Lecture 2a: Phonology – Word Stress

1. Inputs and outputs
2. Cross-linguistic preferences
3. OT and stress
4. The autonomy thesis
5. Autonomy breaking – the interaction of stress and syllabification

1 Inputs and outputs

The representational basis is *metrical phonology* (e.g. Liberman & Prince 1977; Halle & Vergnoud 1987; Hayes 1980, 1995). The central assumption is that stress is a rhythmic phenomenon, encoded by strong-weak relations between syllables.

In short, every prosodic word consists of patterns of alternation (termed *a foot*). Each foot contains one stressed and at most one unstressed syllable. The most common two patterns are

- *trochees* (stressed syllable on the left and at most one stressless syllable on the right), as in English, and
- *iambs* (a stressless-stressed sequence of syllables).

We use brackets to mark feet and áccents to mark prominent vowels. There are unfooted syllables (always stressless).

- **Inputs** are taken as syllabified strings of segments (motivated by morphology).
  Examples: /mi.nə.so.tə/ ; /ə.me.ri.kə/.

- **Outputs** are taken as strings which are analysed at foot-level too. We use brackets to mark feet and áccents to mark prominent vowels.
  Example: (mí.nə)(só.tə) ; ə(mé.ri)kə.

- **The Generator** can be assumed to be relatively free. Each possible input is paired with each possible output supposed the corresponding sequences of the terminal elements agree. The following are outputs generated by the input

  /ə.me.ri.kə/:

  (1) ə(mé.ri)kə
  (2) (ə.mé)(rí.kə)
  (3) ə.me(rí.kə)
  (4) (´ə.me)ri.kə
  (5) (´ə.me)ri(k´ə), ...

And this are several outputs generated by the input

/mi.nə.so.tə/:

(1) (mí.nə)(só.tə)
(2) mi.nə(só.tə)
(3) (mí.nə)so.tə
(4) mi(nə.só)tə
(5) (mi.n´ə)(só.tə)
(6) mi.nə.so.tə, ...

Notice that we drop dots if no misunderstandings are possible. For example, we write X(Y)Z instead of .X.(Y).Z.

2 Cross-linguistic preferences

The four best known common properties of stress languages:

- **The culminative property:** *Words have single prosodic peak.* Many languages impose this requirement on content words only, function words are prosodically dependent on content words.
- **The demarcative property:** *Stress tends to be placed near edges words.* Crosslinguistically favoured positions for primary word stress are (a) the initial syllable, (b) the prefinal syllable and (c) the final syllable (ranked in decreasing order of popularity among the world's languages.

- **The rhythmic property:** *Stress tends to be organized in rhythmic patterns, with strong and weak syllables spaced apart at regular intervals.* The smallest units of linguistic rhythm are metrical feet. *Trochees* are preferred. Languages may also select *iambs* .
- **Quantity-sensitivity:** *Stress prefers to fall on elements which have some intrinsic prominence*. For example, stress tends to be attracted by long vowels rather than by short ones. And stressed vowels tend to lengthen, increasing syllable weight. Mutually reinforcing relations of prominence and quantity are highly typical for stress systems.

3 OT and stress

We present a roughly simplified analysis (based on Hammond 1997) and start with the following three constraints:

Constraint corresponding to the culminative property

Words must be stressed — ROOTING

(another name for this constraint is LXWD≈PRWD: grammatical words must have prosody)

Constraints corresponding to the rhythmic property

Feet are trochaic — TROCHEE

Two unfooted syllables cannot be adjacent — PARSE-SYLLABLE

Ranking for English

ROOT ≫ TROCH ≫ PARSE SYLL

Example

Input: /ə.me.ri.kə/		ROOT	TROCH	PARSE SYLL
1	☞ ə(mé.ri)kə			
2	(ə.mé)(rí.kə)		*	
3	ə.me(rí.kə)			*
4	(ˊə.me)ri.kə			*
5	☞ (ˊə.me)ri(kˊə)			
6	☞ (ˊə.me)(rí.kə)			
7	ə.me.ri.kə	*		

The system predicts multiple (optimal) outputs. For unique solutions we have to add some more constraints. Essentially, we have to consider constraints due to the demarcative property and the property of quantity-sensitivity.

