Computational Phonology

Syntagmatic computing

Dafydd Gibbon

Bielefeld University
Jinan University, Guangzhou

Objectives

- To claim that markedness, defaults and optimality are related, in the form of
  - ‘logical preferences’: *ranking*, *elsewhere conditions*, *exceptions*
  - ‘empirical preferences’: *frequency*, *familiarity*, *statistics*
- To demonstrate computation of the three structural dimensions of the architecture of language and speech:
  - Composition: *ranked*, *grouped*, *parallel syntagmatic relations*
  - Classification: *paradigmatic relations*
  - Interpretation: *modelling relations*
- To show that computation is essential for
  - Phonological theory
  - Phonological hypothesis testing

Types of Computing in Phonology

- Syntagmatic computing (composition)
  - Well-formedness of category combinations
    - Serial: strings, hierarchical grouping
    - Parallel: distinctive features, autosegmental tiers

- Paradigmatic computing (classification)
  - Sets: classification, categorisation
  - Properties: criteria for identifying sets

- Interpretative computing (phonetic modelling)
    - Categorial ←→ physical representation levels
    - Mapping:
      - Derivation (Generative Phonology)
      - Transduction (Finite State Phonology)
      - Selection (Optimality Theory)

Domains of Computational Phonology

- Syntagmatic (compositional) relations:
  - Autosegmental phonology
  - Metrical phonology
  - Finite state phonology
- Paradigmatic (classificatory) relations:
  - Feature theories, feature geometry
  - Inheritance phonology
- Interpretative (mapping) relations:
  - Generative phonologies
  - Optimality theoretic phonologies

Syntagmatic Computing

Syntagmatic computing (compositionality of categories)

- Simultaneous
  - Feature bundles
  - Feature geometry
  - Three-dimensional phonology

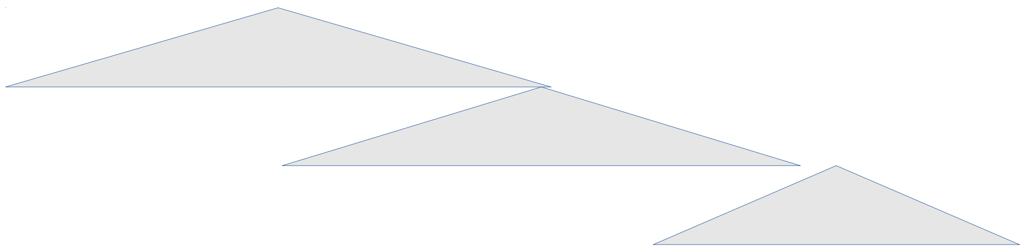
- Sequential
  - Stress cycle
  - Metrical Phonology

- Sequential-simultaneous
  - Autosegmental Phonology
  - Finite State Phonology
  - Inheritance Phonology

FSA, FST
Inc. formal defs.

Syntagmatic Complexity

(3)



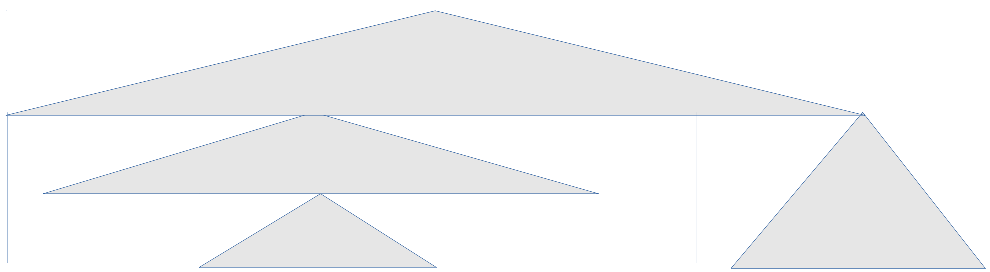
This is the dog that chased the cat that ate the mouse ...

Right-branching linear recursion / iteration.

Regular languages

Chomsky Type 3,
Regular grammar
↔
Finite State Automaton

(2)



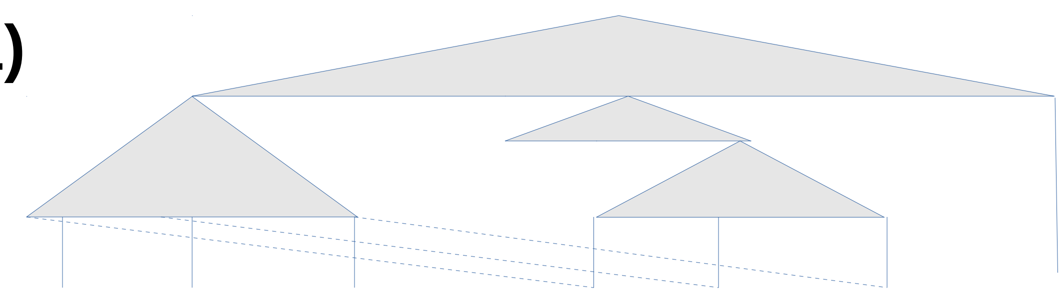
If the man who John met goes home then Jane will smile

Centre-embedding hierarchical recursion.

Context-free languages

Chomsky Type 2,
Context-free grammar
↔
Push-Down Automaton

(1)



June, Jane and Jean love Mick, Dick and Nick, respectively

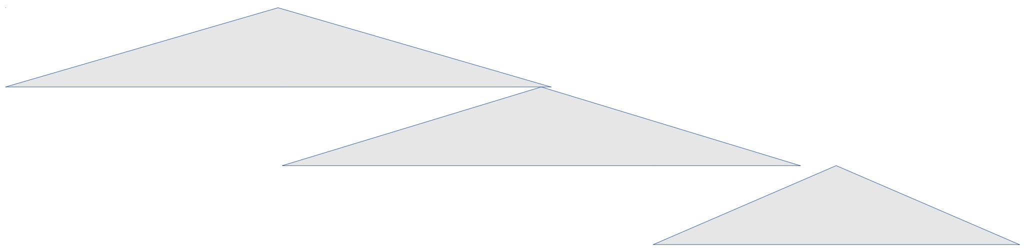
Recursive cross-serial dependency.

Context-sensitive languages

Chomsky Type 1,
Context-sensitive grammar
↔
Linear Bounded Automaton

Syntagmatic Complexity

(3)



This is the dog that chased the cat that ate the mouse ...

Right-branching linear recursion / iteration.

Regular languages

Chomsky Type 3,
Regular grammar
↔
Finite State Automaton

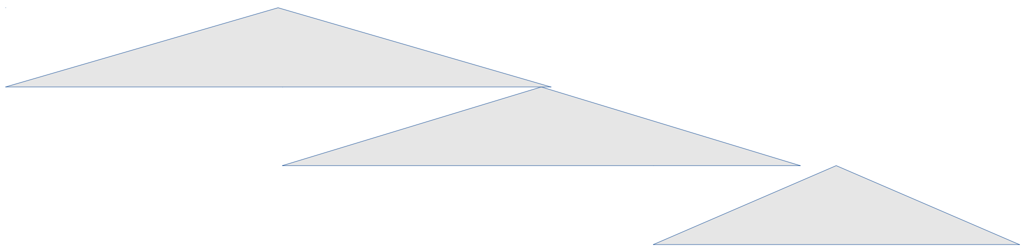
Chomsky maintained in *Syntactic Structures* (1957) that

English is not a finite state language.

- This means that there are structures which are more complex than regular languages.
- But it turns out that these more complex structures are hardly ever found in everyday spontaneous dialogue, and are restricted to formal, rehearsed speech and writing, including mathematics.
- Very many parts of language are indeed ‘finite state’, including phonology and prosody, morphology, most parts of sentence, text and discourse structures.

Syntagmatic Complexity

(3)



This is the dog that chased the cat that ate the mouse ...

Right-branching linear recursion / iteration.

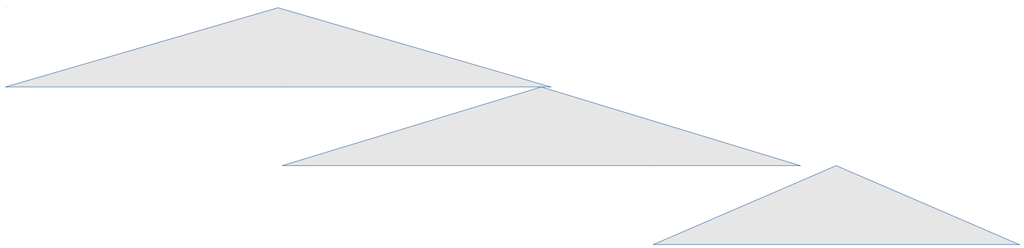
Regular languages

Chomsky Type 3,
Regular grammar
↔
Finite State Automaton

- Very many parts of language are indeed ‘finite state’, including phonology and prosody, morphology, most parts of sentence, text and discourse structures.
- Why is this so?
  - The set of syllables in any language is finite and can be described with a non-iterative finite state automaton or non-recursive regular grammar.
  - The set of words in any language is not finite, but can be described by an iterative finite state automaton or a right-recursive (or left-recursive) regular grammar.

Syntagmatic Complexity

(3)



This is the dog that chased the cat that ate the mouse ...

Right-branching linear recursion / iteration.

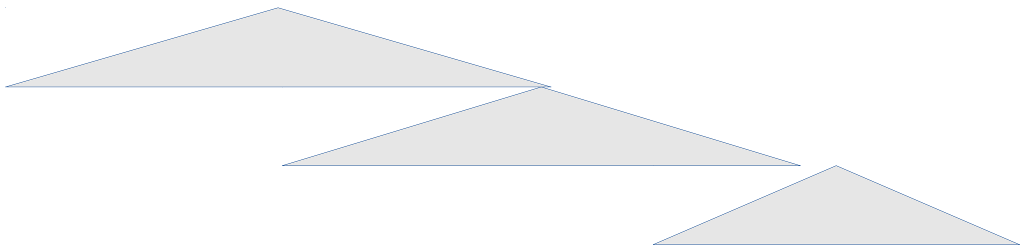
Regular languages

Chomsky Type 3,
Regular grammar
↔
Finite State Automaton

- Very many parts of language are indeed ‘finite state’, including phonology and prosody, morphology, most parts of sentence, text and discourse structures.
- Why is this so?
  - A finite state automaton or regular grammar only requires finite memory.
    All the other more complex kinds of grammar require, in principle, non-finite memory
  - It is plausible that real-time speech uses finite memory
    It is implausible that real-time speech uses non-finite memory.
  - It is plausible that memory can be expanded by rehearsal and by the use of writing, which employs external storage.

Syntagmatic Complexity

(3)



This is the dog that chased the cat that ate the mouse ...

Right-branching linear recursion / iteration.

Regular languages

Chomsky Type 3,
Regular grammar
↔
Finite State Automaton

- Very many parts of language are indeed ‘finite state’, including phonology and prosody, morphology, most parts of sentence, text and discourse structures.
- Why is this so?
  - The set of syllables in any language is finite and can be described with a non-iterative finite state automaton or non-recursive regular grammar.
  - The set of words in any language is not finite, but can be described by an iterative finite state automaton or a right-recursive (or left-recursive) regular grammar.

So what are these – the finite state automaton, the regular grammar?

Syntagmatic computing: State Machines

The most basic computing mode is the *State Machine*

- A set of states of the system
- A set of transitions between states
- Conditions on the transitions
- A starting state
- A set of terminating states

The simplest and classic type: *Finite State Automaton* (*FSA*)

- Finite automaton (DFSA), described by a quintuple:

  < Q, Σ, δ, q0, F >

  Q = a finite set of states

  Σ = a finite, nonempty input alphabet

  δ = a series of transition functions

  q_0 = the starting state

  F = the set of accepting (terminating states

- Deterministic: exactly one transition function for every $\sigma \in \Sigma$ from every $q \in Q$
- Nondeterministic: more than transition function for any $\sigma \in \Sigma$ from any $q \in Q$.

Syntagmatic computing: State Machines

Finite automaton (FSA), described by a quintuple:

< Q, Σ, δ, q0, F >

Q = a finite set of states

Σ = a finite, nonempty input alphabet

δ = a series of transition functions

q_0 = the starting state

F = the set of accepting (terminating states

- Deterministic (DFSA): exactly one transition function for every $\sigma \in \Sigma$ from every $q \in Q$
- Nondeterministic (NDFSA): more than transition function for any $\sigma \in \Sigma$ from any $q \in Q$
- Known principles:
  - An FSA with a transition which has the empty input symbol ϵ is an NDFSA.
  - For any NDFSA there is a weakly equivalent DFSA.
  - For any FSA there is a weakly equivalent *regular grammar* in the Chomsky-Schützenberger hierarchy of formal grammarsand vice versa.

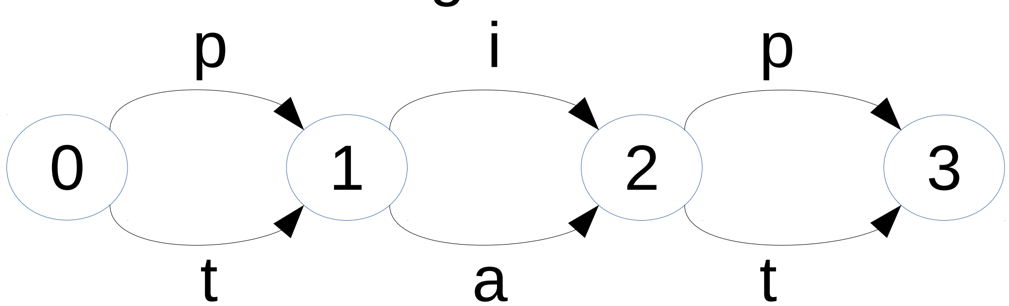
Syntagmatic computing: State Machines

There are several equivalent formalisms for FSTs

Transition table:

	0	1	2	3*
*0		p, t		
1			i, a	
2				p, t

Transition diagram:



Chomsky notation:

$$0 \rightarrow \begin{Bmatrix} p \\ t \end{Bmatrix} 1$$

$$1 \rightarrow \begin{Bmatrix} i \\ a \end{Bmatrix} 2$$

$$2 \rightarrow \begin{Bmatrix} p \\ t \end{Bmatrix}$$

BNF notation:

0 ::= p | t 1
1 ::= i | a 2
2 ::= p | t

State Machines and Grammars

Formal grammars have the structure < N, T, P, S>

N is a set of non-terminal symbols
the non-terminal vocabulary (sometimes called variables)

T is a set of terminal symbols, $N \cap T = \emptyset$
the terminal vocabulary

S is a starting string in $S \in N^*$
for context-free and regular grammars called starting symbol

P is a set of production rules of the form $\alpha \rightarrow \beta$
α and β are strings of symbols from $(N \cup T)^*$
conditions on α and β are different for each type of grammar

