\text{false}) & \quad \rightarrow \text{false}
\end{align*}
\]
Example:

**Evaluation of Formulae**

- **true** → $q_T(\text{true})$
- **false** → $q_F(\text{false})$
- $\neg q_T(x_1)$ → $q_F(\text{false})$
- $\neg q_F(x_1)$ → $q_T(\text{true})$
- $q_T(x_1) \land q_T(x_2)$ → $q_T(\text{true})$
- $q_T(x_1) \land q_F(x_2)$ → $q_F(\text{false})$
- $q_F(x_1) \land q_T(x_2)$ → $q_F(\text{false})$
- $q_F(x_1) \land q_F(x_2)$ → $q_F(\text{false})$
- $q_T(x_1) \lor q_T(x_2)$ → $q_T(\text{true})$
- $q_T(x_1) \lor q_F(x_2)$ → $q_T(\text{true})$
- $q_F(x_1) \lor q_T(x_2)$ → $q_T(\text{true})$
- $q_F(x_1) \lor q_F(x_2)$ → $q_F(\text{false})$
Properties and Complexities of Tree Transducers

**Linear**
- no repeated variables on right hand side
- nonlinearity generates exponential transformations

**Fine-grained**
- rotations, e.g.
- structure elimination requires nonlinearity or pushing

**Deterministic**
- not possible in general

**Non-Erasing**
- all variables appear on right hand side

**Invertible**
- requires linearity
Nonlinearity Generates Exponential Transductions

Generating perfect binary trees:

\[ q(f(x)) \rightarrow g(q(x), q(x)) \]
\[ q(a) \rightarrow a \]

\[ q(f(a)) \rightarrow g(q(a), q(a)) \rightarrow^* g(a, a) \]
\[ q(f(f(a))) \rightarrow g(q(f(a)), q(f(a))) \rightarrow^* g(g(a, a), g(a, a)) \]
\[ q(f(f(f(a)))) \rightarrow^* g(g(g(a, a), g(a, a)), g(g(a, a), g(a, a))) \]
\[ |q(f^n(a))| = 2^n - 1 \]

Exponential growth implies no composition closure
Why Rotations?

Consider

- I like Mary.
- Marie gefällt mir.
Expressing Rotations

\[
q(f(g(x, y), z)) \rightarrow f(q(x), g(q(y), q(z)))
\]

\[
q(f(x_{gxy}, z)) \rightarrow f(q_1(x_{gxy}), g(q_2(x_{gxy}), q(z)))
\]

\[
q_1(g(x, y)) \rightarrow x
\]

\[
q_2(g(x, y)) \rightarrow y
\]

requires
nonlinearity
Why Global Rotations?

Consider

- Dann wird der Doktor dem patienten die Pille geben
- Then the doctor will give the patient the pill

Consider

- \( VP \)
  - \( VP \)
    - \( VP \)
      - \( V \)
        - give
      - \( NP \)
        - the patient
  - \( NP \)
    - the pill

- \( VP \)
  - \( NP \)
    - dem Patienten
  - \( VP \)
    - die Pille
    - \( V \)
      - geben
No Global Rotations

cf. macro tree transducers
Determinization

Determinization of transducers fails even if underlying automata are deterministic.

The goal:
\[
\begin{align*}
f(x, y) &\Rightarrow g(a, x) \\
f(x, y) &\Rightarrow f(x, y) \quad \text{for } x \neq aa \Rightarrow a
\end{align*}
\]

A nondeterministic solution:
\[
\begin{align*}
q(f(x, y)) &\rightarrow g(q_a(x), q(y)) \\
q(f(x, y)) &\rightarrow f(q_{\neg a}(x), q(y)) \\
q(a) &\rightarrow a \\
q_a(a) &\rightarrow a \\
q_{\neg a}(f(x, y)) &\rightarrow g(q_a(x), q(y))
\end{align*}
\]
Summary

Linearity would be helpful
- closure under composition
- no exponential growth
- invertibility

But linearity is insufficient...
- no local rotation
and even nonlinear transducers are insufficient
- global rotations
- fringe
Extended Transducers: Bimorphisms
String Homomorphisms

Given alphabets $\Sigma$ and $\Gamma$ and a finite function $h_0 : \Sigma \rightarrow \Gamma^*$, the homomorphism $h : \Sigma^* \rightarrow \Gamma^*$ is defined as the unique function extending $h_0$ such that for all strings $s, t \in \Sigma^*$, $h(st) = h(s)h(t)$.