Constraint corresponding to quantity-sensitivity

Heavy syllables are stressed WEIGHT-TO-STRESS PRINCIPLE (WSP)

Ranking for English

ROOT, WSP ≫ TROCH ≫ PARSE SYLL

Example (continued)

Input: /ə.me.ri.kə/	ROOT	WSP	TROCH	PARSE SYLL
1 ☞ ə(mé.ri)kə				
2 (ə.mé)(rí.kə)			*	
3 ə.me(rí.kə)		*		*
4 (´ə.me)ri.kə		*		*
5 (´ə.me)ri(k´ə)		*		
6 (´ə.me)(rí.kə)		*		
7 ə.me.ri.kə	*	*		

4 The autonomy thesis

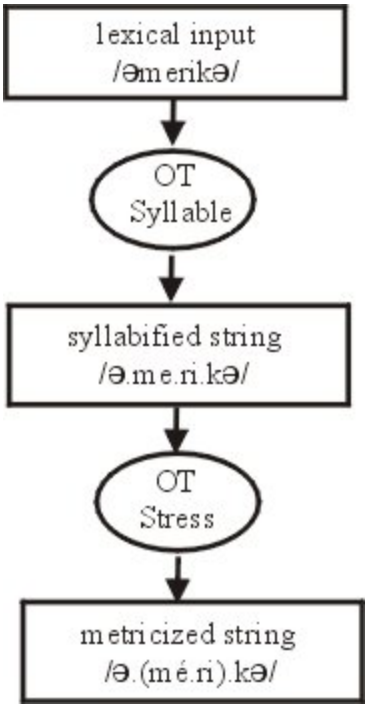
Syllabification parses a strings of segments into a sequence of sub-strings (called syllables). Metrical phonology adds another level of analysis and parses syllables into foots and assigns stress. In the previous section it was assumed that syllabification is independent on stress patterns. In terms of a classical cognitive architecture that means that the outputs of the system of syllabification are the inputs of the stress system. Taken this architecture, the stress system cannot have an effect on syllabification. We may refer to this hypothesis by saying

Syllabification is autonomous with regard to stress

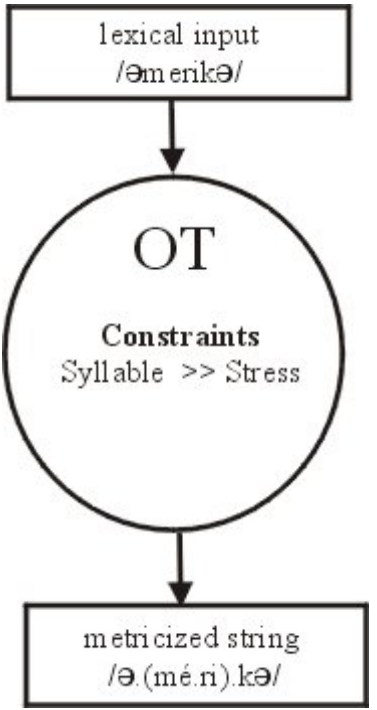
Using OT there are two ways to formulate an autonomy thesis.

The first way, called *representational autonomy*, is a direct transportation from classical architecture into OT. The other way uses hierarchical ranking to express autonomy. It is called *dominance-based autonomy*.

Although both ways are equivalent, they are very different from a conceptual point of view and may be sources of quite different inspirations. This becomes important when violations of autonomy are envisaged.



a. representational autonomy



b. dominance-based autonomy

Representational Autonomy:

- very close to classical (rule-based) architecture.
- separation of representational units and constraints
- the outputs for one system are the inputs for the other one

Dominance-based Autonomy:

- two levels of representation only (input, output)
- no separation of representational elements necessary
- strict separation of the constraints
- the constraints of the autonomous system outrank the constraints of the dependent system

5 Autonomy breaking – the interaction of stress and syllabification

There is ample evidence that syllabification is influenced by stress, contrary to the autonomy thesis.