State Machines and Grammars

Type 0: Unrestricted Grammars

- $\alpha \in (N \cup T)^* N (N \cup T)^*$
- $\beta \in (N \cup T)^*$

Type 1: Context-sensitive Grammars

$|\alpha| <= |\beta|$, where there is no deletion

Type 2: Context-free Grammars

Phrase Structure Grammars, Constituent Structure Grammars

like Type 1, but

$\alpha \in N, |\alpha| = 1$

Type 3: Regular Grammars

like Type 2 but

1) $\beta \in T$, or
2) Either left regular or right regular, but not mixed:

left regular: $\beta = B\ a$, right regular: $\beta = a\ B$

for $a \in T$, $B \in N$

State Machines and Grammars

Type 0: Unrestric

- $\alpha \in (N \cup T)^*$
- $\beta \in (N \cup T)^*$

Type 1: Context-s

$|\alpha| <= |\beta|$, whe

Type 2: Context-

Phrase Stru ucture Grammars

like Type 1, but

$\alpha \in N$, $|\alpha| = 1$

The linguist’s favourite type:

In principle: Type 2

In Practice: Type 3
right-branching
(right regular)

Type 3: Regular Grammars

like Type 2 but

1) $\beta \in T$, or

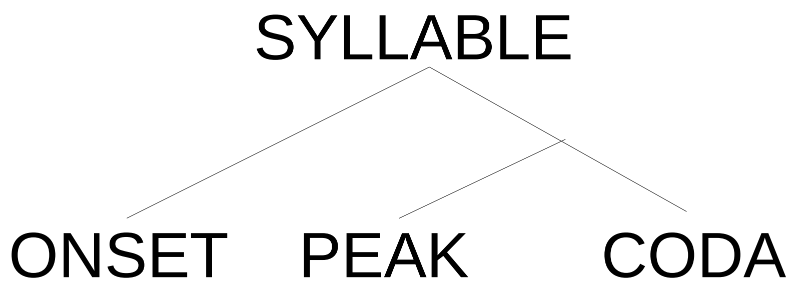
2) Either left regular or right regular, but not mixed:

left regular: β = B a, right regular: β = a B

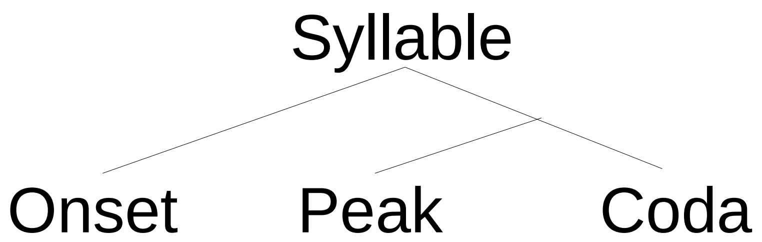
for $a \in T$, $B \in N$

Computational Syllable Phonotactics

Computational Syllable Phonotactics



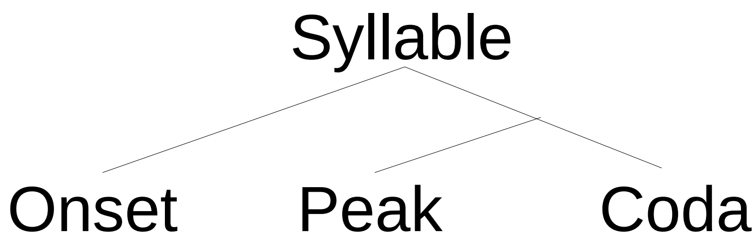
Computational Syllable Phonotactics

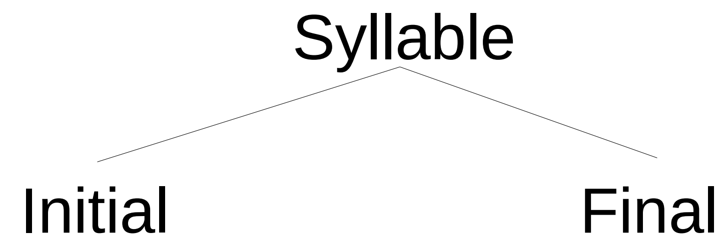


Notice that this is right-branching.

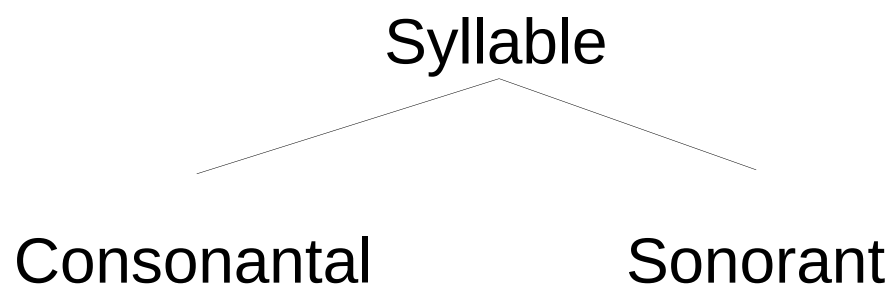
And what about Pǔtōnghuà?

Computational Syllable Phonotactics

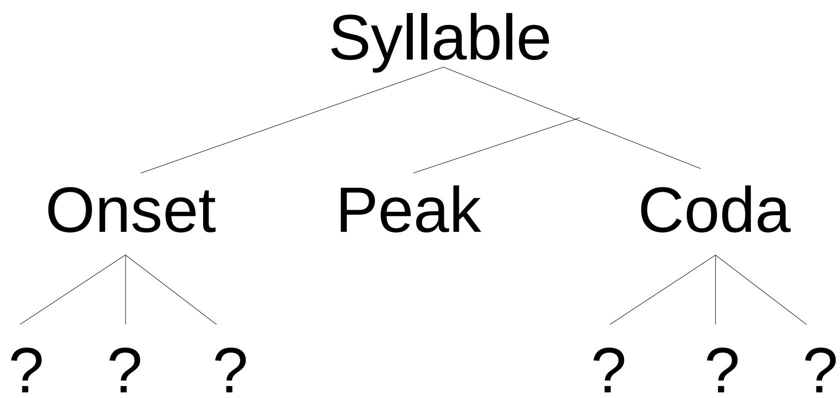


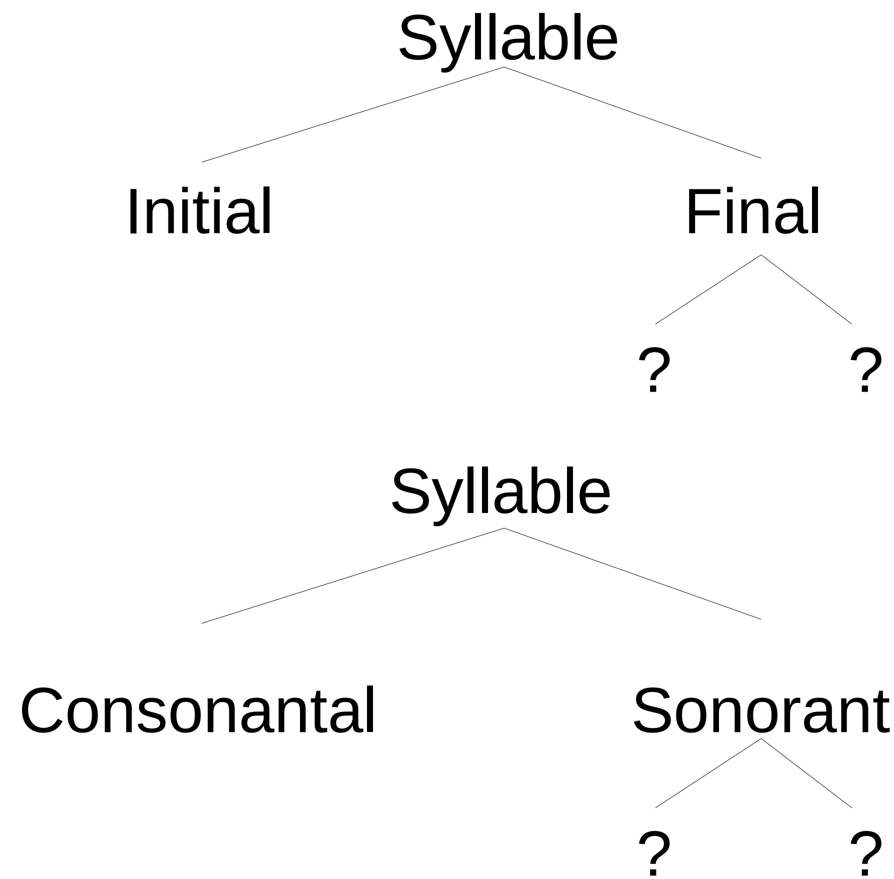


Notice that this is right-branching.



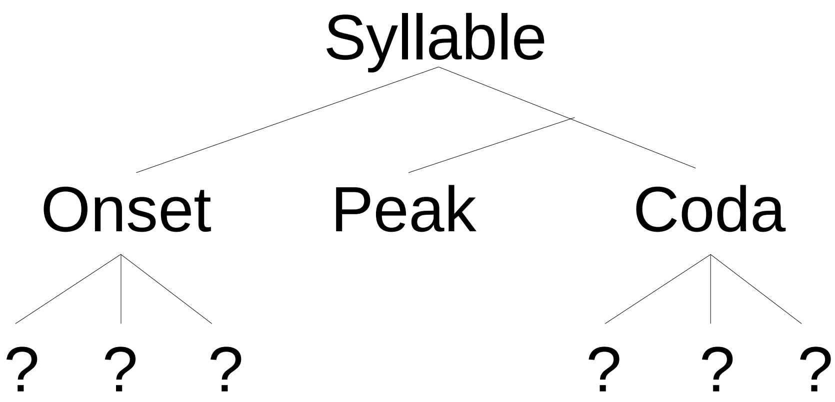
Computational Syllable Phonotactics

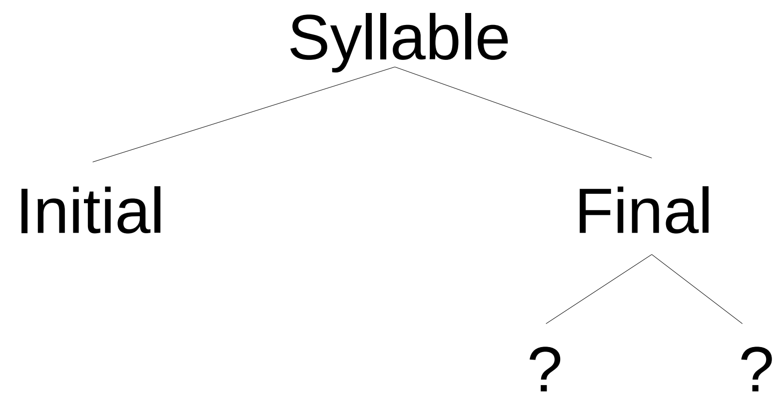




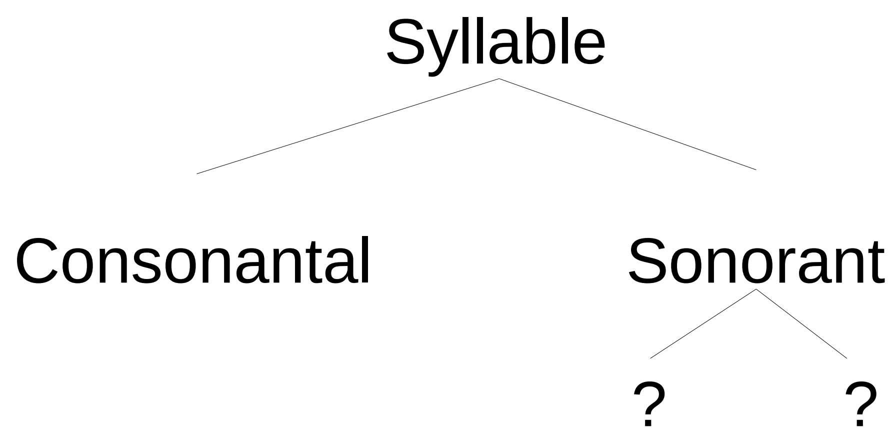
What do we call these finer grained end nodes?

Computational Syllable Phonotactics





SYLLABLE → ONSET NUCLEUS
NUCLEUS → PEAK CODA

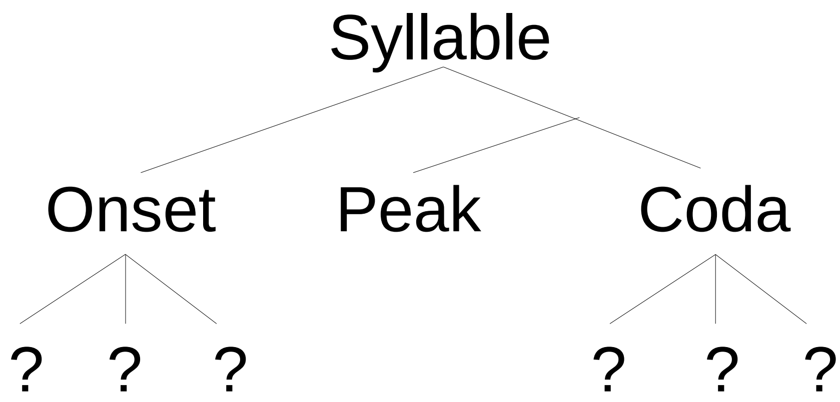


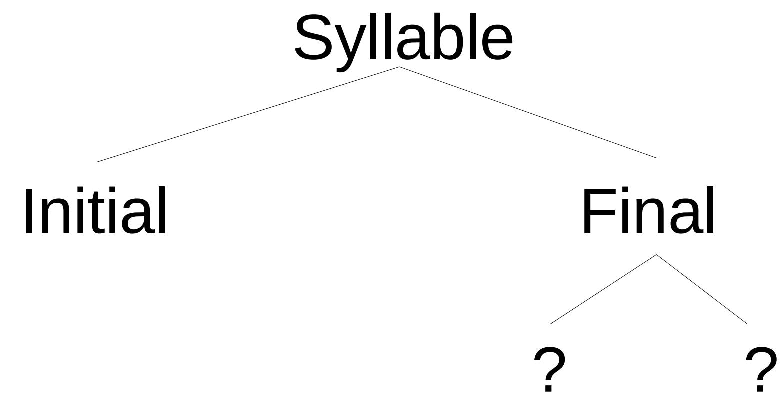
Presupposed by constraints in Optimality Theory:

ONSET
PEAK
NOCODA

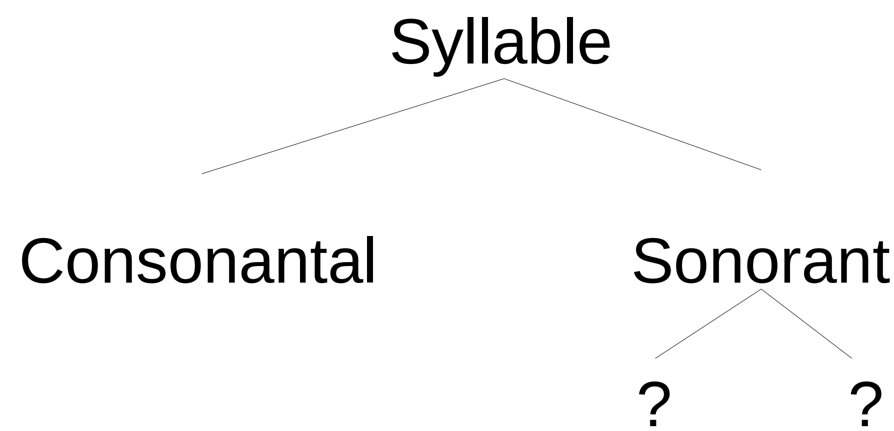
There are many more constraints.