A string homomorphism is $\epsilon$-free if $h(s) = \epsilon$ only when $s = \epsilon$.

A string bimorphism is a triple $\langle h_{in}, L, h_{out} \rangle$ where $L$ is a regular language and $h_{in}$ and $h_{out}$ are homomorphisms.
String Bimorphisms

Finite state transducers and string bimorphisms are equivalent.

Proof sketch for one direction: Given a FST $\langle Q, \Sigma, \Gamma, \Delta, q_0 \rangle$, construct FSA $A_L = \langle Q, \Sigma^* \times \Gamma^*, \Delta', q_0 \rangle$ where $\Delta'$ contains transitions of the form

$$q \langle s, t \rangle \rightarrow q'$$

for each transition in $\Delta$ of the form

$$q s \rightarrow t q'$$

Construct homomorphism $h_{in} : \Sigma^* \times \Gamma^* \rightarrow \Sigma$ extending

$$h_{in}(\langle s, t \rangle) = s$$

and $h_{out} : \Sigma^* \times \Gamma^* \rightarrow \Gamma$ extending

$$h_{in}(\langle s, t \rangle) = t$$

The required bimorphism is $\langle h_{in}, L(A_L), h_{out} \rangle$. 

Division of Engineering and Applied Sciences
Harvard University
String Bimorphisms

Finite state transducers and string bimorphisms are equivalent.

\[ \langle h_{in}, A^* D B^* E C^*, h_{out} \rangle \]
Tree Homomorphisms

Let $h_F : \mathcal{F} \rightarrow \mathcal{T}(\mathcal{F}', \mathcal{X})$ be a function mapping each $f \in \mathcal{F}$ of arity $n$ to a tree $h_F(f) : \mathcal{T}(\mathcal{F}', X_n)$. The tree homomorphism extending $h_F$ is the function $h : \mathcal{T}(\mathcal{F}) \rightarrow \mathcal{T}(\mathcal{F}')$ such that for all $n$ and all $f$ of arity $n$

$$h(f(t_1, \ldots, t_n)) = h_F(f)[h(t_1), \ldots, h(t_n)]$$

Equivalent to one-state tree transducers
Example: Perfect Binary Trees

\[
\begin{align*}
f & \mapsto \quad g \\
x & \mapsto \quad x \\
a & \mapsto \quad a
\end{align*}
\]
Tree Bimorphisms

A tree bimorphism over input alphabet \( \Sigma \) and output alphabet \( \Gamma \) is a triple \( \langle h_i, L, h_o \rangle \) where \( L \) is a rational tree language over an alphabet \( \Delta \) and \( h_i : T(\Delta) \rightarrow T(\Sigma) \) and \( h_o : T(\Delta) \rightarrow T(\Gamma) \) are tree homomorphisms.

A tree bimorphism generates a tree relation \( R \) from \( T(\Sigma) \) to \( T(\Gamma) \) as follows: \( R(s, t) \) holds just in case there is a tree \( d \in L \) such that \( h_i(d) = s \) and \( h_o(d) = t \).

Equivalently,

\[
R = h_i^{-1} \circ L \circ h_o
\]
Restricting Bimorphisms

\( h_{perf}^{-1} \circ \mathcal{T} \circ h_{id} \)
Restricting Homomorphisms

Linear
- no repeated variables

Complete
- no dropped variables

Epsilon free
- some structure on output

Symbol to symbol
- output of height 1
- implies epsilon free

Delabeling
- = linear complete symbol to symbol
Intuition for Restriction

\[ T^Q(\mathcal{F}, \mathcal{X}_n) \times T^Q(\mathcal{F}, \mathcal{X}_n) \]

\[ T(\mathcal{F}, \mathcal{X}_n) \quad (\approx \mathcal{F}) \]

delabeling \( LCS \) \quad arbitrary morphism \( M \)

\[ B(LCS, M) \]
Bimorphism Characterization

The class of bottom up tree transductions is equivalent to the relations defined by tree bimorphisms where the first homomorphism is a delabeling.

If the homomorphisms are linear epsilon free, complete, the bimorphism characterizes a linear resp., epsilon free, complete transduction.

