Transducers as a Substrate for Natural-Language Processing

Stuart M. Shieber

Division of Engineering and Applied Sciences
Harvard University

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Please do not cite, quote, copy or redistribute.

Copyright © 2003. Stuart M. Shieber.

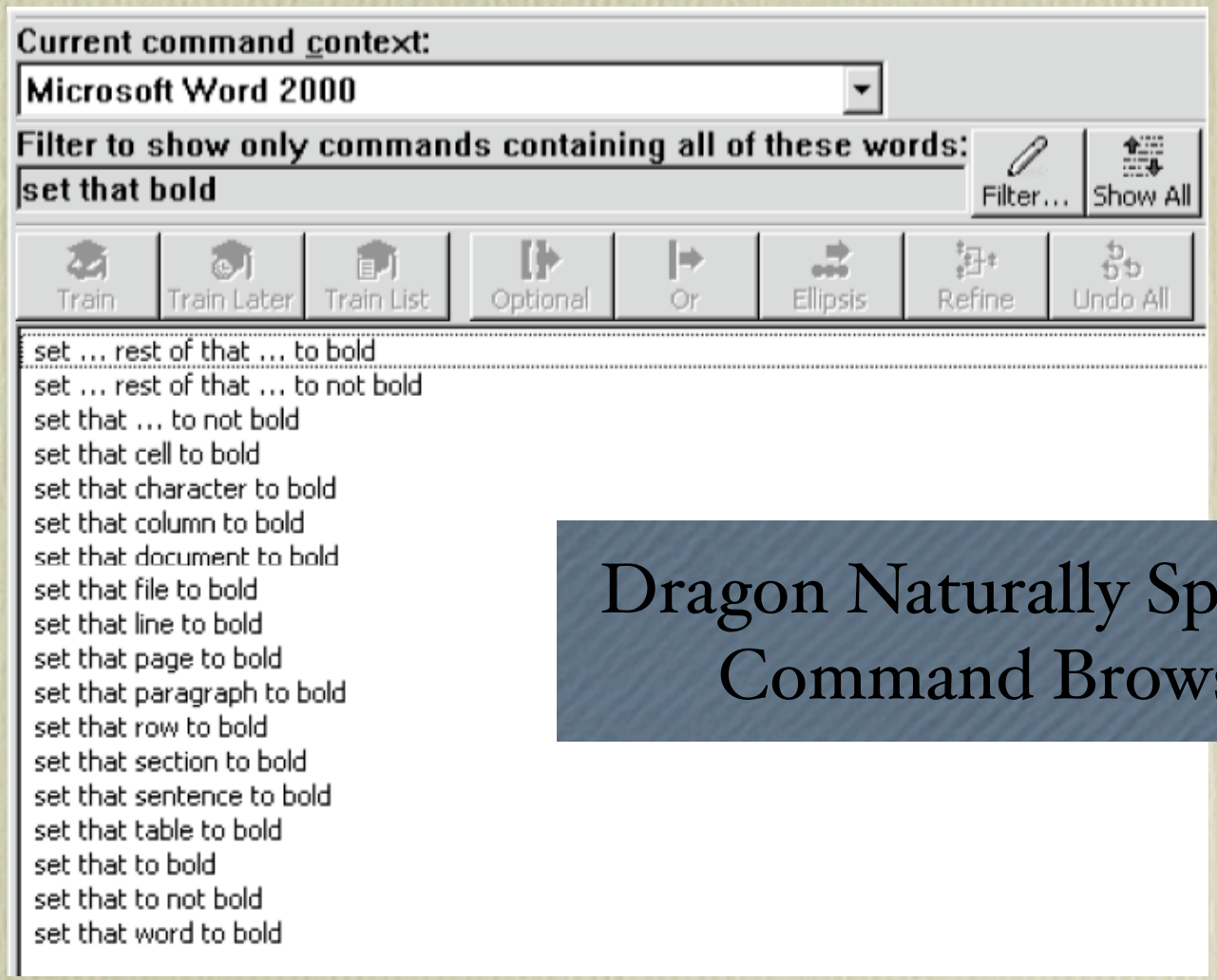
Division of Engineering and Applied Sciences
Harvard University

Motivation

Deckard: Enhance 224 to 176. Enhance, stop. Move in, stop. Pull out, track right, stop. Center in, pull back. Stop. Track 45 right. Stop. Center and stop. Enhance 34 to 36. Pan right and pull back. Stop. Enhance 34 to 46. Pull back. Wait a minute, go right, stop. Enhance 57 to 19. Track 45 left. Stop. Enhance 15 to 23. Give me a hard copy right there.

—*Blade Runner,* 1982

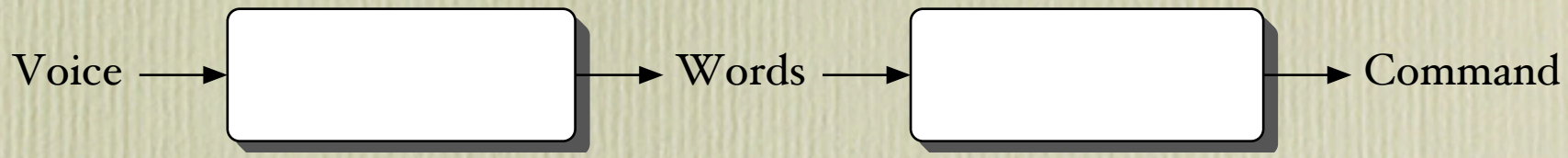
Motivation



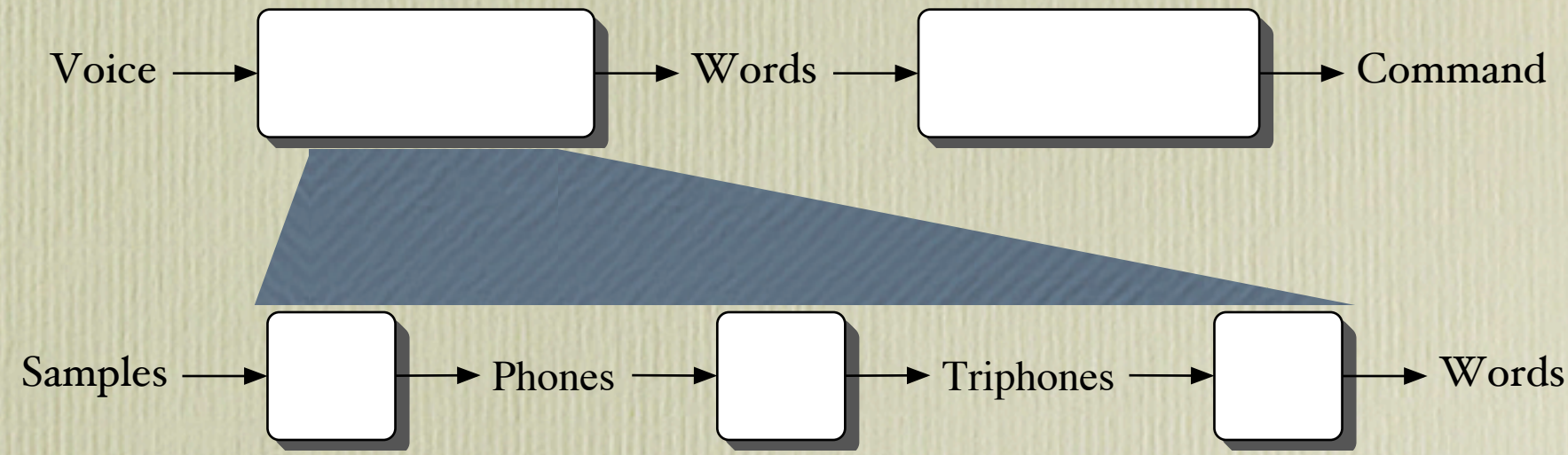
Dragon Naturally Speaking Command Browser

Voice to Commands

Voice to Commands



Voice to Commands



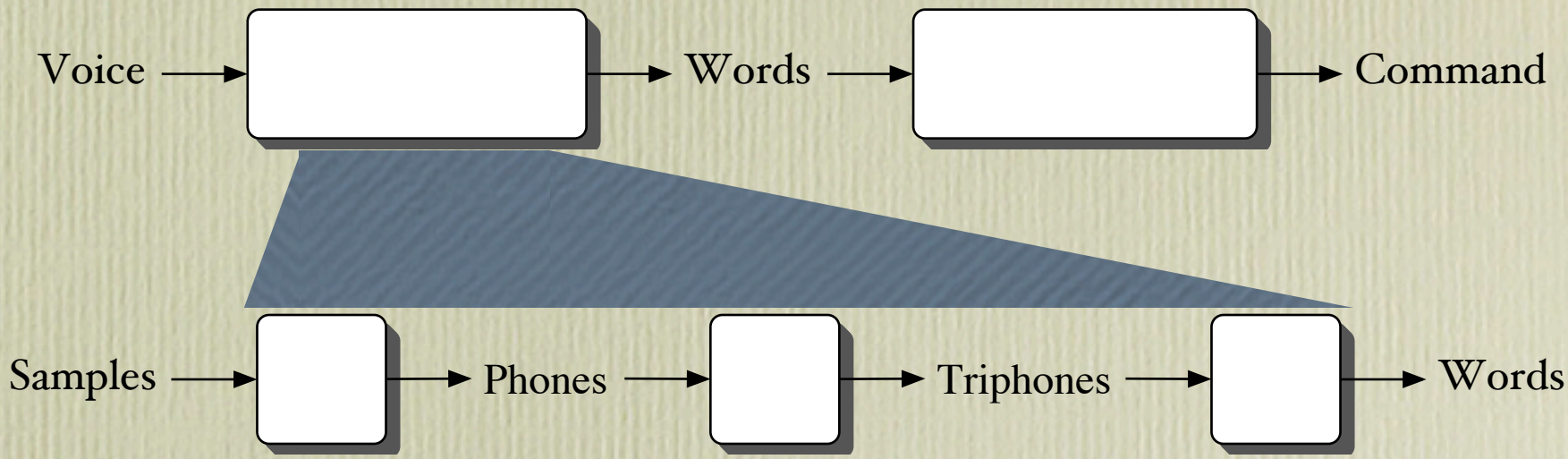
Hidden Markov Models

phonotactic models

dictionaries

language models

Voice to Commands



Hidden Markov Models

phonotactic models

Weighted Finite State Transducers

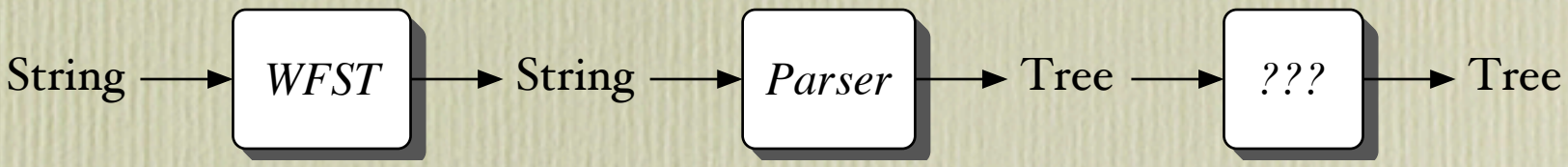
dictionaries

language models

Voice to Commands

Voice to Commands

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

Division of Engineering and Applied Sciences
Harvard University

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

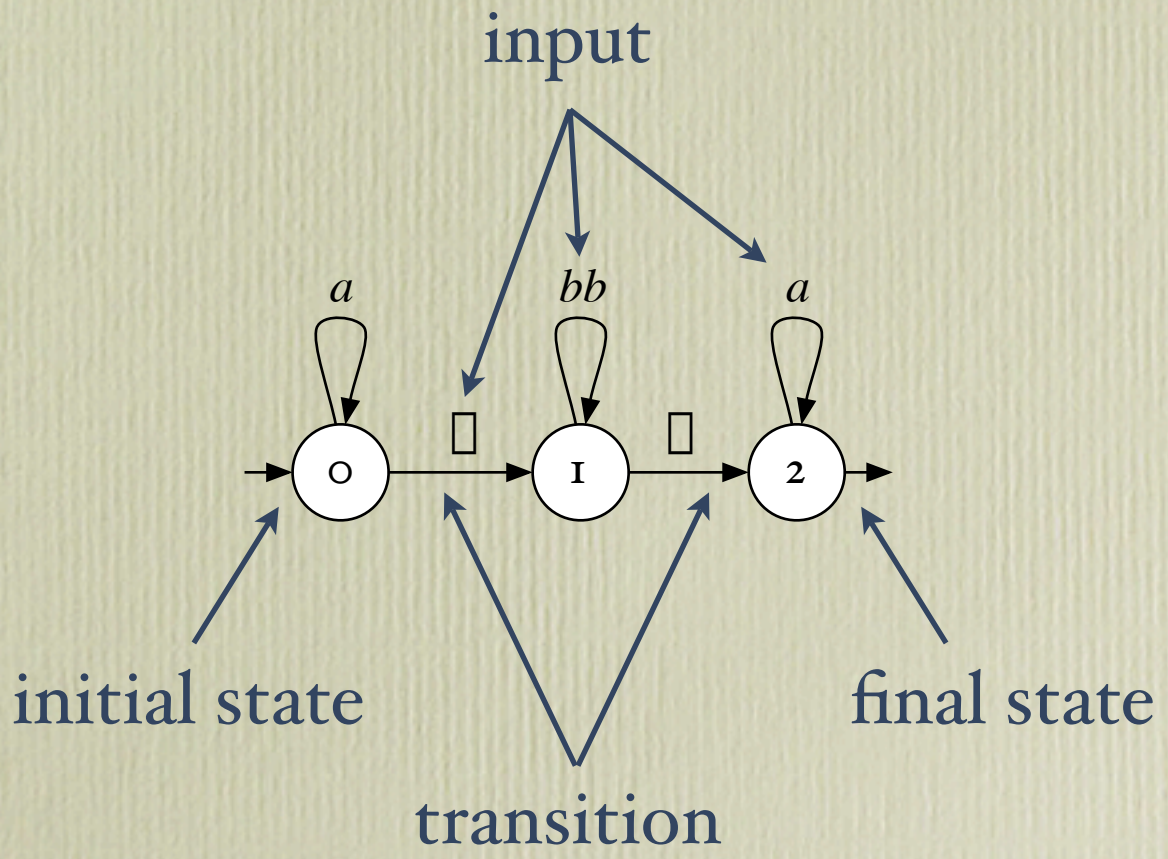
Notation for regular languages:

- Empty string: ϵ
- Singleton language: $\boldsymbol{a}$
- Union: $x \mid y$
- Concatenation: $x\,y$
- Iteration: x^*

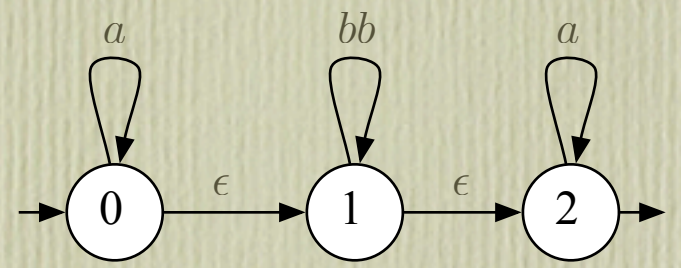
Example (alphabet = $a \ldots z$):

$$((be \mid it \mid let)\,\_)^* \qquad \ni let\_it\_be\_$$

Finite-State Automata



Finite-State Automata



A derivation:

o *aaabba#* → o *aabba#* → o *abba#* → o *bba#*

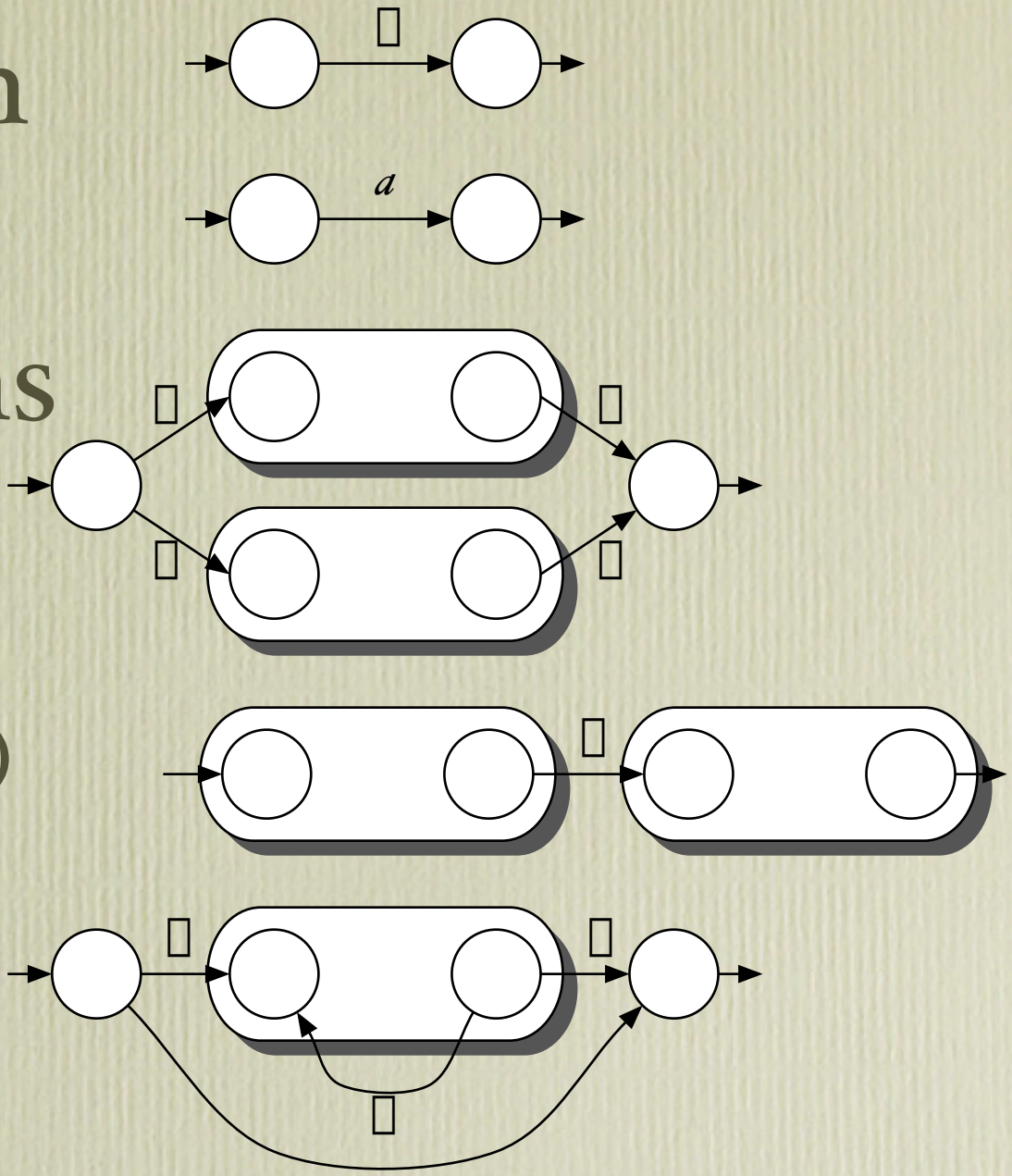
→ 1 *bba#* → 1 *a#* → 2 *a#* → 2 *#* → *#*

Definition of recognition:

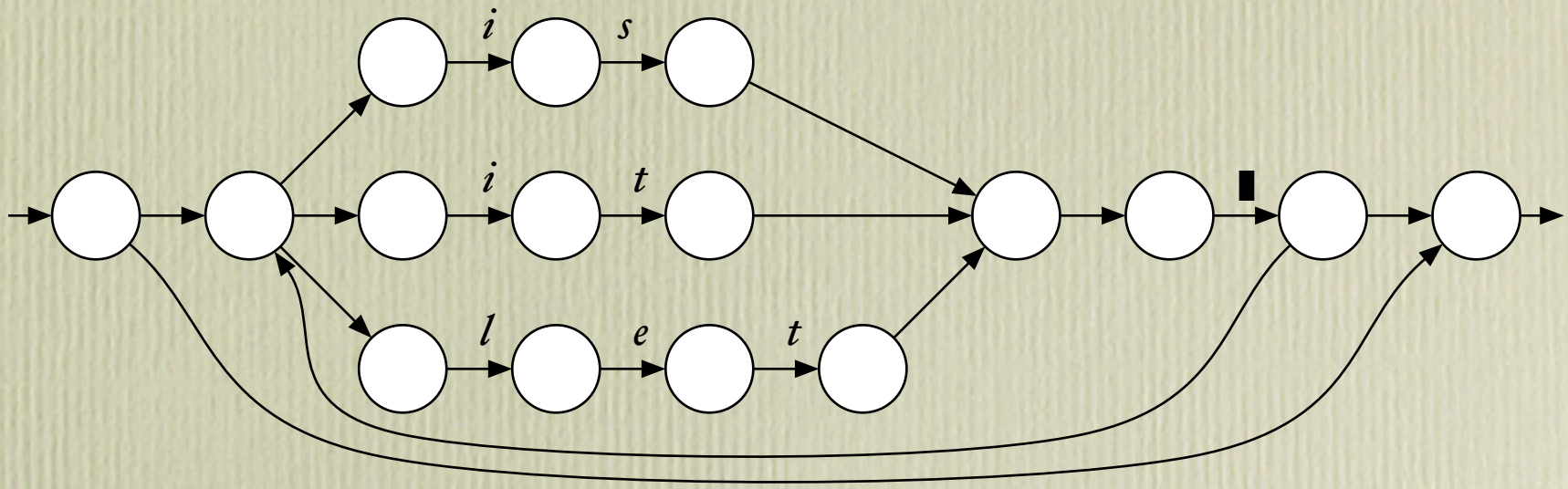
- accept *w* if and only if o *w#* →* *#*

FSAs from Regular Expressions

(Thompson's Construction)



Sample FSA



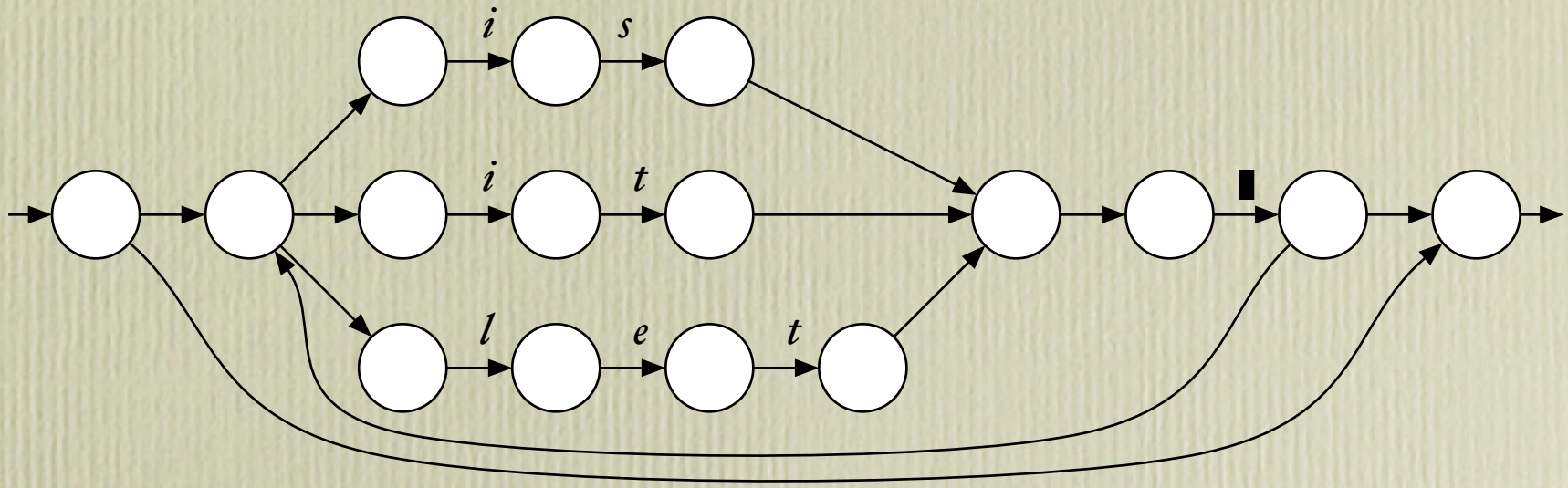
$$((is \mid it \mid let)\sqcup)^*$$

Degrees of Freedom

Lots of things make no difference:

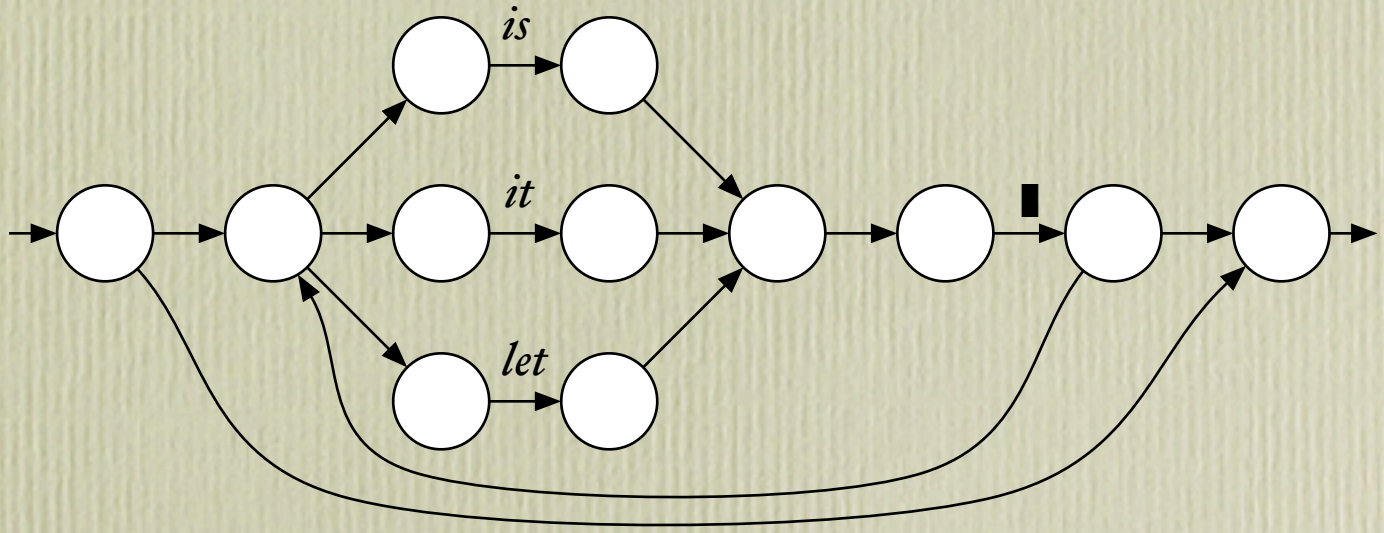
- Granularity
- Epsilon removal
- Left-right reversal
- Determinization
- Minimization