As a case in point consider pronunciation of /h/. This phoneme is pronounced at the beginning of words (+syllables) but not at the ends. Consequently, we can use the pronunciation of /h/ as a check for syllabification.

Now consider the pair *véhicle – vehícular*. In the first case /h/ isn't pronounced, in the second case it is. Consequently, our test suggests the syllabification in (i) which contrasts with standard theory (ii):

(i) /véh.i.cle/ - /ve.hí.cu.lar/ (empirically)

(ii) /ve.hi.cle/ - /ve.hi.cu.lar/ (standard theory)

Conclusion: Stress influences syllabification. Intervocalic consonants are affiliated with the syllable to its left if the following vowel is stressless.

Another example is aspiration. For example, we have the generalization that /t/ is aspirated syllable-initially but not at the ends. Consider for

example the word *hotél* where the /t/ is aspirated contrasting with the word *vánity* where /t/ isn't aspirated.

(i) /ho.tél/ - /vá.nit.y/ (empirically)

(ii) /ho.tel/ - /va.ni.ty/ (standard theory)

The observed facts seem to obey the following constraint:

Constraint corresponding to a kind of demarcative property

Stressless medial syllables are onsetless NOONSET

Obviously, this constraint is part of the stress family and conflicts with the constraint ONSET of syllable theory. The constraint NOONSET must outrank ONSET to be effective:

NOONSET ≫ ONSET

Autonomy breaking occurs since autonomy of syllable theory would demand that all constraints of syllable theory outrank those of the stress theory.

Interaction of stress and syllabification

Input: /vehikl/	NoOnset	Onset	NoCoda
(vé.hikl)	*		*
☞ (véh.ikl)		*	**

Input: /vehiculə/	NoOnset	Onset	NoCoda
☞ ve(hí.cu)lə			
veh(í.cu)lə		*	*

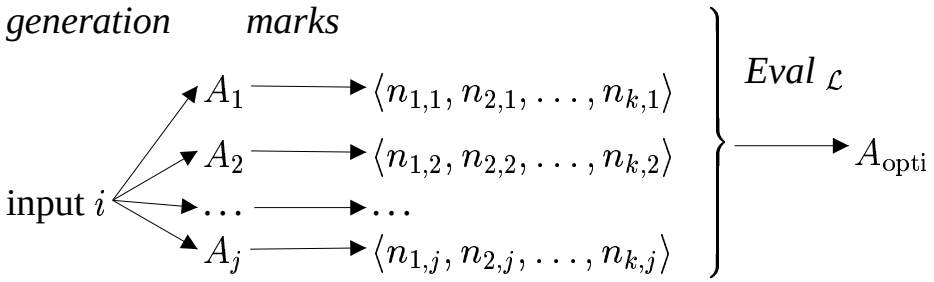
Sources: Frank and Satta 1998, Karttunen 1998

Lecture 2b: Computational aspects of OT

1 Computational issues

Naive evaluation algorithm for an OT system

(1) 1. construct candidates
    2. apply constraints
    3. pick most harmonic candidate(s)



Computational issues (1):

- **Infinity of candidate set**
  - Since candidates can violate faithfulness constraints, the candidate set is generally infinite

(2) Input: laugh(x), x = Ann, tense = past

a.	Ann laughed	f.	Ann
b.	Ann did laugh	g.	
c.	Ann did did laugh	h.	she laughed
d.	it laughed Ann	i.	Ann yawned
e.	laughed	j.	…

  - even with large finite candidate sets, processing will get extremely costly

Computational issues (2):

- **Directionality issues**
  Definition of expressive optimization is based on generation from underlying input forms – how can one decide that a given surface form is an optimal output?
  - requires processing in opposite direction to determine possible inputs (cf. robust interpretive parsing)
  - this may cause additional infinity issues (even in unidirectional optimization)

(3)

A:	Bill saw Sue.	B: And Ann.
A′:	John claimed that Bill saw Sue.	B: And Ann.
A″:	Mary reported that John claimed that Bill saw Sue.	B: And Ann.