This asymmetry explains, e.g., lack of invertibility.
Types of Bimorphisms

\[ B(x, y) \] bimorphisms with homomorphisms of type \( x \) and \( y \)

- \( M \) any homomorphism
- \( L \) linear
- \( C \) complete
- \( F \) epsilon free
- \( D \) delabeling

\[ B D, M \] equivalent to tree transducers
Regaining Symmetry

$B$ $M, M$
- very powerful; composition is Turing equivalent

$B$ $L, L$
- expands input power; contracts output power
- not closed under composition
- $B(L, L) \subset B(L, L)^2 \subset B(L, L)^3 \subset B(L, L)^4 = B(L, L)^5$

$B$ $LC, LC$
- = synchronous tree substitution grammars

$B$ $LCF, LCF$
- $B(LCF, LCF) \subset B(LCF, LCF)^2 = B(LCF, LCF)^3$
Synchronous Grammars

Context free grammars as tree substitution

NP
Kim

S
NP
VP
likes
VP
V
NP
Sandy

S
NP
VP
Kim

NP
Kim

NP
VP
V
NP
Synchronous Context Free Grammars

Diagram showing the structure of sentences in a synchronous context-free grammar.
Synchronous Context Free Grammars
Problems With SCFG

Domain of locality is too small

I like Mary

I like Marie

gefällt mir
Tree Substitution Grammars

Expands domain of locality to *elementary tree*.

```
NP  NP  S  NP  VP  NP
  Kim Kuchen S  NP  VP
       V   isst
       NP

S  NP  VP
  NP  V  NP
    NP  isst
       NP
         Kim
          Kuchen
```
Tree Substitution Grammars

Expands domain of locality to *elementary tree*.

No additional expressive power over CFG.
Synchronous Tree Substitution Grammars

\[ S \rightarrow NP \rightarrow V \rightarrow NP \]
\[ S \rightarrow NP \rightarrow V \rightarrow NP \]

\[ NP \rightarrow \text{eats} \]
\[ NP \rightarrow \text{isst} \]

\[ NP \rightarrow \text{Kim} \]
\[ NP \rightarrow \text{cake} \]
\[ NP \rightarrow \text{Kuchen} \]
Synchronous Tree Substitution Grammars
The STSG Payoff

eats
isst
likes
gefällt
STSG as Bimorphism

$hi(\alpha_1) = S(x, VP(V(eats), y))$
$ho(\alpha_1) = S(x, VP(V(isst), y))$
$hi(\alpha_2) = NP(cake)$
$ho(\alpha_2) = NP(Kuchen)$
$hi(\alpha_3) = NP(Kim)$
$ho(\alpha_3) = NP(Kim)$
$hi(\alpha_4) = S(x, VP(V(likes), y))$
$ho(\alpha_4) = S(y, VP(V(gefällt), x))$
$hi(\alpha_5) = NP(I)$
$ho(\alpha_5) = NP(mir)$

NB: linear, complet
Substitution and Adjunction

substitutio#

adjunctio#
Tree Adjoining Grammars

Elementary trees extend domain of locality
Combination by substitution and adjunction
Trans context free power

- CFG: \( a^n b^n \)
- TAG: \( a^n b^n c^n d^n \)

Examples of adjunction:

- *Kim likes cake* → *Kim really likes cake*
- *the cake* → *the chocolate cake*
  → *the chocolate cake that I baked*
Synchronous Tree Adjoining Grammars

Proposed for transductions to characterize

- semantics
- natural language generation
- machine translation
Extended Transducers: Macro Tree Transducers and Deterministic Tree Rewriting
More Powerful Tree Transducers
Corpus Normalization

Identifying Complements and Adjuncts in the Penn Treebank

We add the “C” suffix to all non terminals in training data that satisfy the following conditions:

1. The non terminal must be: 1 an NP, SBAR, or S whose parent is an S; 2 an NP, SBAR, S, or VP whose parent is a VP; or 3 an S whose parent is an SBAR.
2. The non terminal must not have one of the following semantic tags: ADV, VOC, BNF, DIR, EXT, LOC, MNR, TMP, CLR or PRP. ...

In addition, the first child following the head of a prepositional phrase is marked as a complement.

...

Punctuation#

This section describes our treatment of “punctuation” in the model, where “punctuation” is used to refer to words tagged as a comma or colon. ...