Granularity of Input



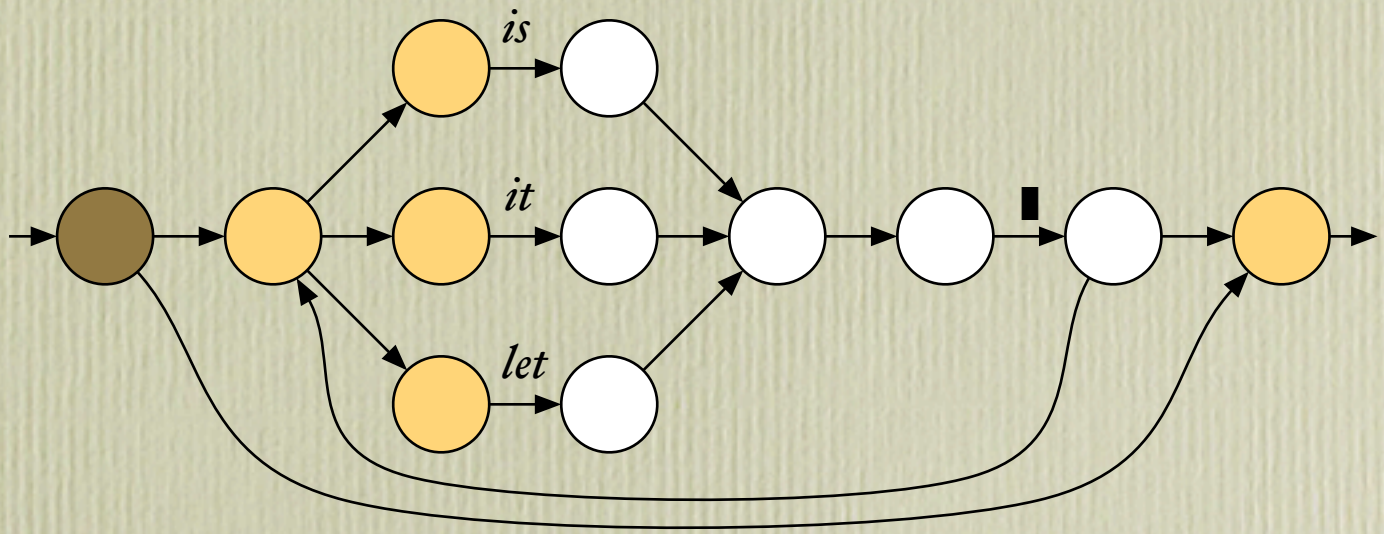
$$((is \mid it \mid let)\sqcup)^*$$

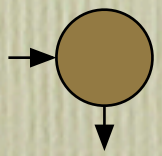
Granularity of Input



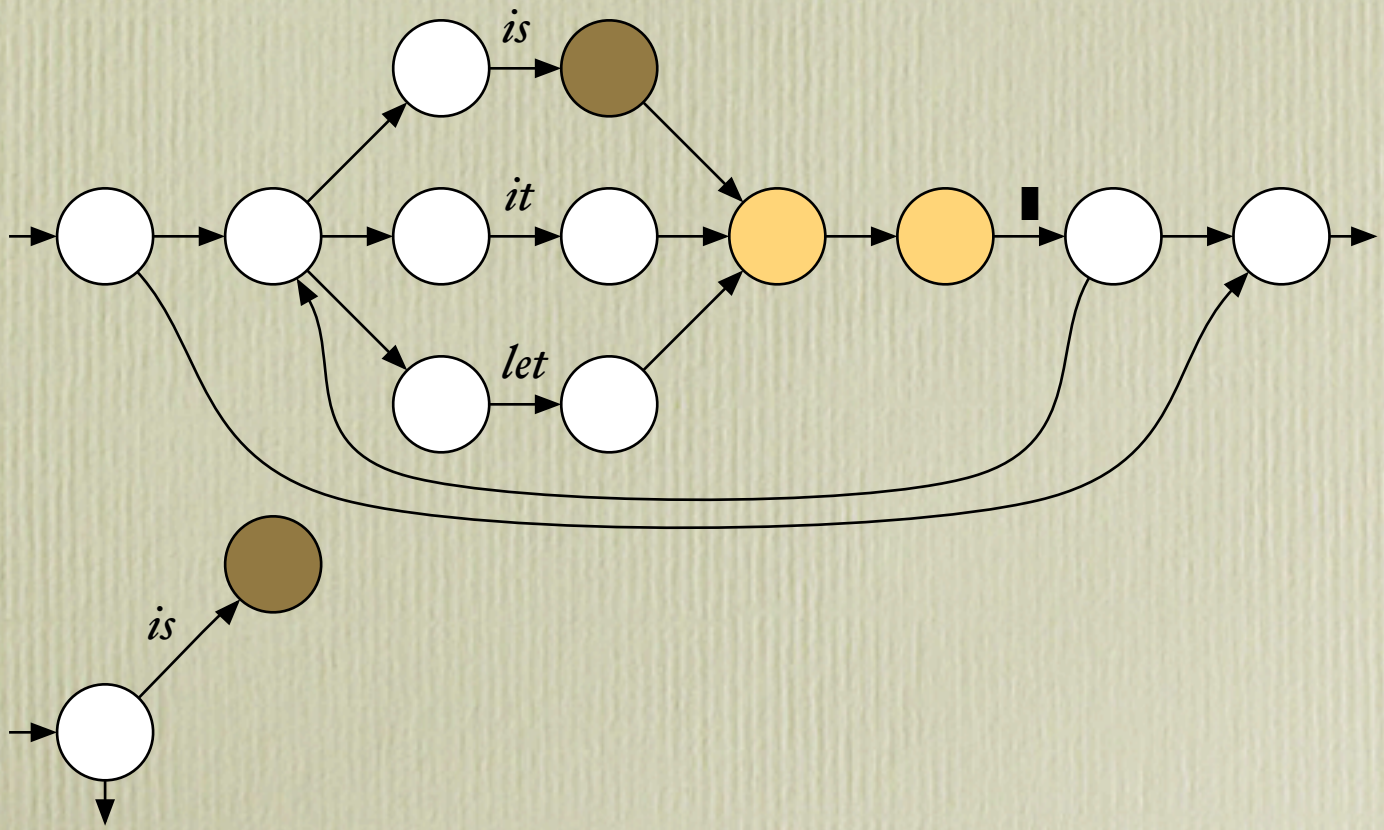
$$((is \mid it \mid let)\_)^*$$

Epsilon Removal

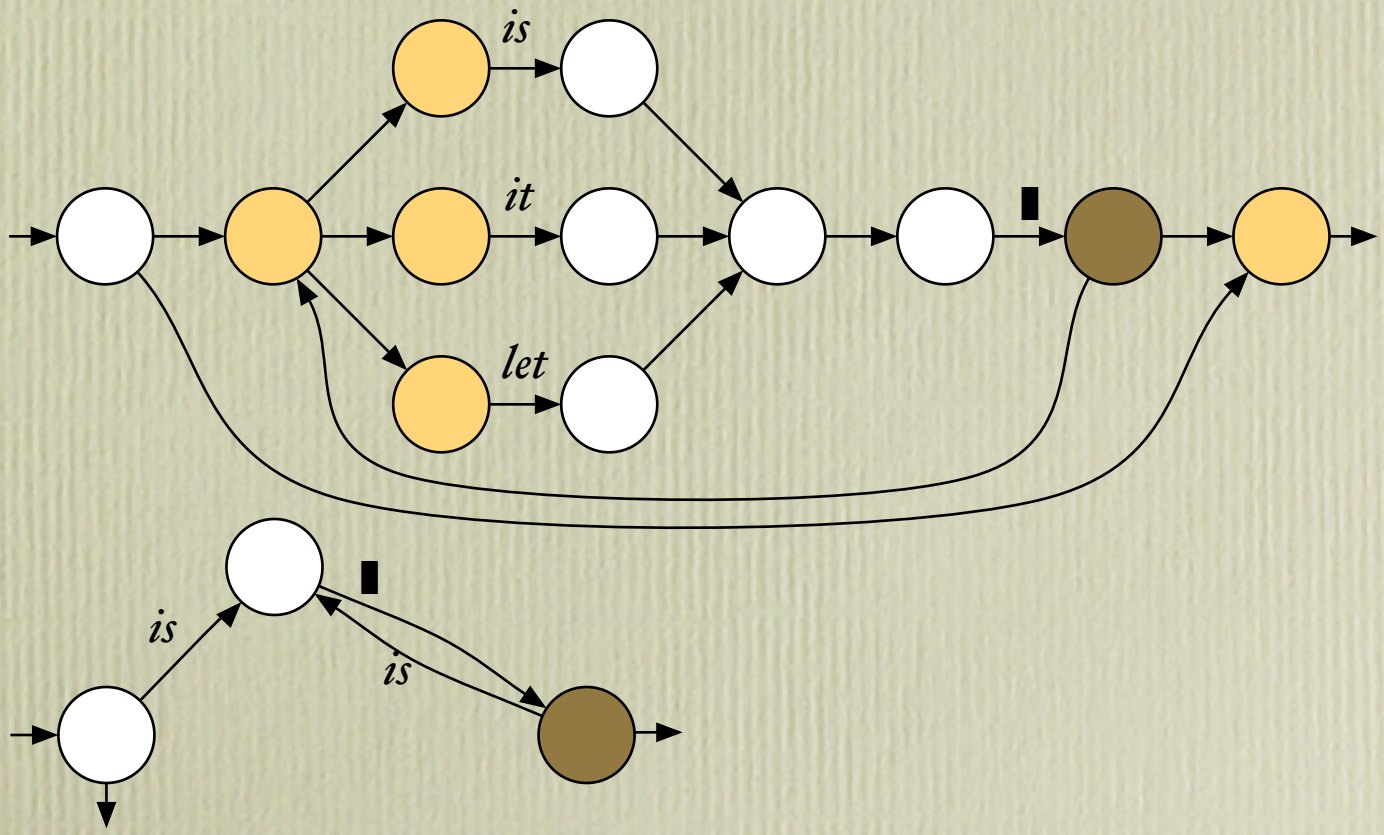




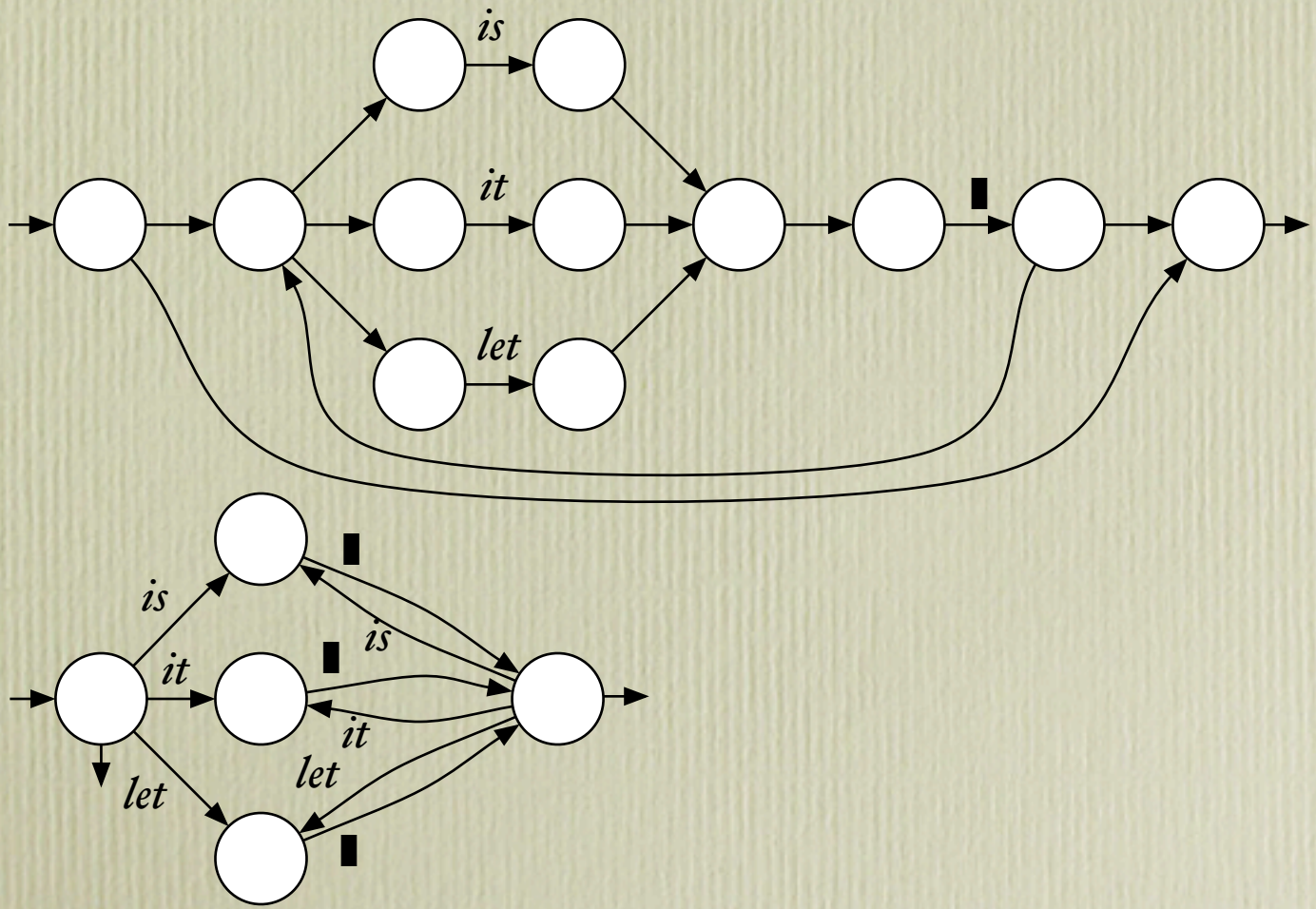
Epsilon Removal



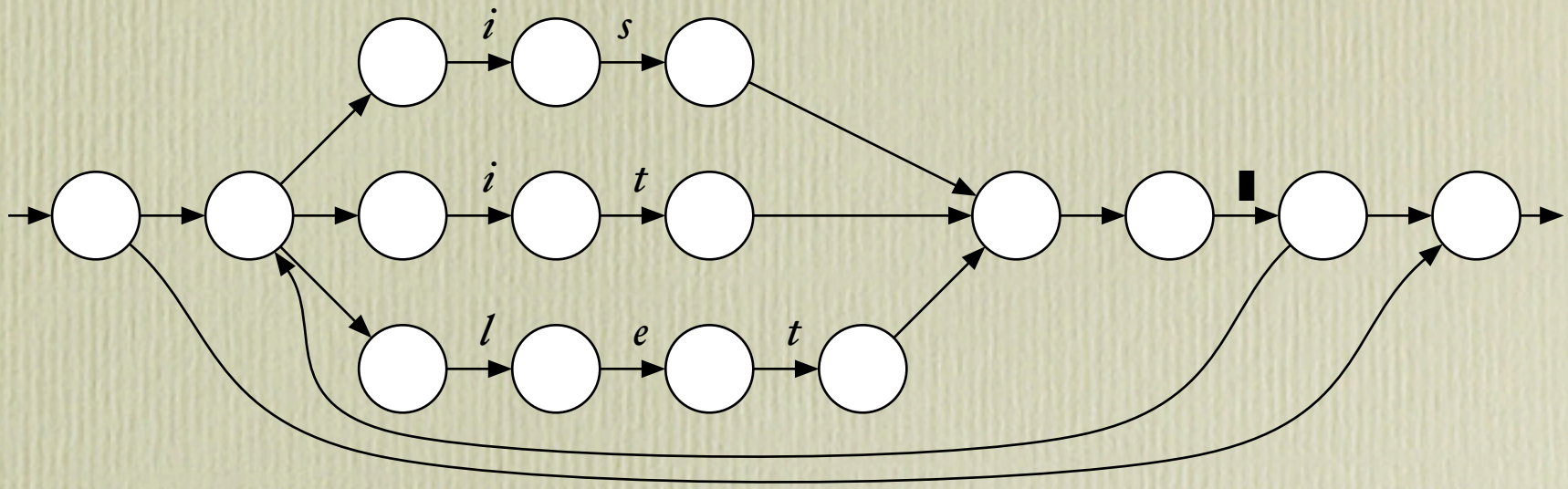
Epsilon Removal



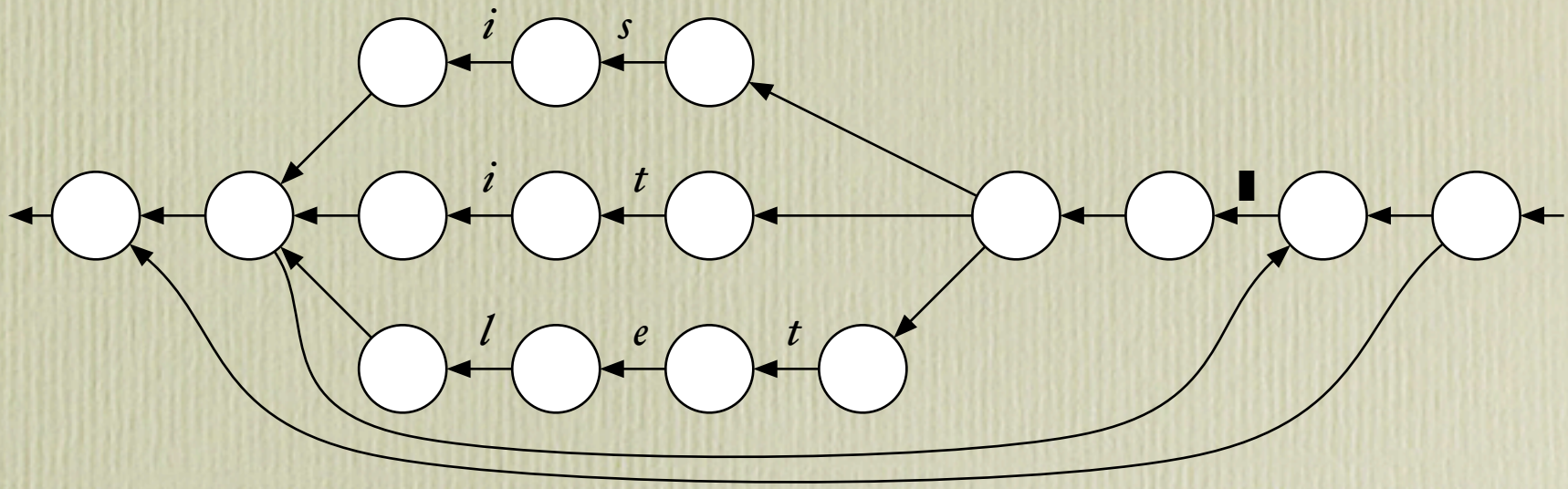
Epsilon Removal



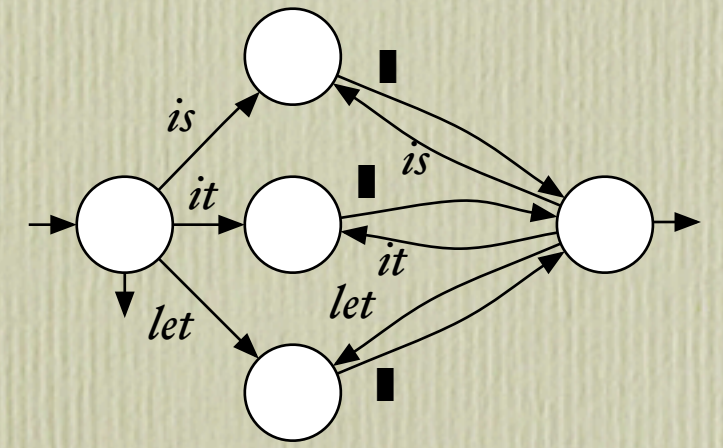
Left-Right Reversal



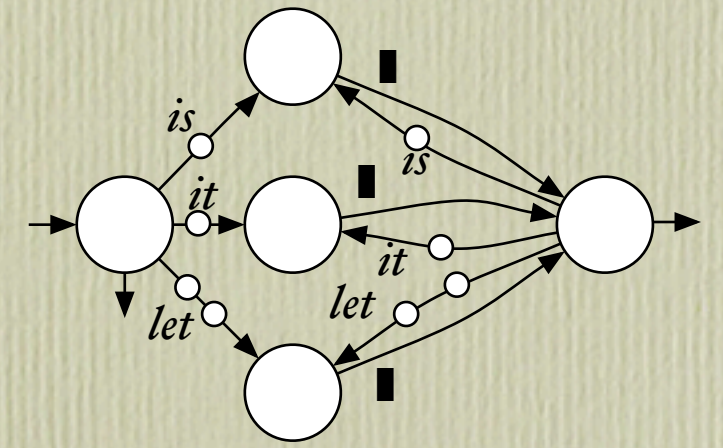
Left-Right Reversal



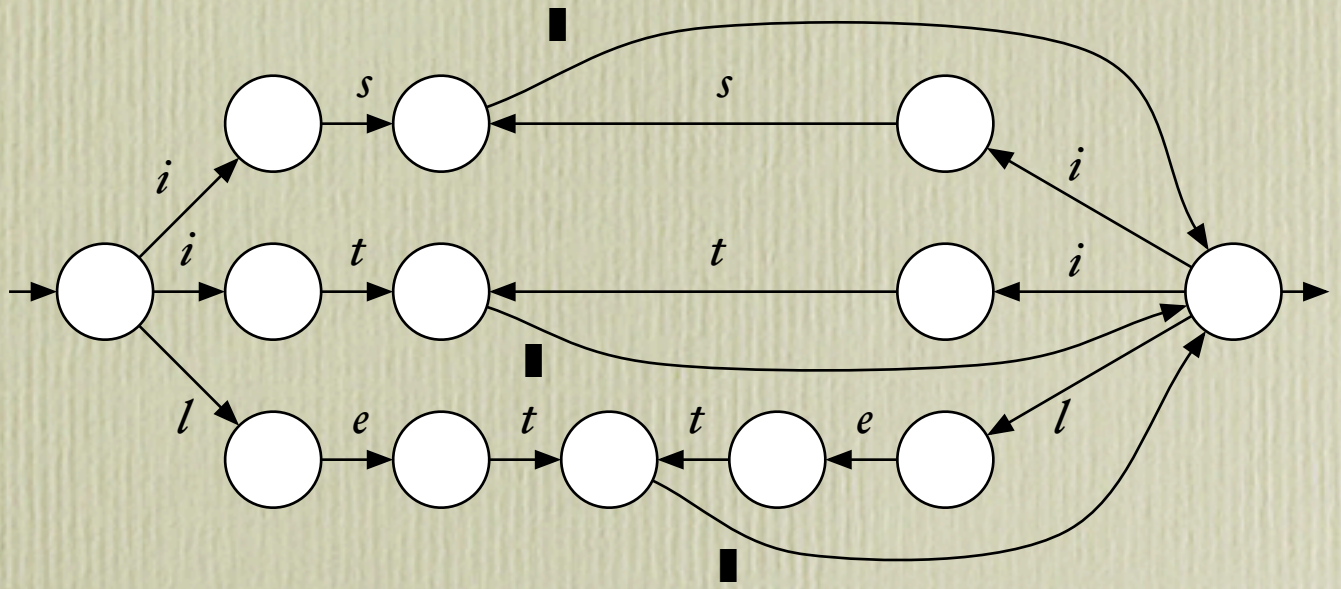
Determinization



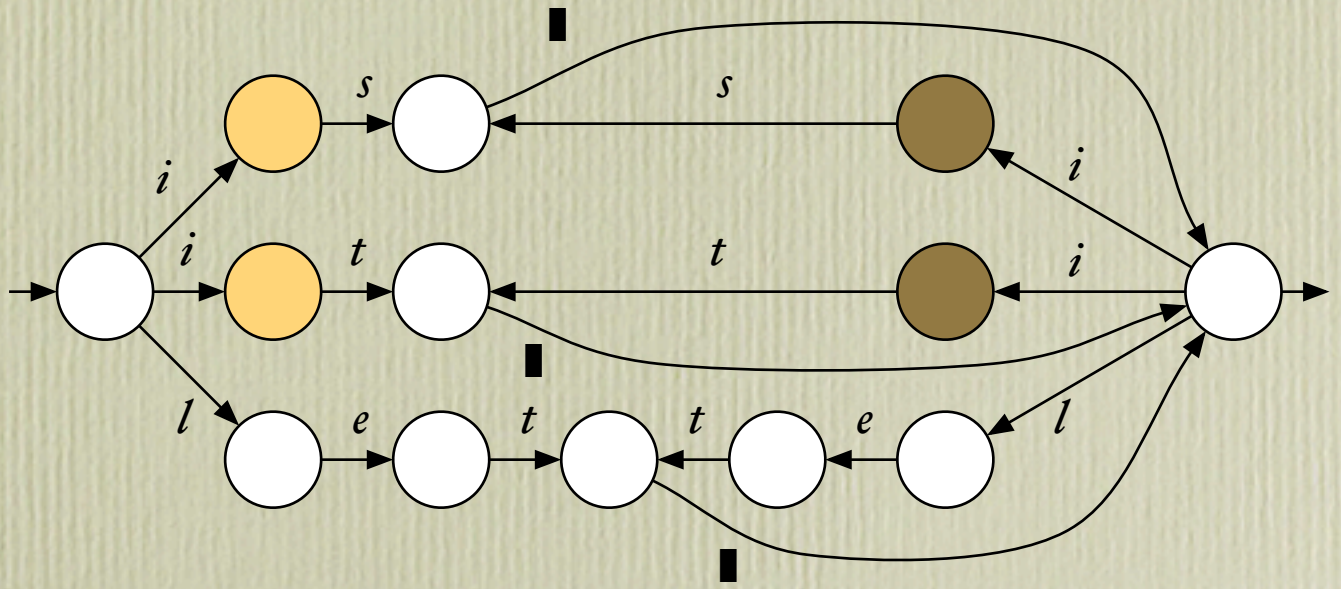
Determinization



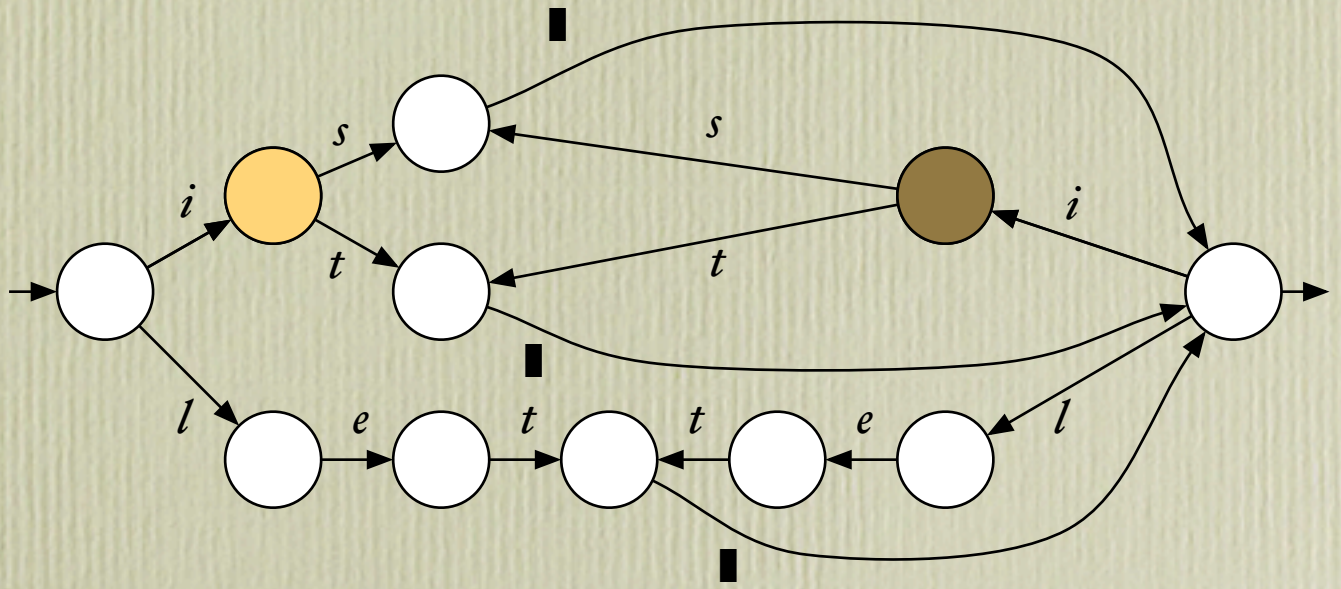
Determinization



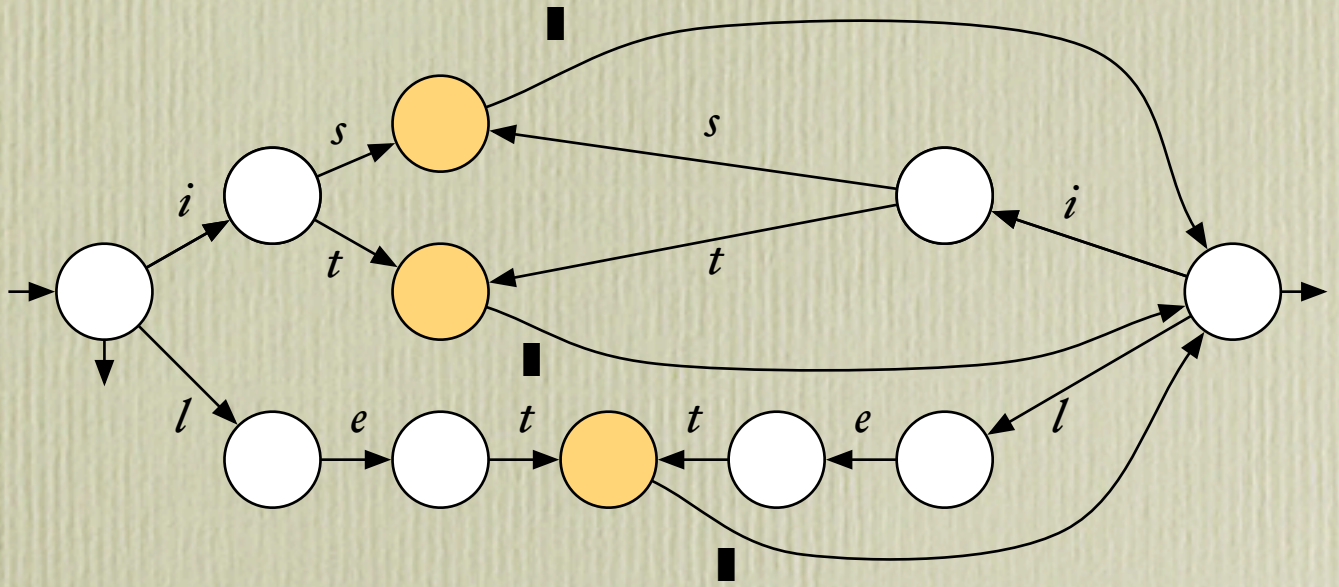
Determinization



Determinization

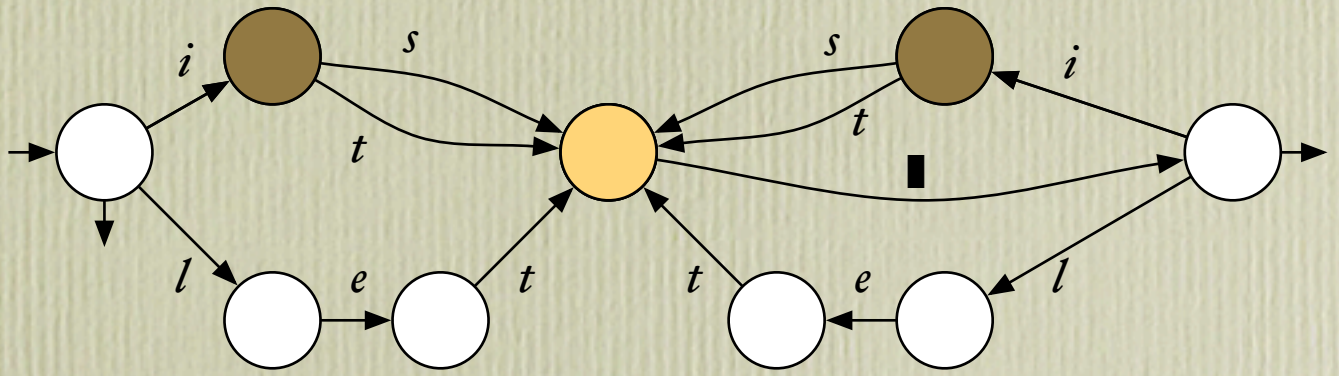


Minimization



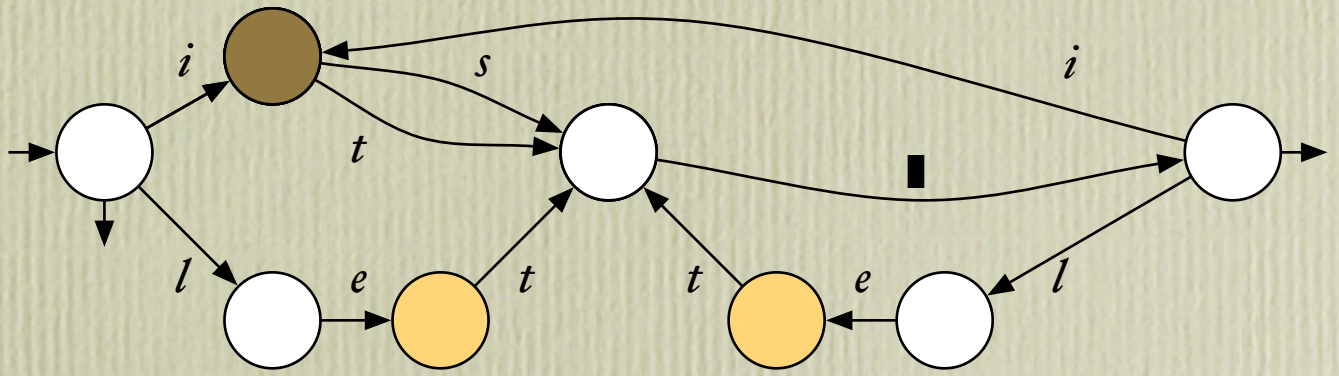
States correspond to equivalence classes of suffixes. States corresponding to identical equivalence classes can be merged.

Minimization



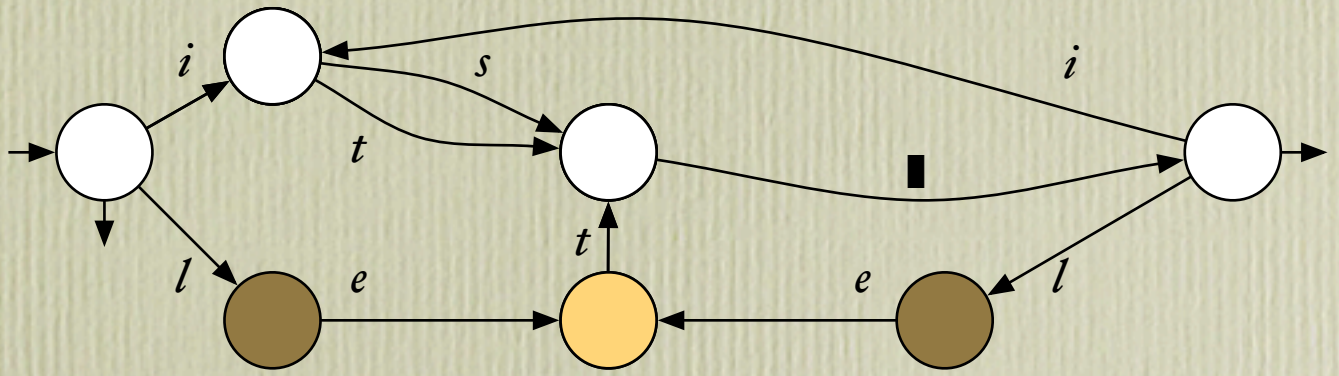
States correspond to equivalence classes of suffixes. States corresponding to identical equivalence classes can be merged.

Minimization



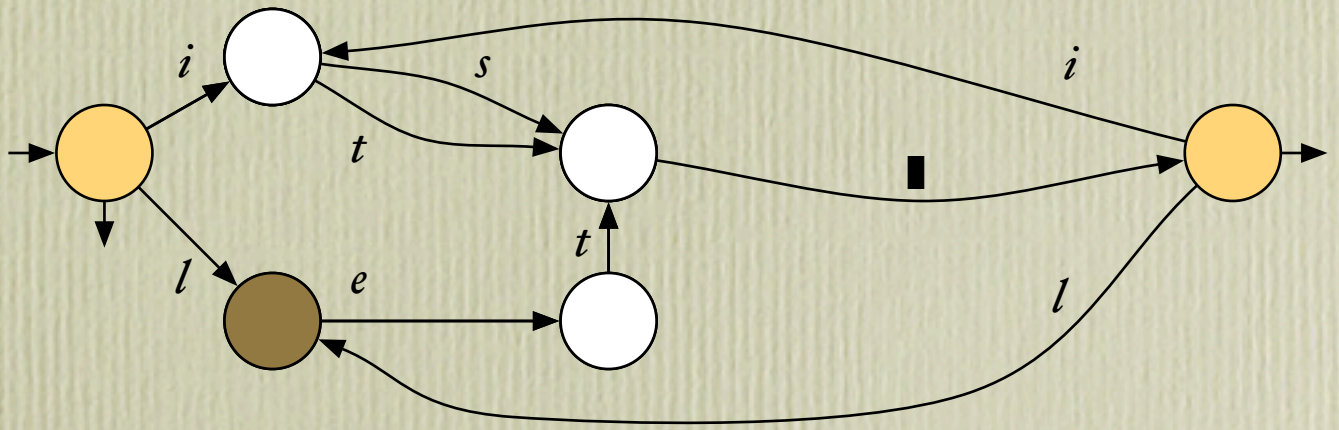
States correspond to equivalence classes of suffixes. States corresponding to identical equivalence classes can be merged.

Minimization



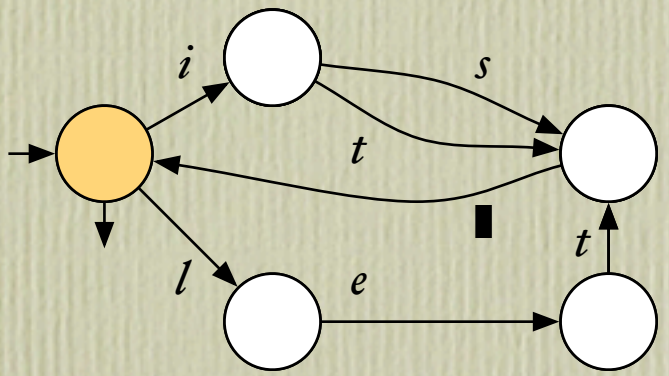
States correspond to equivalence classes of suffixes. States corresponding to identical equivalence classes can be merged.

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

Division of Engineering and Applied Sciences
Harvard University

Regular Relations

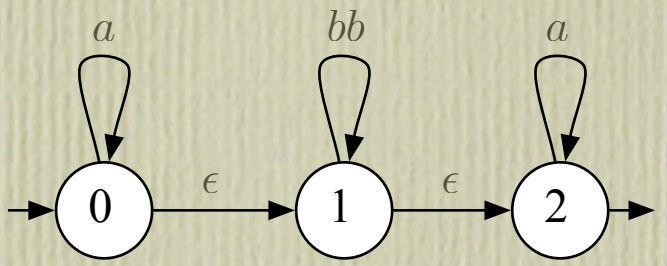
A string relation is a set of *pairs of* strings.

$$input : output$$

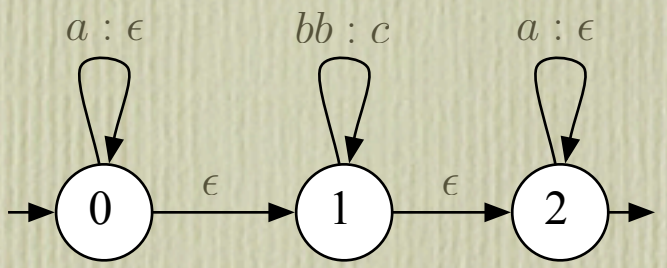
Regular relations is the smallest class of relations including:

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

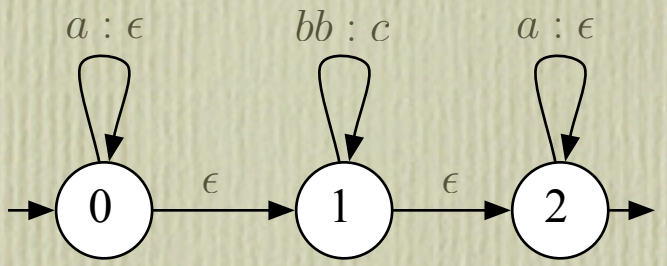
Sample FST



Sample FST



Sample FST



A derivation:

○ *aaabba#* → ○ *aabba#* → ○ *abba#* → ○ *bba#*

→ 1 *bba#* → *c* 1 *a#* → *c* 2 *a#* → *c* 2 *#* → *c #*

Definition of recognition:

- accept *s*:*t* if and only if ○ *s #* →* *t #*

Degrees of Freedom

granularity of input and output

left-right reversal

epsilon removal *

pushing

determinization *

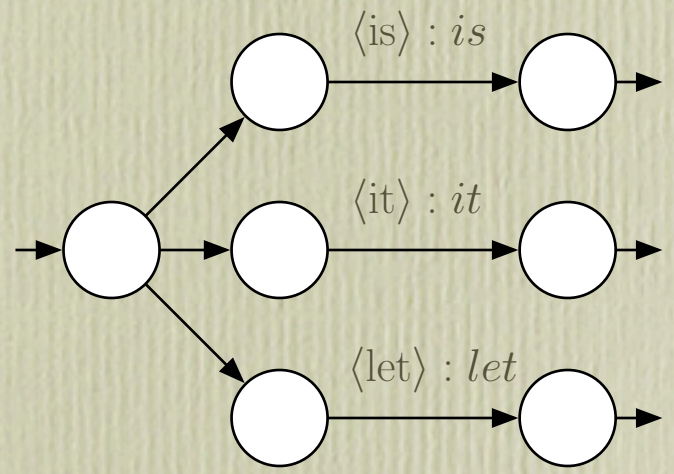
minimization *

inversion

composition

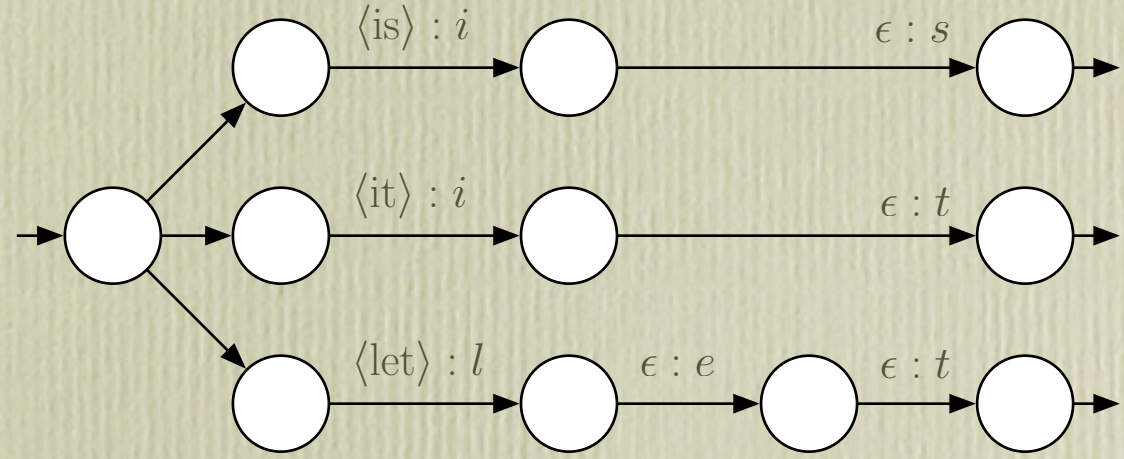
* sometimes

Granularity of Output

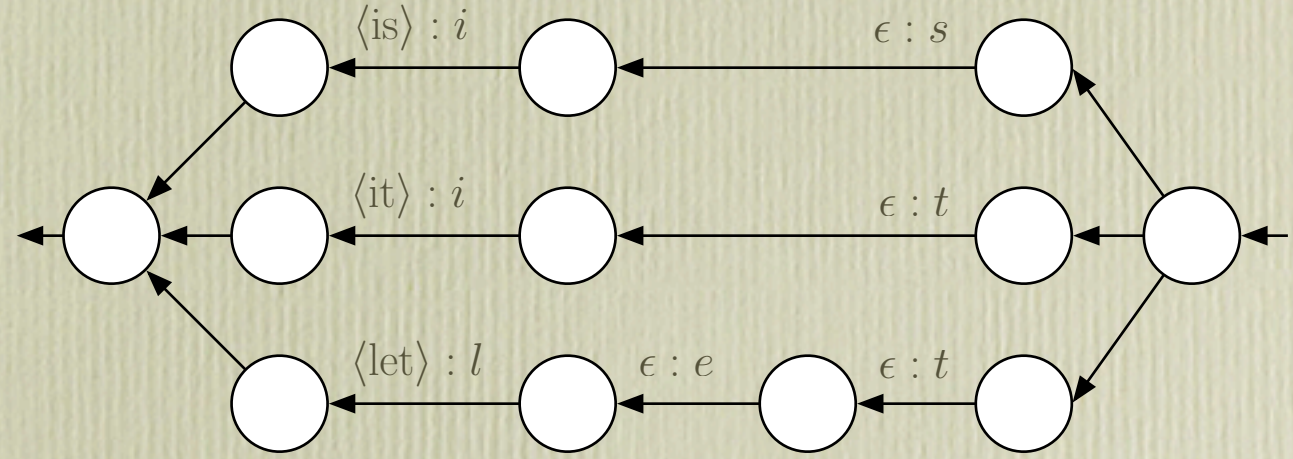


A Spelling Dictionary

Granularity of Output

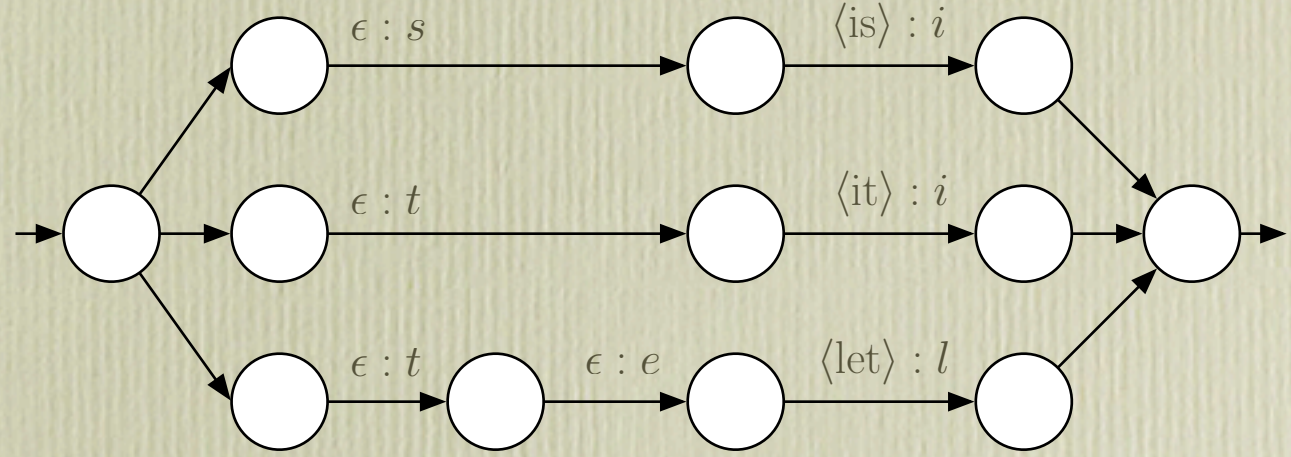


Left-Right Reversal



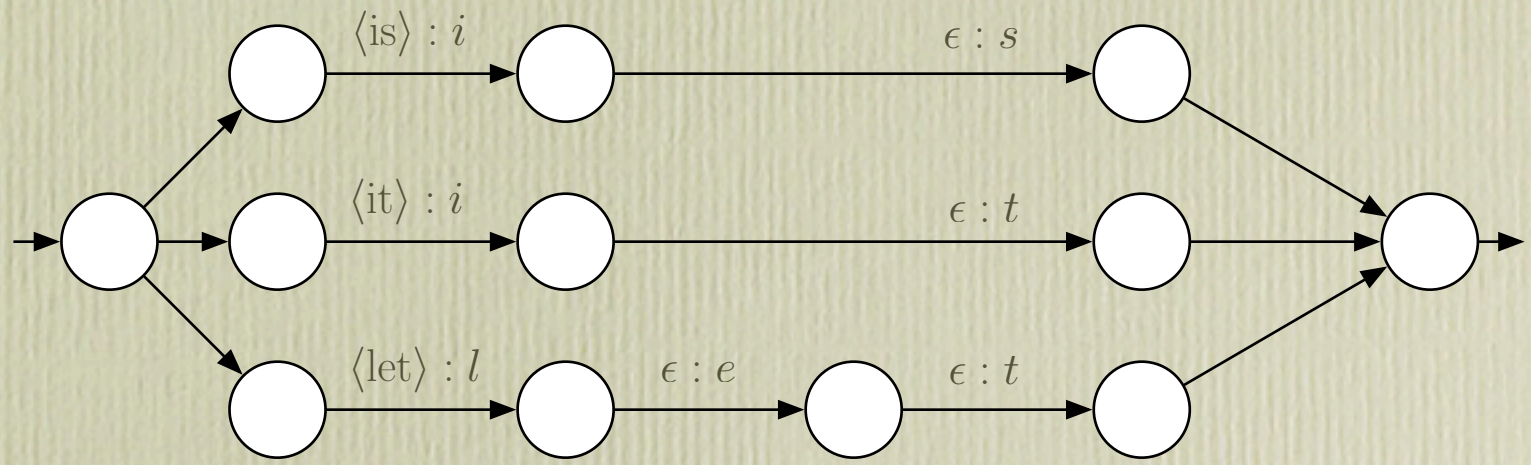
...by treating FST as FSA over cross-product vocabulary.