Computational Syllable Phonotactics





SYLLABLE → ONSET NUCLEUS
NUCLEUS → PEAK CODA



Presupposed by constraints in Optimality Theory:

- ONSET
- PEAK
- NOCODA

Linguists love drawing trees, but tend to forget about the underlying grammars!

Computational Phonotactics: English asymmetries

Computational Phonotactics: English asymmetries

What do the constraints on English syllable patterns really look like?

Let’s take a look at a Finite State Automaton

Computational Phonotactics: English asymmetries

English onset constraints:
1 rule per context

\# s + ...
q1 → s q2

\# post-s AnteriorVoicelessCons
q2 → p
q2 → t
q2 → k
q2 → m
q2 → n
q2 → l
q2 → w

\# post-s VoicelessStop + Gliquid
q2 → t q4
q2 → p q6
q2 → k q7

\# post-s VoicelessStop + Gliquid
q4 → r

\# post-s VoicelessStop + Gliquid
q6 → r
q6 → l

\# post-s VoicelessStop + Gliquid
q7 → r
q7 → l
q7 → w

\# post-s VoicessCons + j
q2 → p q3
q2 → t q3
q2 → k q3

\# Consonant + j + u
q3 → j

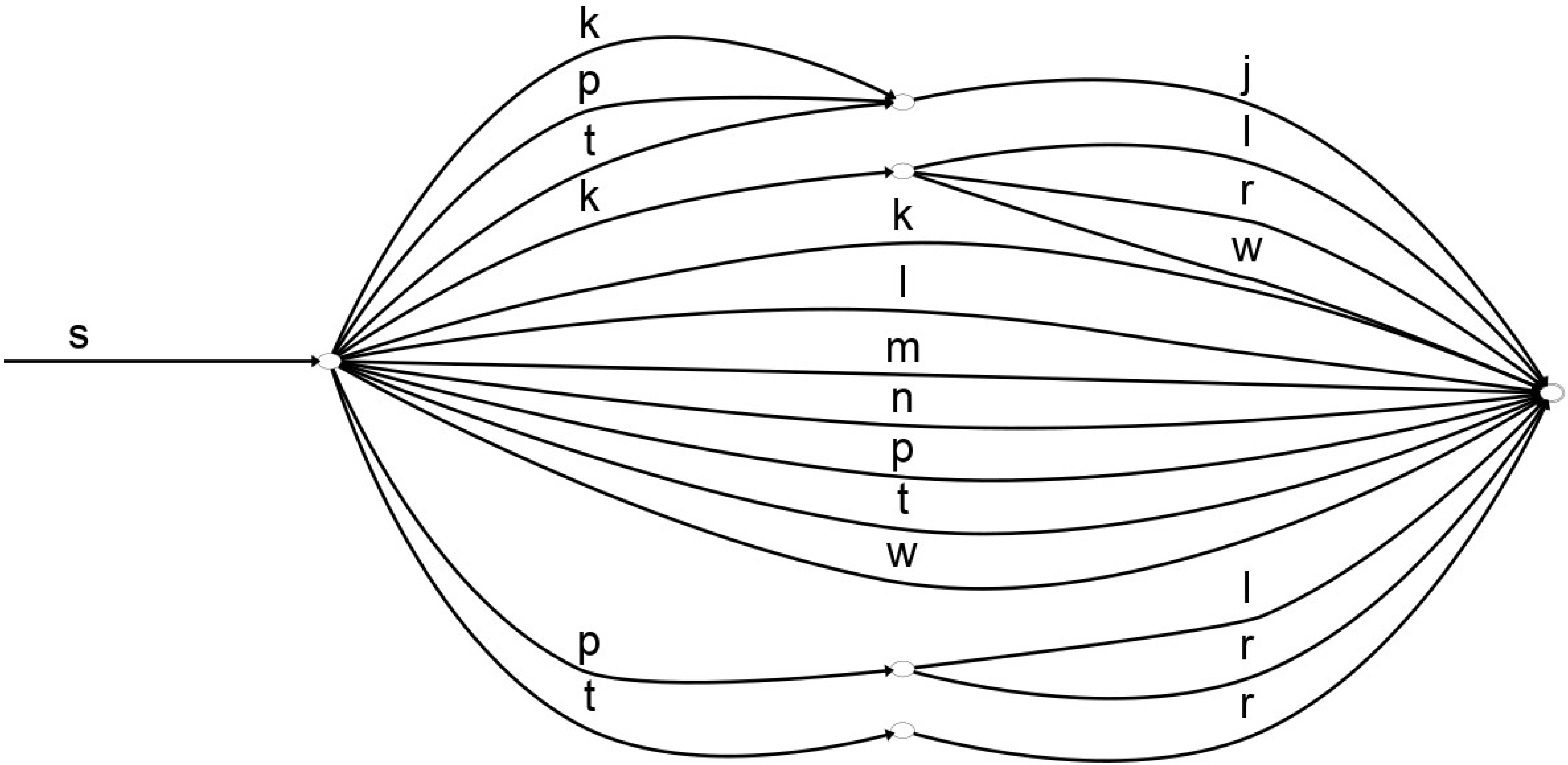
Computational Phonotactics: English asymmetries

English #s__ onset constraints,
Implementation as an NDFST

```
#   s + ...
     q1,s,q2;

#  post-s AnteriorVoicelessCons
     q2,p,q9;
     q2,t,q9;
     q2,k,q9;
     q2,m,q9;
     q2,n,q9;
     q2,l,q9;
     q2,w,q9;

#   post-s VoicelessStop + Gliquid
     q2,t,q4;
     q2,p,q6;
     q2,k,q7;

#   post-s VoicelessStop + Gliquid
     q4,r,q9;

#   post-s VoicelessStop + Gliquid
     q6,r,q9;
     q6,l,q9;

#   post-s VoicelessStop + Gliquid
     q7,r,q9;
     q7,l,q9;
     q7,w,q9;

#  post-s VoicessCons + j
     q2,p,q3;
     q2,t,q3;
     q2,k,q3;

#  Consonant + j + u
     q3,j,q9;
```

Computational Phonotactics: English asymmetries

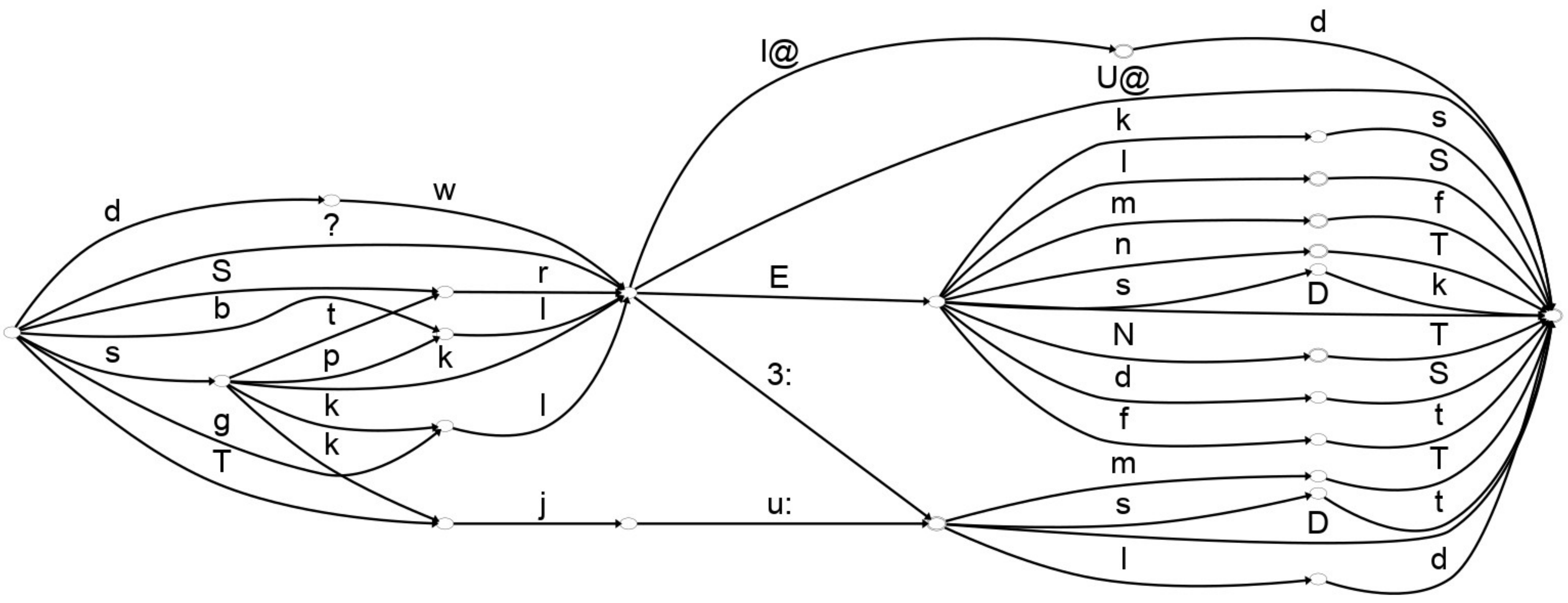
English #s__ onset constraints



http://localhost/Syllables/English/english-syllonsets-demo.html

Computational Phonotactics: English asymmetries

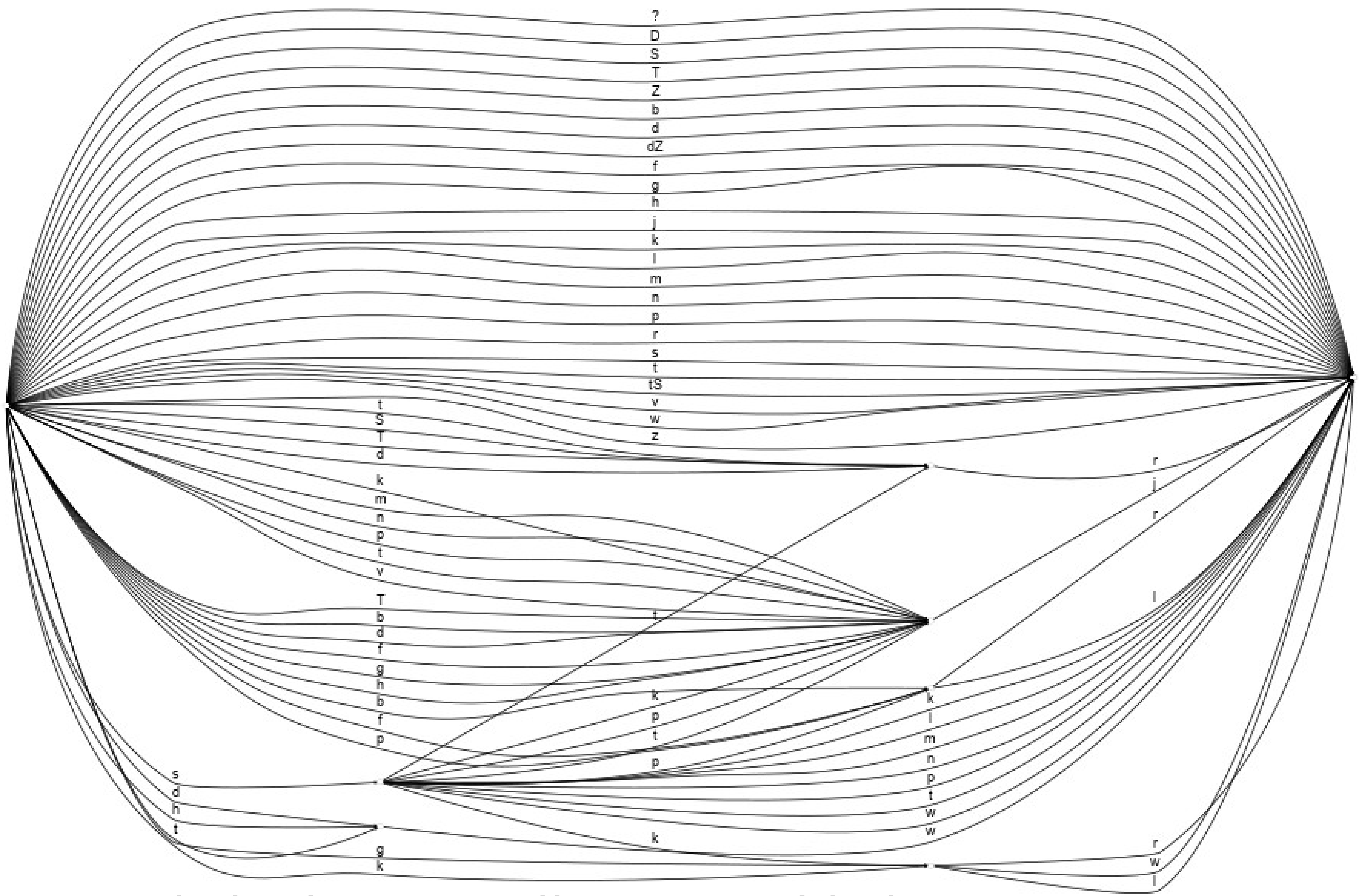
English syllable structure, parallel transitions reduced to one, with a single vocabulary item