Our first step, for consistency, is to raise punctuation as high in the parse trees as possible. Punctuation at the beginning or end of sentences is removed from the training/test data altogether...

Collins, Head Driven Statistical Models for Natural Language Parsing, 1999
More Corpus Normalization

Table 3: Transformations from N-ary to binary branching structures

,  Goodman, Efficient algorithms for parsing the DOP model, 1996

Note the implicit need for global rotations.
Programming Rotation

\[\text{rotate}(x) = \text{rotate}(x, \#)\]
\[\text{rotate}(\#, x) = x\]
\[\text{rotate}(f(x, y), z) = \text{rotate}(x, f(y, z))\]

\[\text{rotate}(f(f(f(\#, a), b), c))\]
\[= \text{rotate}(f(f(f(\#, a), b), c), \#)\]
\[= \text{rotate}(f(f(\#, a), b), f(c, \#))\]
\[= \text{rotate}(f(\#, a), f(b, f(c, \#)))\]
\[= \text{rotate}(\#, f(a, f(b, f(c, \#))))\]
\[= f(a, f(b, f(c, \#)))\]
Macro Tree Transducers

A macro tree transducer (MTT) is a tuple \(\langle Q, F, \Delta, q_0 \rangle\) where

- \(Q\) is a ranked alphabet of states;
- \(F\) is a ranked alphabet;
- \(\Delta \in \bigcup_{r \geq 1, q(r) \in Q, k \geq 0, f(k) \in F} \mathcal{T}_{\text{mlhs}}(q(r), f(k), X_k, Y_r) \times \mathcal{T}_{\text{mrhs}}(Q, F, X_k, Y_r)\) is a set of transitions;
- \(t_0 \in \mathcal{T}_{\text{mrhs}}(Q, F, X_1, \emptyset)\) is a distinguished initial tree.
Macro Tree Transducers

The set of macro tree transducer left-hand sides over \( r \)-ary state \( q \), \( k \)-ary symbol \( f \), variables \( X_k \), and parameters \( Y_r \), notated \( T_{mlhs}(q^{(r)}, f^{(k)}, X_k, Y_r) \), is the singleton set comprising the tree of the form

\[
q(f(x_1, \ldots, x_k), y_1, \ldots, y_r)
\]

The set of macro tree transducer right-hand sides over states \( Q \), alphabet \( \mathcal{F} \), variables \( X_k \), and parameters \( Y_r \), notated \( T_{mrhs}(Q, \mathcal{F}, X_k, Y_r) \), is the smallest set of trees in \( T(Q \cup \mathcal{F}, X_k \cup Y_r) \) such that

1. \( Y_r \subseteq T_{mrhs}(Q, \mathcal{F}, X_k, Y_r) \)
2. for all \( k \geq 0 \) and \( f \in \mathcal{F}^{(k)} \) and \( t_1, \ldots, t_k \in T_{mrhs}(Q, \mathcal{F}, X_k, Y_r) \),
   \[
f(t_1, \ldots, t_k) \in T_{mrhs}(Q, \mathcal{F}, X_k, Y_r)
   \]
3. for all \( r \geq 1 \) and \( q \in Q^{(r)} \) and \( x_i \in X_k \) and
   \[
t_1, \ldots, t_{r-1} \in T_{mrhs}(Q, \mathcal{F}, X_k, Y_r),
   \]
   \[
q(x_i, t_1, \ldots, t_{r-1}) \in T_{mrhs}(Q, \mathcal{F}, X_k, Y_r)
   \]
MTT Examples:

Global Rotation

\[ \text{rotate}(x) = \text{rotate}(x, \#) \]
\[ \text{rotate}(\#, x) = x \]
\[ \text{rotate}(f(x, y), z) = \text{rotate}(x, f(y, z)) \]

\[ q_0(x_1) \rightarrow q_r(x_1, \#) \]
\[ q_r(\#, y_1) \rightarrow y_1 \]
\[ q_r(f(x_1, x_2), y_1) = q_r(x_1, f(x_2, y_1)) \]
MTT Examples:

Local Rotation

\[
q(f(x_1, x_2)) \rightarrow q'(x_1, x_2)
\]
\[
q'(f(x_1, x_2), y_1) \rightarrow f(q(x_1), f(q(x_2), q(y_1)))
\]
\[
q(a) \rightarrow a
\]
\[
q(b) \rightarrow b
\]

\[
q(f(f(a, f(f(b, a), a)), b))
\rightarrow q'(f(a, f(f(b, a), a)), b)
\rightarrow f(q(a), f(q(f(f(b, a), a)), q(b)))
\rightarrow f(a, f(q'(f(b, a), a), b))
\rightarrow f(a, f(f(q(b), f(q(a), q(a))), b))
\rightarrow f(a, f(f(b, f(a, a)), b))
\rightarrow f(a, f(f(b, f(a, a)), b))
\]
MTT Examples: Reversal

$q_0(x_1) \rightarrow q(x_1, \#)$
$q(f(x_1, x_2), y) \rightarrow q(x_2, f(x_1, y))$
$q(\#, y) \rightarrow y$

$q_0(f(a, f(b, f(c, \#))))$
$\rightarrow q(f(a, f(b, f(c, \#))), \#)$
$\rightarrow q(f(b, f(c, \#)), f(a, \#))$
$\rightarrow q(f(c, \#), f(b, f(a, \#)))$
$\rightarrow q(\#, f(c, f(b, f(a, \#))))$
$\rightarrow f(c, f(b, f(a, \#)))$
MTT Examples:

Reversal

\[ q_0(x_1) \rightarrow q(x_1, \#) \]
\[ q(f(x_1, x_2), y) \rightarrow q(x_1, q(x_2, y)) \]
\[ q(a, y) \rightarrow f(a, y) \]
\[ q(b, y) \rightarrow f(b, y) \]

\[ q_0(f(f(a, b), f(b, a))) \]
\[ \rightarrow q(f(f(a, b), f(b, a)), \#) \]
\[ \rightarrow q(f(a, b), q(f(b, a), \#)) \]
\[ \rightarrow q(f(a, b), q(b, q(a, \#))) \]
\[ \rightarrow q(a, q(b, q(b, q(a, \#)))) \]
\[ \rightarrow^4 f(a, f(b, f(b, f(a, \#)))) \]
MTT Examples:

Frontier

\[ q_0(x_1) \rightarrow q(x_1, \#) \]
\[ q(f(x_1, x_2), y) \rightarrow q(x_1, q(x_2, y)) \]
\[ q(a, y) \rightarrow f(a, y) \]
\[ q(b, y) \rightarrow f(b, y) \]

\[ q_0(f(f(a, b), f(b, a))) \]
\[ \rightarrow q(f(f(a, b), f(b, a)), \#) \]
\[ \rightarrow q(f(a, b), q(f(b, a), \#)) \]
\[ \rightarrow q(f(a, b), q(b, q(a, \#))) \]
\[ \rightarrow q(a, q(b, q(b, q(a, \#)))) \]
\[ \rightarrow^4 f(a, f(b, f(b, f(a, \#)))) \]
Deterministic Tree Rewriting

Formalism designed for specifying speech command and control systems for Kurzweil Lernout and Hauspie Scansoft

Basis for natural language like voice commands in

- Kurzweil VoiceXpress
- Lernout and Hauspie VoiceXpress
- Scansoft Dragon NaturallySpeaking
Overview

Cascade of bimorphisms: $B L, M$

Where $h_{in}^{-1}$ is ambiguous, it is determinized by explicit ordering.

Extended to unranked alphabet through sequence variables

Succinct notation for rewrite rules
Rewrite Rules

\[ \langle \text{pattern} \rangle \implies \langle \text{result} \rangle \]

\[ \text{NP}(\text{Det}, N) \implies \text{Det}(N) \]

\[ q(np(x_{\text{det}}, x_n)) \rightarrow det(q(x_n)) \]