Left-Right Reversal

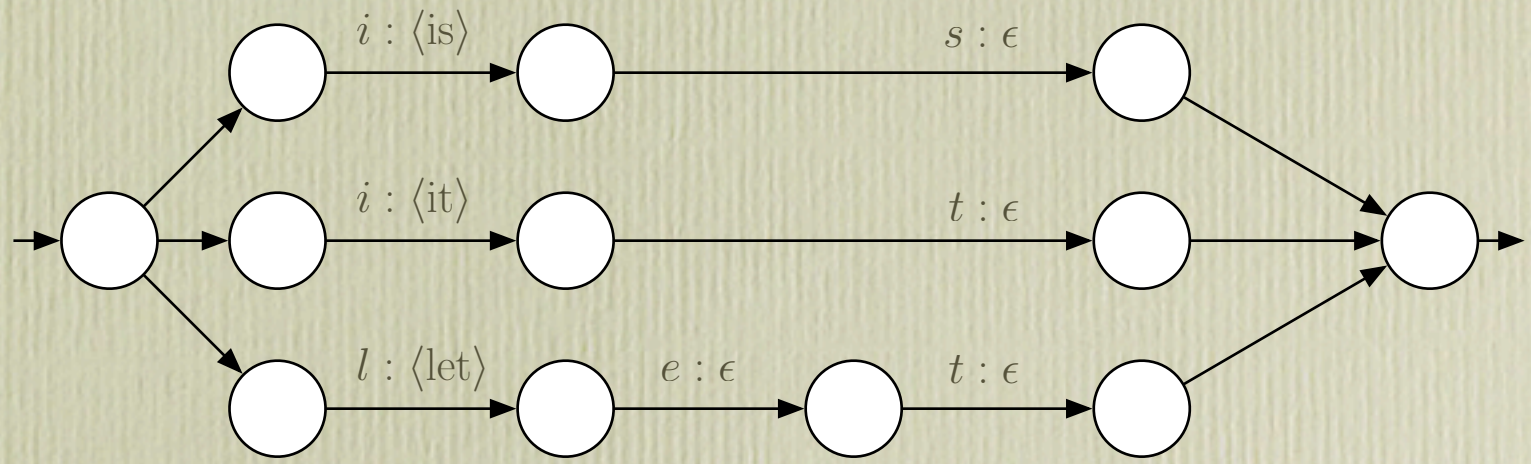


...by treating FST as FSA over cross-product vocabulary.

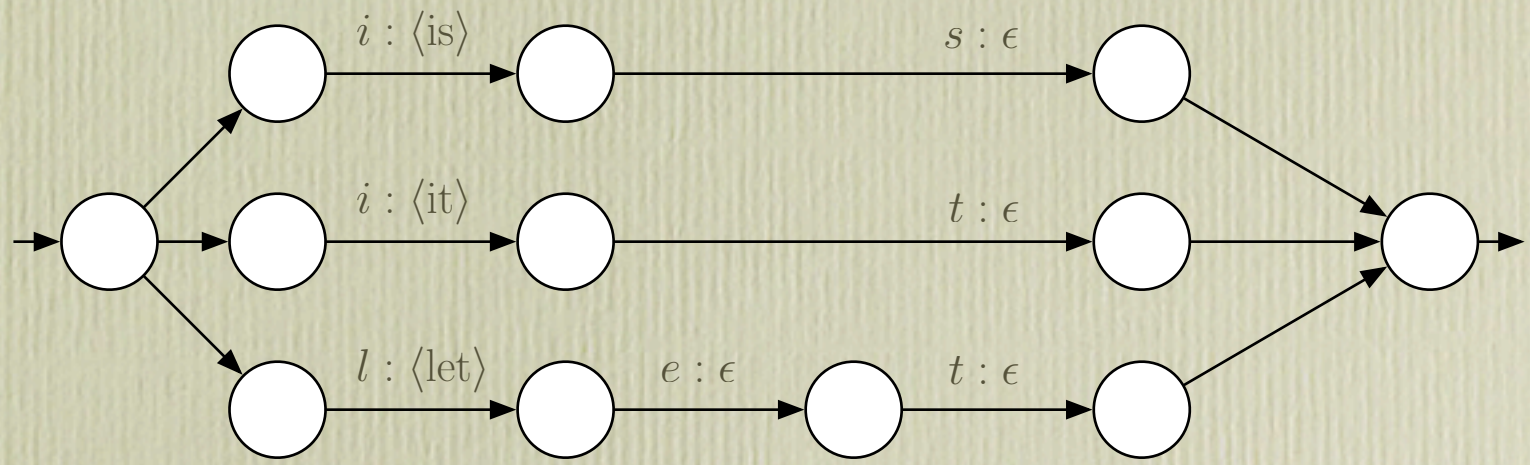
Inversion



Inversion

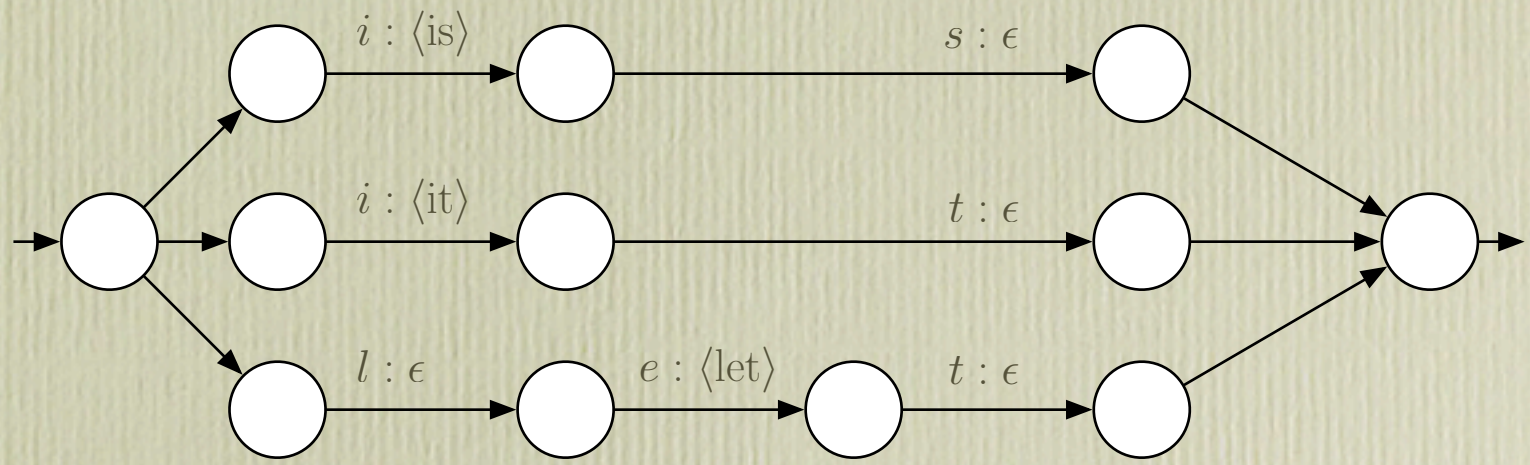

$i : \langle \text{is} \rangle$
$s : \epsilon$
$i : \langle \text{it} \rangle$
$t : \epsilon$
$l : \langle \text{let} \rangle$
$e : \epsilon$
$t : \epsilon$

Pushing

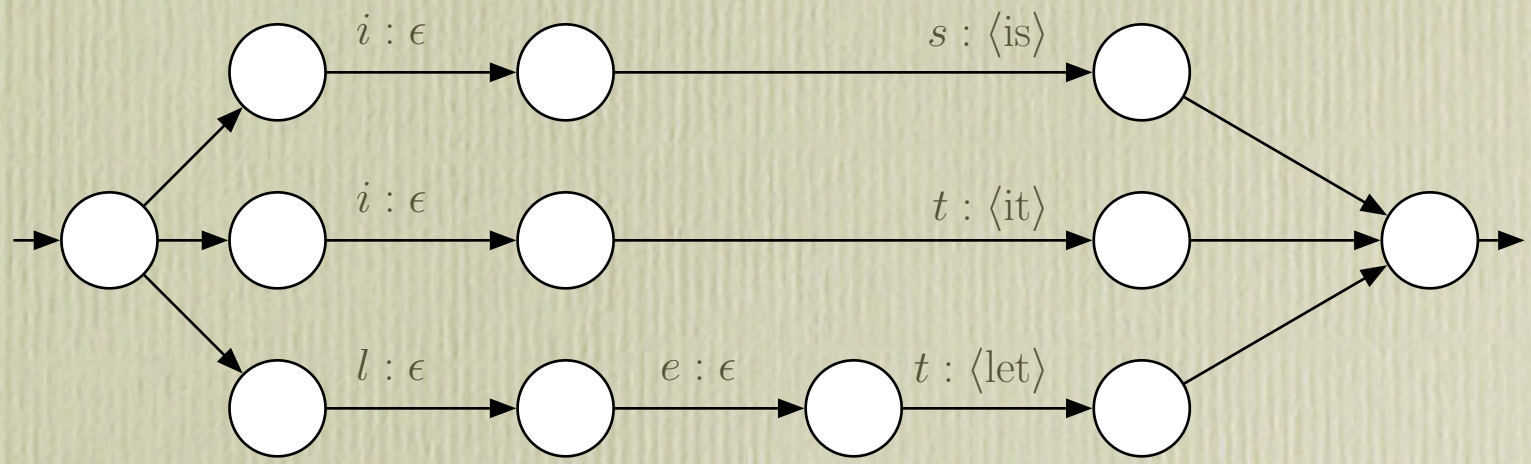


Movement of output symbols along a path.

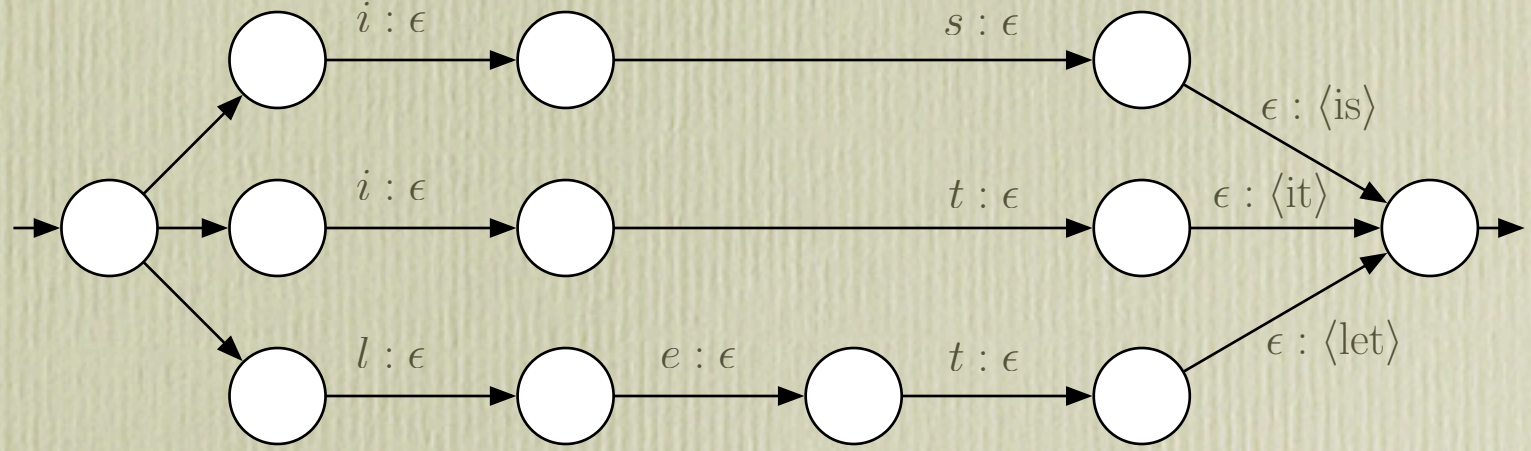
Pushing


$i : \langle \text{is} \rangle$
$s : \epsilon$
$i : \langle \text{it} \rangle$
$t : \epsilon$
$l : \epsilon$
$e : \langle \text{let} \rangle$
$t : \epsilon$

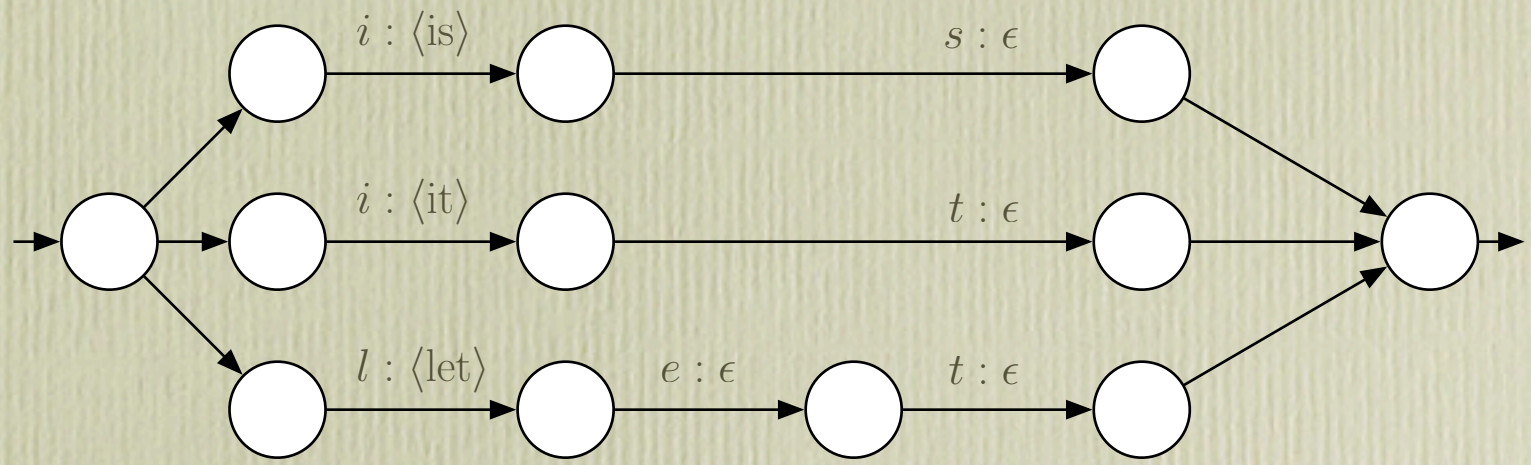
Pushing


$i : \epsilon$
$s : \langle \text{is} \rangle$
$i : \epsilon$
$t : \langle \text{it} \rangle$
$l : \epsilon$
$e : \epsilon$
$t : \langle \text{let} \rangle$

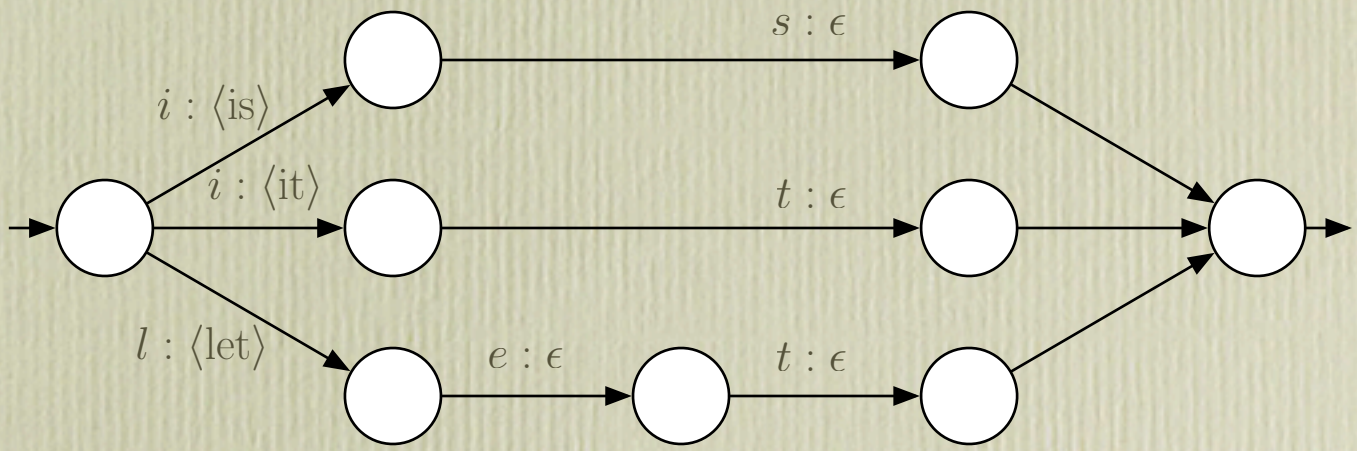
Pushing


$i : \epsilon$
$s : \epsilon$
$\epsilon : \langle \text{is} \rangle$
$i : \epsilon$
$t : \epsilon$
$\epsilon : \langle \text{it} \rangle$
$l : \epsilon$
$e : \epsilon$
$t : \epsilon$
$\epsilon : \langle \text{let} \rangle$

Determinization

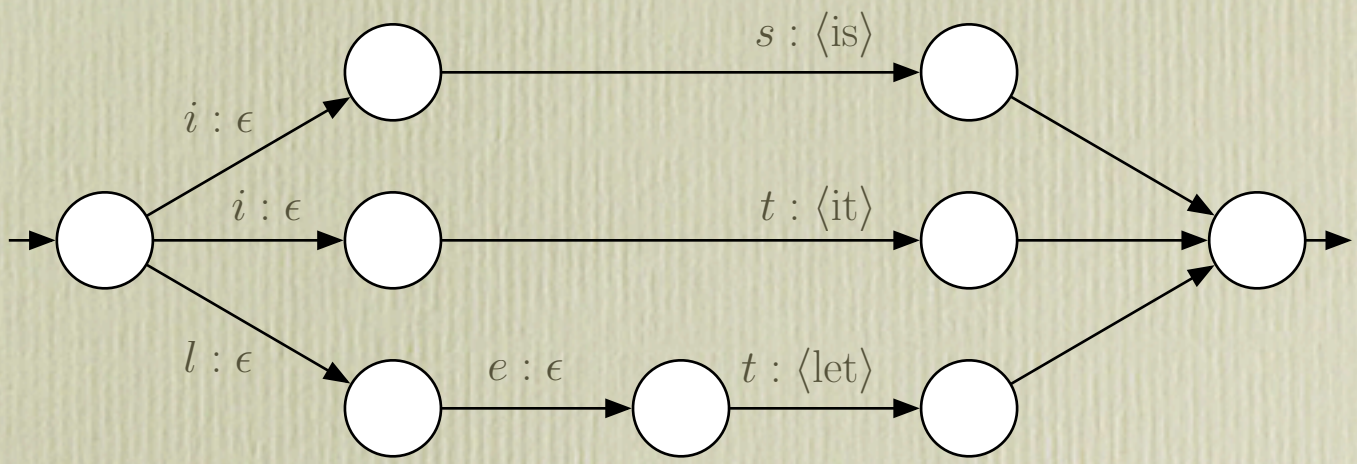


Determinization



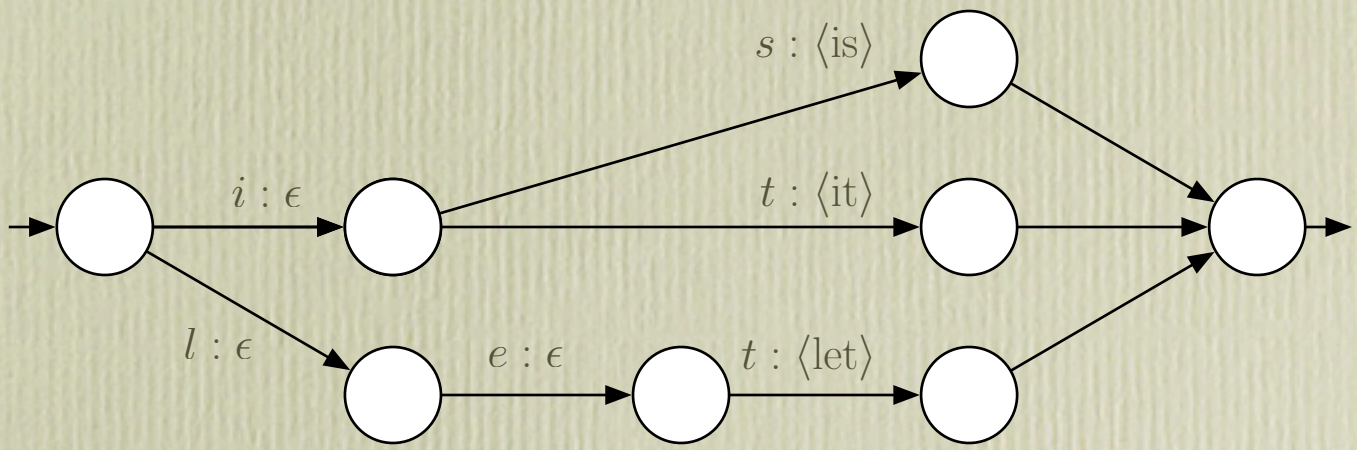
No more determinization possible...

Determinization



No determinization possible (without pushing)...

Determinization

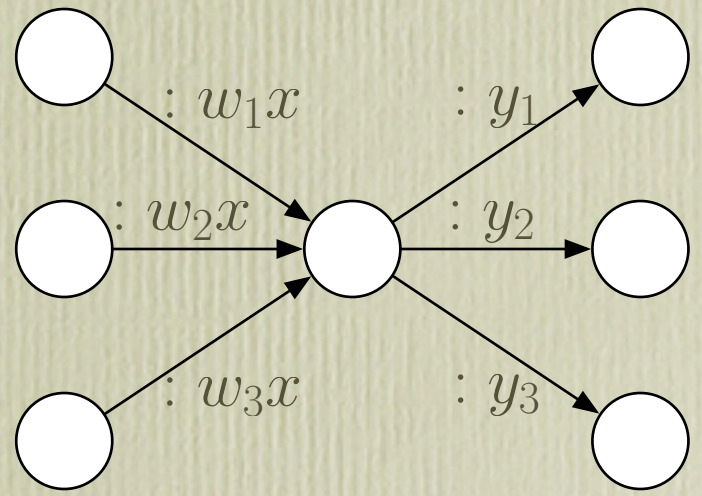


(The transducer determinization algorithm performs forward pushing implicitly.)

Pushing

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

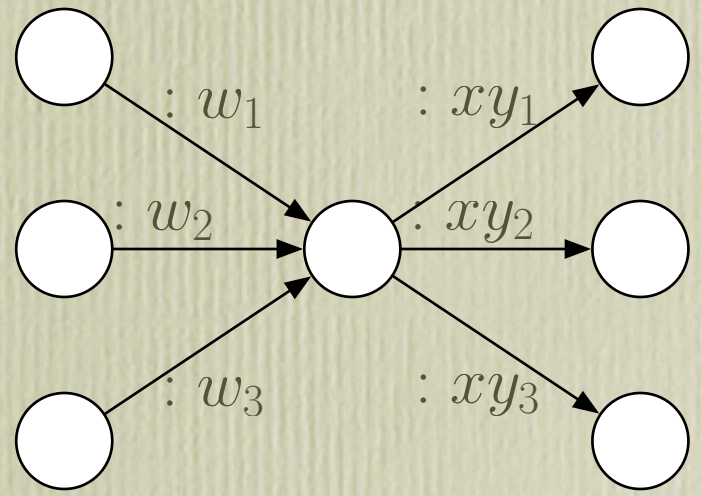
- remove x from end of all in-edges
- add x to start of all out-edges.



Pushing

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

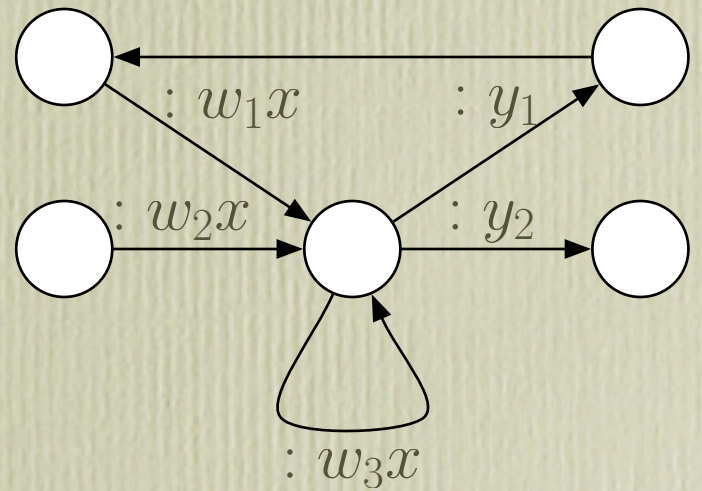
- remove x from end of all in-edges
- add x to start of all out-edges.



Pushing

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

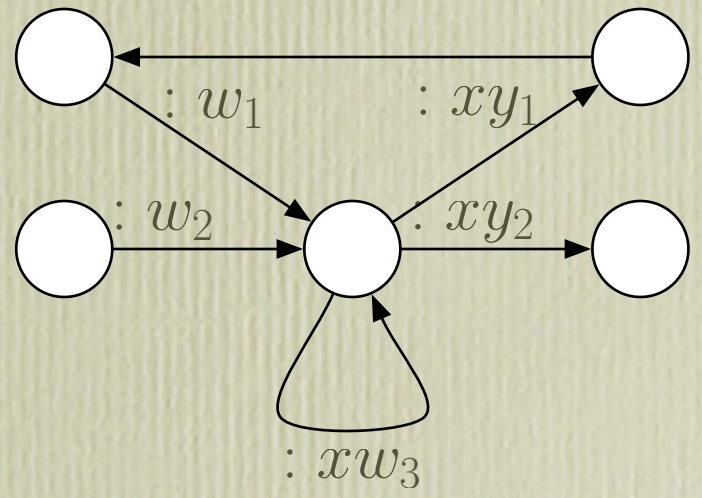
- remove x from end of all in-edges
- add x to start of all out-edges.



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.



Limitations on Determinization

Phrasal verb marking:

1. John *called* the teacher *up*.
2. John *called* the dogs *off*.

Idealized transduction problem:

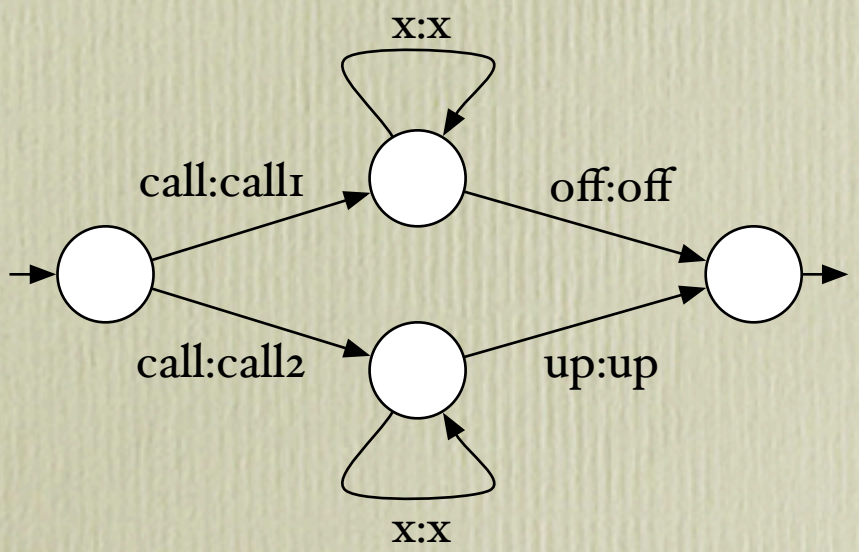
- call x* up → call1 x* up
- call x* off → call2 x* off

(based on Roche and Schabes (1996))

Limitations on Determinization

Idealized transduction problem:

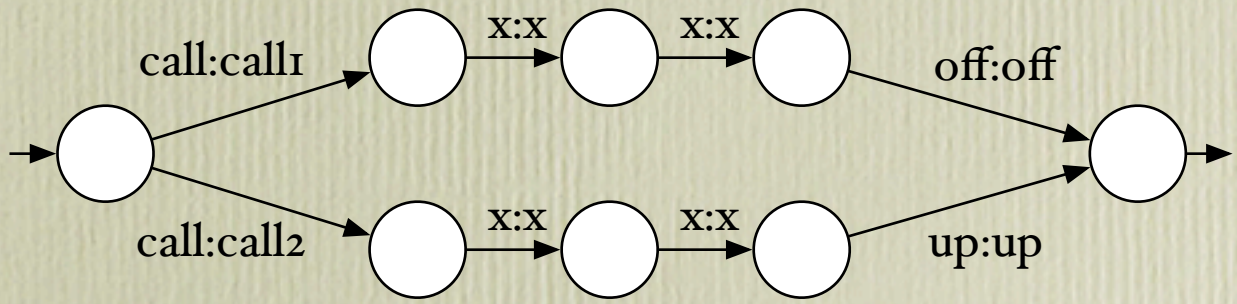
- call x* up → call1 x* up
- call x* off → call2 x* off



Limitations on Determinization

Idealized transduction problem:

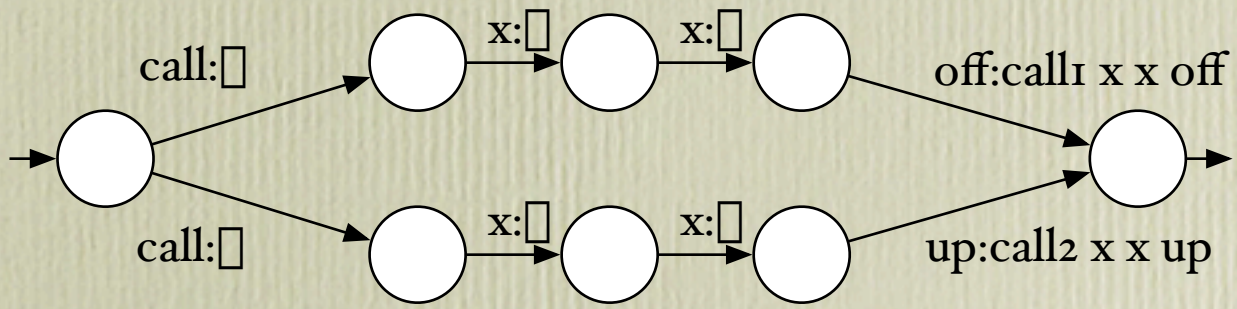
- call x x up → call1 x x up
- call x x off → call2 x x off



Limitations on Determinization

Idealized transduction problem:

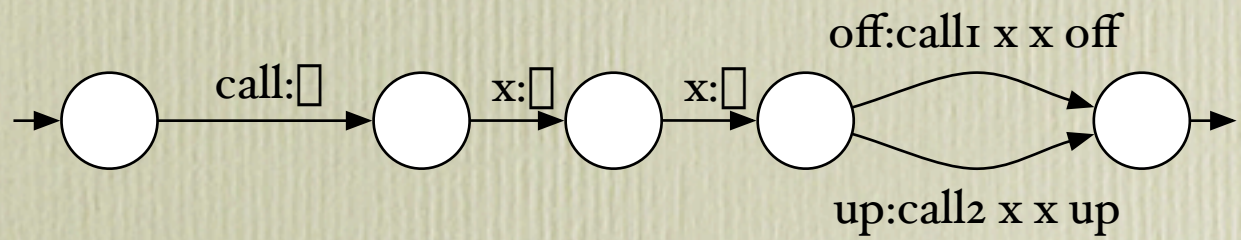
- call x x up → call1 x x up
- call x x off → call2 x x off



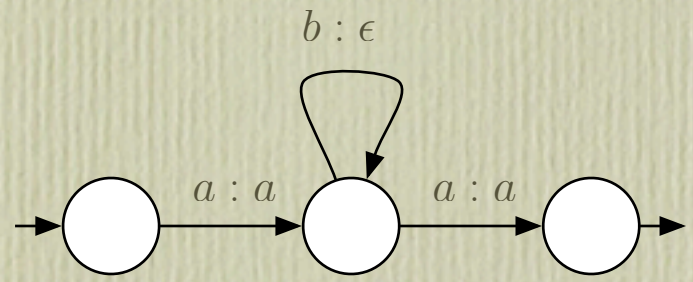
Limitations on Determinization

Idealized transduction problem:

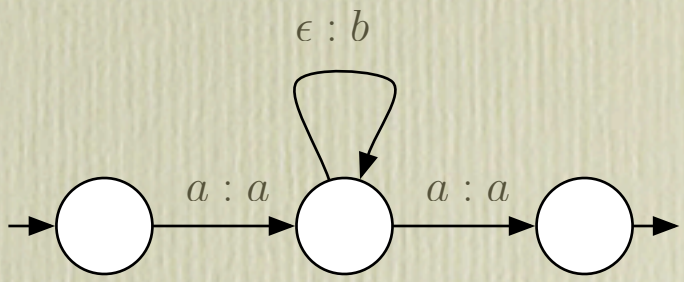
- call x x up → call1 x x up
- call x x off → call2 x x off



Limitations on Epsilon Removal



b removal
(deterministic)



b insertion
(undeterminizable)

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

$$T_1 \circ T_2 = \langle Q \times Q', \Sigma, \Sigma'', \Delta'', \langle q_0, q_0' \rangle \rangle, F \times F' \rangle$$