http://localhost/Syllables/English/english-syllables-demo.html

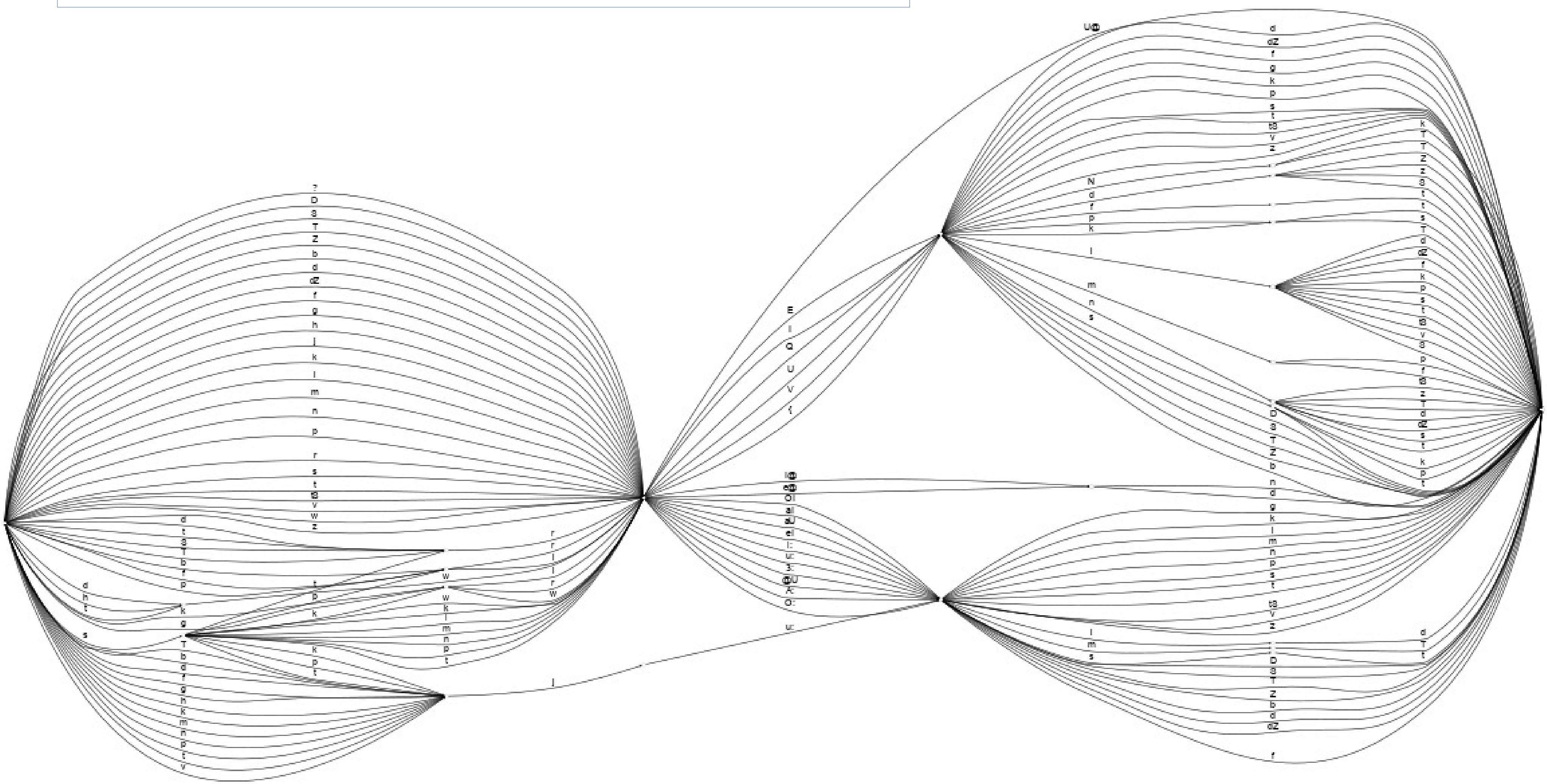
Computational Phonotactics: English asymmetries

English onset structure in full detail,
one transition per vocabulary item



Computational Phonotactics: English asymmetries

English syllable structure in full detail,
one transition per vocabulary item

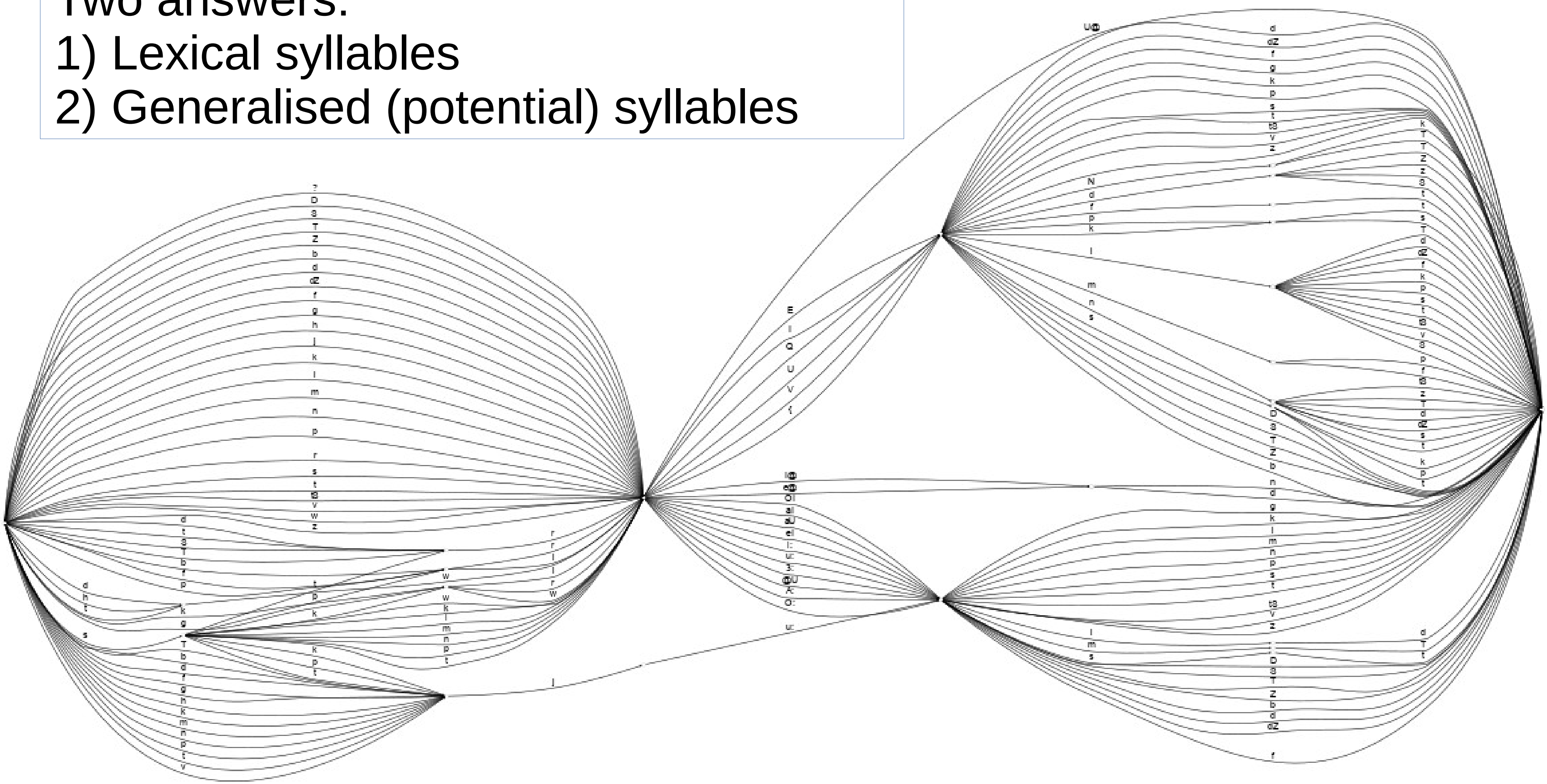


http://localhost/Syllables/English/english-syllables-demo.html

Computational Phonotactics: English asymmetries

How many English syllables are there?

Two answers:
1) Lexical syllables
2) Generalised (potential) syllables



http://localhost/Syllables/English/english-syllables-demo.html

Computational Phonotactics: Pǔtōnghuà symmetries

Computational Phonotactics: Pǔtōnghuà symmetries

	b	p	m	f	d	t	n	l	g	k	h	z	c	s	zh	ch	sh	r	j	q	x	⁂
a	ba	pa	ma	fa	da	ta	na	la	ga	ka	ha	za	ca	sa	zha	cha	sha					a
o	bo	po	mo	fo																		o
e			me		de	te	ne	le	ge	ke	he	ze	ce	se	zhe	che	she	re				e
ai	bai	pai	mai		dai	tai	nai	lai	gai	kai	hai	zai	cai	sai	zhai	chai	shai					ai
ei	bei	pei	mei	fei	dei	tei	nei	lei	gei	kei	hei	zei			zhei		shei					ei
ao	bao	pao	mao		dao	tao	nao	lao	gao	kao	hao	zao	cao	sao	zhao	chao	shao	rao				ao
ou		pou	mou	fou	dou	tou	nou	lou	gou	kou	hou	zou	cou	sou	zhou	chou	shou	rou				ou
an	ban	pan	man	fan	dan	tan	nan	lan	gan	kan	han	zan	can	san	zhan	chan	shan	ran				an
ang	bang	pang	mang	fang	dang	tang	nang	lang	gang	kang	hang	zang	cang	sang	zhang	chang	shang	rang				ang
en	ben	pen	men	fen	den		nen		gen	ken	hen	zen	cen	sen	zhen	chen	shen	ren				en
eng	beng	peng	meng	feng	deng	teng	neng	leng	geng	keng	heng	zeng	ceng	seng	zheng	cheng	sheng	reng				eng
ong					dong	tong	nong	long	gong	kong	hong	zong	cong	song	zhong	chong		rong				ong
u	bu	pu	mu	fu	du	tu	nu	lu	gu	ku	hu	zu	cu	su	zhu	chu	shu	ru				wu *
ua									gua	kua	hua				zhua	chua	shua	rua				wa *
uo					duo	tuo	nuo	luo	guo	kuo	huo	zuo	cuo	suo	zhuo	chuo	shuo	ruo				wo *
uai									guai	kuai	huai				zhuai	chuai	shuai					wai *
ui					dui	tui			gui	kui	hui	zui	cui	sui	zhui	chui	shui	rui				wei * 1
uan					duan	tuan	nuan	luan	guan	kuan	huan	zuan	cuan	suan	zhuan	chuan	shuan	ruan				wan *
uang									guang	kuang	huang				zhuang	chuang	shuang					wang *
un					dun	tun	nun	lun	gun	kun	hun	zun	cun	sun	zhun	chun	shun	run				wen * 2
ueng																						weng *
i	bi	pi	mi		di	ti	ni	li				zi †	ci †	si †	zhi ‡	chi ‡	shi ‡	ri ‡	ji	qi	xi	yi +
ia					dia			lia											jia	qia	xia	ya +
ie	bie	pie	mie		die	tie	nie	lie											jie	qie	xie	ye +
iao	biao	piao	miao		diao	tiao	niao	liao											jiao	qiao	xiao	yao +
iu			miu		diu		niu	liu											jiu	qiu	xiu	you + 3
ian	bian	pian	mian		dian	tian	nian	lian											jian	qian	xian	yan +
iang							niang	liang											jiang	qiang	xiang	yang +
in	bin	pin	min				nin	lin											jin	qin	xin	yin +
ing	bing	ping	ming		ding	ting	ning	ling											jing	qing	xing	ying +
iong																			jiong	qiong	xiong	yong +
ü							nü	lü											ju ※	qu ※	xu ※	yu ※
üe							nüe	lüe											jue ※	que ※	xue ※	yue ※
üan																			juan ※	quan ※	xuan ※	yuan ※
ün																			jun ※	qun ※	xun ※	yun ※

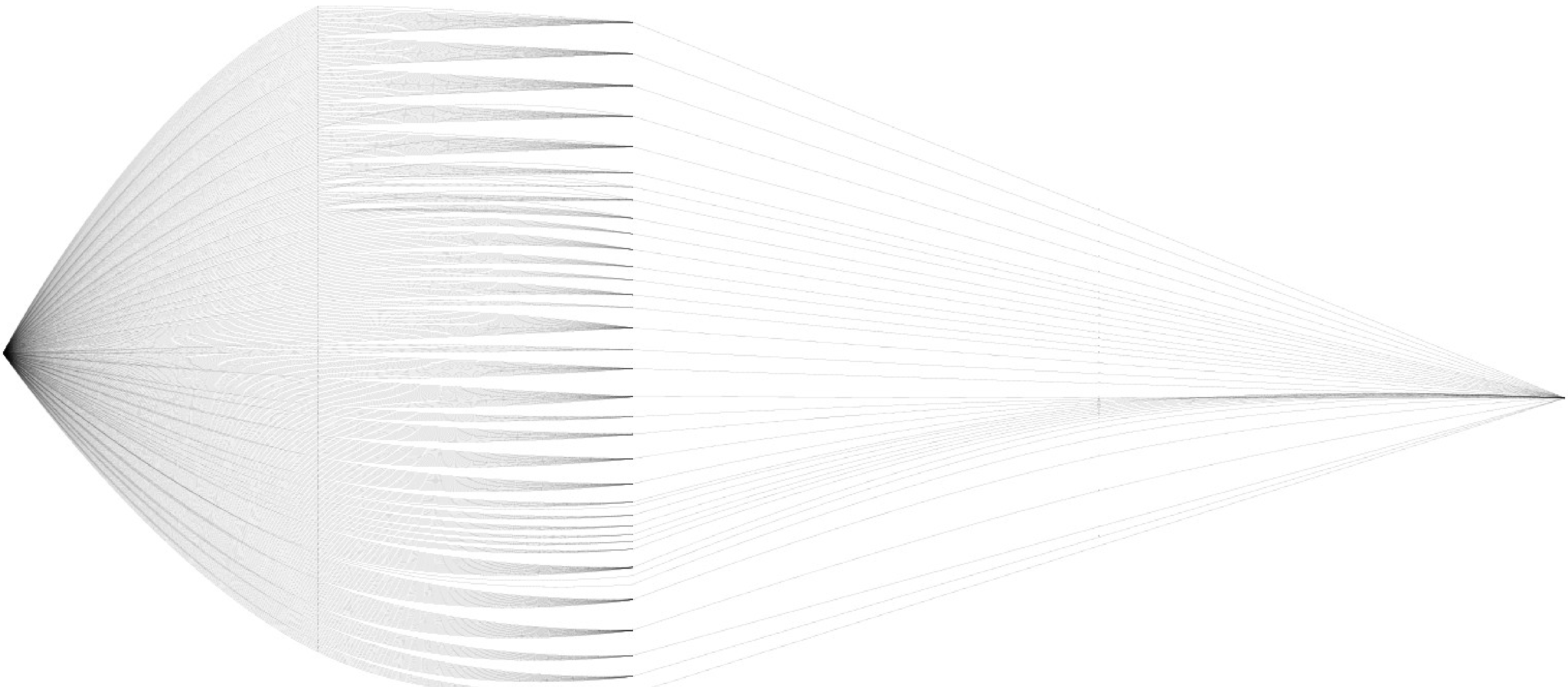
Computational Phonotactics: Pǔtōnghuà symmetries

Model 1:

Pinyin table, grouped initials, non-deterministic

Exact model: sound and complete

How many syllables?