(4) *Expressive optimization for (3-B) in reply to (3-A′)*

Input (simplified): claim(j,see(b, a))	*STRUCT	FAITH
a. and John claimed that Bill saw Ann	*****	
b. and that Bill saw Ann	***	**
c. ☞ and Ann	*	****

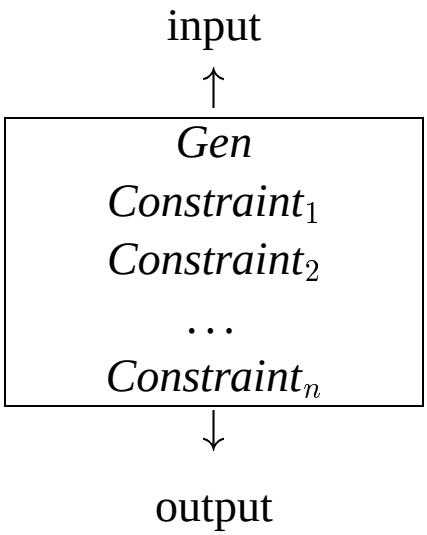
- Is *And Ann* grammatical?
  What are possible underlying forms for *And Ann*?

(5)
a. see(b, a)
b. claim(j,see(b, a))
c. report(m,claim(j,see(b, a)))
d. …

Ways of addressing the infinity issue:

- Control candidate construction based on constraint violations
  dynamic (or chart-based) programming
  (Tesar 1995, Kuhn 2000a,b, 2001)

- Pre-compute the set of distinctions between relevant candidates and the respective winner ($\Rightarrow$ no online construction of competing candidates)
  (Karttunen 1998, based on results by Frank and Satta 1998)

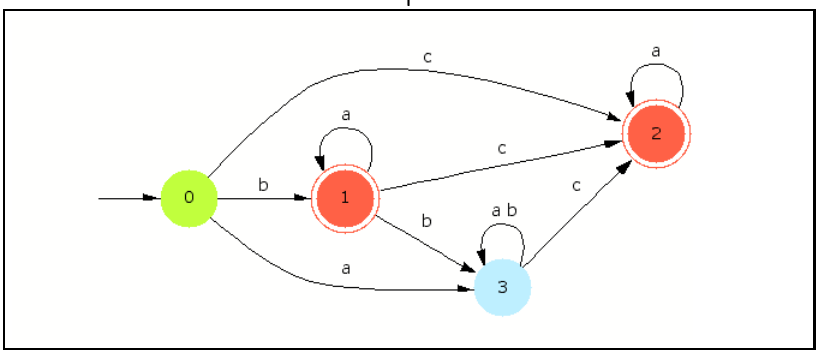


Second option will be discussed in detail

2 Some background on formal languages

Formal language = set of strings over an alphabet (Σ)

Different ways of characterizing a formal language:

<table>
<tr><td>list of strings
(for finite languages)</td><td colspan="4">{ ϵ, a, b, aa, ab, bb, ba }
(note: ϵ is the empty string)</td></tr>
<tr><td>algebraic expression</td><td colspan="4">$a^n b^n, n \geq 1$</td></tr>
<tr><td>formal grammar</td><td>non-terminals
S, A, B</td><td>terminals
a, b</td><td>productions
S $\rightarrow$ A S B
S $\rightarrow \epsilon$
A $\rightarrow$ a
B $\rightarrow$ b</td><td>start symbol
S</td></tr>
<tr><td>abstract automaton</td><td colspan="4">.....
 