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)
    }*).
```


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

Division of Engineering and Applied Sciences
Harvard University

Why Weights?

FSAs have multiple paths

How to adjudicate?

- Notion of *best path*

Other applications:

- Numeric functions over strings
  - perfect hashing

Semirings

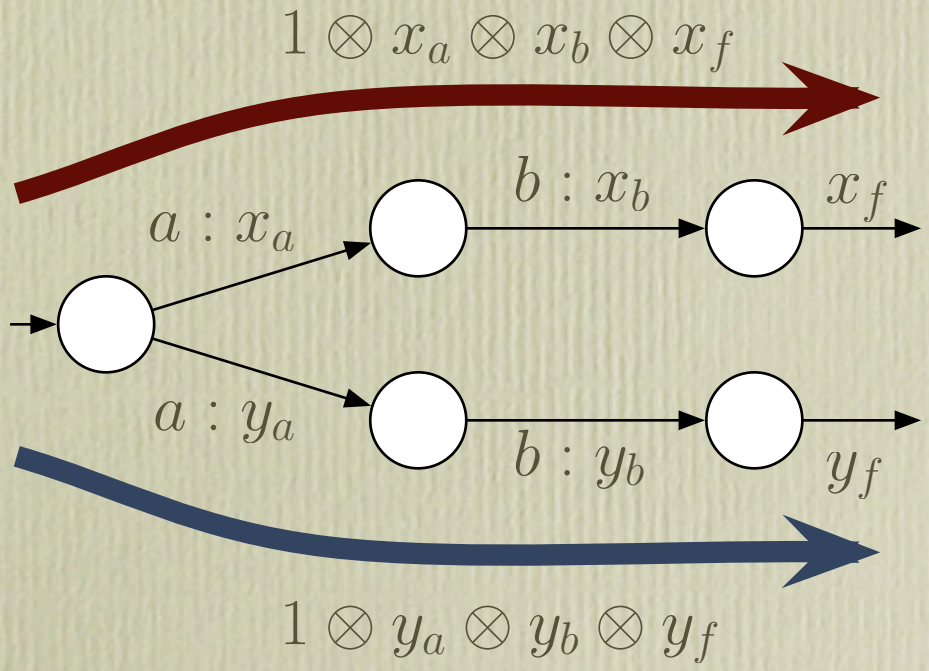
A set K along with operations of sum and product obeying the following algebraic laws:

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



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

Example Semiring: Strings

String sets form a semiring with:

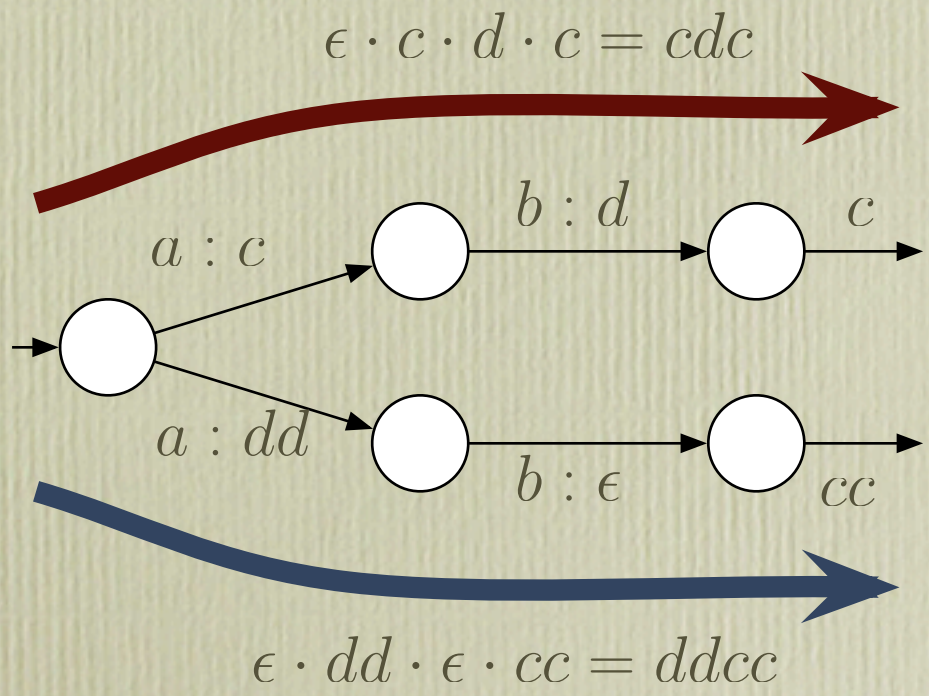
- Sum: union
- Product: concatenation
- 0: {} (the empty language)
- 1: {□} (the language containing the empty string)

Transducers can generate their output in any semiring.

String Semiring = FST

Product ( ): along paths

Sum (∪): among paths



$cdc \cup ddcc = \{cdc, ddcc\}$

Example Semiring: Probabilities

Probabilities $[0...1]$ form a semiring with

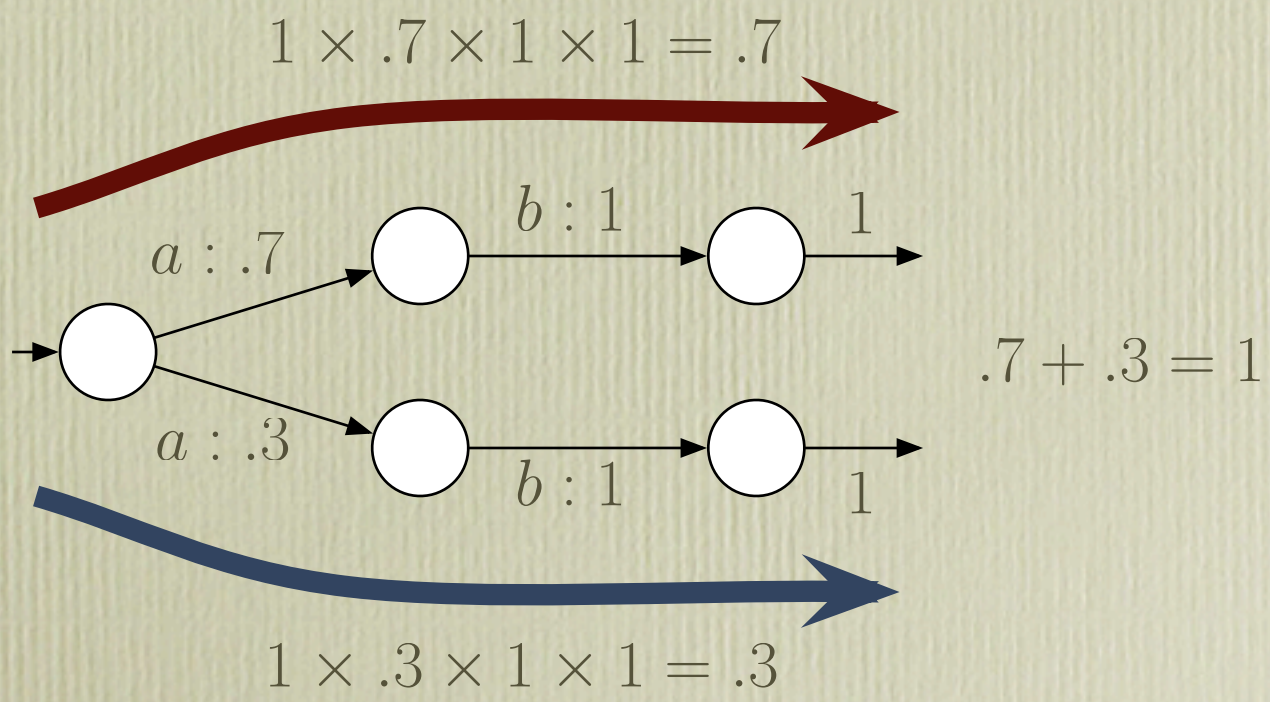
- Sum: $+$
- Product: $\times$
- 0: 0
- 1: 1

Weights place *relative values* on transitions, hence paths (as product of transitions), hence inputs (as sum over paths).

Probability Semiring = WFSA

Product (×): along paths

Sum (+): among paths

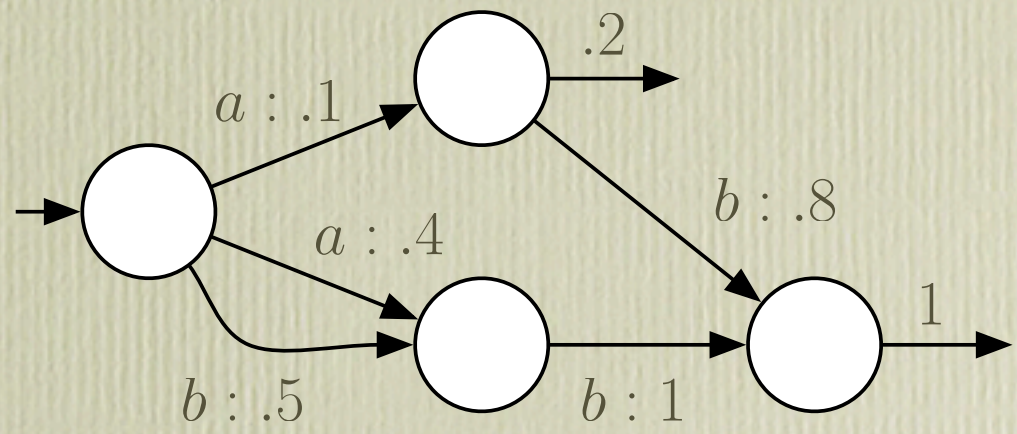


$.7 + .3 = 1$

Weighted Finite-State Automaton

Defines probability distribution over strings:

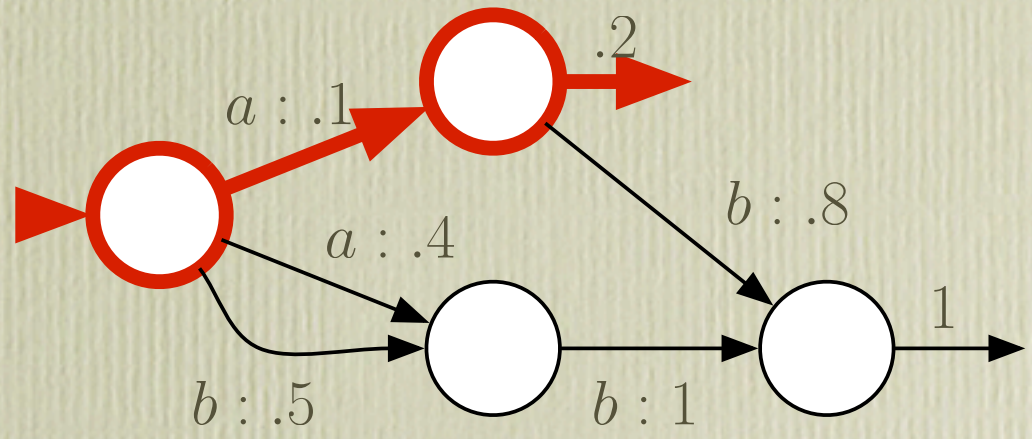
- *a*
- *bb*
- *ab*
- 1.0



Weighted Finite-State Automaton

Defines probability distribution over strings:

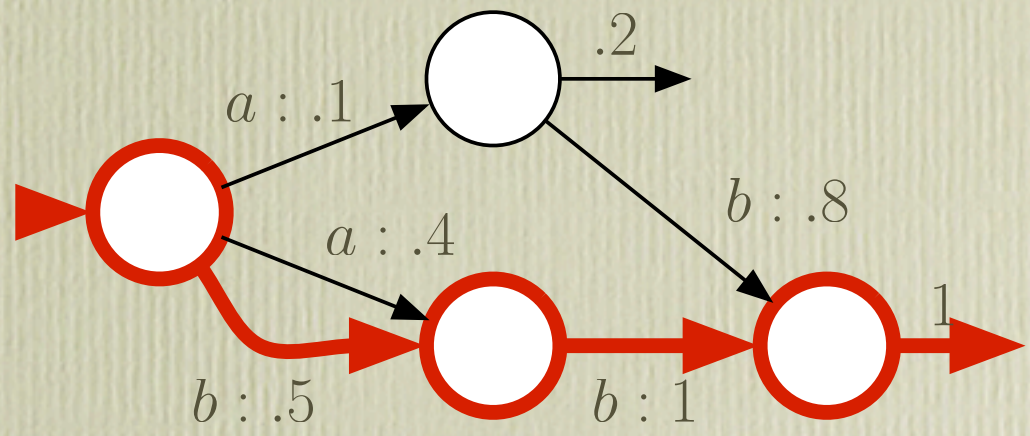
- *a* .02
- *bb*
- *ab*
- 1.0



Weighted Finite-State Automaton

Defines probability distribution over strings:

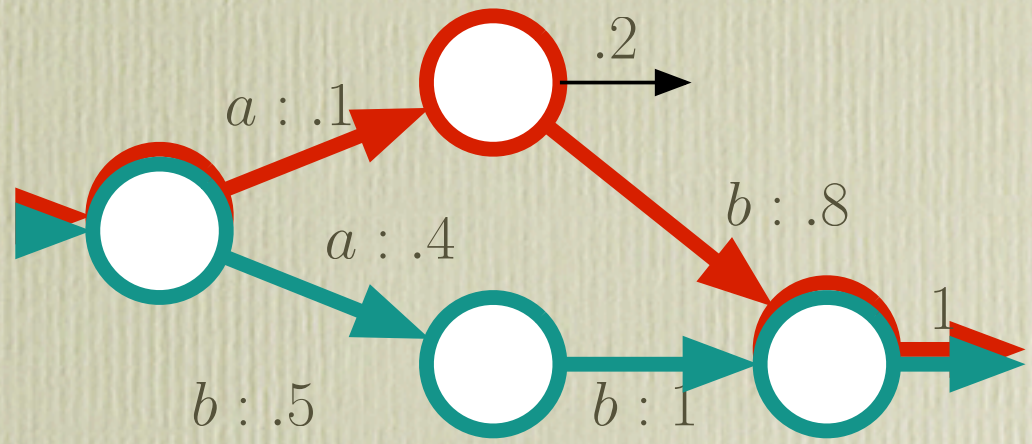
- *a* .02
- *bb* .5
- *ab*
- 1.0



Weighted Finite-State Automaton

Defines probability distribution over strings:

- *a* .02
- *bb* .5
- *ab* .48
- 1.0



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

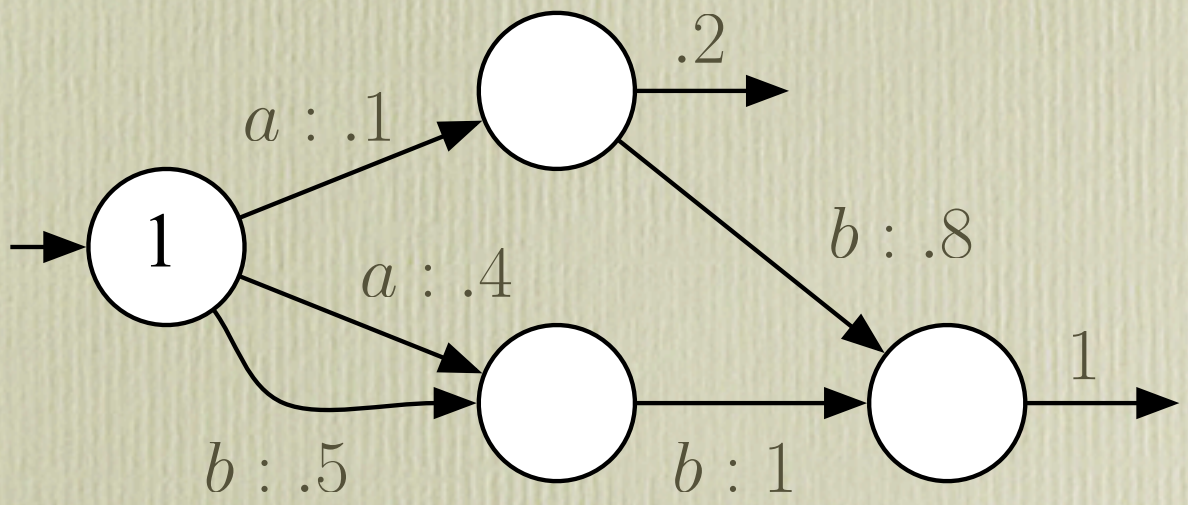
$$\delta_q(0) = \begin{cases} 1 & \text{if } q = q_0 \\ 0 & \text{otherwise} \end{cases}$$

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

  - for a given input, just intersect

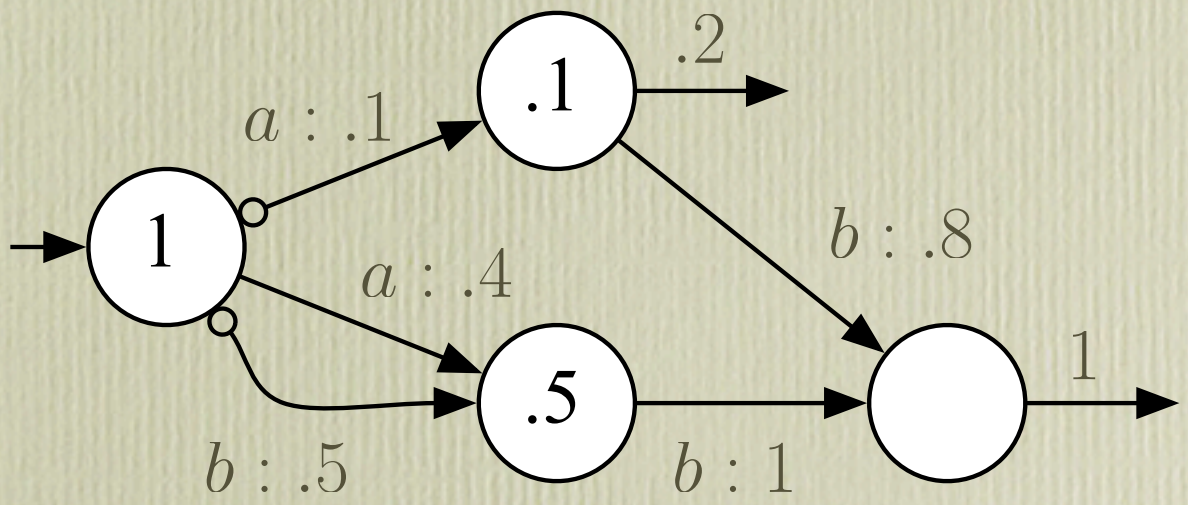
Best Path Computation

Example



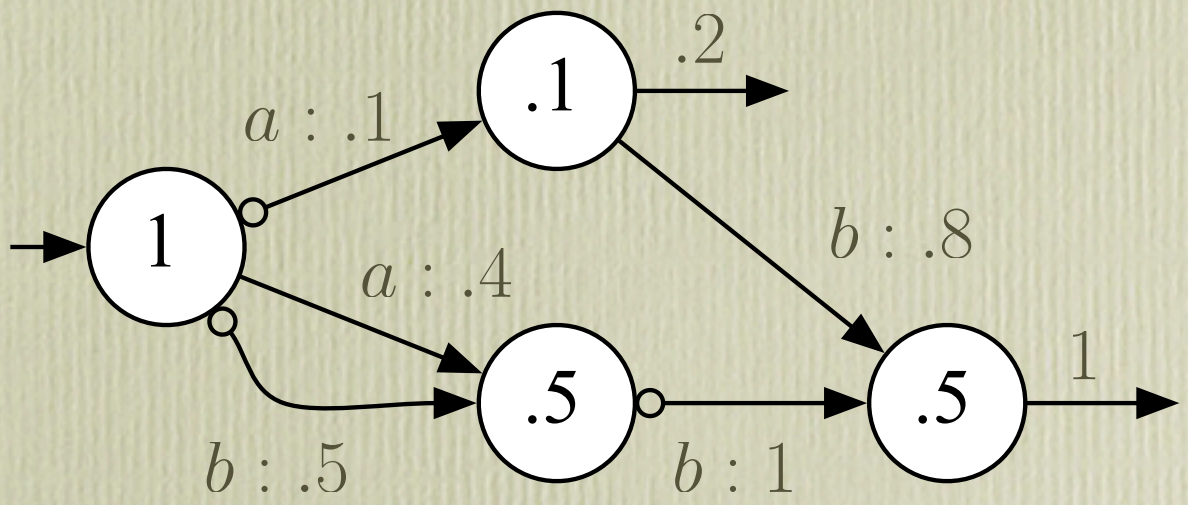
Best Path Computation

Example



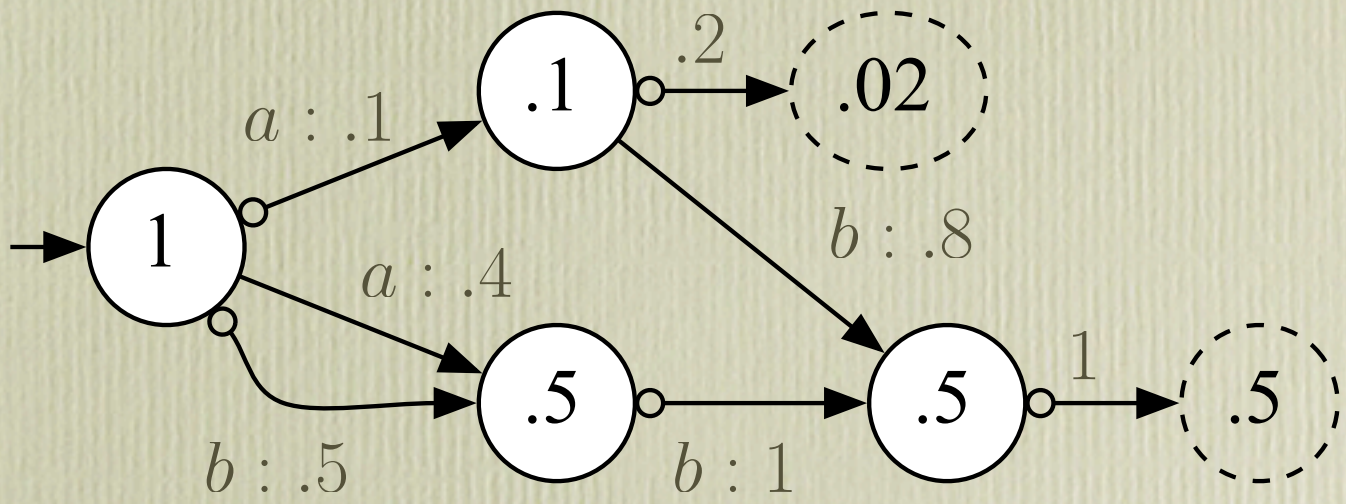
Best Path Computation

Example



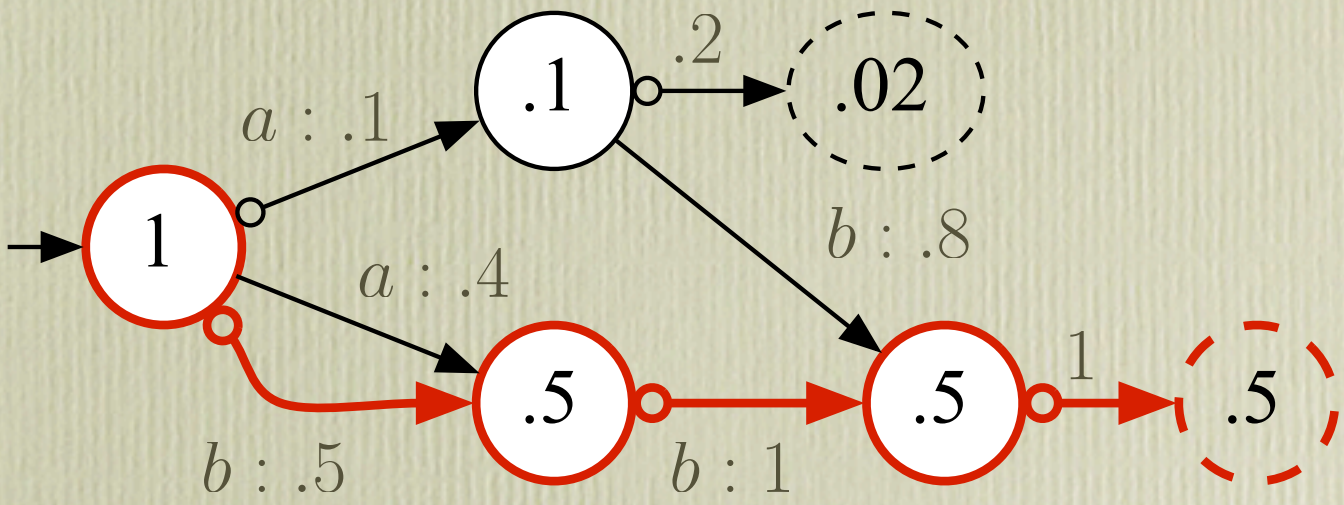
Best Path Computation

Example

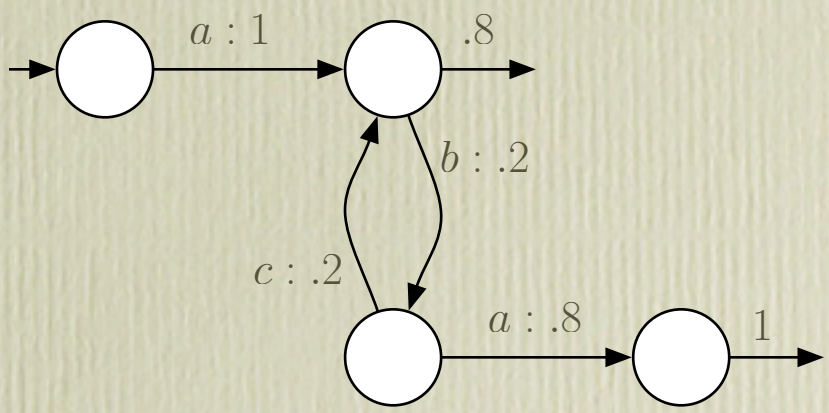


Best Path Computation

Example

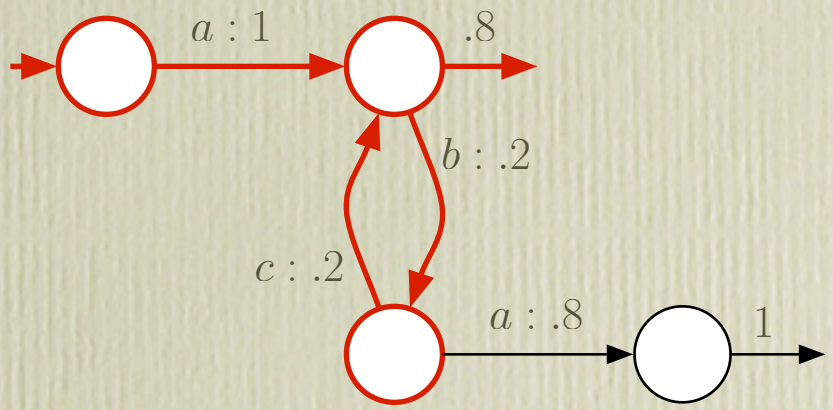


Weights Over Cycles



Weights Over Cycles

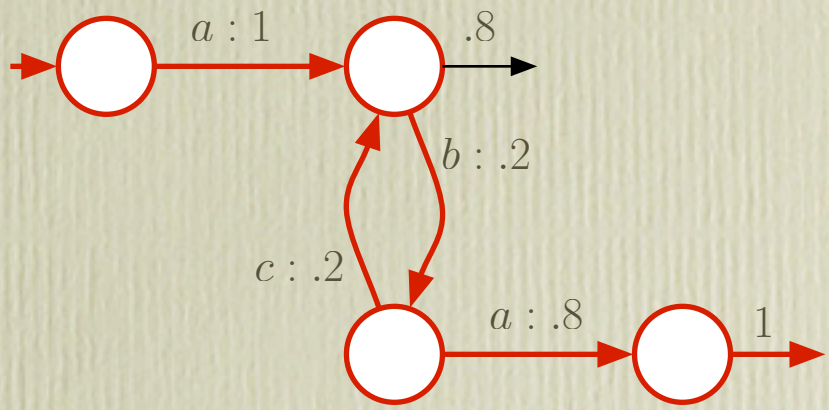
$$P(a(bc)^*) = \sum_{n=0}^{\infty} (.2 \cdot .2)^n .8 = .8 \frac{1}{1 - .04} = .8\overline{3}$$



Weights Over Cycles

$$P(a(bc)^*) = \sum_{n=0}^{\infty} (.2 \cdot .2)^n .8 = .8 \frac{1}{1 - .04} = .8\overline{3}$$

$$P(a(bc)^* ba) = \sum_{n=0}^{\infty} (.2 \cdot .2)^n \cdot .2 \cdot .8 = .16 \frac{1}{1 - .04} = .1\overline{6}$$



Application:

Language Modeling

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

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

Approximate under a Markovian assumption that words depend only on the previous $N-1$

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

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:

$$P(w_1 \mid w_{-N} \cdots w_0) = P(w_1 \mid \overbrace{\vdash \cdots \vdash}^{N-1 \text{ times}})$$

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 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

Word	*Count*	*1-gram MLE*
be	62	.276
to	41	.182
it	33	.147
sure	31	.138
let	27	.120
mine	17	.076
when	8	.036
is	6	.027
total	225	

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 to be sure let it be 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

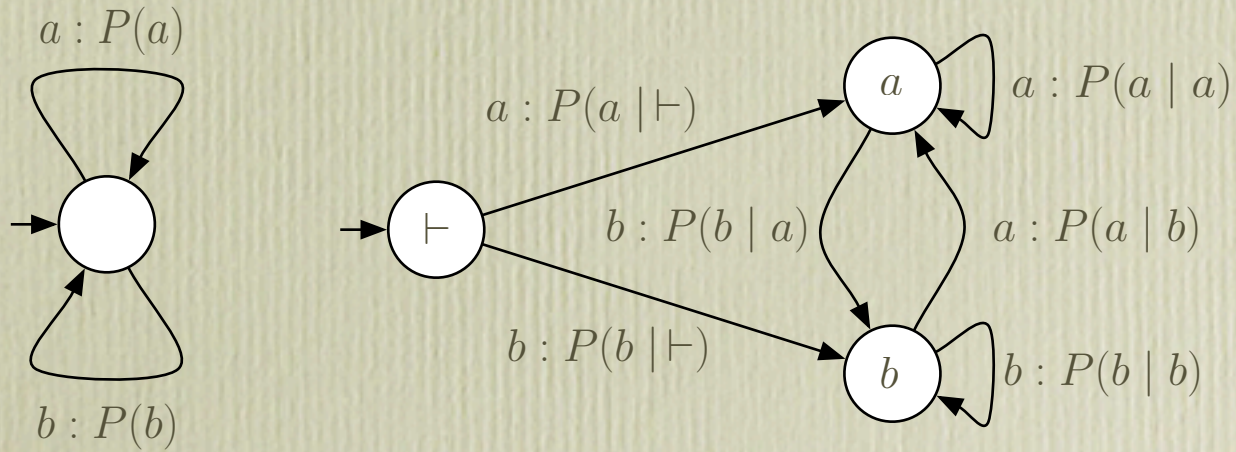
Trigram	*MLE Prob*
it be be	0
it be is	0
it be it	0
it be let	0.083
it be mine	0.125
it be sure	0
it be to	0.708
it be when	0.083

WFSA for N-gram Models

One state per (N–1-gram) conditioning context

Transitions among states to change context

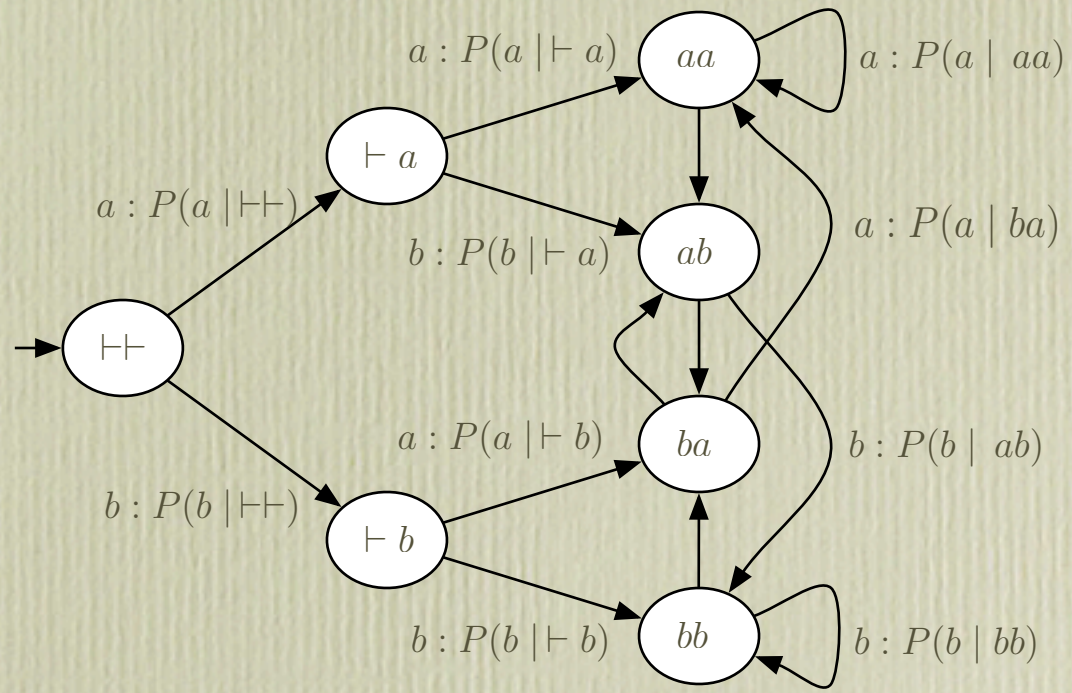
Start state is $\langle \vdash^{N-1} \rangle$



Unigram *Bigram*

Application: Language Modeling

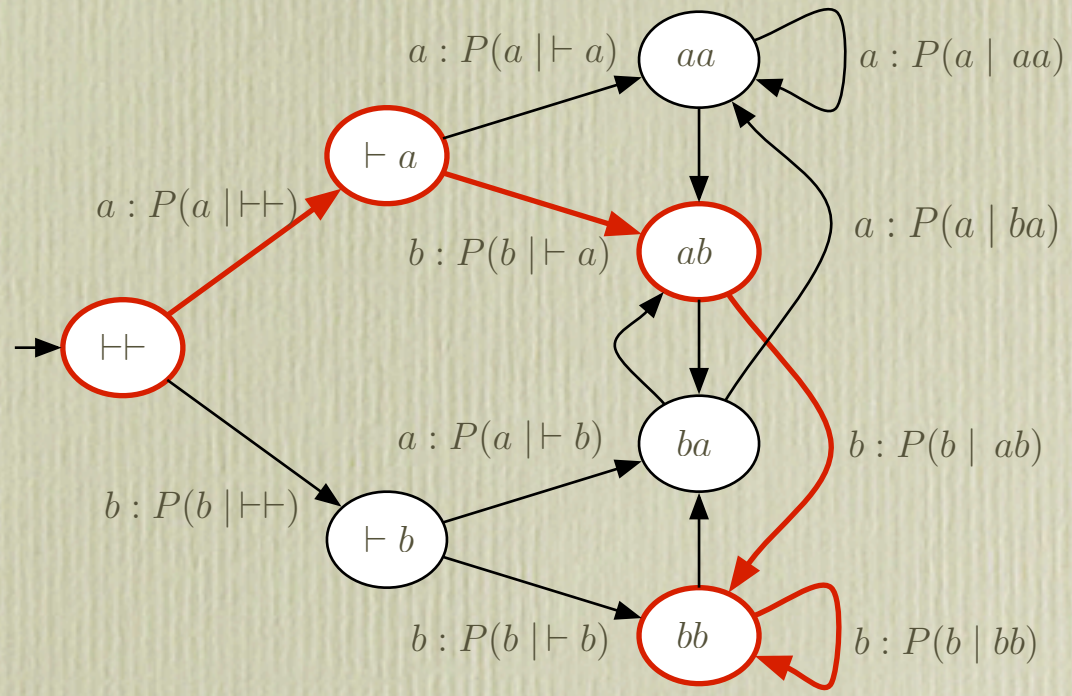
WFSA for N-gram Models



Trigram

Application: Language Modeling

WFSA for N-gram Models



$$P(abbb) = P(a \mid \vdash\vdash) \cdot P(b \mid \vdash a) \cdot P(b \mid ab) \cdot P(b \mid bb)$$

Application: Language Modeling

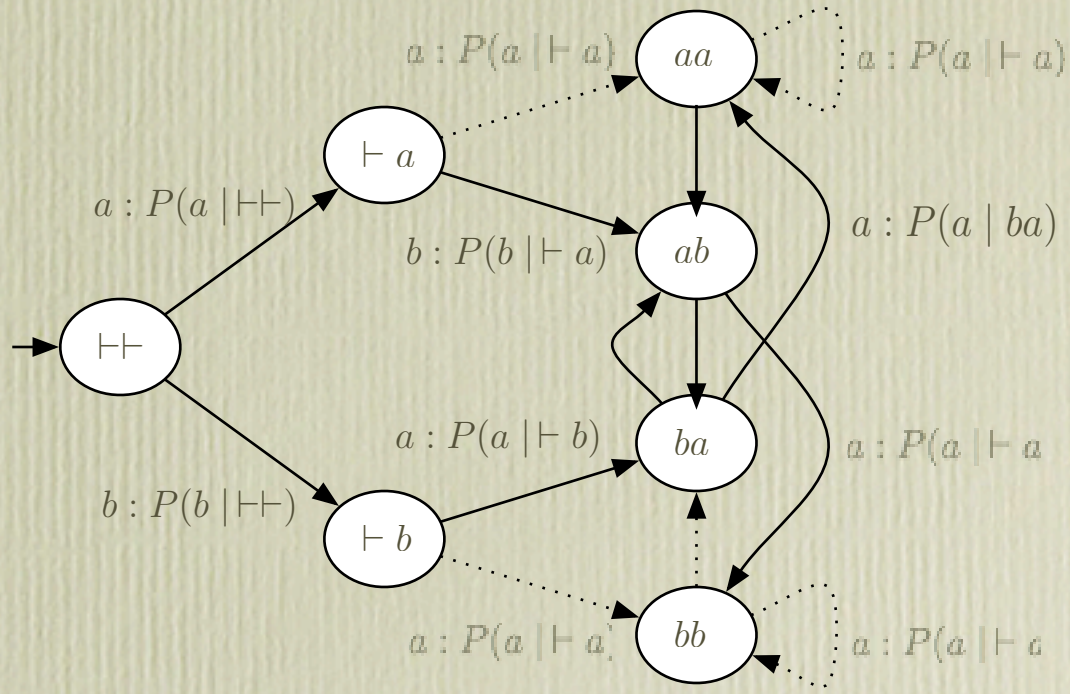
Smoothing

Training corpus:

- *abaab*
- *babb*

Test corpus:

- *aab*



Application: Language Modeling

Why Smoothing?

	Possible	*Attested*	*Sparsity*
trigrams	584	39	6.7%
bigrams	72	22	30.5%
unigrams	8	8	100%

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 to be sure let it be 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

Smoothing

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

First, reserve some probability mass

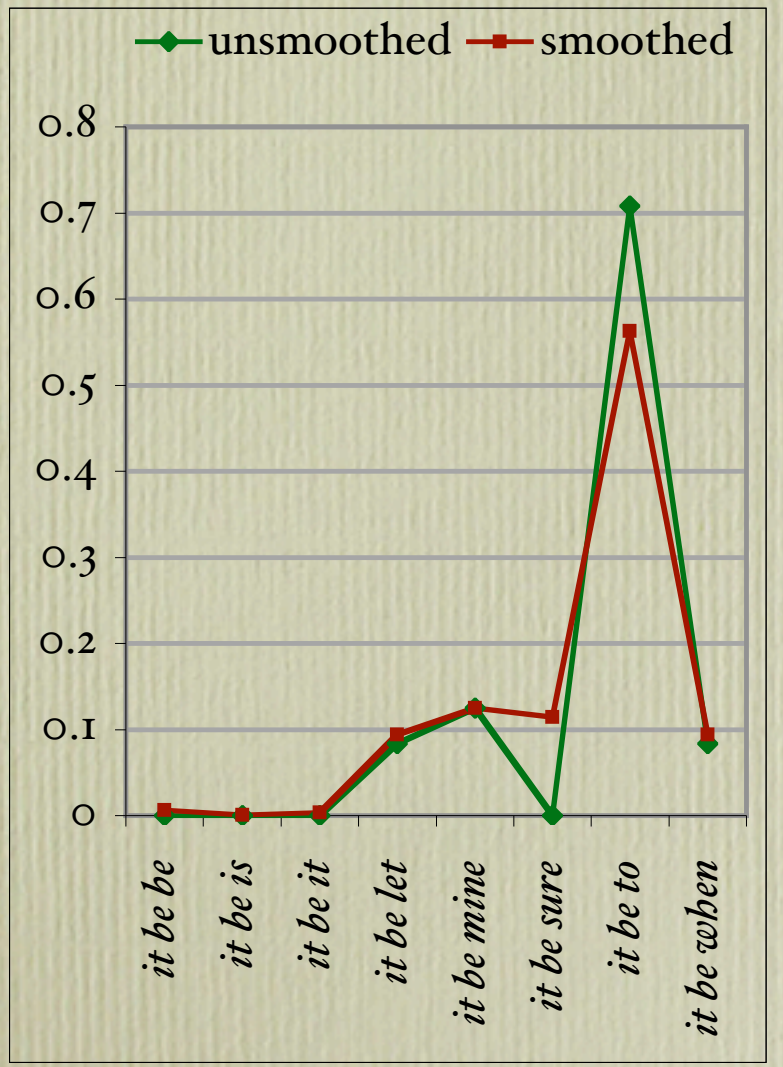
- Add-delta smoothing

Then, redistribute held-out mass to unseen events

- Katz back-off

Application: Language Modeling

Effect of Smoothing



Trigram	*Unsmoothed*	*Smoothed*
it be be	0	0.0066
it be is	0	0.0006
it be it	0	0.0035
it be let	0.083	0.0938
it be mine	0.125	0.1250
it be sure	0	0.1143
it be to	0.708	0.5625
it be when	0.083	0.0938

Reserving Probability

Add-delta smoothing

- add a fictitious "count" of δ 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 δ to each N-gram

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

Reserving Probability

Add-delta smoothing

- add a fictitious "count" of δ 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

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

Reserving Probability

trigram	*count*	*unsmoothed*	$\tilde{P}_3$
it be sure	0	0.0000	
it be when	2	0.0833	0.0938
it be is	0	0.0000	
it be it	0	0.0000	
it be let	2	0.0833	0.0938
it be mine	3	0.1250	0.1250
it be to	17	0.7083	0.5625
it be be	0	0.0000	
$\tilde{P}_{held}$			0.1250
$\tilde{P}_{back}$			
total	24	1.0000	1.0000

Application: Language Modeling

Distributing Probability

Katz backoff

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

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

Application: Language Modeling

Distributing Probability

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