- Variables/nonterminals uppercase or _
- Terminals lowercase
- Recursive rewriting implicit
- Nonterminals play dual role:
  - variables
  - node labels
Rewrite Rules

\[ \text{pattern} \implies \text{result} \]

\[
\begin{align*}
\text{NP}(\text{Det}, N) & \implies \text{Det}(N) \\
q(np(x_{\text{det}}, x_n)) & \implies \text{det}(q(x_n))
\end{align*}
\]

\[
\begin{align*}
\text{Units(lines)} & \implies \text{line} \\
q(\text{units(lines)}) & \implies \text{line}
\end{align*}
\]

\[
\begin{align*}
\text{Units(_unit)} & \implies _\text{unit} \\
q(\text{units(x\_unit)}) & \implies q(x_{\text{unit}})
\end{align*}
\]

\[
\begin{align*}
\text{Command(move, down, Number, Units)} & \implies \text{Move(down, Number, Units)} \\
q(\text{command(move, down, x\_number, x\_units)}) & \implies \text{move(down, q(x\_number), q(x\_units))}
\end{align*}
\]
Example

\[
\text{Command(move, down, Number, Units)}
\quad \Rightarrow \quad \text{Move(down, Number, Units)}
\]

\[
\text{Number(_n)} = _n
\]

\[
\text{Units(lines)} \Rightarrow \text{line}
\]

\[
\text{Units(_unit)} \Rightarrow _\text{unit}
\]

\[
\text{Command(move, down, Number(3), Units(lines))}
\quad \Rightarrow \ast \quad \text{Move(down, 3, line)}
\quad \Rightarrow \ast \quad \text{Move(down, 3, lines)}
\]
Extended Example

Number Name Normalization

// Front end grammar

/* Starting nonterminal is NatNum
   NatNum  covers natural numbers 0 through 10^12 - 1
   NatNumX covers natural numbers 1 through 10^X - 1
   NatDigit covers natural digits 1 through 9
   NatLeadDig similarly
   NatTeen covers 10 through 19
   NatTy  covers the multiples of 10, 20 through 90
*/

NatNum   --> zero | NatNum12
NatNum12 --> NatNum3 | NatNum3 billion NatNum9
NatNum9  --> NatNum3 | NatNum3 million NatNum6
NatNum6  --> NatNum3 | NatNum3 thousand NatNum3
NatNum3  --> NatNum2 | NatLeadingDigit hundred {and} NatNum2
NatNum2  --> NatDigit | NatTeen | NatTy NatDigit
NatDigit --> one | two | three | four | five
          | six  | seven | eight | nine
NatLeadDig --> a | NatDigit
NatTeen  --> ten | eleven | twelve | thirteen | fourteen
          | fifteen | sixteen | seventeen | eighteen | nineteen
NatTy    --> twenty | thirty | forty | fifty
          | sixty | seventy | eighty | ninety
Extended Example

Number Name Normalization

//.....................
Pass "compute expression"

// Rewrites natural numbers into an arithmetic expression
// tree that computes the corresponding numeric value.

NatNum3(NatNum2) ==> NatNum2
NatNum3(NatLeadingDigit, hundred, _, NatNum2)
  ==> Plus(NatNum2, Times(NatLeadDig,
      Exp(10, 2)))

NatNum2(NatDigit) ==> NatDigit
NatNum2(NatTy, NatDigit) ==> Plus(NatTy, NatDigit)
NatNum2(NatTeen) ==> NatTeen
Extended Example

Number Name Normalization

NatDigit(one) ==> 1
NatDigit(two) ==> 2
NatDigit(three) ==> 3
...
NatDigit(nine) ==> 9

NatLeadDig(NatDigit) ==> NatDigit
NatLeadDig(a) ==> 1

NatTeen(ten) ==> 10
NatTeen(eleven) ==> 11
NatTeen(twelve) ==> 12
...
NatTeen(nineteen) ==> 19

NatTy(twenty) ==> 20
NatTy(thirty) ==> 30
NatTy(forty) ==> 40
...
NatTy(ninety) ==> 90
Extended Example

Number Name Normalization

// Pass "generate code"

// Converts arithmetic Expression trees into
// corresponding VB string

Plus(X, Y) ==> "(" . X . " + " . Y . ")"
Times(X, Y) ==> "(" . X . " * " . Y . ")"
Exp(X, Y) => "(" . X . " ^ " . Y . ")"
_ => _
Extended Example

Number Name Normalization

string:
  • seven hundred and thirty five

parse:
  • NatNum3(NatLD(NatDigit(seven)),
         hundred, and,
         NatNum2(NatTy(thirty),
                  NatDigit(five)))

pass “compute expression”:
  • Plus(Plus(30, 5),
            Times(7, Exp(10, 2)))

pass “generate code”:
  • “((30+5)+(7*(10^2)))”

evaluation:
  • 735