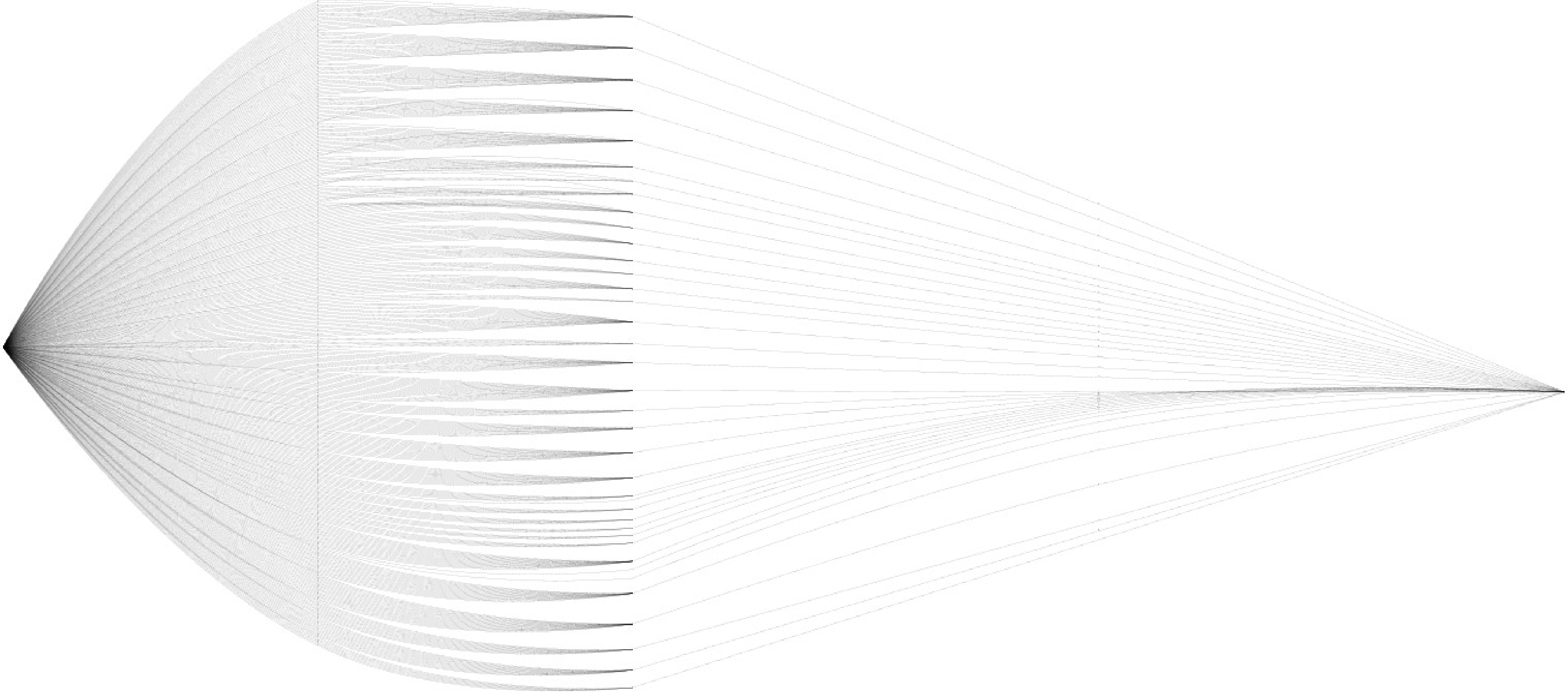
http://localhost/Syllables/Mandarin/

Computational Phonotactics: Pǔtōnghuà symmetries

Model 1:

Pinyin table, grouped initials, non-deterministic

Exact model: sound and complete



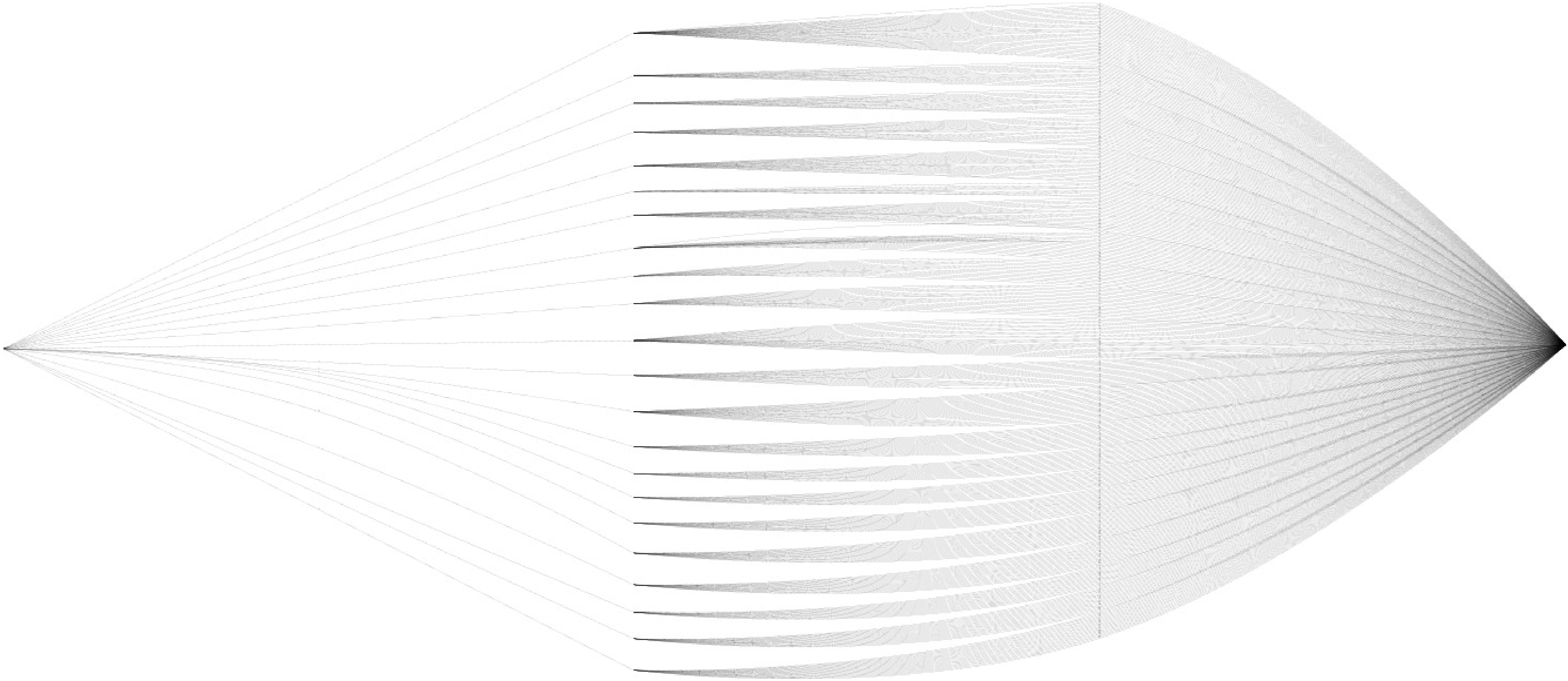
http://localhost/Syllables/Mandarin/

Computational Phonotactics: Pǔtōnghuà symmetries

Model 2:

Pinyin table, grouped finals, deterministic

Exact model: sound and complete)



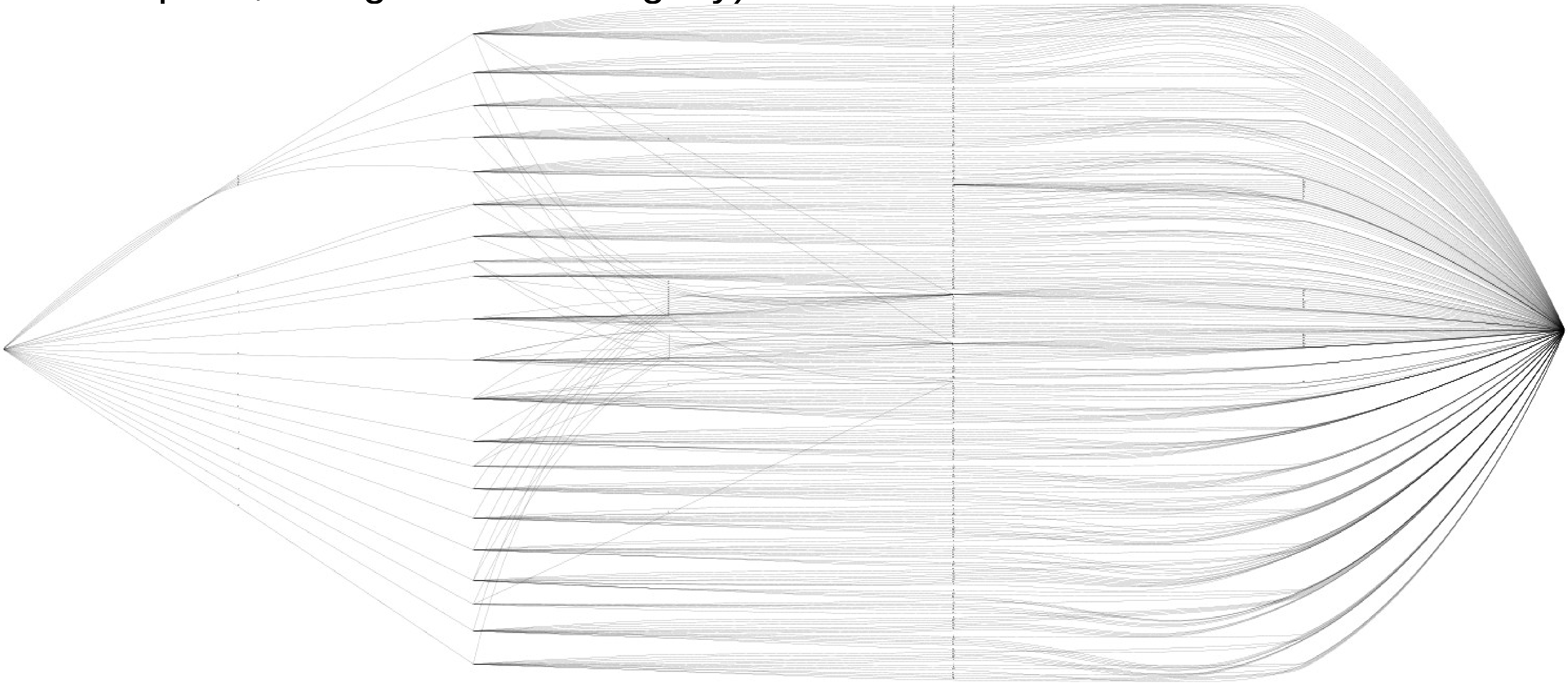
http://localhost/Syllables/Mandarin/

Computational Phonotactics: Pǔtōnghuà symmetries

Model 3 (full):

Node inserted for onset glides

Complete, overgeneralises slightly)



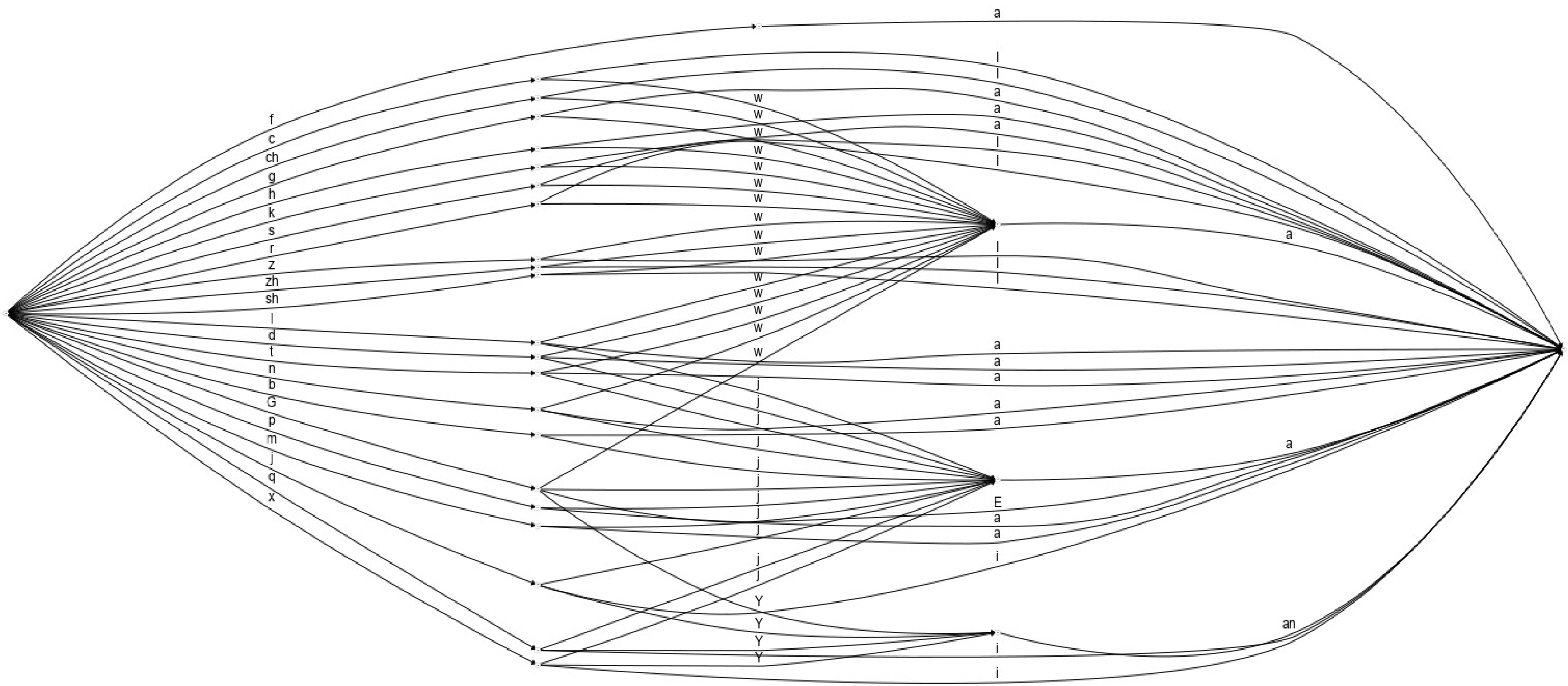
http://localhost/Syllables/Mandarin/

Computational Phonotactics: Pǔtōnghuà symmetries

Model 3 (compact):

Node inserted for onset glides

Complete, overgeneralises slightly)