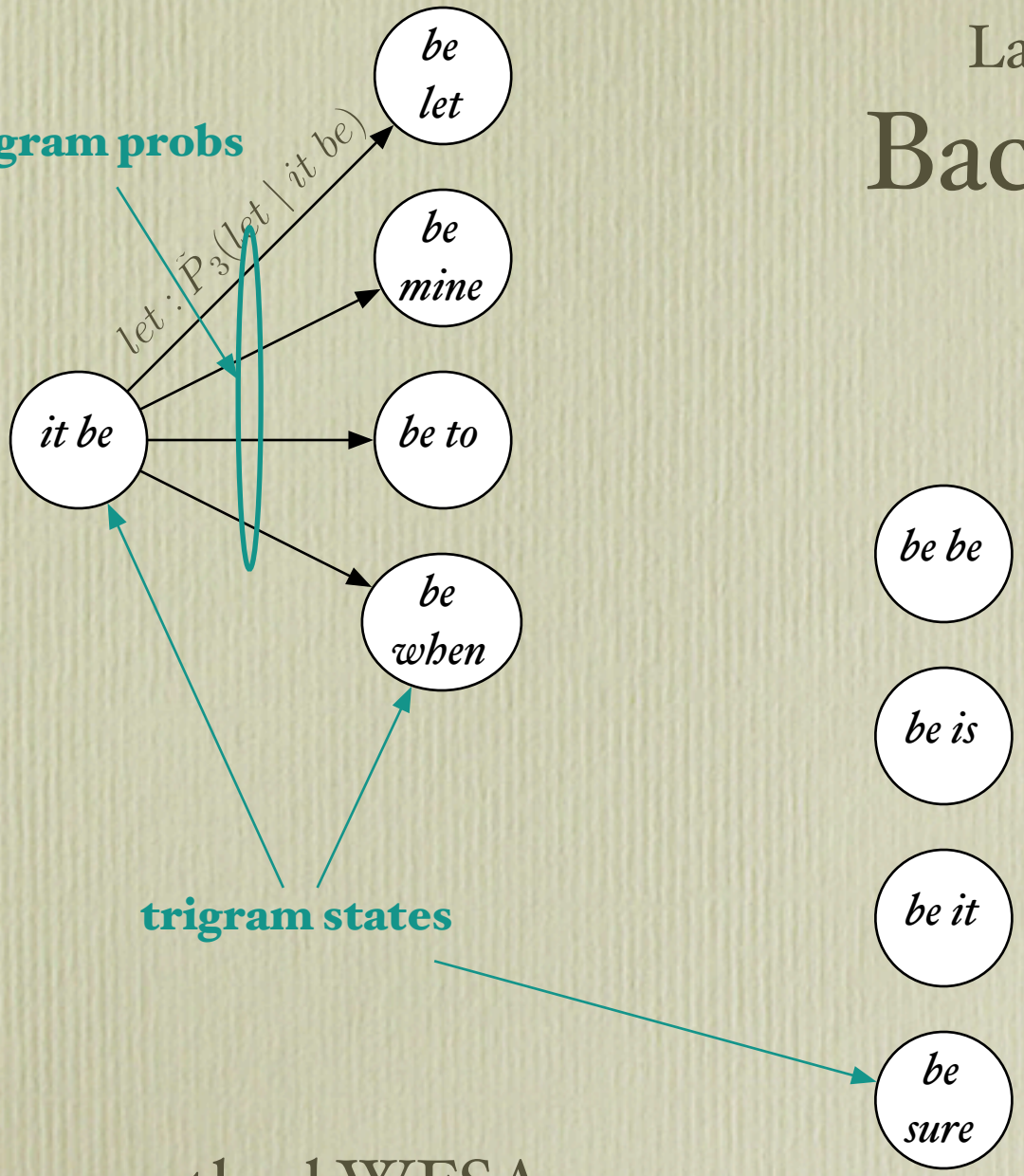
$$\tilde{P}_{held}(w_1 \cdots w_{N-1}) = 1 - \sum_{w_N : c(w_1 \cdots w_N) > 0} \tilde{P}(w_N \mid w_1 \cdots w_{N-1})$$

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

$$\alpha(w_1 \cdots w_{N-1}) = \frac{\tilde{P}_{held}(w_1 \cdots w_{N-1})}{\tilde{P}_{back}(w_1 \cdots w_{N-1})}$$

Application: Language Modeling

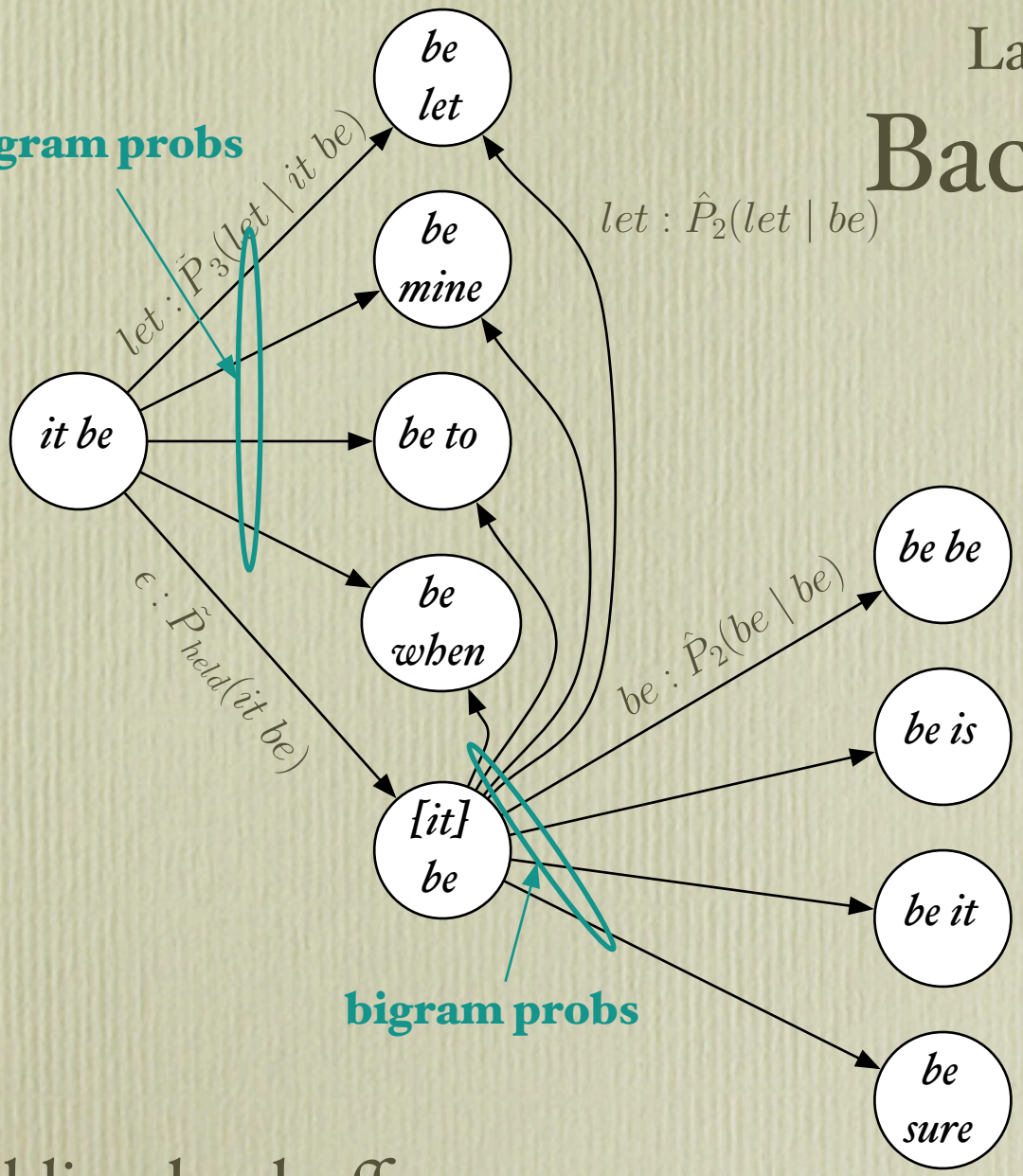
Backoff WFSA



Unsmoothed WFSA

Application:
Language Modeling

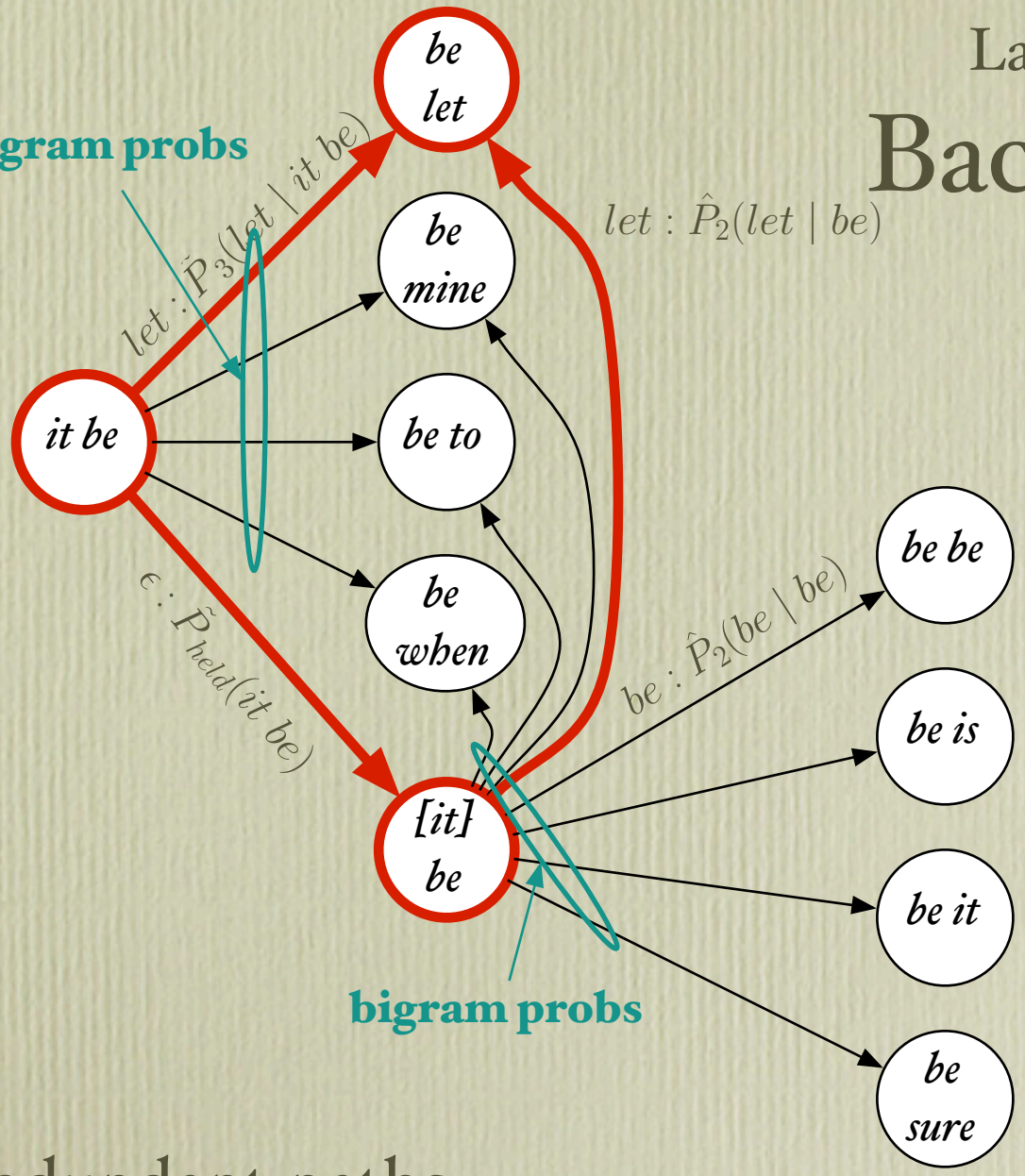
Backoff WFSA



Adding backoff

Application:
Language Modeling

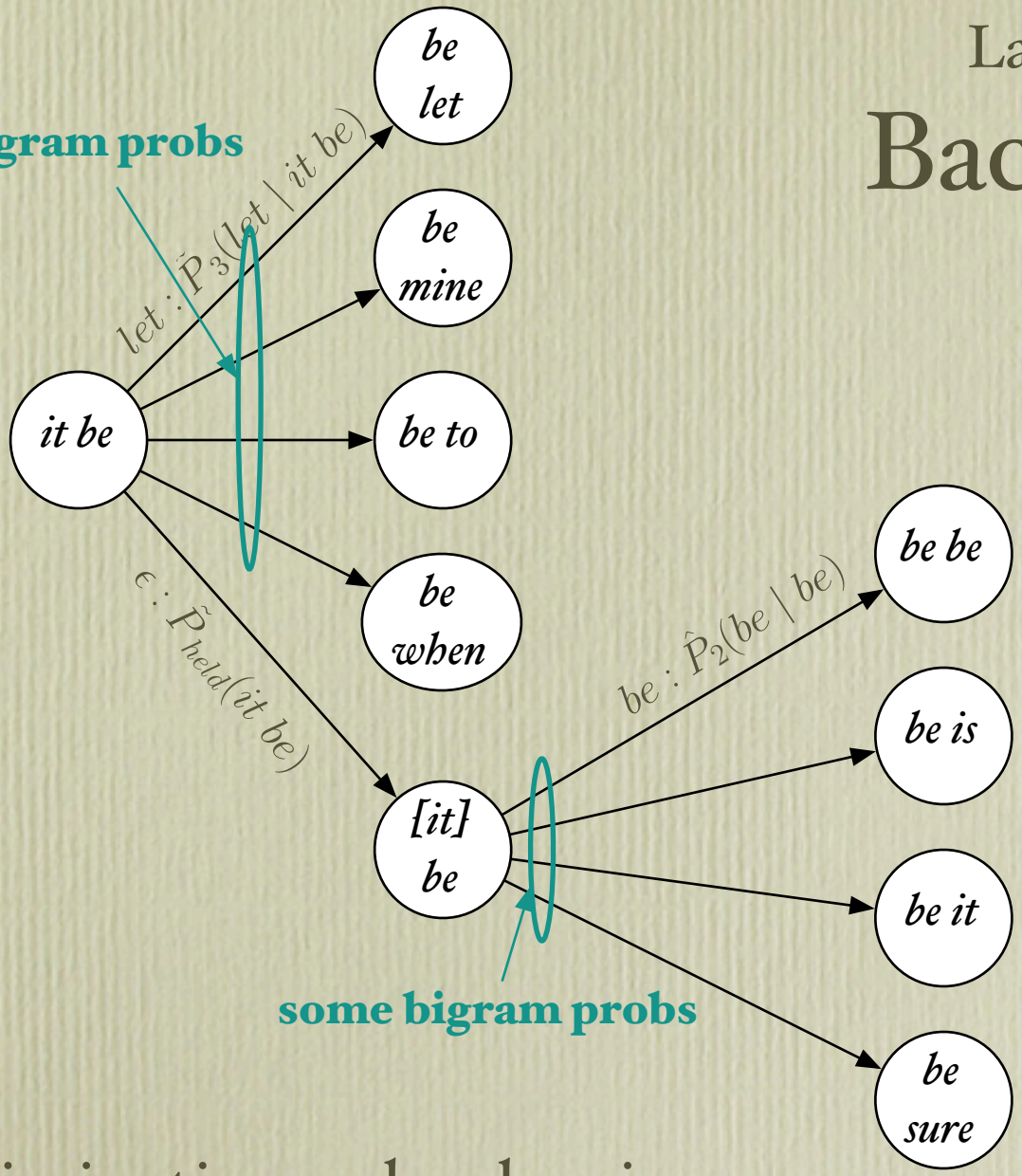
Backoff WFSA



Redundant paths

Application:
Language Modeling

Backoff WFSA

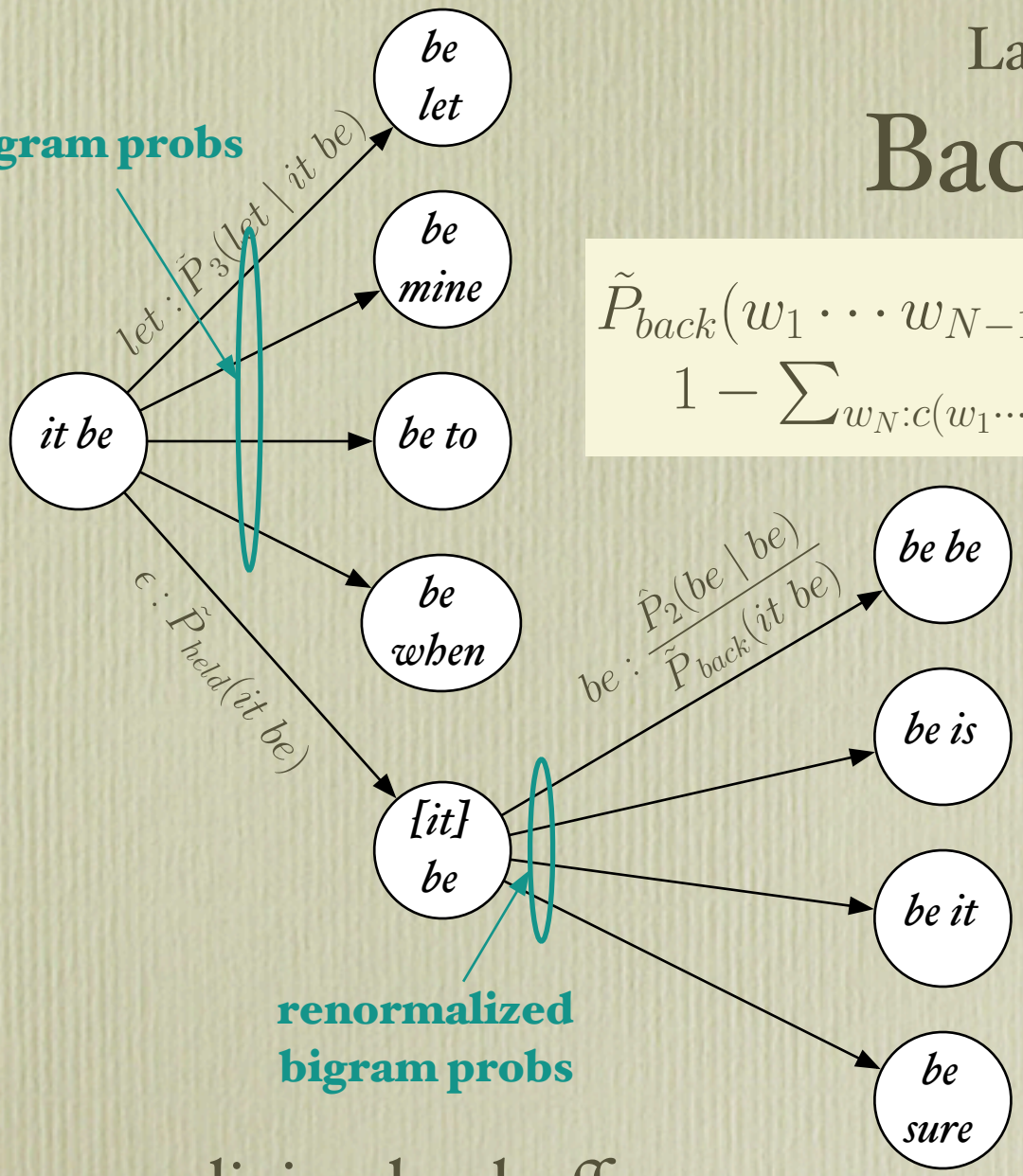


Eliminating redundancies

Application:
Language Modeling

Backoff WFSA

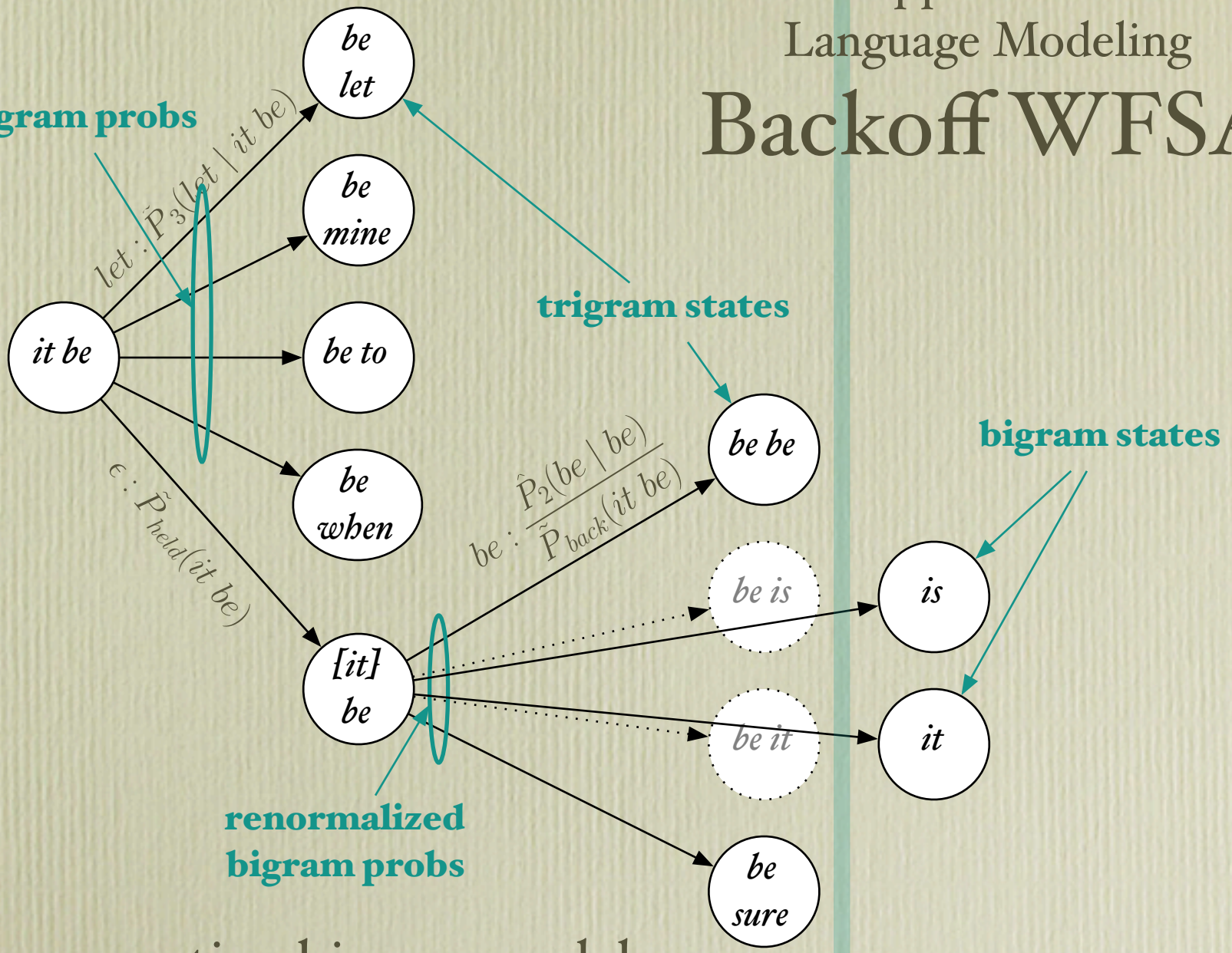
$$\tilde{P}_{back}(w_1 \cdots w_{N-1}) = 1 - \sum_{w_N : c(w_1 \cdots w_N) > 0} \tilde{P}(w_N \mid w_2 \cdots w_{N-1})$$



Renormalizing backoff

Application:
Language Modeling

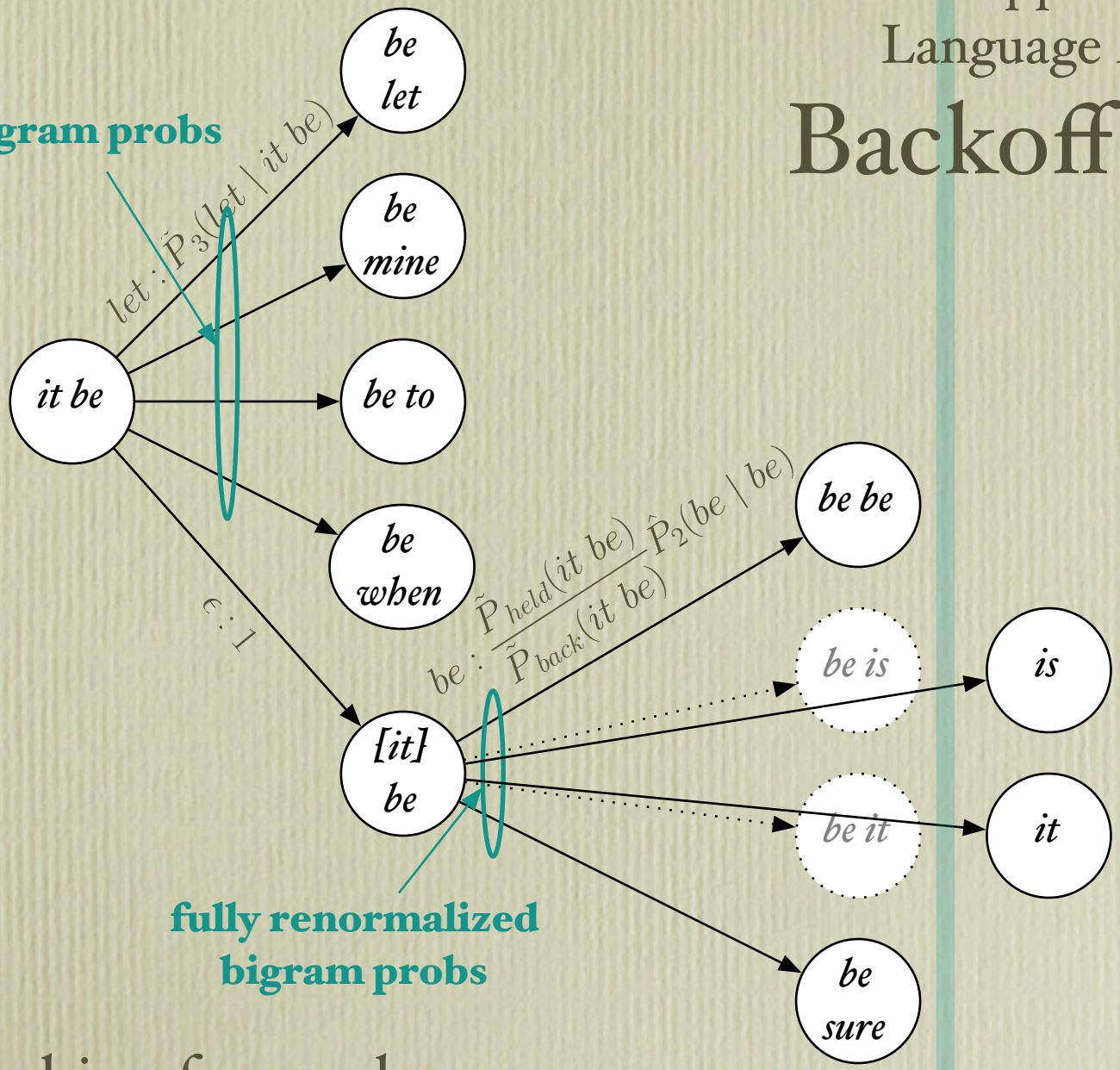
Backoff WFSA



Incorporating bigram model

Application:
Language Modeling

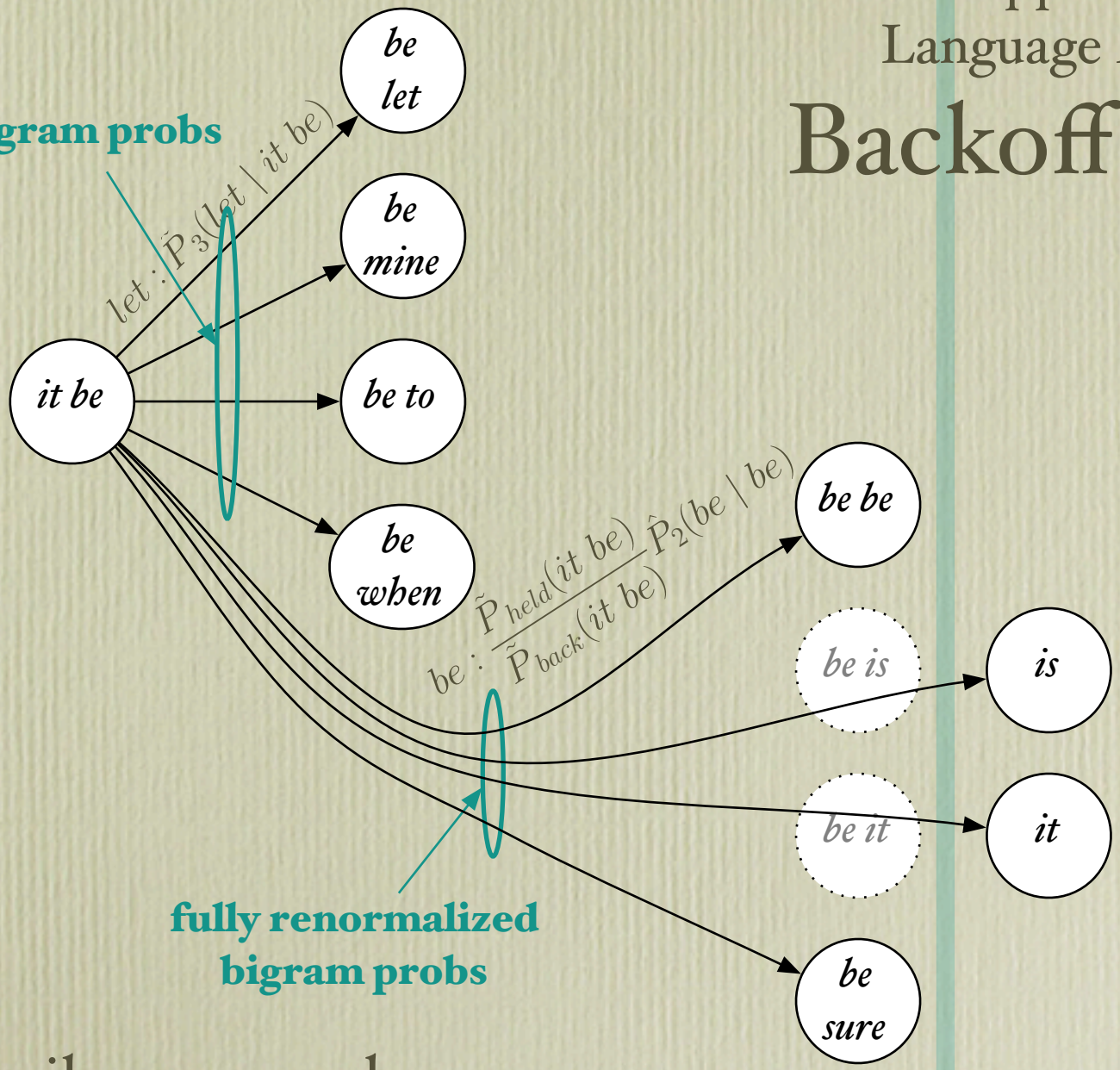
Backoff WFSA



Pushing forward

Application:
Language Modeling

Backoff WFSA



Epsilon removal

Application: Language Modeling

Distributing Probability

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

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

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

$$\alpha(w_1 \cdots w_{N-1}) = \frac{\tilde{P}_{held}(w_1 \cdots w_{N-1})}{\tilde{P}_{back}(w_1 \cdots w_{N-1})}$$

Sample Smoothing

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

Weighted Transducers

Combining output and weighting

Examples:

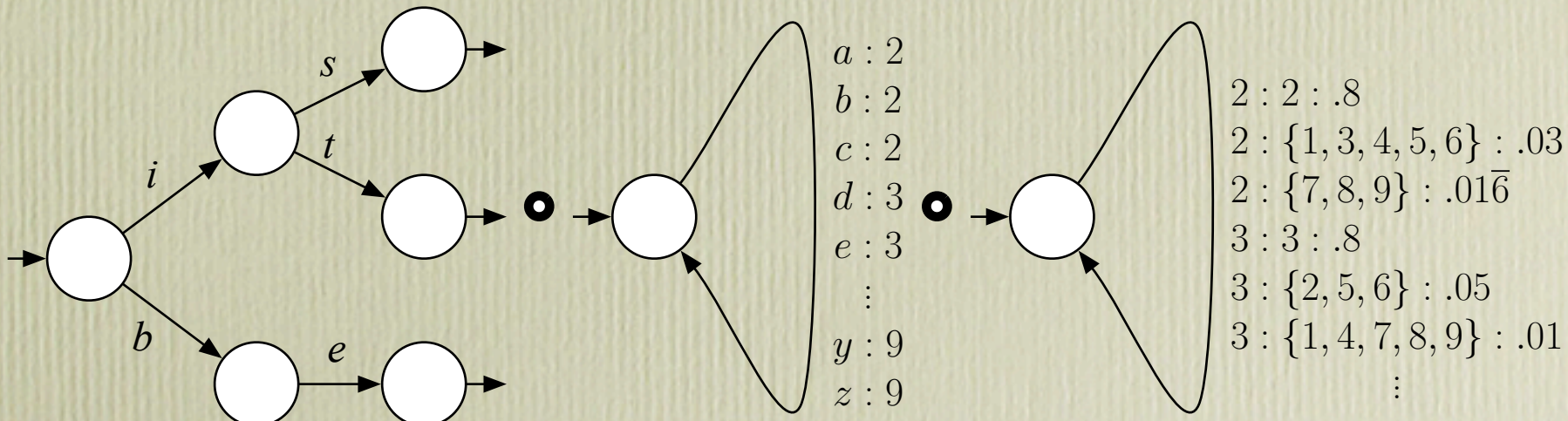
- typing models

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

‹an› ‹example› ‹of› ‹NUM› ‹words›

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)

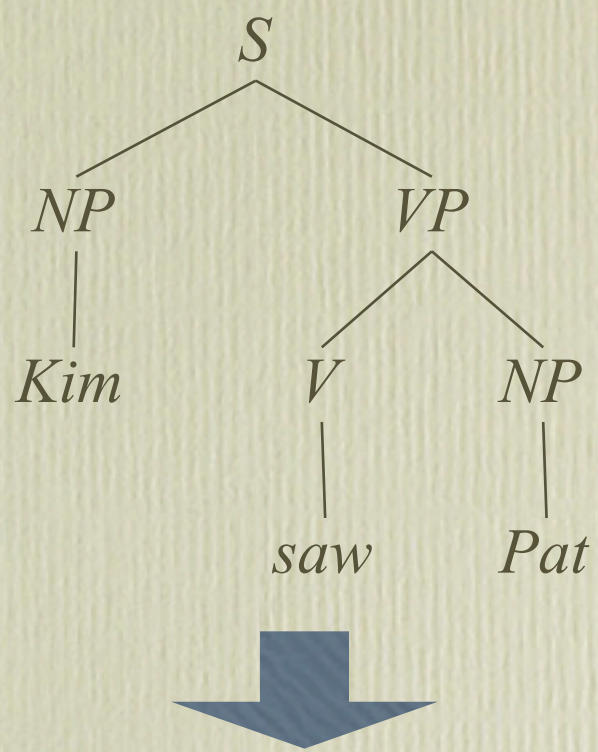
Tree Automata

Division of Engineering and Applied Sciences
Harvard University

Trees as Terms

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

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



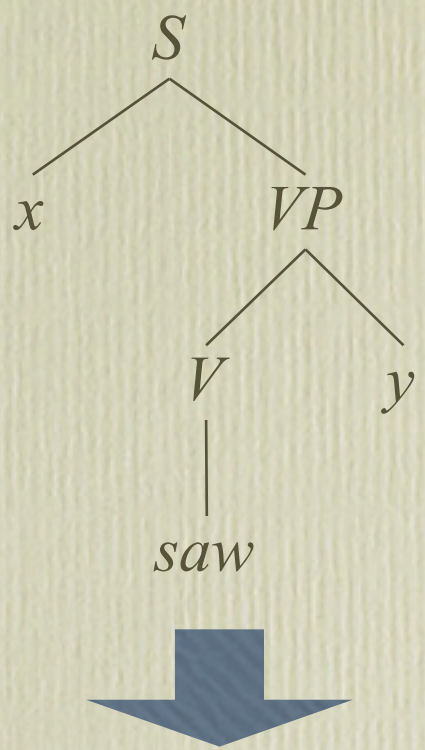
$$S(NP(Kim), \\ VP(V(saw), NP(Pat)))$$

Incomplete Trees

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

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

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



$$S(x, VP(V(saw), y))$$

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:
$f \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ for all $f \in \mathcal{F}$ such that $\mathit{arity}(f) = 0$;

Variables at leaves:
$x \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ for all $x \in \mathcal{X}$;

Internal nodes:
$f(t_1, \ldots, t_p) \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ for all $p \geq 1$ 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:

$$\mathcal{T}(\mathcal{X})$$

$$\mathcal{T}$$

n numerically ordered variables:

$$\mathcal{X}_n = \{x_1, \ldots, x_n\}$$

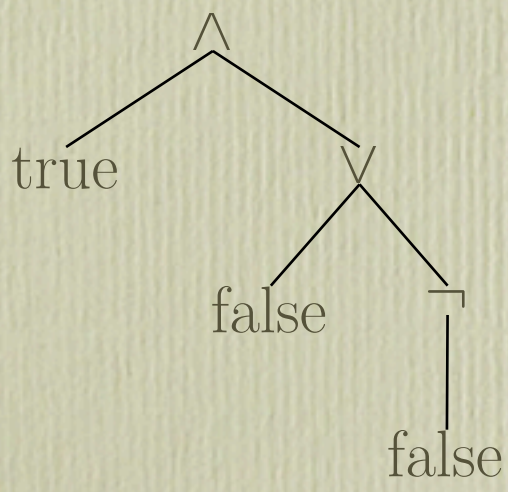
Example:

Propositional Formulae

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

$$\mathcal{F}_{prop} = \{\wedge_2, \vee_2, \neg_1, \textsc{true}_0, \textsc{false}_0\}$$

(Arities are given in the subscripts.)



$\text{true} \wedge (\text{false} \vee \neg\text{false})$

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:

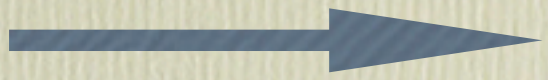
$$(f_m(u_1, \ldots, u_m))[t_1, \ldots, t_n] = f_m(u_1[t_1, \ldots, t_n], \ldots, u_m[t_1, \ldots, t_n])$$

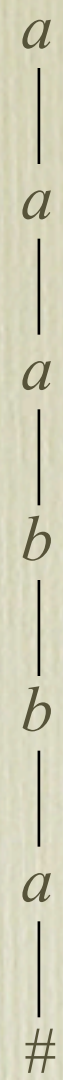
$$x_i[t_1, \ldots, t_n] = t_i$$

Strings as Trees

Ranked alphabet

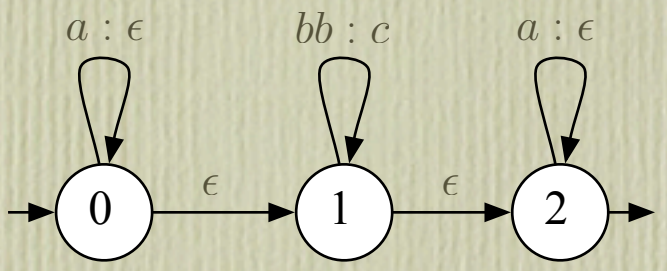
- Vocabulary as unary symbols
- End marker (#) as sole nullary symbol
-

$aaabbaa$ 



$a(a(a(b(b(a(\#))))))$ =

Finite-State Derivations



$$
\begin{aligned}
0\,a\cdots &\rightarrow 0\cdots \\
0\cdots &\rightarrow 1\cdots \\
1\,bb\cdots &\rightarrow c\,1\cdots \\
1\cdots &\rightarrow 2\cdots \\
2\,a\cdots &\rightarrow 2\cdots \\
2\,\sharp &\rightarrow \sharp
\end{aligned}
$$

A derivation:

$$
\begin{aligned}
0\,aaabba\sharp &\rightarrow 0\,aabba\sharp \\
&\rightarrow 0\,abba\sharp \\
&\rightarrow 0\,bba\sharp \\
&\rightarrow 1\,bba\sharp \\
&\rightarrow c\,1\,a\sharp \\
&\rightarrow c\,2\,a\sharp \\
&\rightarrow c\,2\,\sharp \\
&\rightarrow c\,\sharp
\end{aligned}
$$

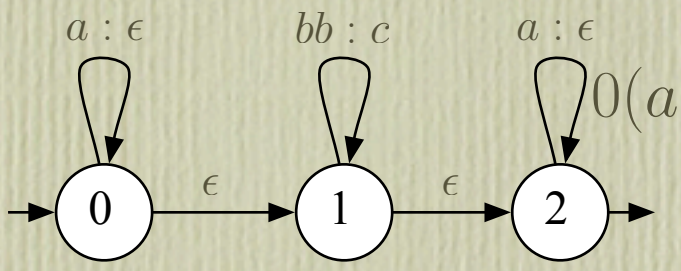
Definition of recognition:

accept $s : t$ if and only if

$$0\,s\sharp \rightarrow^* t\sharp$$

Division of Engineering and Applied Sciences
Harvard University

Strings as Trees



$$
\begin{aligned}
0(a(x)) &\rightarrow 0(x) \\
0(x) &\rightarrow 1(x) \\
1(b(b(x))) &\rightarrow c(1(x)) \\
1(x) &\rightarrow 2(x) \\
2(a(x)) &\rightarrow 2(x) \\
2(\sharp) &\rightarrow \sharp
\end{aligned}
$$

A derivation:

$$
\begin{aligned}
0(a(a(a(b(b(a(\sharp))))))) &\rightarrow 0(a(a(b(b(a(\sharp)))))) \\
&\rightarrow 0(a(b(b(a(\sharp))))) \\
&\rightarrow 0(b(b(a(\sharp)))) \\
&\rightarrow 1((b(b(a(\sharp))))) \\
&\rightarrow c(1(a(\sharp))) \\
&\rightarrow c(2(a(\sharp))) \\
&\rightarrow c(2(\sharp)) \\
&\rightarrow c(\sharp)
\end{aligned}
$$

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 Q, and variables $\mathcal{X}$, notated $\mathcal{T}^Q(\mathcal{F}, \mathcal{X})$, is the set of trees $q(t)$ where $q \in Q$ and $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$.

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

- **Nullary symbols at leaves:** $f \in \mathcal{T}_Q(\mathcal{F}, \mathcal{X})$ for all $f \in \mathcal{F}$ such that $\mathit{arity}(f) = 0$;
- **States over variables at leaves:** $q(x) \in \mathcal{T}_Q(\mathcal{F}, \mathcal{X})$ for all $x \in \mathcal{X}$ and $q \in Q$;
- **Internal nodes:** $f(t_1, \ldots, t_p) \in \mathcal{T}_Q(\mathcal{F}, \mathcal{X})$ for all $p \geq 1$ and $t_1, \ldots, t_p \in \mathcal{T}_Q(\mathcal{F}, \mathcal{X})$.

Examples

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

$$\mathcal{T}^Q(\mathcal{F}_{fg}, \mathcal{X}_3) \supset \{\, q_0(x_3)$$
$$q_1(f(g(x_1, x_2), x_3))$$
$$q_0(a)\ \}$$

$$\mathcal{T}_Q(\mathcal{F}_{fg}, \mathcal{X}_3) \supset \{\, q_0(x_3)$$
$$f(g(q_0(x_1), q_0(x_2)), q_1(x_3))$$
$$a\ \}$$

Tree Linearity and Height

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

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

$$\begin{aligned} height(x) &= 0 \text{ for all } x \in \mathcal{X} \\ height(f) &= 1 \text{ for all } f \in \mathcal{F} \text{ where } arity(f) = 0 \\ height(f(t_1, \ldots, t_n)) &= 1 + \max_{i=1}^{n} height(t_i) \\ &\quad \text{for all } f \in F \text{ where } arity(f) = n \geq 1 \end{aligned}$$

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

Simple Subclasses of Trees

The consecutively numbered linear upper trees of height 1

$$\mathsf{T}^Q$$

are of the form

$$q(f_n(x_1, \ldots, x_n))$$

The consecutively numbered linear lower trees of height 1

$$\mathsf{T}_Q$$

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.

$$\begin{aligned} 0(a(x)) &\to 0(x) \\ 0(x) &\to 1(x) \\ 1(b(b(x))) &\to c(1(x)) \\ 1(x) &\to 2(x) \\ 2(a(x)) &\to 2(x) \\ 2(\sharp) &\to \sharp \end{aligned}$$

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

We notate transitions

$$q(f_n(x_1, \ldots, x_n)) \longrightarrow f_n(q_1(x_1), \ldots, q_n(x_n))$$

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

$$t' = C[u'[u_1, \ldots, u_m]] \quad .$$

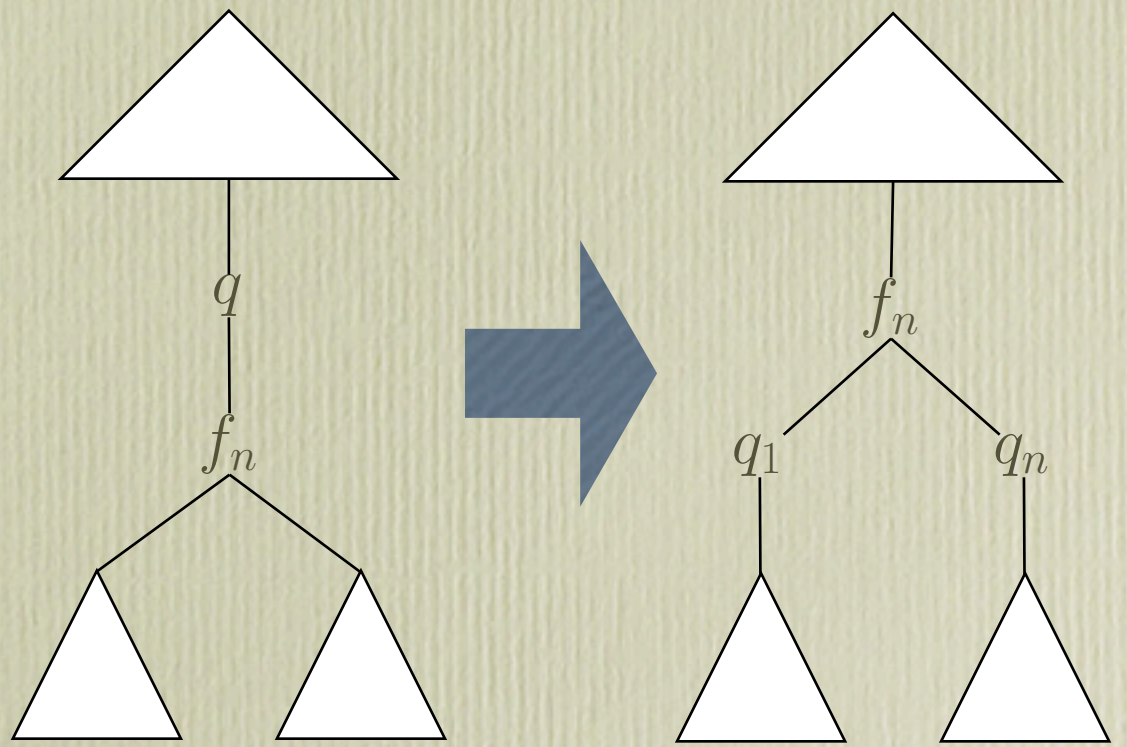
A tree $t \in \mathcal{T}(\mathcal{F})$ is accepted by an NTTA just in case

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

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

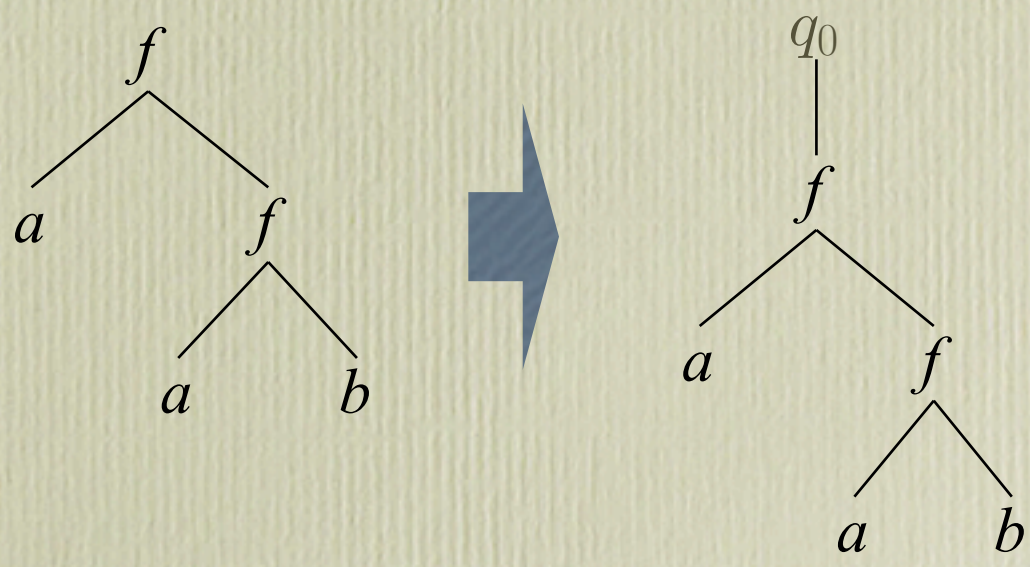
Tree Automaton Derivation

$$q(f_n(x_1, \ldots, x_n)) \rightarrow f_n(q_1(x_1), \ldots, q_n(x_n))$$



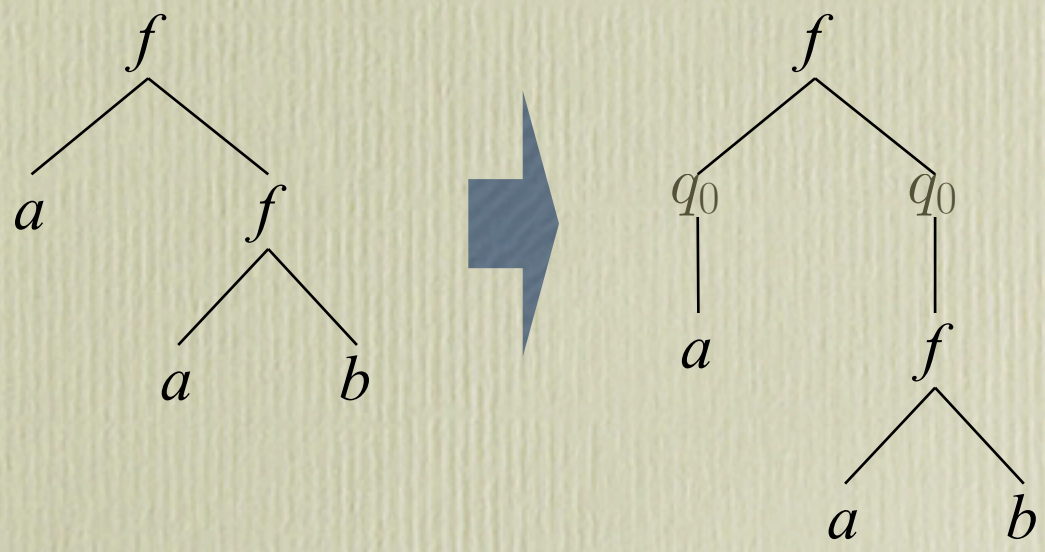
Example NTTA

- $q_0(f(x_1, x_2)) \rightarrow f(q_0(x_1), q_0(x_2))$
  $q_0(a) \rightarrow a$
  $q_0(b) \rightarrow b$



Example NTTA

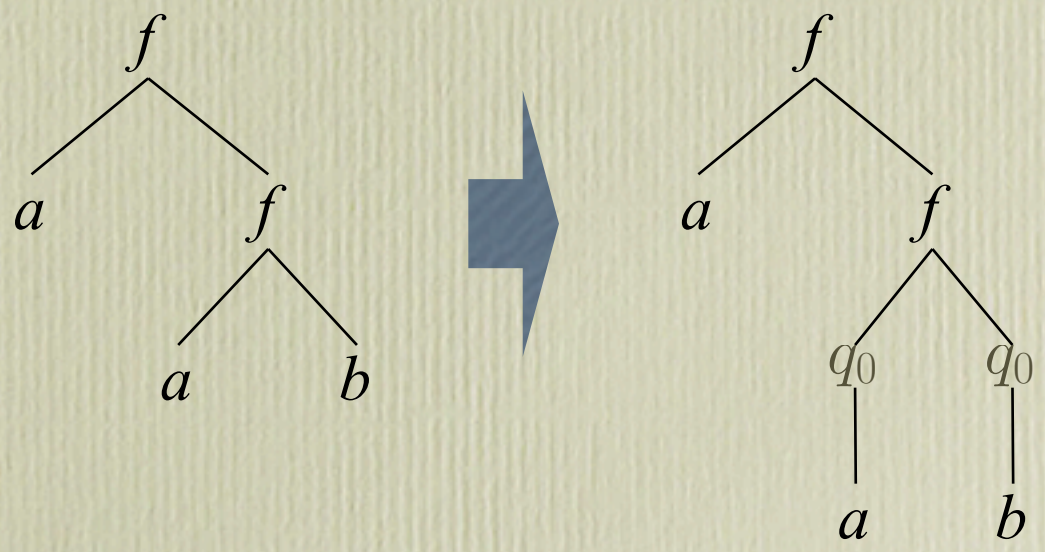
- $q_0(f(x_1, x_2)) \rightarrow f(q_0(x_1), q_0(x_2))$
- $q_0(a) \rightarrow a$
  $q_0(b) \rightarrow b$



Example NTTA

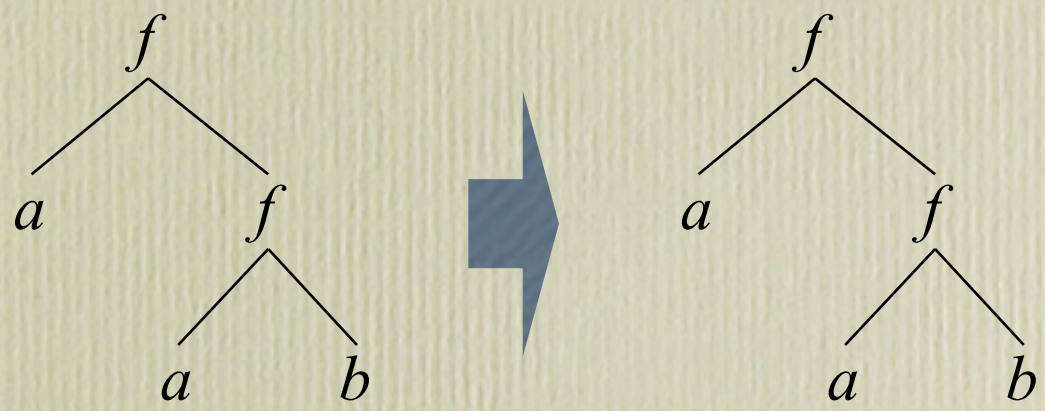
$q_0(f(x_1, x_2)) \rightarrow f(q_0(x_1), q_0(x_2))$

- $q_0(a) \rightarrow a$
- $q_0(b) \rightarrow b$



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

$$
\begin{aligned}
q_t(\text{true}) &\rightarrow \text{true} \\
q_f(\text{false}) &\rightarrow \text{false}
\end{aligned}
$$

$$
\begin{aligned}
q_t(\neg x_1) &\rightarrow q_f(x_1) \\
q_f(\neg x_1) &\rightarrow q_t(x_1)
\end{aligned}
$$

$$
\begin{aligned}
q_t(x_1 \wedge x_2) &\rightarrow q_t(x_1) \wedge q_t(x_2) \\
q_f(x_1 \wedge x_2) &\rightarrow q_t(x_1) \wedge q_f(x_2) \\
q_f(x_1 \wedge x_2) &\rightarrow q_f(x_1) \wedge q_t(x_2) \\
q_f(x_1 \wedge x_2) &\rightarrow q_f(x_1) \wedge q_f(x_2)
\end{aligned}
$$

$$
\begin{aligned}
q_t(x_1 \vee x_2) &\rightarrow q_t(x_1) \vee q_t(x_2) \\
q_t(x_1 \vee x_2) &\rightarrow q_t(x_1) \vee q_f(x_2) \\
q_t(x_1 \vee x_2) &\rightarrow q_f(x_1) \vee q_t(x_2) \\
q_f(x_1 \vee x_2) &\rightarrow q_f(x_1) \vee q_f(x_2)
\end{aligned}
$$

$$
\begin{aligned}
& q_t(\text{false} \vee \neg\text{false}) \\
\rightarrow\ & q_f(\text{false}) \vee q_t(\neg\text{false}) \\
\rightarrow\ & \text{false} \vee q_t(\neg\text{false}) \\
\rightarrow\ & \text{false} \vee \neg q_f(\text{false}) \\
\rightarrow\ & \text{false} \vee \neg\text{false}
\end{aligned}
$$

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;
- $\Delta \in \mathsf{T}^Q(\mathcal{F}, \mathcal{X}_n) \times \mathsf{T}_Q(\mathcal{F}, \mathcal{X}_n)$ 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

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

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 $\langle Q, \mathcal{F}, \Delta, Q_f \rangle$ where

- Q is a finite set of states;
- $\mathcal{F}$ is a ranked alphabet;
- $\Delta \in \mathsf{T}_Q(\mathcal{F}, \mathcal{X}_n) \times \mathsf{T}^Q(\mathcal{F}, \mathcal{X}_n)$ is a set of transitions;
- $Q_f \subseteq Q$ is a distinguished set of final states.

A tree $t \in \mathcal{T}(\mathcal{F})$ 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

$$
\begin{aligned}
&q(f(x,x)) \to f(q_a(x), q_a(x)) \\
&q(f(x,x)) \to f(q_b(x), q_b(x)) \\
&q_a(a) \to a \\
&q_b(b) \to b
\end{aligned}
$$

$$\{f(a,a), f(b,b)\}$$

Tree Transducers

Division of Engineering and Applied Sciences
Harvard University

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;
- $\Delta \in \mathsf{T}^Q(\mathcal{F}, \mathcal{X}_n) \times \mathsf{T}_Q(\mathcal{F}, \mathcal{X}_n)$ 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

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

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

Top-Down Tree Transducers

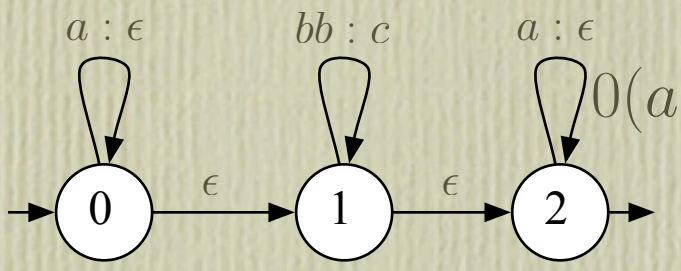
A nondeterministic top-down tree transducer (NTTT) is a tuple $\langle Q, \mathcal{F}, \Delta, q_0 \rangle$ where

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

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

$$q_0(s) \rightarrow^* t \qquad .$$

Example NTTT (FST)



$$
\begin{aligned}
0(a(x)) &\to 0(x) \\
0(x) &\to 1(x) \\
1(b(b(x))) &\to c(1(x)) \\
1(x) &\to 2(x) \\
2(a(x)) &\to 2(x) \\
2(\sharp) &\to \sharp
\end{aligned}
$$

A derivation:

$$
\begin{aligned}
0(a(a(a(b(b(a(\sharp))))))) &\to 0(a(a(b(b(a(\sharp)))))) \\
&\to 0(a(b(b(a(\sharp))))) \\
&\to 0(b(b(a(\sharp)))) \\
&\to 1((b(b(a(\sharp))))) \\
&\to c(1(a(\sharp))) \\
&\to c(2(a(\sharp))) \\
&\to c(2(\sharp)) \\
&\to c(\sharp)
\end{aligned}
$$

Definition of recognition:

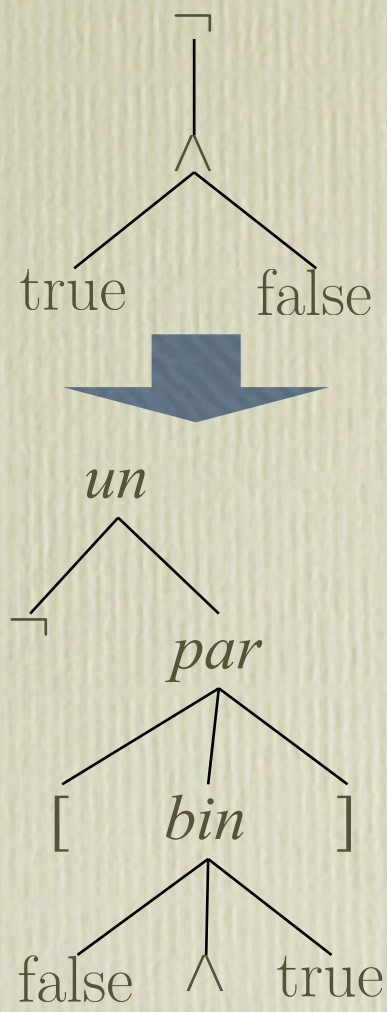
accept $s : t$ if and only if

$$0(s) \to^* t$$

Example:

Concrete Syntax of Formulae

$$
\begin{aligned}
q_0(\neg x) &\rightarrow un(\neg, q_{\wedge\vee}(x)) \\
q_0(x \wedge y) &\rightarrow bin(q_{\wedge\vee}(x), \wedge, q_{\vee}(y)) \\
q_0(x \vee y) &\rightarrow bin(q_{\vee}(x), \vee, q_0(y)) \\
q_0(\text{true}) &\rightarrow \text{true} \\
q_0(\text{false}) &\rightarrow \text{false} \\
q_{\wedge\vee}(\neg x) &\rightarrow un(\neg, q_{\wedge\vee}(x)) \\
q_{\wedge\vee}(x \wedge y) &\rightarrow par([, bin(q_{\wedge\vee}(x), \wedge, q_{\vee}(y)),]) \\
q_{\wedge\vee}(x \vee y) &\rightarrow par([, bin(q_{\vee}(x), \vee, q_0(y)),]) \\
q_{\wedge\vee}(\text{true}) &\rightarrow \text{true} \\
q_{\wedge\vee}(\text{false}) &\rightarrow \text{false} \\
q_{\vee}(\neg x) &\rightarrow un(\neg, q_{\wedge\vee}(x)) \\
q_{\vee}(x \wedge y) &\rightarrow bin(q_{\wedge\vee}(x), \wedge, q_{\vee}(y)) \\
q_{\vee}(x \vee y) &\rightarrow par([, bin(q_{\vee}(x), \vee, q_0(y)),]) \\
q_{\vee}(\text{true}) &\rightarrow \text{true} \\
q_{\vee}(\text{false}) &\rightarrow \text{false}
\end{aligned}
$$

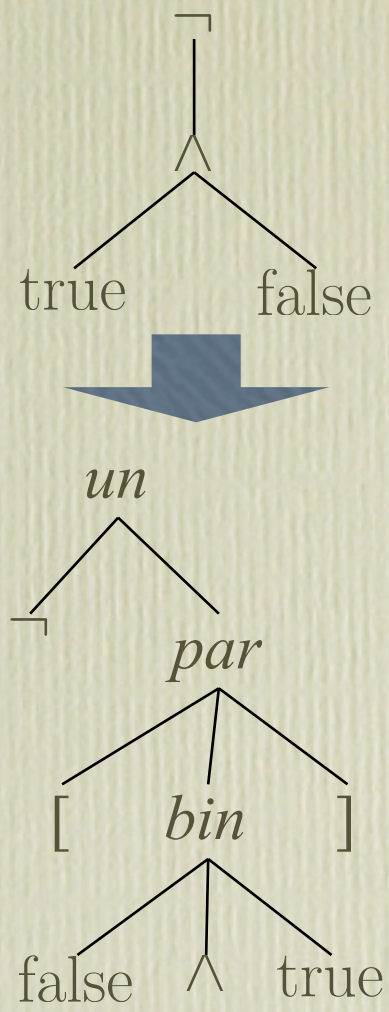


Example:

Concrete Syntax of Formulae

$$
\begin{array}{ll}
q_0(\neg x) & \rightarrow un(\neg, q_{\wedge\vee}(x)) \\
q_0(x \wedge y) & \rightarrow bin(q_{\wedge\vee}(x), \wedge, q_{\vee}(y)) \\
q_0(x \vee y) & \rightarrow bin(q_{\vee}(x), \vee, q_0(y)) \\
q_0(\text{true}) & \rightarrow \text{true} \\