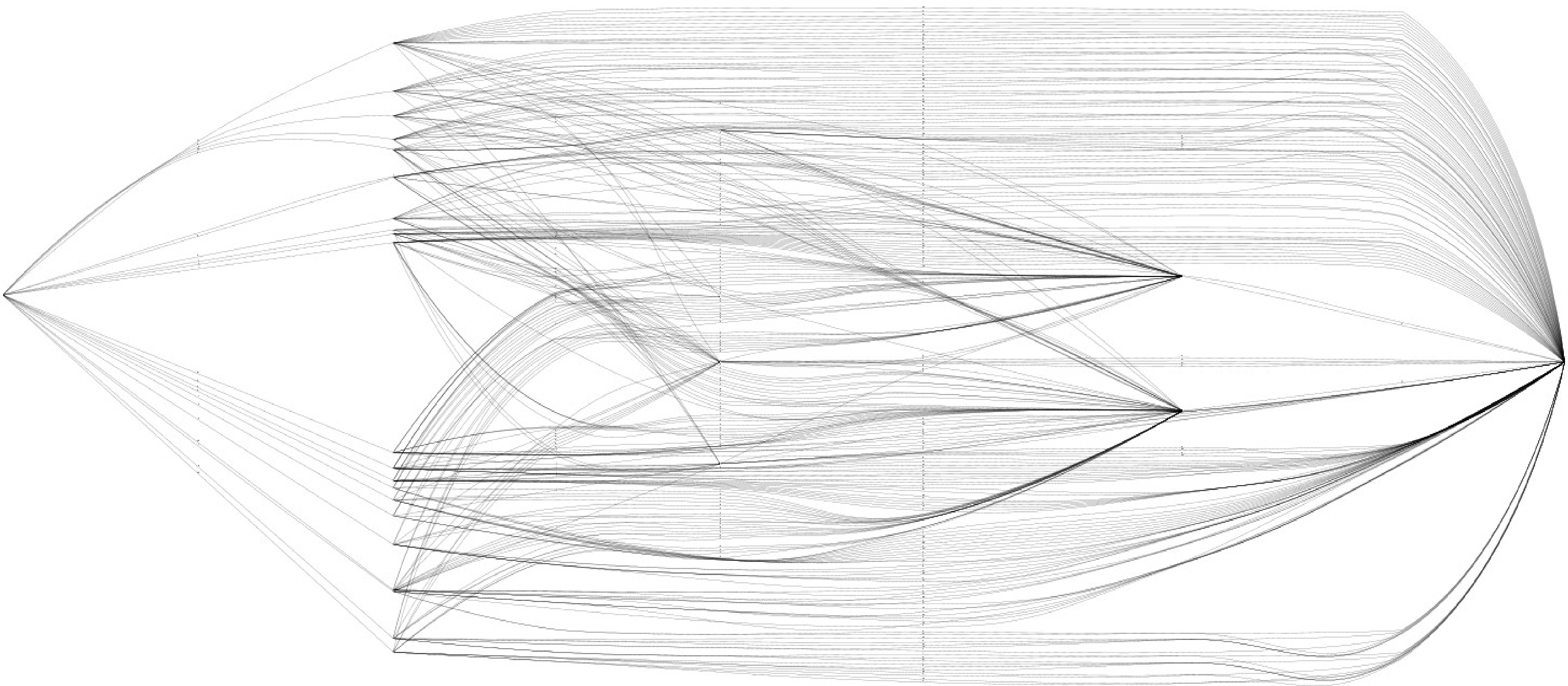
http://localhost/Syllables/Mandarin/

Computational Phonotactics: Pǔtōnghuà symmetries

Model 4 (full):

Nodes inserted for onset glides and coda nasals

Complete, overgeneralises slightly, the most complex model



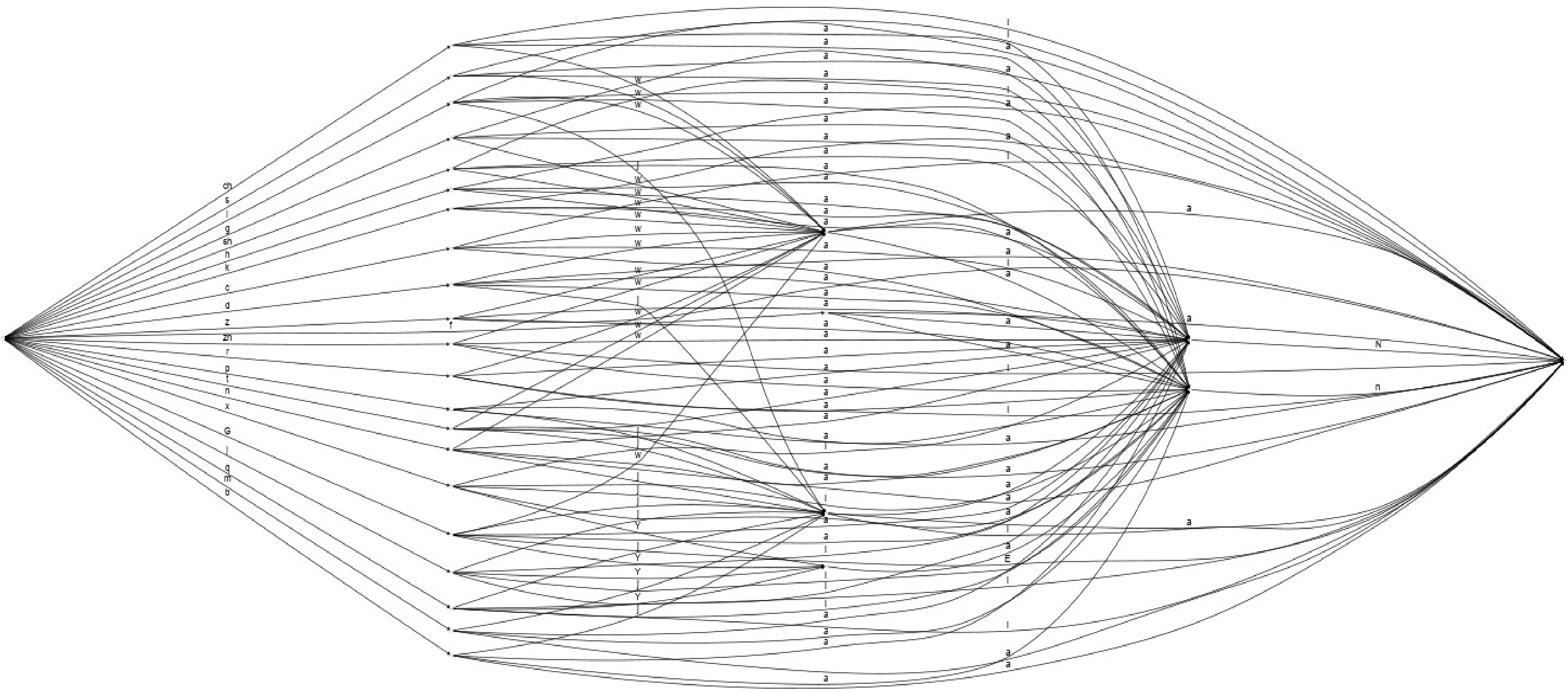
http://localhost/Syllables/Mandarin/

Computational Phonotactics: Pǔtōnghuà symmetries

Model 4 (compact):

Nodes inserted for onset glides and coda nasals

Complete, overgeneralises slightly, the most complex model



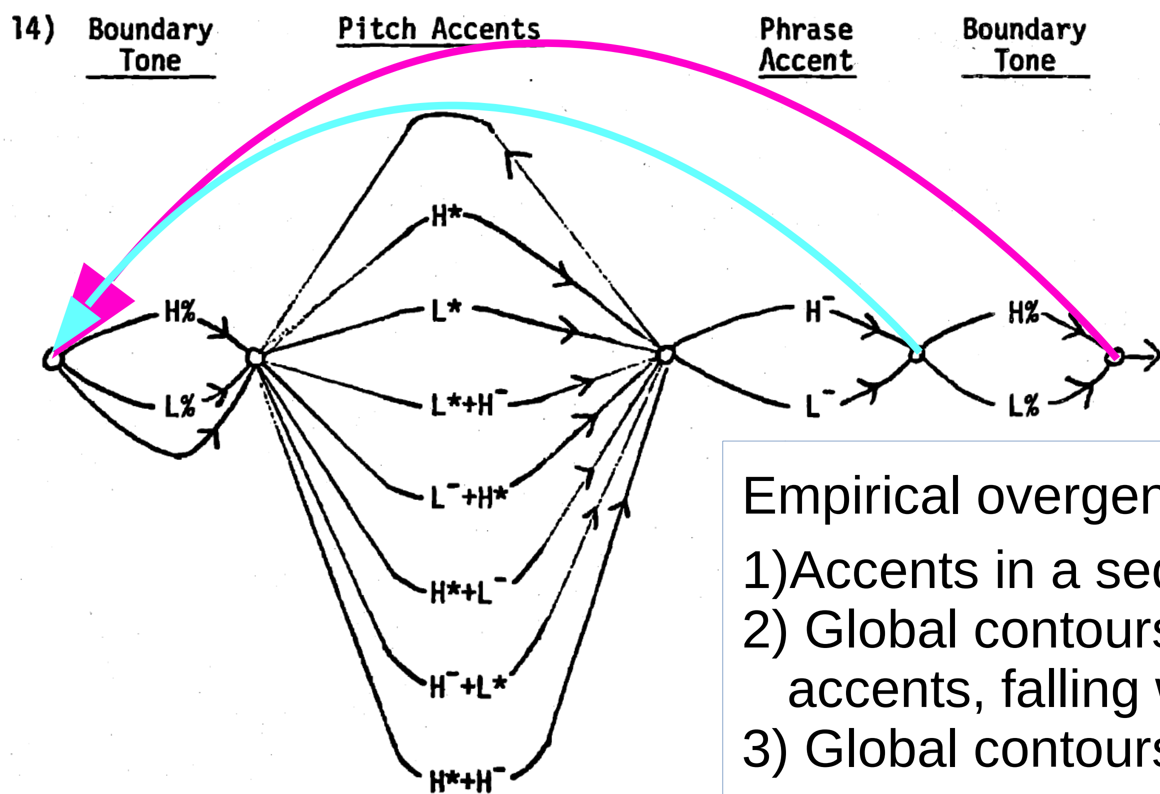
http://localhost/Syllables/Mandarin/

Computational Sentence Prosody - Pierrehumbert

Computational Sentence Prosody - Pierrehumbert

Intonational iteration as a layered hierarchy of loops (linear abstract oscillations)

Pierrehumbert's regular grammar / finite state transition network



Not the first (cf. Reich, ’t Hart et al., Fujisaki, …)

But linguistically the most interesting.

Empirical overgeneration:

1)Accents in a sequence tend to be all H* or all L*
2) Global contours tend to be rising with L* accents, falling with H* accents
3) Global contours may span more than 1 turn

Empirical undergeneration:

1)Paratone hierarchy not included
2)No time constraints

Computational Sentence Prosody - Pierrehumbert

initial=q0

terminal=q4

fst=

q0,H%,x,q1;

q0,L%,x,q1;

q0, ,x,q1;

q1,H*,x,q2;

q1,L*,x,q2;

q1,L*+H-,x,q2;

q1,L-+H*,x,q2;

q1,H*+L-,x,q2;

q1,H-+L*,x,q2;

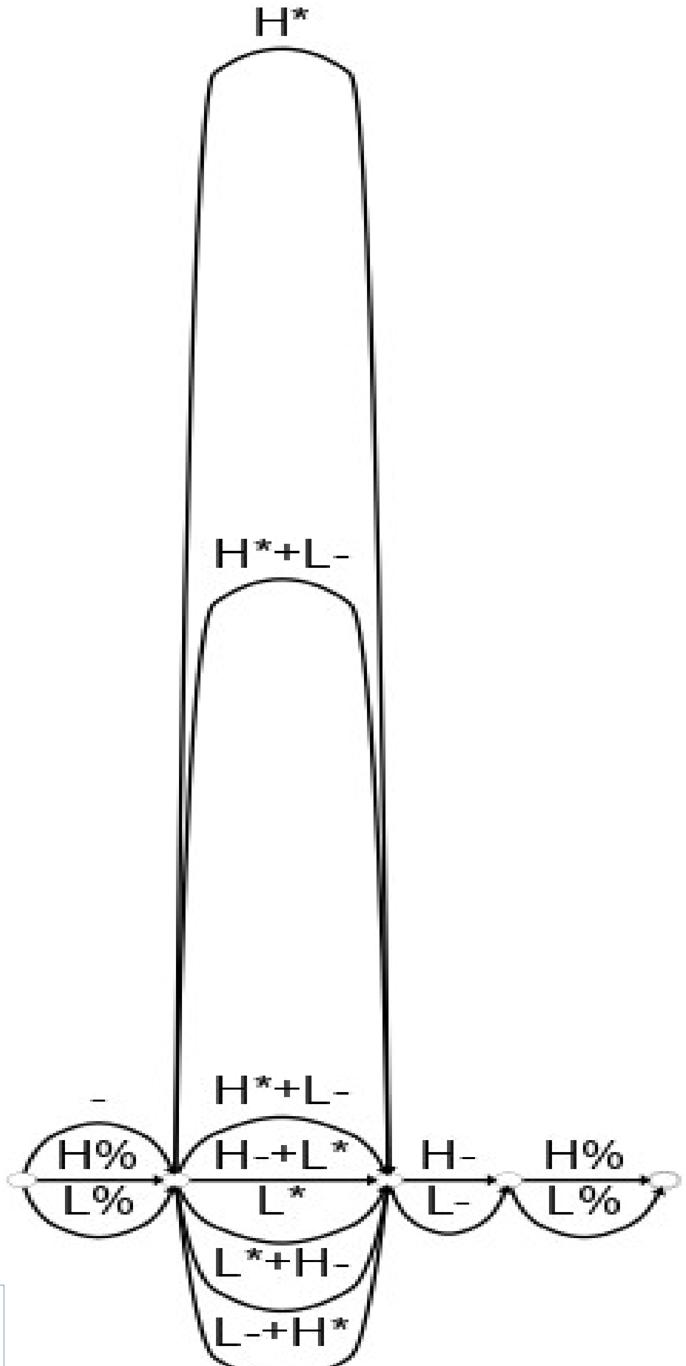
q1,H*+L-,x,q2;

q2,H-,x,q3;

q2,L-,x,q3;

q3,H%,x,q4;

q3,L%,x,q4;

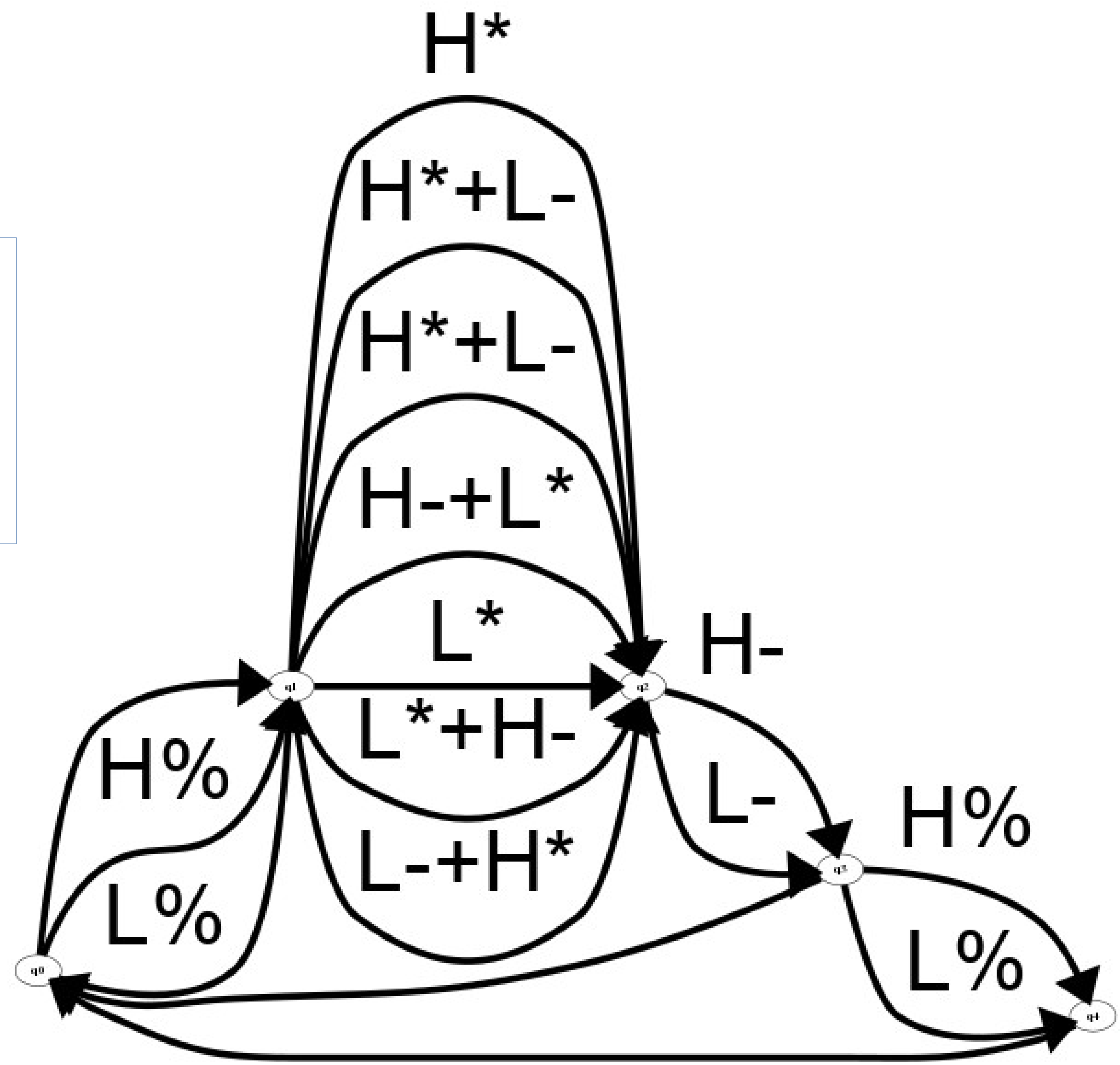


http://localhost/Syllables/Prosody/pierrehumbert.html

Computational Sentence Prosody - Pierrehumbert

Pierrehumbert’s FST with additional iteration loops for

- Intermediate Phrase
- Intonation Phrase

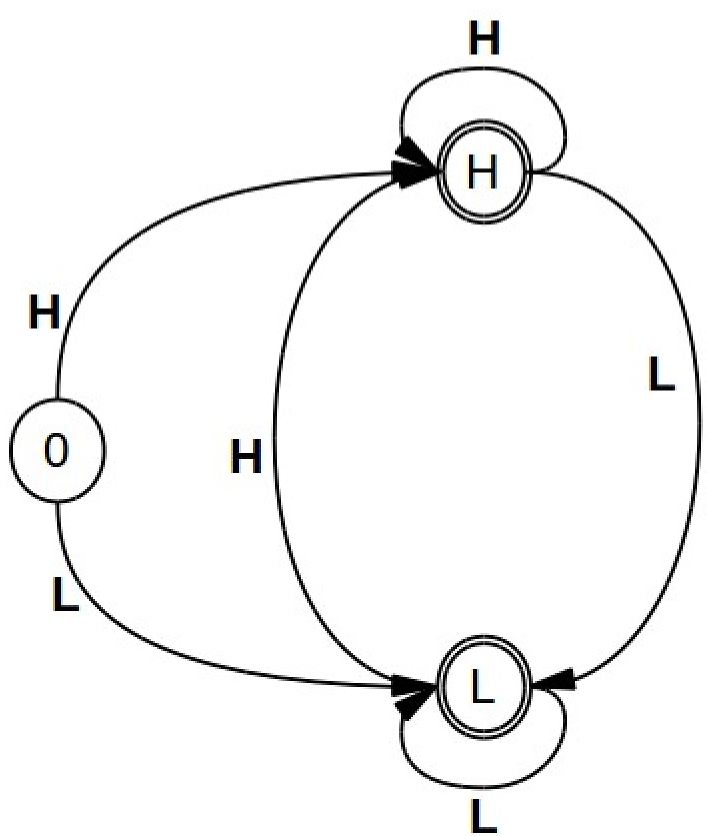


http://localhost/Syllables/Prosody/pierrehumbert.html

Computational lexical prosody: Niger-Congo terraced tone

Computational lexical prosody: Niger-Congo terraced tone

Niger-Congo Iterative Tonal Sandhi – a 1-tape FSA



At the most abstract level, just one node with H and L cycling around it.

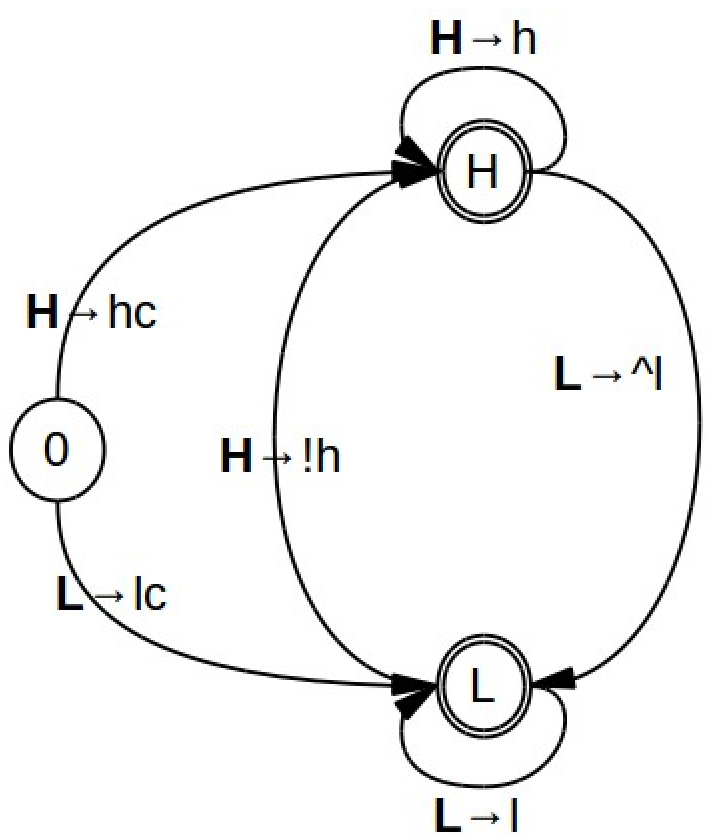
From an allotonic point of view:

- 3 cycles
- 1-tape (1-level) transition network

Computational lexical prosody: Niger-Congo terraced tone

Niger-Congo Iterative Tonal Sandhi – a 2-tape FST

Syntagmatic + Interpretative Computing



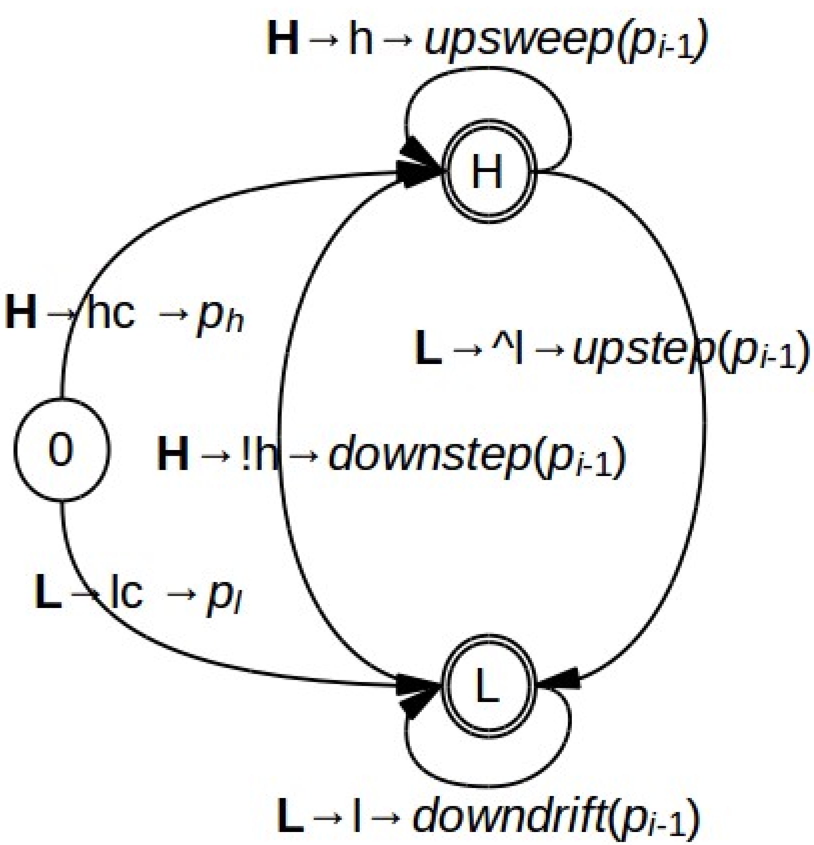
From an allotonic point of view:

- 3 cycles
- 2-tape (= 2-level) transition network

Computational lexical prosody: Niger-Congo terraced tone

Niger-Congo Iterative Tonal Sandhi – a 3-tape FST

Syntagmatic + Interpretative Computing



From phonetic signal processing point of view:

- 3 cycles
- 3-tape (= 3-level) transition network

Computational lexical prosody: Niger-Congo terraced tone

Implementation as FST

Niger-Congo terraced tone sandhi, 2 tones

initial=q0

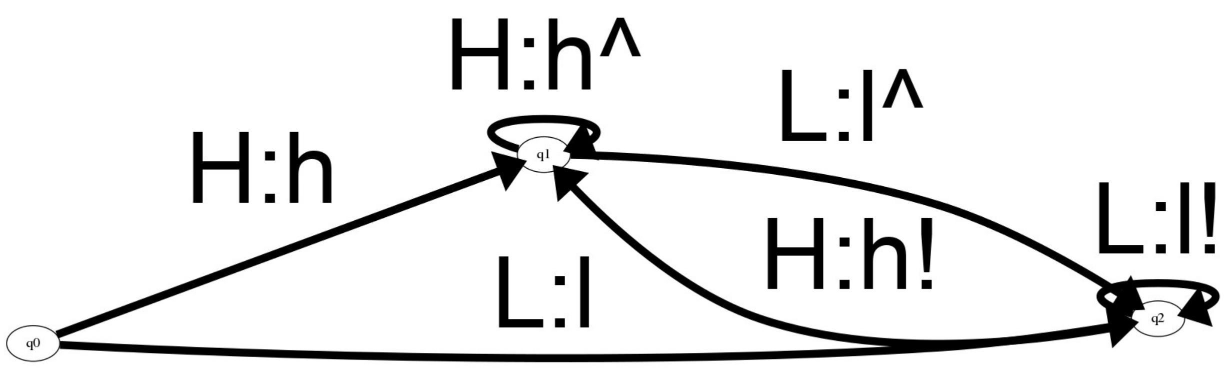
terminal=q1,q2

fst=

q0,H,h,q1;
q0,L,l,q2;

q1,H,h^,q1;
q1,L,l^,q2;

q2,L,l!,q2;
q2,H,h!,q1;