q_0(\text{false}) & \rightarrow \text{false} \\
q_{\wedge\vee}(\neg x) & \rightarrow un(\neg, q_{\wedge\vee}(x)) \\
q_{\wedge\vee}(x \wedge y) & \rightarrow par([, bin(q_{\wedge\vee}(x), \wedge, q_{\vee}(y)),]) \\
q_{\wedge\vee}(x \vee y) & \rightarrow par([, bin(q_{\vee}(x), \vee, q_0(y)),]) \\
q_{\wedge\vee}(\text{true}) & \rightarrow \text{true} \\
q_{\wedge\vee}(\text{false}) & \rightarrow \text{false} \\
q_{\vee}(\neg x) & \rightarrow un(\neg, q_{\wedge\vee}(x)) \\
q_{\vee}(x \wedge y) & \rightarrow bin(q_{\wedge\vee}(x) \\
q_{\vee}(x \vee y) & \rightarrow par([, bin(\\
q_{\vee}(\text{true}) & \rightarrow \text{true} \\
q_{\vee}(\text{false}) & \rightarrow \text{false}
\end{array}
$$

$$fringe(T(\neg(\wedge(\text{true}, \text{false})))) = \ulcorner\neg[\text{true} \wedge \text{false}]\urcorner$$



Example:

Evaluation of Formulae

$$
\begin{aligned}
&\text{true} && \rightarrow q_T(\text{true}) \\
&\text{false} && \rightarrow q_F(\text{false}) \\
&\neg q_T(x_1) && \rightarrow q_F(\text{false}) \\
&\neg q_F(x_1) && \rightarrow q_T(\text{true}) \\
&q_T(x_1) \wedge q_T(x_2) && \rightarrow q_T(\text{true}) \\
&q_T(x_1) \wedge q_F(x_2) && \rightarrow q_F(\text{false}) \\
&q_F(x_1) \wedge q_T(x_2) && \rightarrow q_F(\text{false}) \\
&q_F(x_1) \wedge q_F(x_2) && \rightarrow q_F(\text{false}) \\
&q_T(x_1) \vee q_T(x_2) && \rightarrow q_T(\text{true}) \\
&q_T(x_1) \vee q_F(x_2) && \rightarrow q_T(\text{true}) \\
&q_F(x_1) \vee q_T(x_2) && \rightarrow q_T(\text{true}) \\
&q_F(x_1) \vee q_F(x_2) && \rightarrow q_F(\text{false})
\end{aligned}
$$

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:

$$\begin{aligned} q(f(x)) &\to g(q(x), q(x)) \\ q(a) &\to a \end{aligned}$$

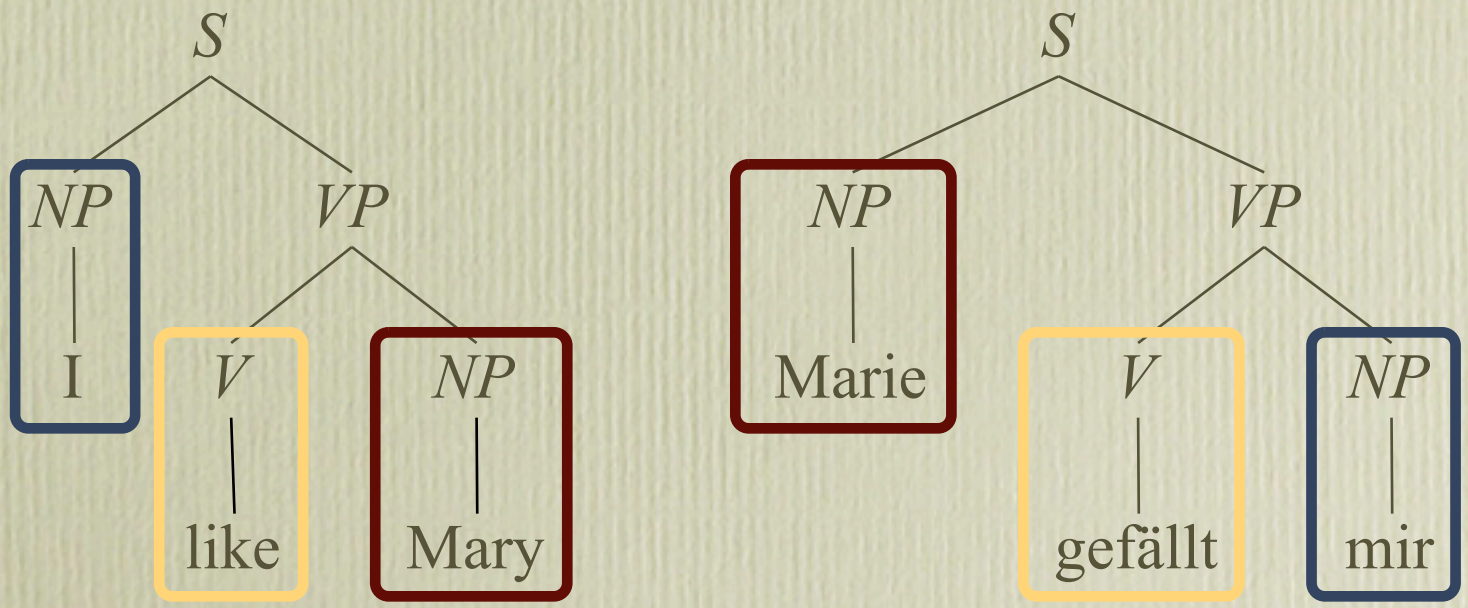
$$q(f(a)) \to g(q(a), q(a)) \to^* g(a, a)$$
$$q(f(f(a))) \to g(q(f(a)), q(f(a))) \to^* g(g(a, a), g(a, a))$$
$$q(f(f(f(a)))) \to^* g(g(g(a, a), g(a, a)), g(g(a, a), g(a, a)))$$
$$|q(f^n(a))| = 2^n - 1$$

Exponential growth implies no composition closure

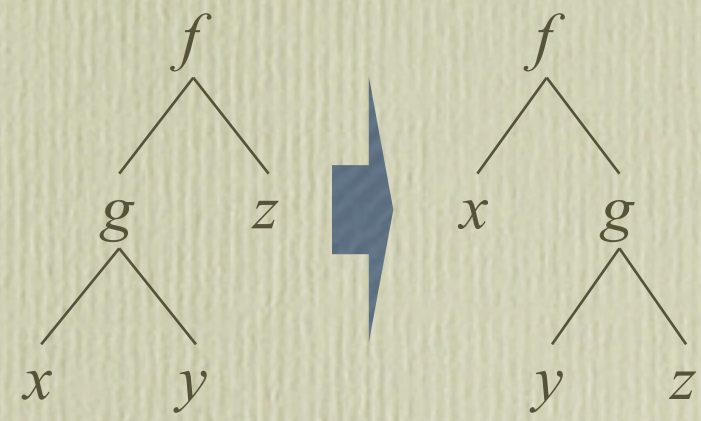
Why Rotations?

Consider

- I like Mary.
- Marie gefällt mir.



Expressing Rotations



$$q(f(g(x,y),z)) \rightarrow f(q(x),g(q(y),q(z)))$$

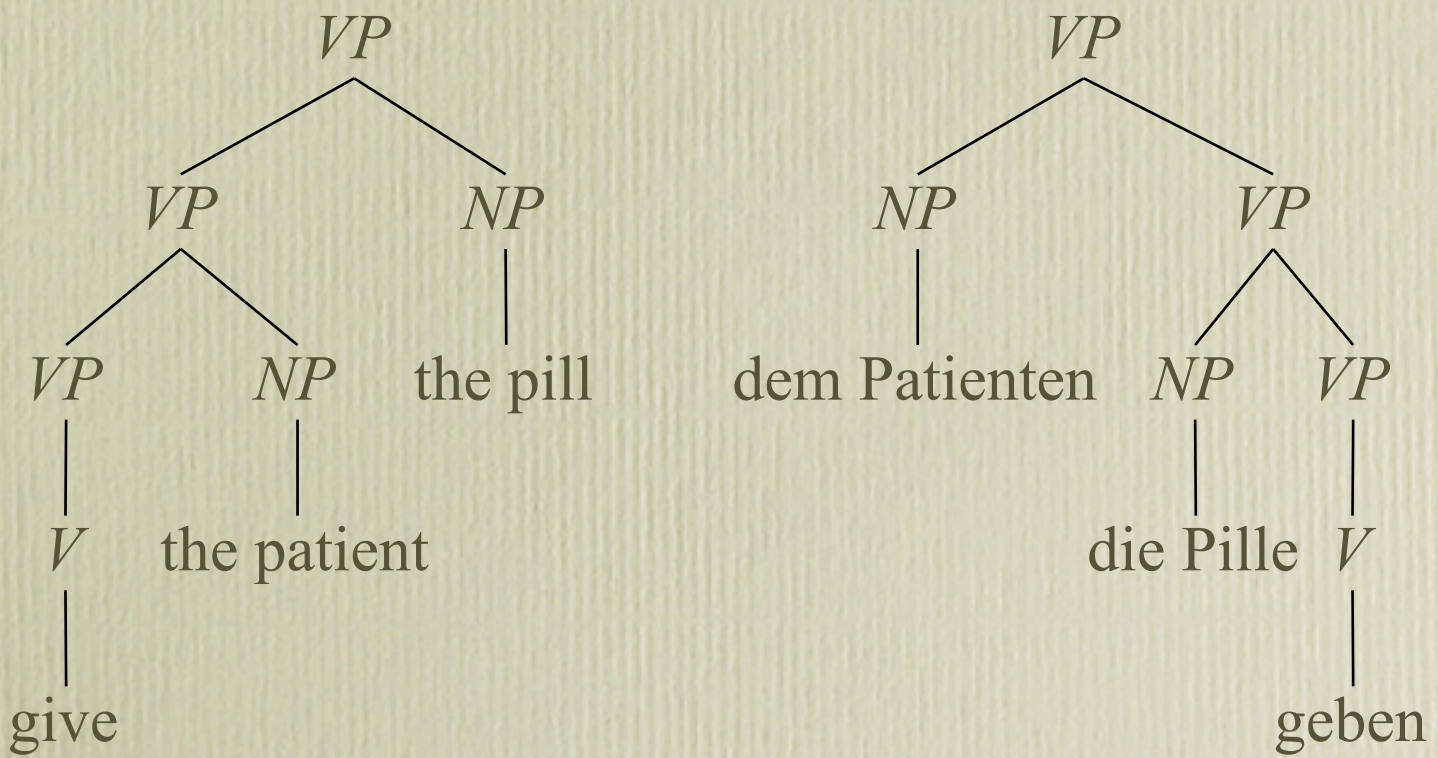
$$\begin{aligned} q(f(x_{gxy},z)) &\rightarrow f(q_1(x_{gxy}),g(q_2(x_{gxy}),q(z))) \\ q_1(g(x,y)) &\rightarrow x \\ q_2(g(x,y)) &\rightarrow y \end{aligned}$$

requires nonlinearity

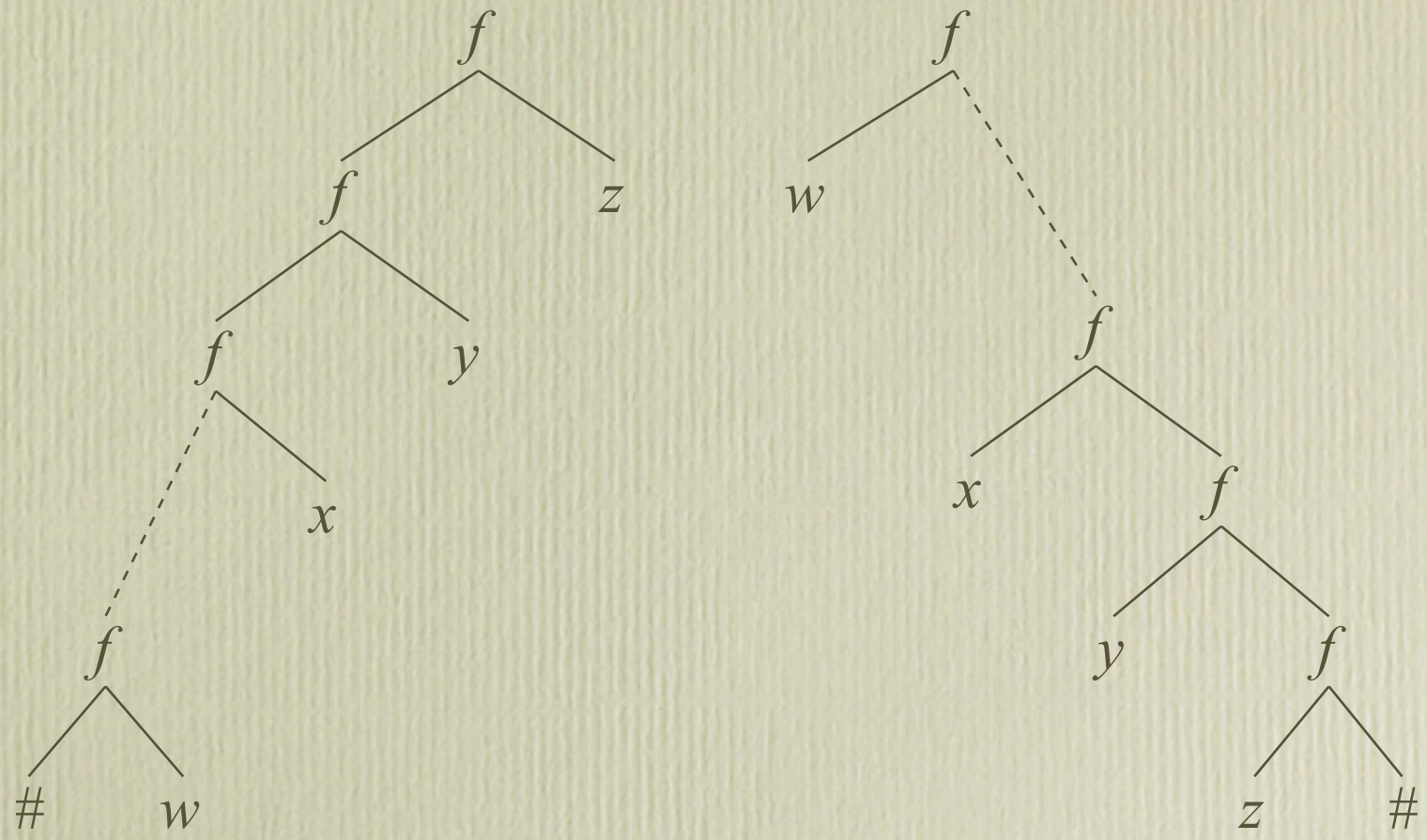
Why Global Rotations?

Consider

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



No Global Rotations



cf. macro tree transducers

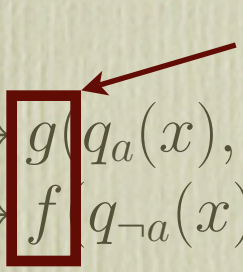
Determinization

Determinization of transducers fails even if underlying automata are deterministic.

The goal:

$$f(x,y) \Rightarrow g(a,x)$$
$$f(x,y) \Rightarrow f(x,y) \quad \text{for } x \neq aa \Rightarrow a$$

A nondeterministic solution:



$$\begin{aligned}
q(f(x,y)) &\to g(q_a(x), q(y)) \\
q(f(x,y)) &\to f(q_{\neg a}(x), q(y)) \\
q(a) &\to a \\
q_a(a) &\to a \\
q_{\neg a}(f(x,y)) &\to g(q_a(x), q(y))
\end{aligned}$$

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

Division of Engineering and Applied Sciences
Harvard University

String Homomorphisms

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

A string homomorphism is ϵ-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 Δ' contains transitions of the form

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

for each transition in Δ of the form

$$q\, s \rightarrow t\, q' \quad .$$

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

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

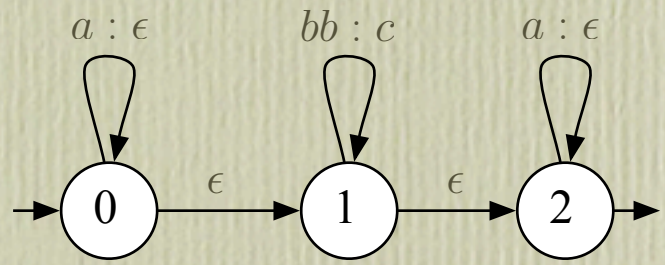
and $h_{out} : \Sigma^* \times \Gamma^* \rightarrow \Gamma$ extending

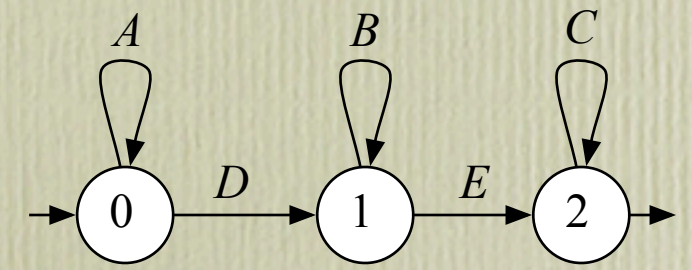
$$h_{in}(\langle s, t \rangle) = t \quad .$$

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

String Bimorphisms

Finite-state transducers and string bimorphisms are equivalent.





$$\langle h_{in}, A^*DB^*EC^*, h_{out} \rangle$$

$$h_{in}(A) = a$$
$$h_{in}(B) = bb$$
$$h_{in}(C) = a$$
$$h_{in}(D) = \epsilon$$
$$h_{in}(E) = \epsilon$$

$$h_{out}(A) = \epsilon$$
$$h_{out}(B) = c$$
$$h_{out}(C) = \epsilon$$
$$h_{out}(D) = \epsilon$$
$$h_{out}(E) = \epsilon$$

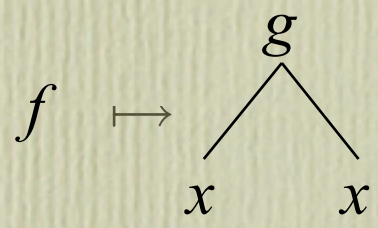
Tree Homomorphisms

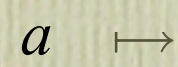
Let $h_{\mathcal{F}} : \mathcal{F} \to \mathcal{T}(\mathcal{F}', \mathcal{X})$ be a function mapping each $f \in \mathcal{F}$ of arity n to a tree $h_{\mathcal{F}}(f) : \mathcal{T}(\mathcal{F}', X_n)$. The tree homomorphism extending $h_{\mathcal{F}}$ is the function $h : \mathcal{T}(\mathcal{F}) \to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

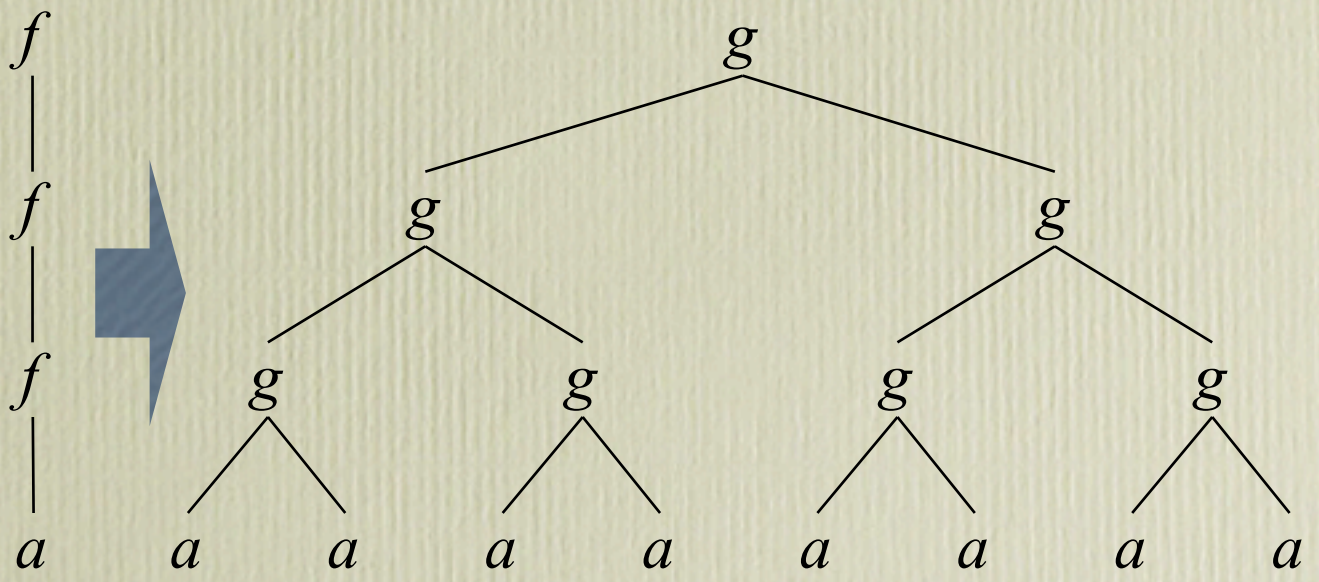
$$h(f(t_1, \ldots, t_n)) = h_{\mathcal{F}}(f)[h(t_1), \ldots, h(t_n)]$$

Equivalent to one-state tree transducers

Example: Perfect Binary Trees







Tree Bimorphisms

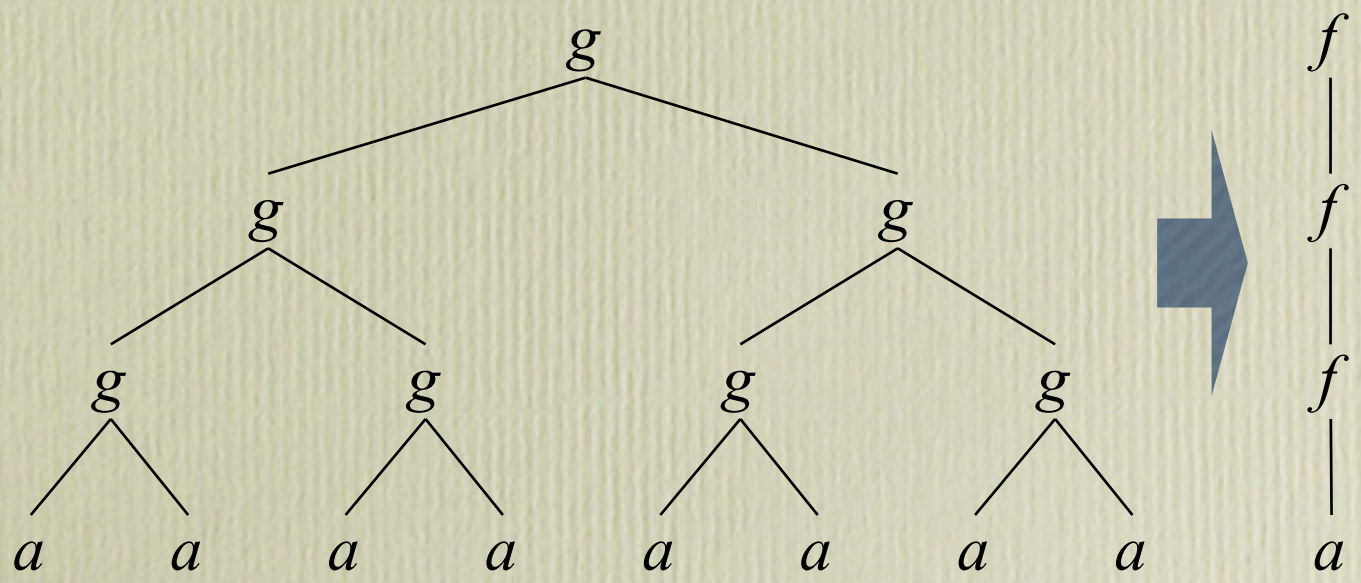
A tree bimorphism over input alphabet Σ and output alphabet Γ is a triple $\langle h_i, L, h_o \rangle$ where L is a rational tree language over an alphabet Δ and $h_i : \mathcal{T}(\Delta) \to \mathcal{T}(\Sigma)$ and $h_o : \mathcal{T}(\Delta) \to \mathcal{T}(\Gamma)$ are tree homomorphisms.

A tree bimorphism generates a tree relation R from $\mathcal{T}(\Sigma)$ to $\mathcal{T}(\Gamma)$ 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,

$$R = h_i^{-1} \circ L \circ h_o \qquad .$$

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

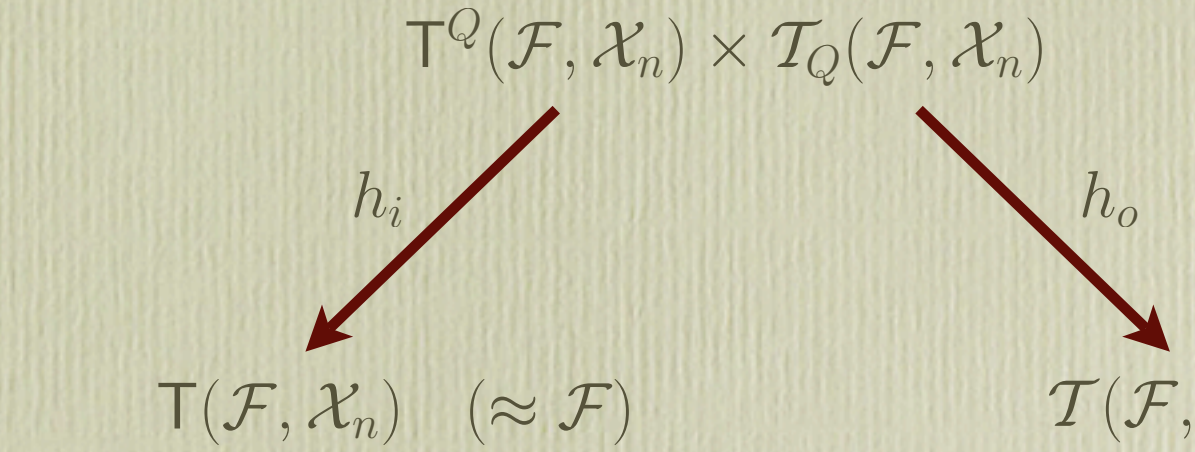
Symbol-to-symbol

- output of height 1
- (implies epsilon-free)

Delabeling

- = linear complete symbol-to-symbol

Intuition for Restriction



delabeling (*LCS*)

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*)

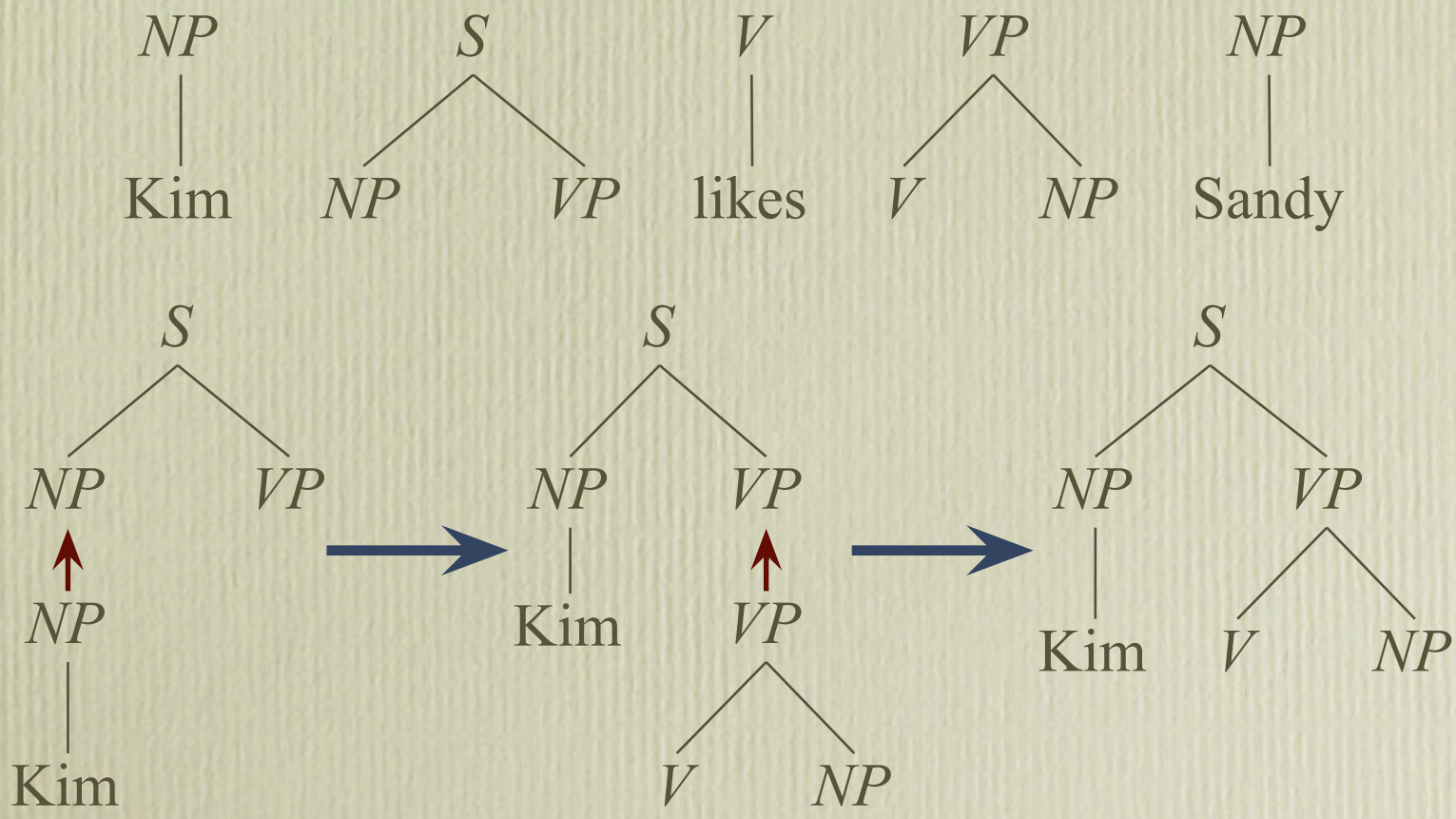
- = synchronous tree substitution grammars

B(*LCF*, *LCF*)

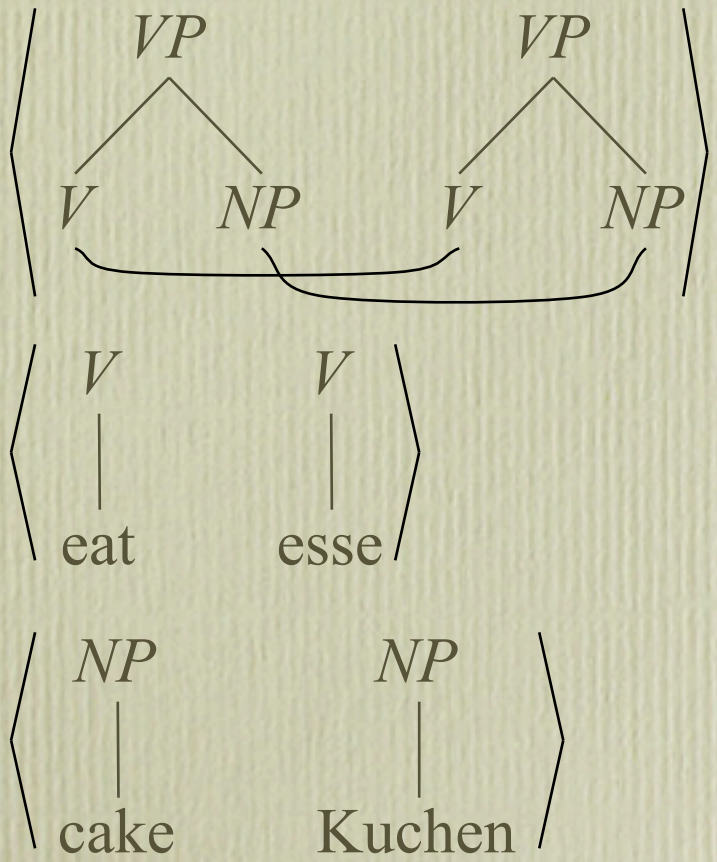
- $B(LCF, LCF) \subset B(LCF, LCF)^2 = B(LCF, LCF)^3$

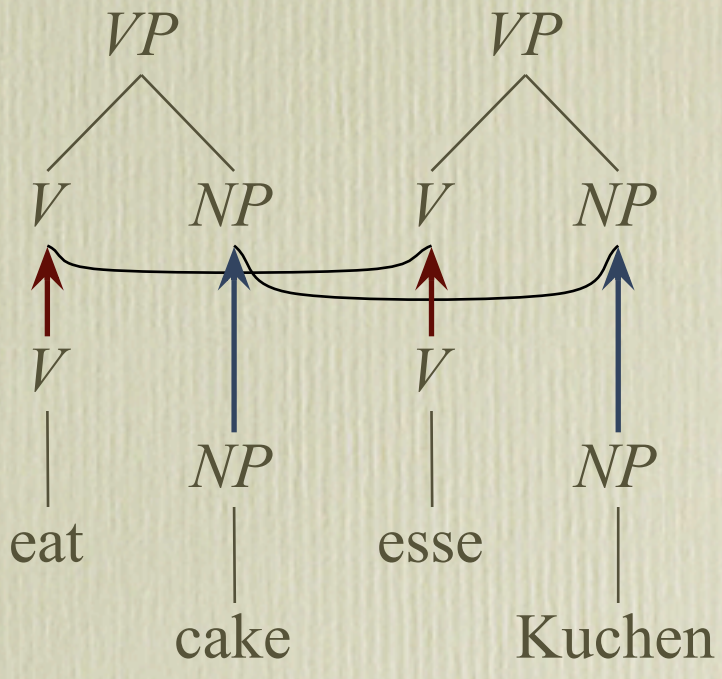
Synchronous Grammars

Context-free grammars as tree substitution

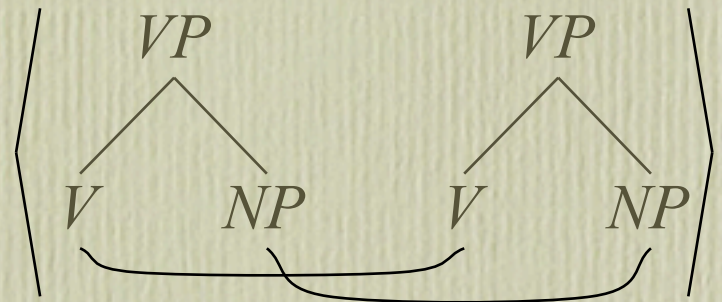


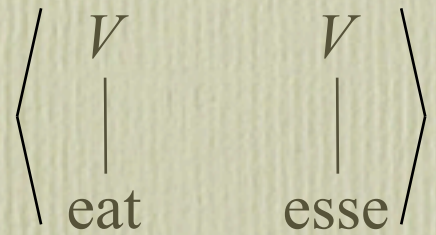
Synchronous Context-Free Grammars

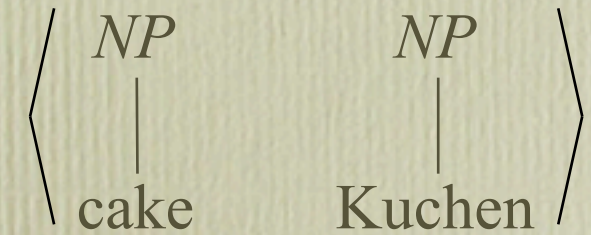


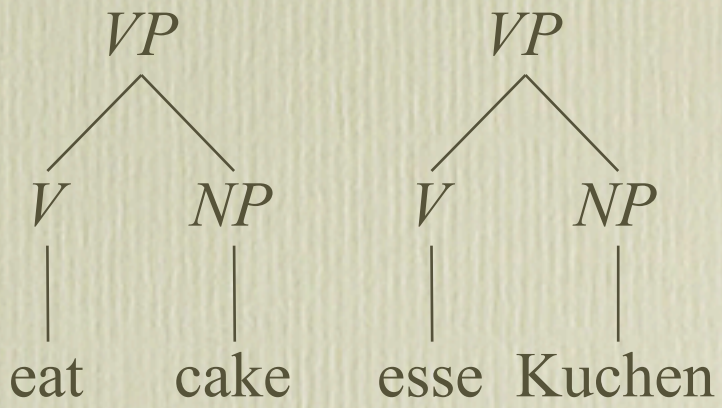


Synchronous Context-Free Grammars



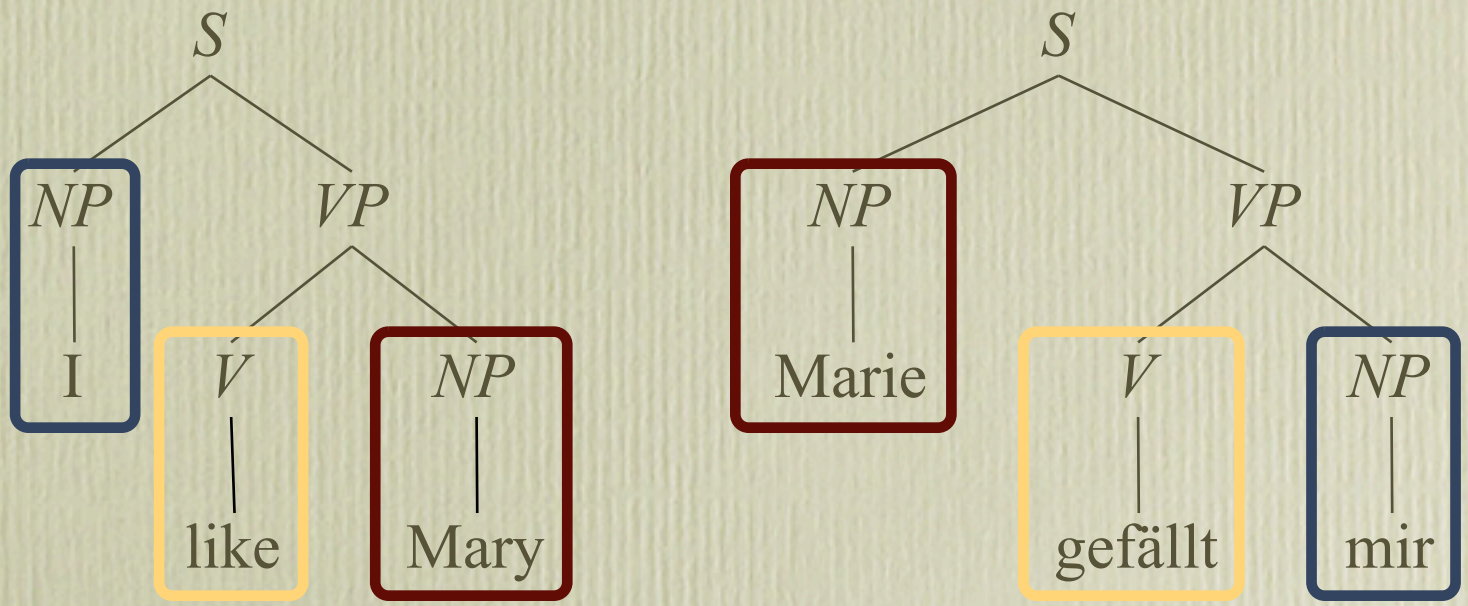






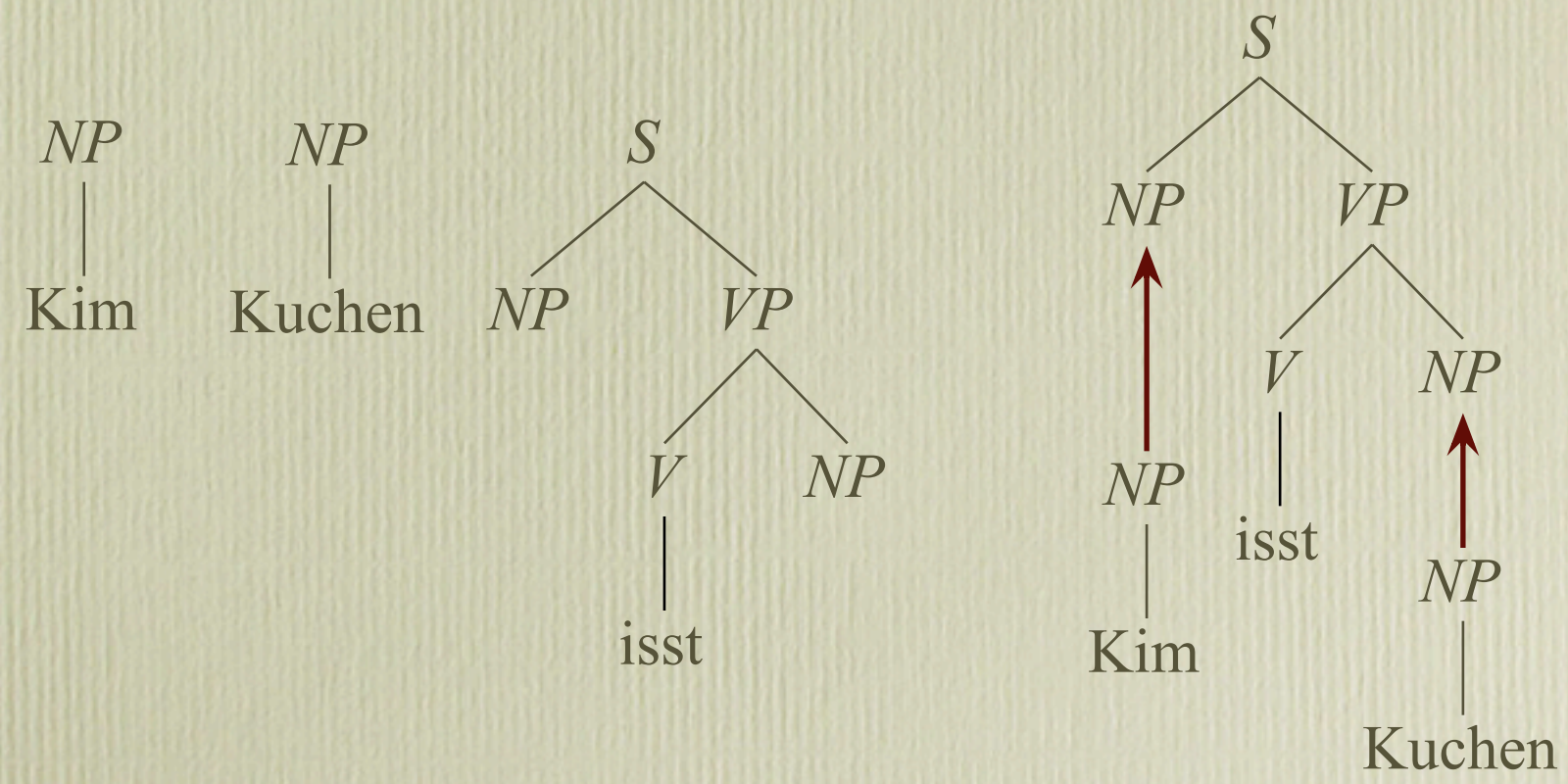
Problems With SCFG

Domain of locality is too small



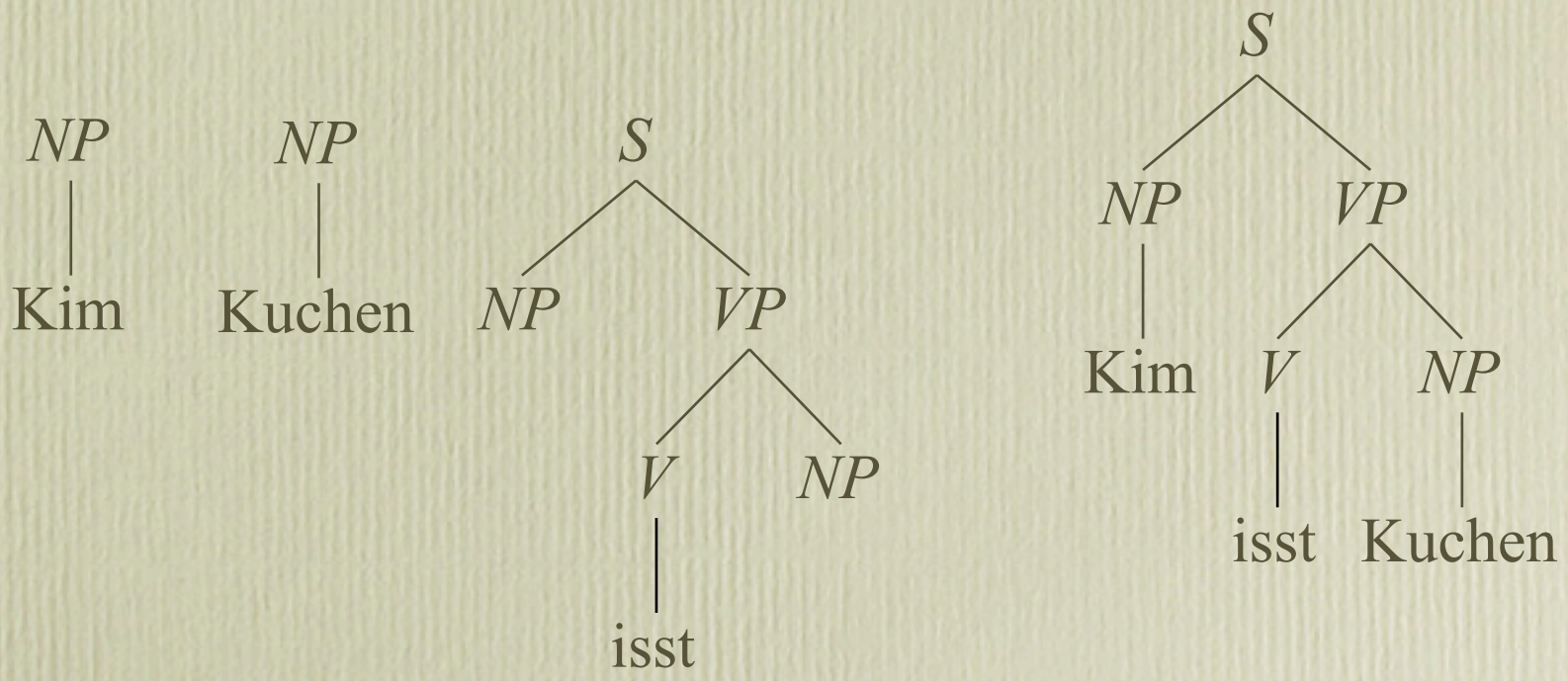
Tree Substitution Grammars