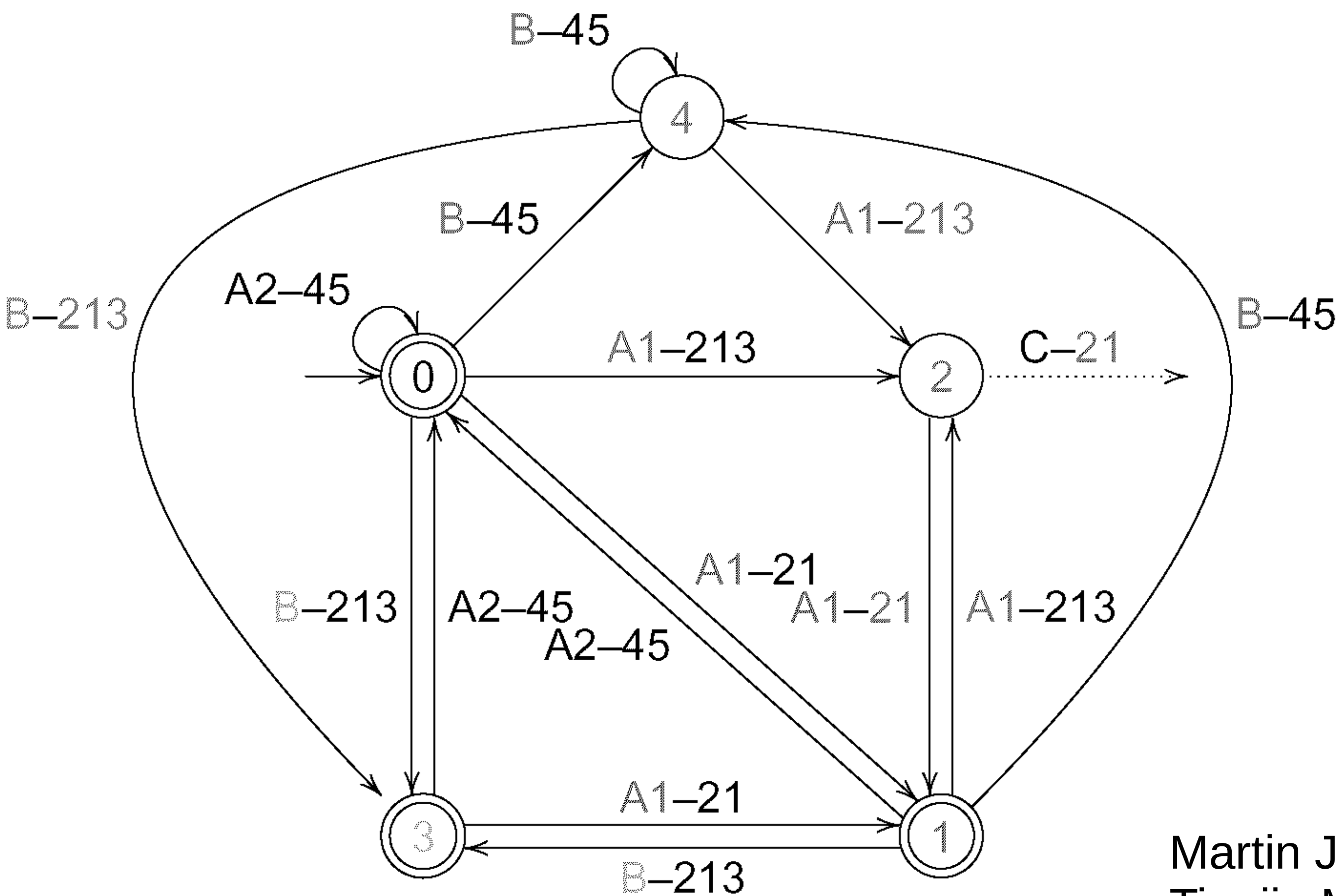


http://localhost/Syllables/Prosody/nigercongo.html

Computational lexical prosody: Tianjin tone sandhi

Computational lexical prosody: Niger-Congo terraced tone

Tianjin Dialect Iterative Tonal Sandhi

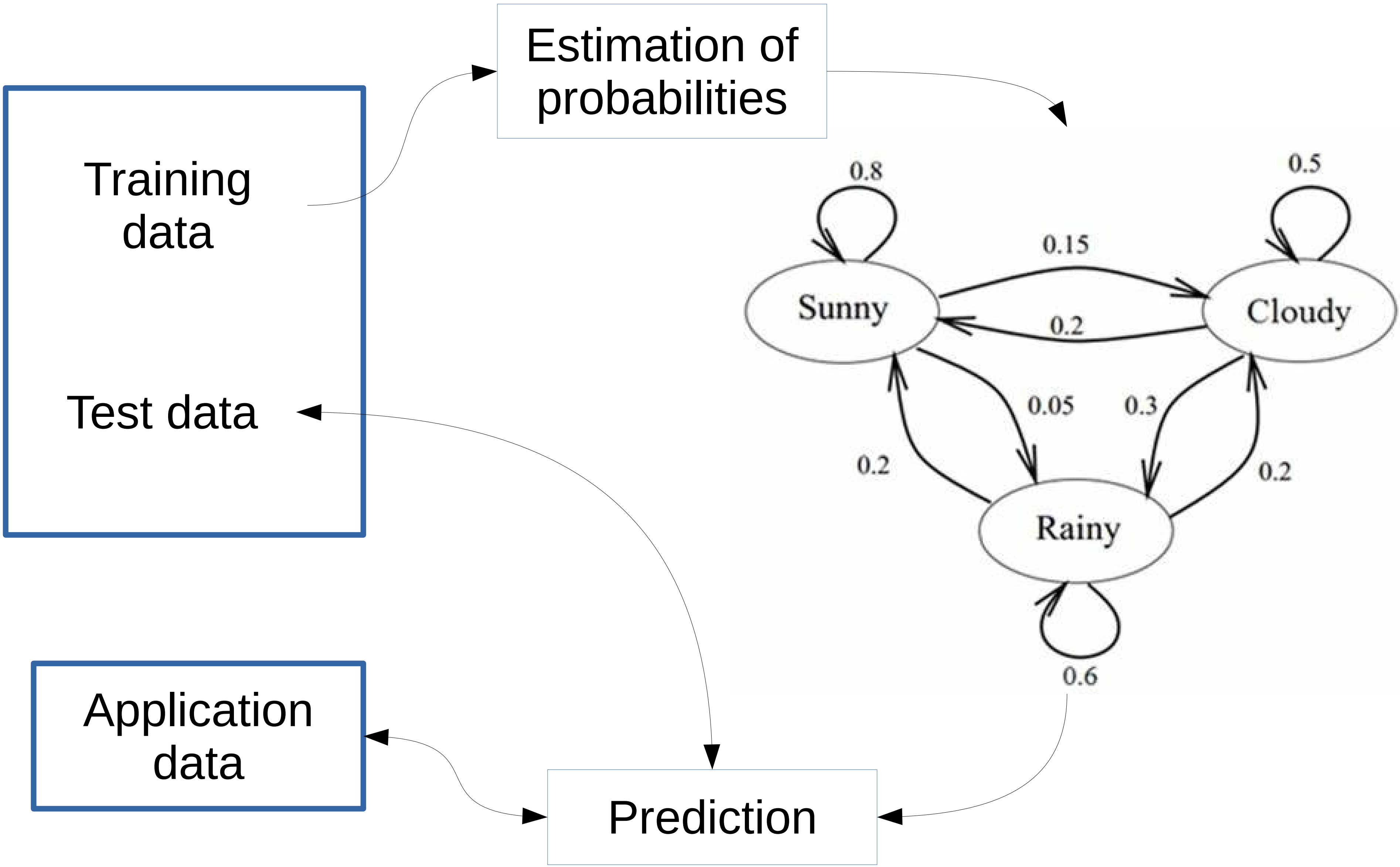


Martin Jansche 1998
Tianjin Mandarin tone sandhi

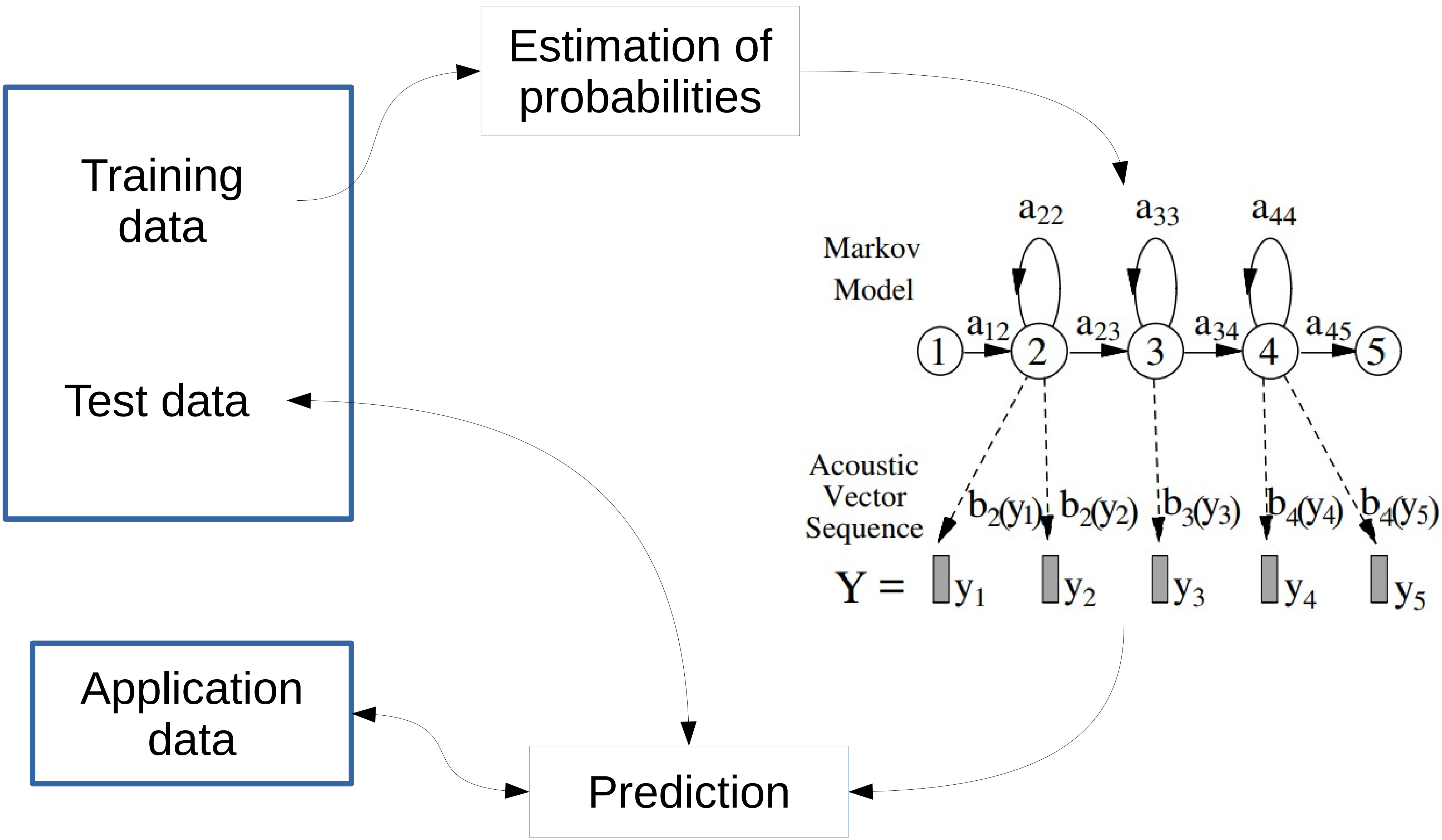
FSTs in Automatic Speech Recognition:

Hidden Markov Models (HMMs)

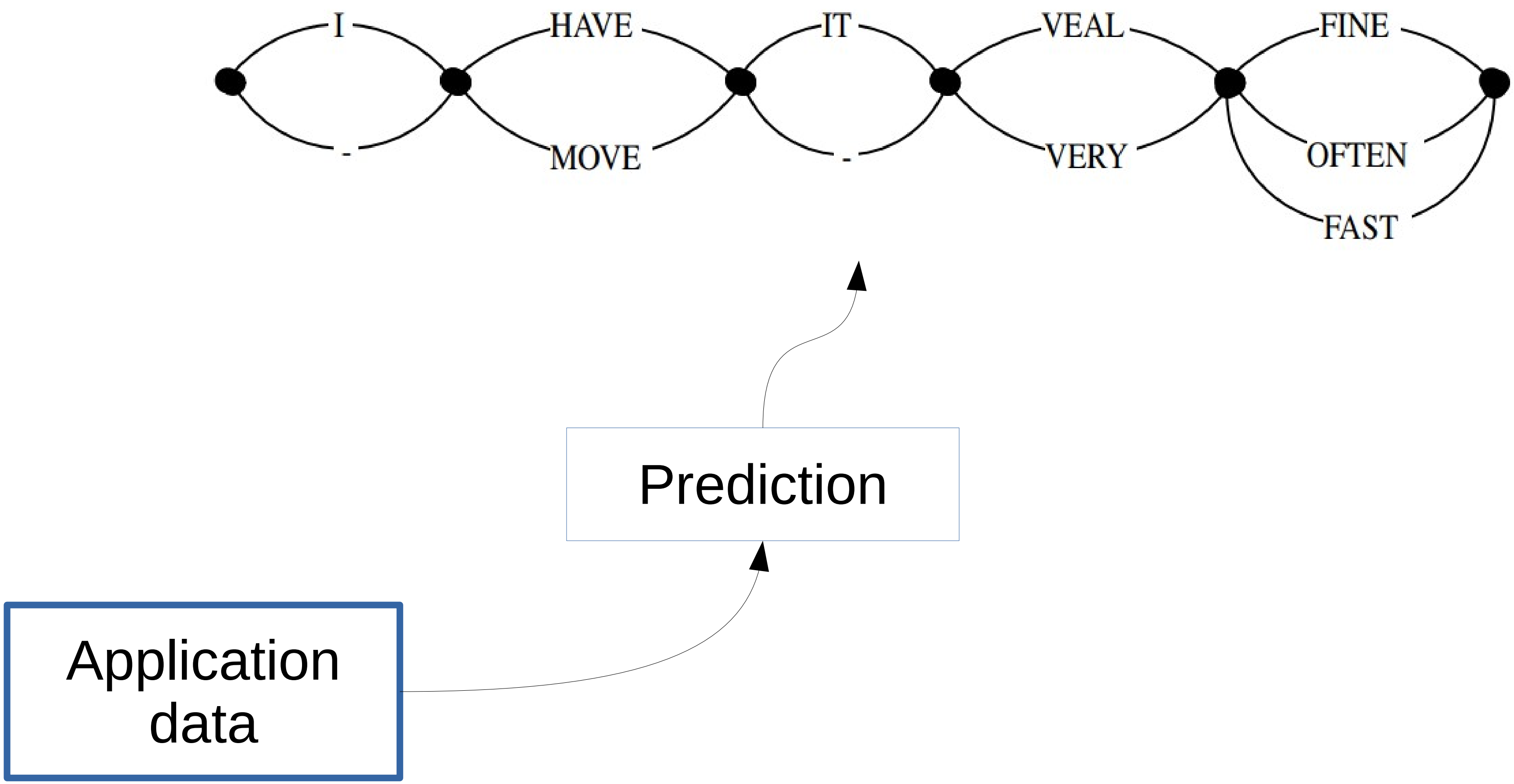
FSTs in Automatic Speech Recognition: HMMs



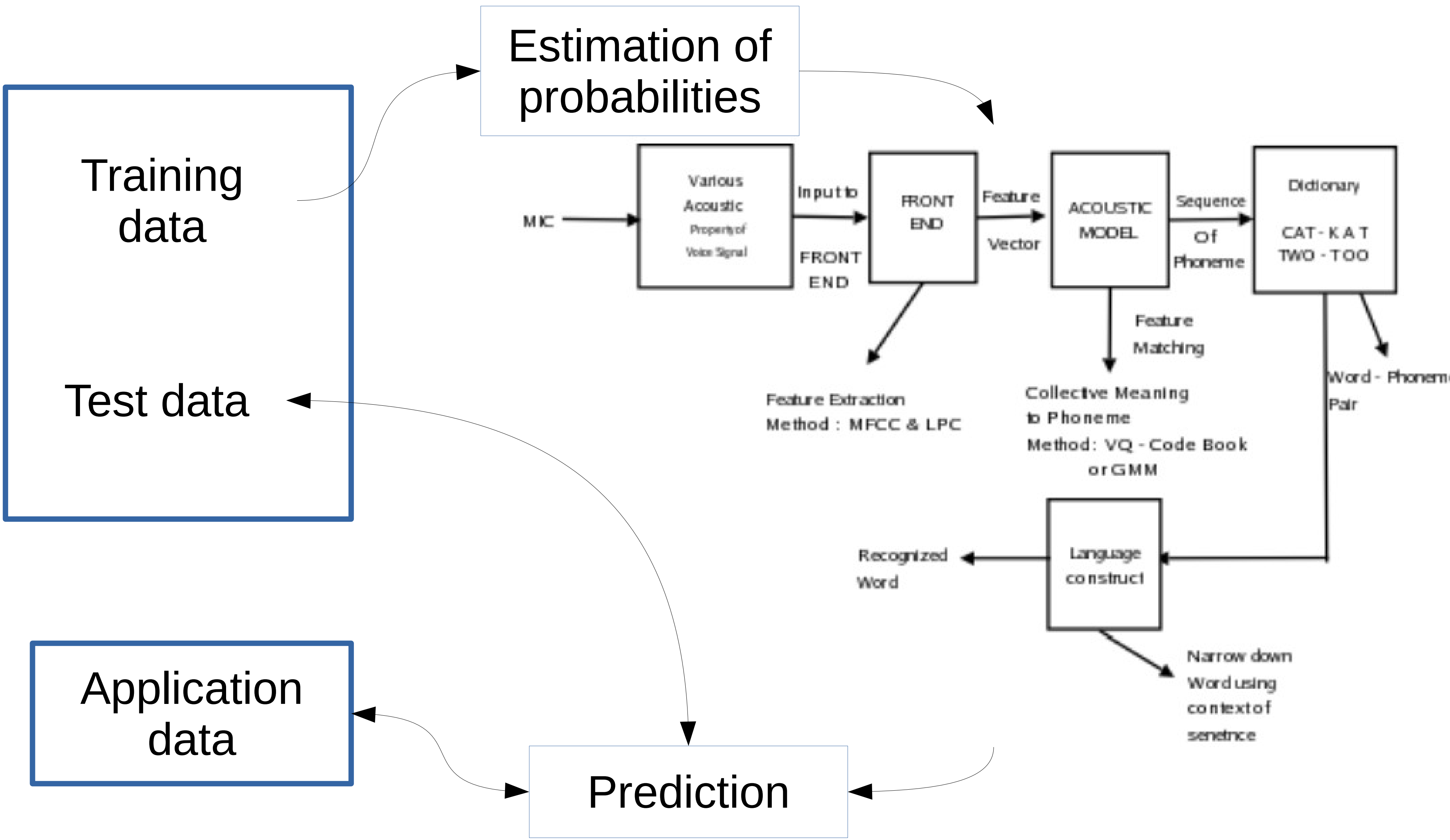
FSTs in Automatic Speech Recognition: HMMs



FSTs in Automatic Speech Recognition: HMMs



FSTs in Automatic Speech Recognition: HMMs



To be continued ...