Expands domain of locality to ***elementary tree***.



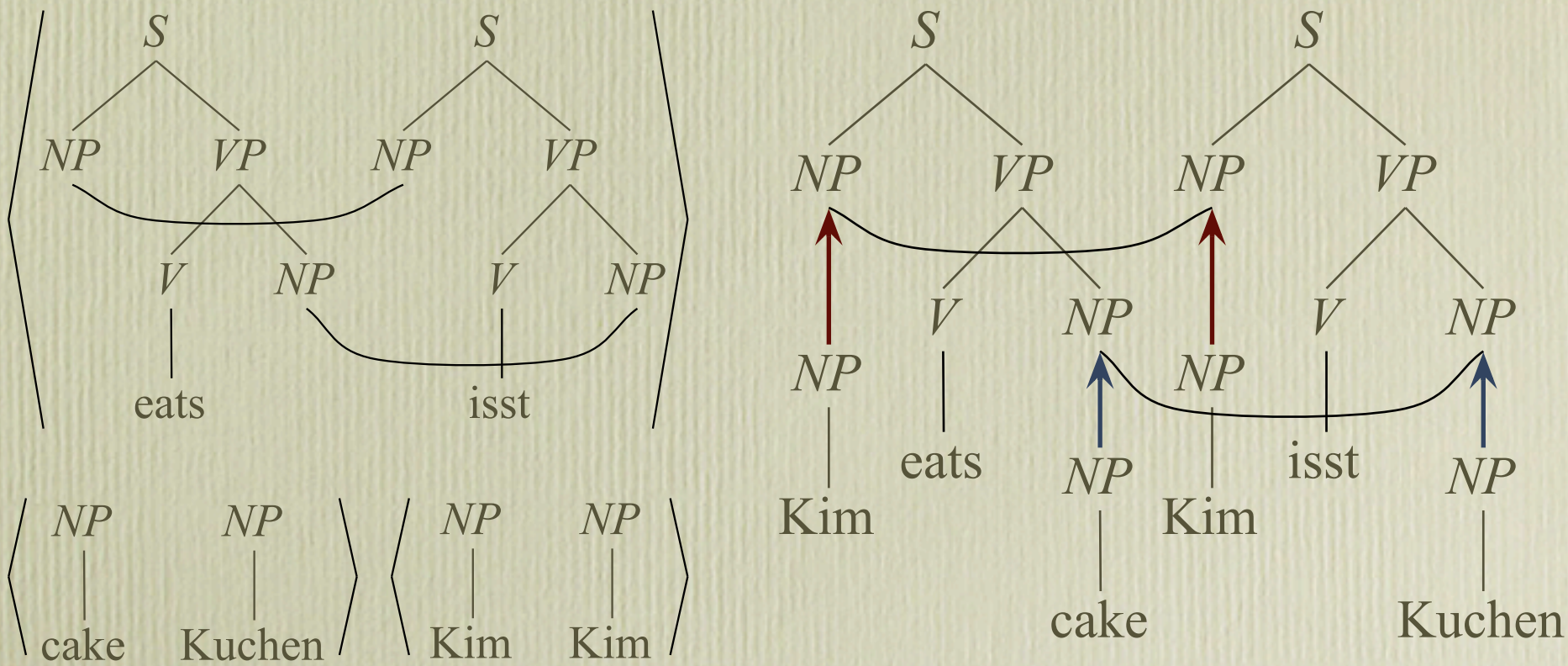
Tree Substitution Grammars

Expands domain of locality to *elementary tree*.

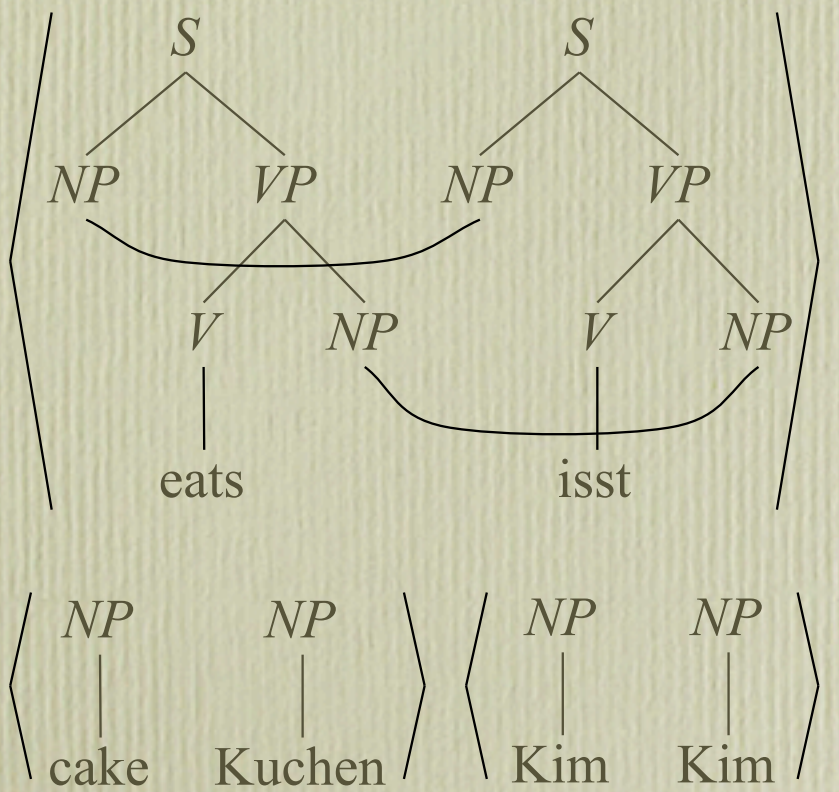


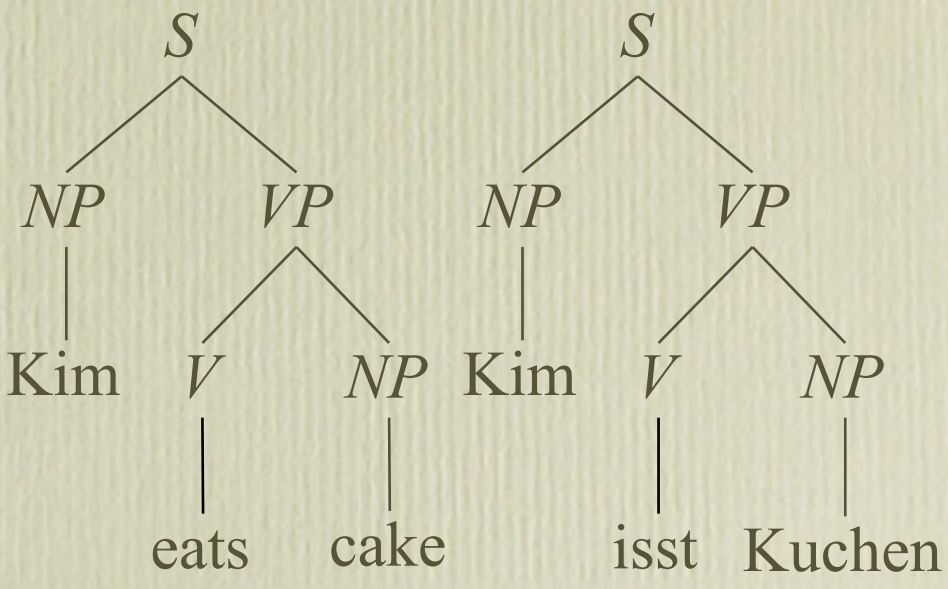
No additional expressive power over CFG.

Synchronous Tree-Substitution Grammars

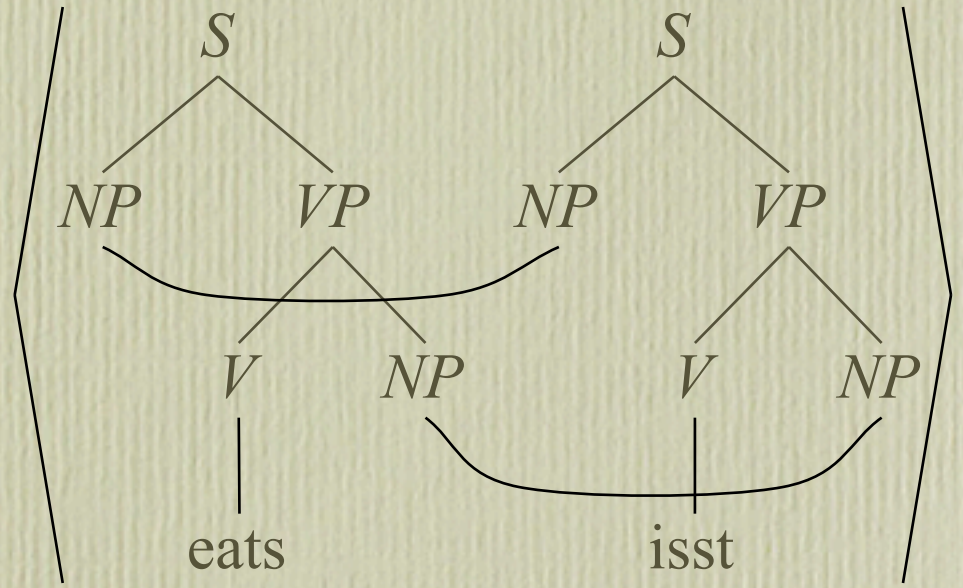


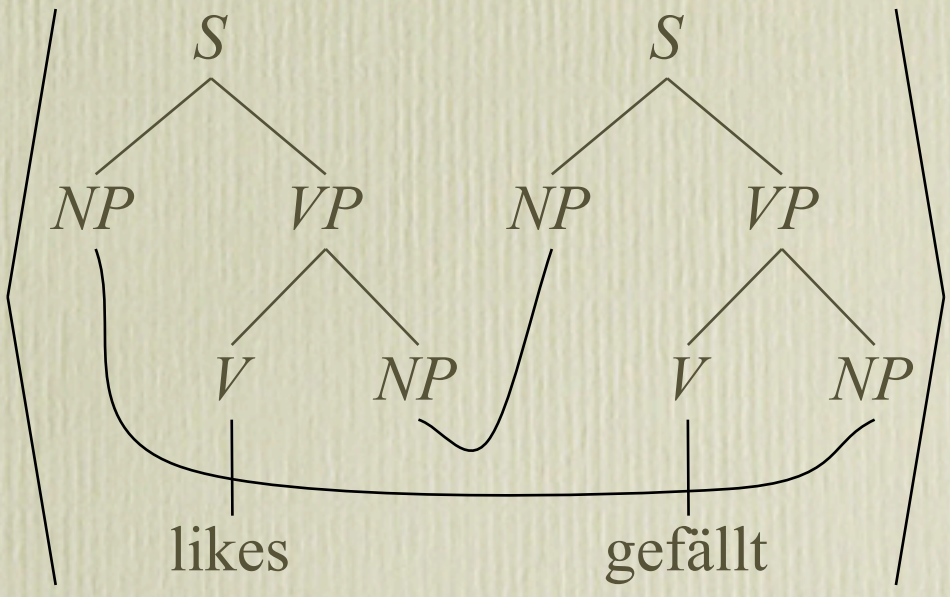
Synchronous Tree-Substitution Grammars



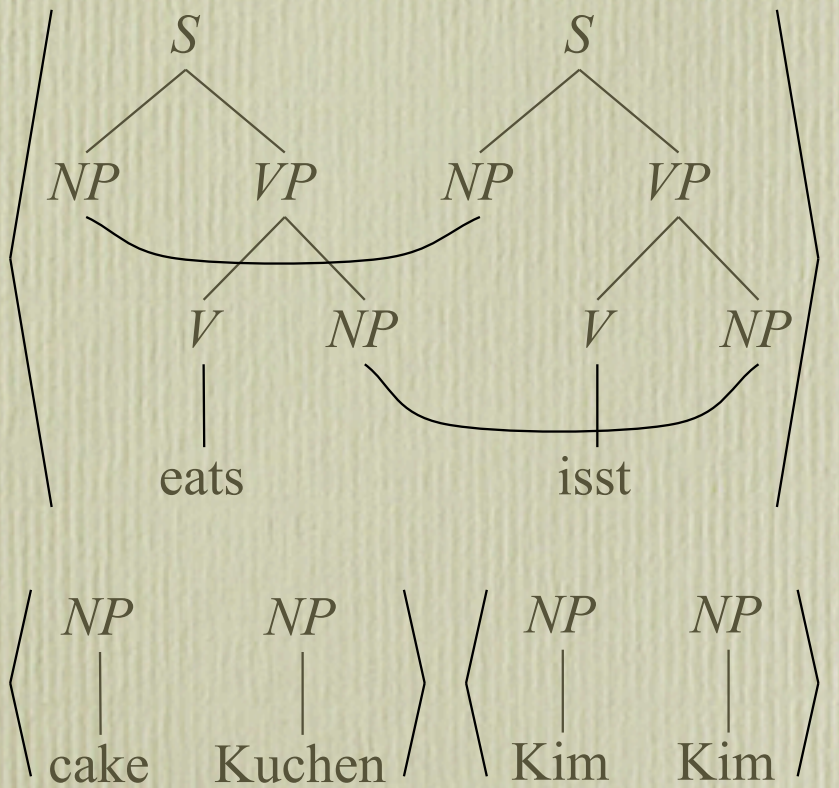


The STSG Payoff





STSG as Bimorphism

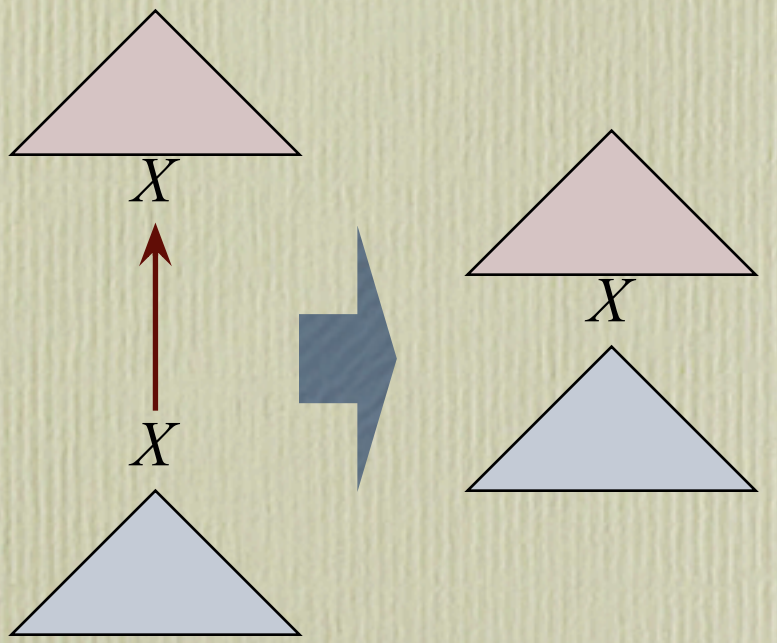


$$h_i(\alpha_1) = S(x, VP(V(eats), y)$$
$$h_o(\alpha_1) = S(x, VP(V(isst), y)$$
$$h_i(\alpha_2) = NP(cake)$$
$$h_o(\alpha_2) = NP(Kuchen)$$
$$h_i(\alpha_3) = NP(Kim)$$
$$h_o(\alpha_3) = NP(Kim)$$

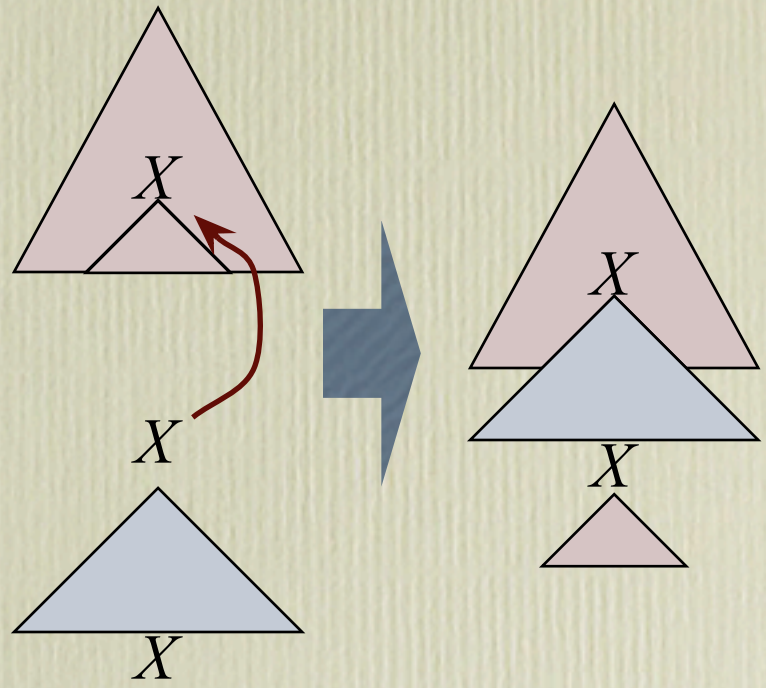
$$h_i(\alpha_4) = S(x, VP(V(likes), y)$$
$$h_o(\alpha_4) = S(y, VP(V(gefällt), x)$$
$$h_i(\alpha_5) = NP(I)$$
$$h_o(\alpha_5) = NP(mir)$$

NB: linear, complete

Substitution and Adjunction



substitution



adjunction

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

Division of Engineering and Applied Sciences
Harvard University

Synchronous Tree-Adjoining Grammars

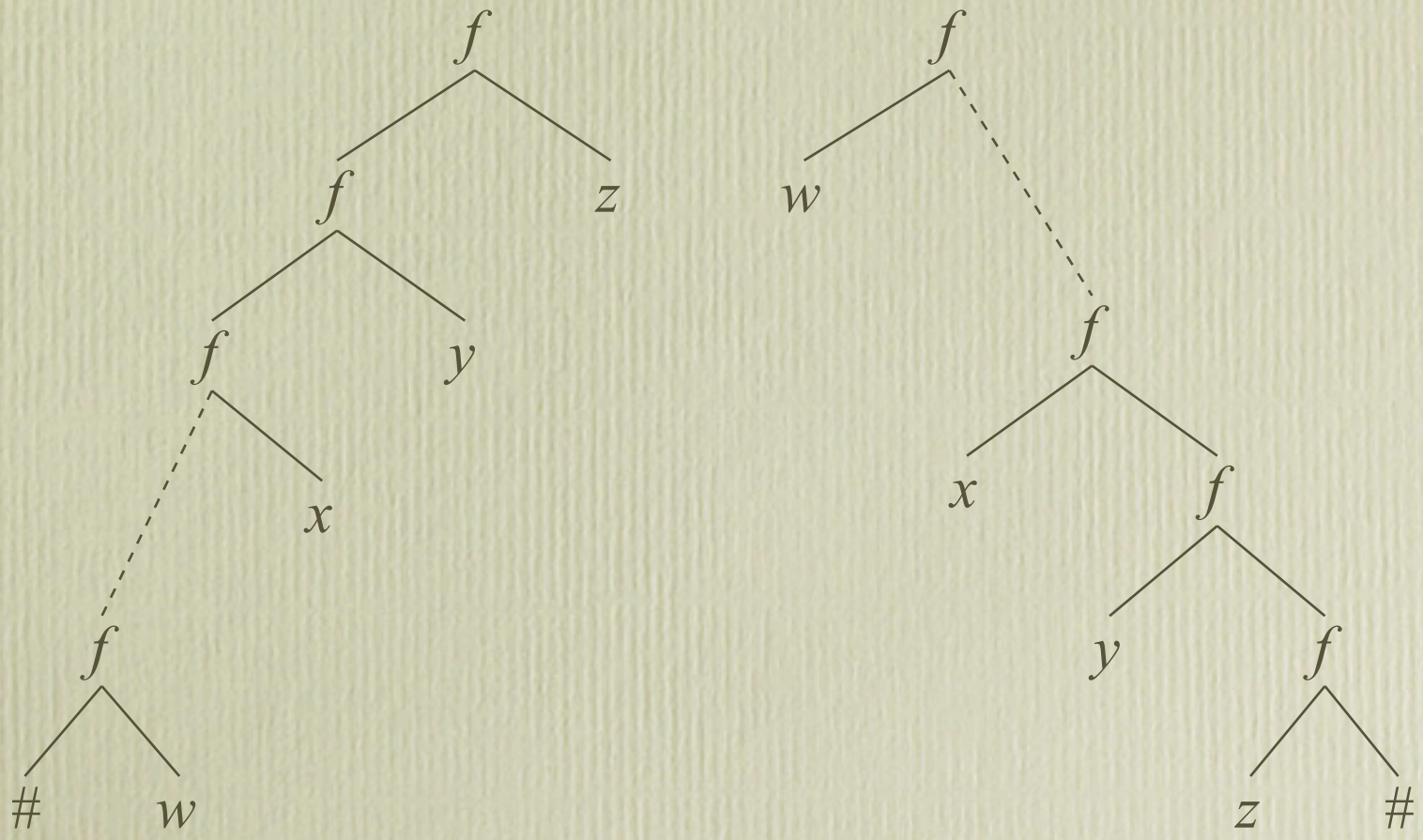
Proposed for transductions to characterize

- semantics
- natural-language generation
- machine translation

Extended Transducers: Macro Tree Transducers and Deterministic Tree Rewriting

Division of Engineering and Applied Sciences
Harvard University

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

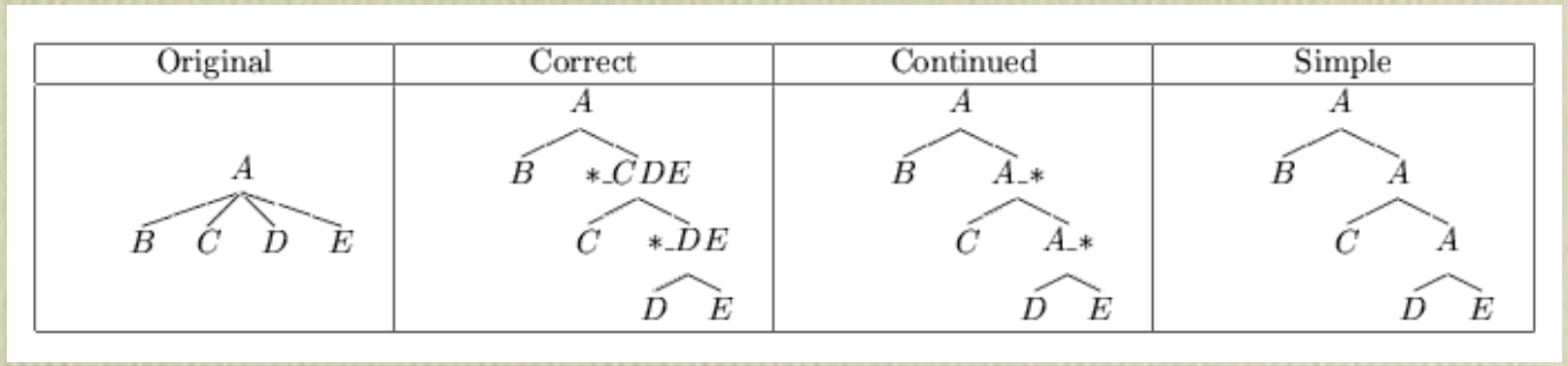


Table 3: Transformations from N-ary to binary branching structures

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

(Note the implicit need for global rotations.)

Programming Rotation

$$rotate(x) = rotate(x, \sharp)$$
$$rotate(\sharp, x) = x$$
$$rotate(f(x, y), z) = rotate(x, f(y, z))$$

$$rotate(f(f(f(\sharp, a), b), c))$$
$$= rotate(f(f(f(\sharp, a), b), c), \sharp)$$
$$= rotate(f(f(\sharp, a), b), f(c, \sharp))$$
$$= rotate(f(\sharp, a), f(b, f(c, \sharp)))$$
$$= rotate(\sharp, f(a, f(b, f(c, \sharp))))$$
$$= f(a, f(b, f(c, \sharp)))$$

Macro Tree Transducers

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

- Q is a ranked alphabet of states;
- $\mathcal{F}$ is a ranked alphabet;
- $\Delta \in \bigcup_{r \geq 1, q^{(r)} \in Q, k \geq 0, f^{(k)} \in \mathcal{F}} \mathcal{T}_{mlhs}(q^{(r)}, f^{(k)}, \mathcal{X}_k, \mathcal{Y}_r) \times \mathcal{T}_{mrhs}(Q, \mathcal{F}, \mathcal{X}_k, \mathcal{Y}_r)$ is a set of transitions;
- $t_0 \in \mathcal{T}_{mrhs}(Q, \mathcal{F}, \mathcal{X}_1, \emptyset)$ 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 $\mathcal{X}_k$, and parameters $\mathcal{Y}_r$, notated $\mathcal{T}_{mlhs}(q^{(r)}, f^{(k)}, \mathcal{X}_k, \mathcal{Y}_r)$, 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 $\mathcal{X}_k$, and parameters $\mathcal{Y}_r$, notated $\mathcal{T}_{mrhs}(Q, \mathcal{F}, \mathcal{X}_k, \mathcal{Y}_r)$, is the smallest set of trees in $\mathcal{T}(Q \cup F, \mathcal{X}_k \cup \mathcal{Y}_r)$ such that

(1) $\mathcal{Y}_r \subseteq \mathcal{T}_{mrhs}(Q, \mathcal{F}, \mathcal{X}_k, \mathcal{Y}_r)$
(2) for all $k \geq 0$ and $f \in \mathcal{F}^{(k)}$ and $t_1, \ldots, t_k \in \mathcal{T}_{mrhs}(Q, \mathcal{F}, \mathcal{X}_k, \mathcal{Y}_r)$, $f(t_1, \ldots, t_k) \in \mathcal{T}_{mrhs}(Q, \mathcal{F}, \mathcal{X}_k, \mathcal{Y}_r)$
(3) for all $r \geq 1$ and $q \in Q^{(r)}$ and $x_i \in X_k$ and $t_1, \ldots, t_{r-1} \in \mathcal{T}_{mrhs}(Q, \mathcal{F}, \mathcal{X}_k, \mathcal{Y}_r)$, $q(x_i, t_1, \ldots, t_{r-1}) \in \mathcal{T}_{mrhs}(Q, \mathcal{F}, \mathcal{X}_k, \mathcal{Y}_r)$

MTT Examples:

Global Rotation

$$rotate(x) = rotate(x, \sharp)$$
$$rotate(\sharp, x) = x$$
$$rotate(f(x, y), z) = rotate(x, f(y, z))$$

$$q_0(x_1) \rightarrow q_r(x_1, \sharp)$$
$$q_r(\sharp, y_1) \rightarrow y_1$$
$$q_r(f(x_1, x_2), y_1) = q_r(x_1, f(x_2, y_1))$$

MTT Examples:

Local Rotation

$$q(f(x_1, x_2)) \to q'(x_1, x_2)$$
$$q'(f(x_1, x_2), y_1) \to f(q(x_1), f(q(x_2), q(y_1)))$$
$$q(a) \to a$$
$$q(b) \to b$$

$$\begin{aligned}
&q(f(f(a, f(f(b, a), a)), b)) \\
&\to q'(f(a, f(f(b, a), a)), b) \\
&\to f(q(a), f(q(f(f(b, a), a)), q(b))) \\
&\to^3 f(a, f(q'(f(b, a), a), b)) \\
&\to f(a, f(f(q(b), f(q(a), q(a))), b)) \\
&\to^3 f(a, f(f(b, f(a, a)), b))
\end{aligned}$$

Division of Engineering and Applied Sciences
Harvard University

MTT Examples:

Reversal

$$q_0(x_1) \to q(x_1, \sharp)$$
$$q(f(x_1, x_2), y) \to q(x_2, f(x_1, y))$$
$$q(\sharp, y) \to y$$

$$\begin{aligned}
&q_0(f(a, f(b, f(c, \sharp)))) \\
&\to q(f(a, f(b, f(c, \sharp))), \sharp) \\
&\to q(f(b, f(c, \sharp)), f(a, \sharp)) \\
&\to q(f(c, \sharp), f(b, f(a, \sharp))) \\
&\to q(\sharp, f(c, f(b, f(a, \sharp)))) \\
&\to f(c, f(b, f(a, \sharp)))
\end{aligned}$$

MTT Examples:

Reversal

$$q_0(x_1) \to q(x_1, \sharp)$$
$$q(f(x_1, x_2), y) \to q(x_1, q(x_2, y))$$
$$q(a, y) \to f(a, y)$$
$$q(b, y) \to f(b, y)$$

$$q_0(f(f(a, b), f(b, a)))$$
$$\to q(f(f(a, b), f(b, a)), \sharp)$$
$$\to q(f(a, b), q(f(b, a), \sharp))$$
$$\to q(f(a, b), q(b, q(a, \sharp)))$$
$$\to q(a, q(b, q(b, q(a, \sharp))))$$
$$\to^4 f(a, f(b, f(b, f(a, \sharp))))$$

MTT Examples:

Frontier

$$q_0(x_1) \to q(x_1, \sharp)$$
$$q(f(x_1, x_2), y) \to q(x_1, q(x_2, y))$$
$$q(a, y) \to f(a, y)$$
$$q(b, y) \to f(b, y)$$

$$q_0(f(f(a, b), f(b, a)))$$
$$\to q(f(f(a, b), f(b, a)), \sharp)$$
$$\to q(f(a, b), q(f(b, a), \sharp))$$
$$\to q(f(a, b), q(b, q(a, \sharp)))$$
$$\to q(a, q(b, q(b, q(a, \sharp))))$$
$$\to^4 f(a, f(b, f(b, f(a, \sharp))))$$

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

‹pattern› ==> *‹result›*

```
NP(Det, N) ==> Det(N)
```

$$q(np(x_{det}, x_n)) \rightarrow det(q(x_n))$$

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

Rewrite Rules

‹pattern› ==> ‹result›

```
NP(Det, N) ==> Det(N)
```

$q(np(x_{det}, x_n)) \rightarrow det(q(x_n))$

```
Units(lines) ==> line
```

$q(units(lines)) \rightarrow line$

```
Units(_unit) ==> _unit
```

$q(units(x_{unit})) \rightarrow q(x_{unit})$

```
Command(move, down, Number, Units)
     ==> Move(down, Number, Units)
```

$q(command(move, down, x_{number}, x_{units}))$
$\rightarrow move(down, q(x_{number}), q(x_{units}))$

Example

```
Command(move, down, Number, Units)
    ==> Move(down, Number, Units)

Number(_n) = _n

Units(lines) ==> line

Units(_unit) ==> _unit


Command(move, down,
        Number(3), Units(lines))
==>*  Move(down, 3, line)
=/=>* 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`