</td></tr>
</table>

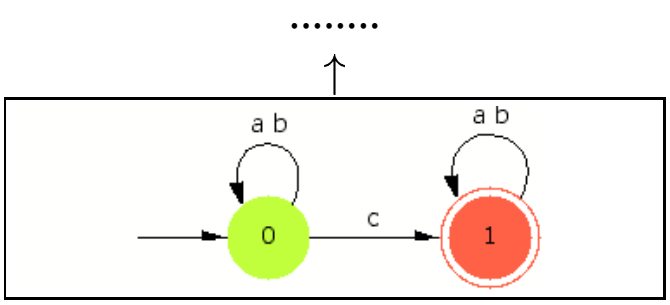
Classes of formal languages (Chomsky hierarchy):

- regular languages
- context-free languages
- context-sensitive languages
- recursively enumerable languages

(Classification based on restrictions on formal grammars; equivalent classes follow from specific types of automata)

Regular languages

Characterized (equivalently) by

regular expressions	expressions over alphabet formed by concatenation, union and Kleene closure {a,b}*c{a,b}*
regular grammars	all productions have the form A $\rightarrow$ wB or A $\rightarrow$ w, where A, B: nonterminals, w: terminal string S $\rightarrow$ A A $\rightarrow$ aA A $\rightarrow$ bA A $\rightarrow$ cC C $\rightarrow$ aC C $\rightarrow$ bC C $\rightarrow$ ϵ
finite-state automata	machine with a finite set of states and state transitions triggered by input symbols (from the alphabet) or ϵ 

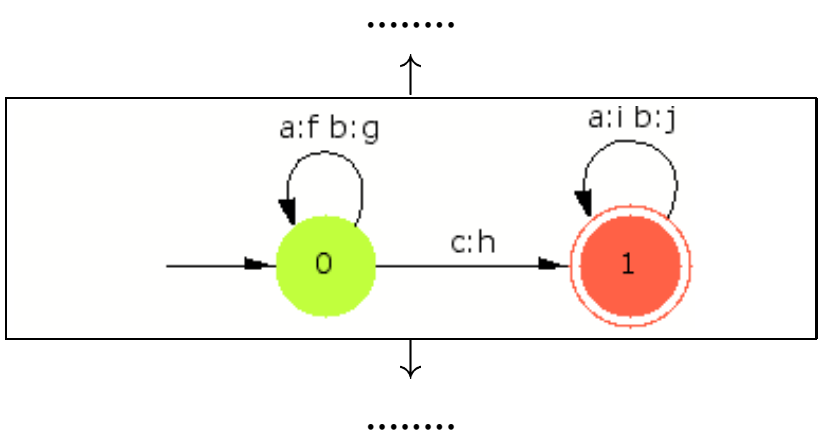
Important properties of regular languages:

- closed under union, intersection, complementation
- recognizers are very efficient: linear time complexity (i.e., computation with double input length will only take twice as long)

Note: as only a finite number of states can be distinguished, a language like $a^n b^n$, $n \geq 1$ is not regular

Finite-state transducers (FST's)/Rational relations

- a finite-state automaton with two tapes is called a finite-state transducer
- specifies a relation between two regular languages (so-called rational relation)
  - state transitions are marked with two symbols `a:b`
  - extension of regular expression notation is used to specify transducers
  - one can view one side of the transducer as the input, which is transformed into the form(s) on the other side
  - nondeterminism may lead to several possibilities in the mapping
  - the upper and lower side can be swapped



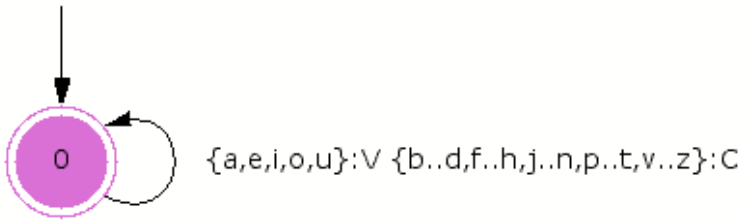
- FST's are widely used for phonological, morphological and "shallow" syntactic processing

Examples of FST's

Specification:

```
{{a,e,i,o,u}:V,
{b,c,d,f,g,h,j,k,l,m,n,p,q,r,s,t,v,w,x,y,z}:C}*
```

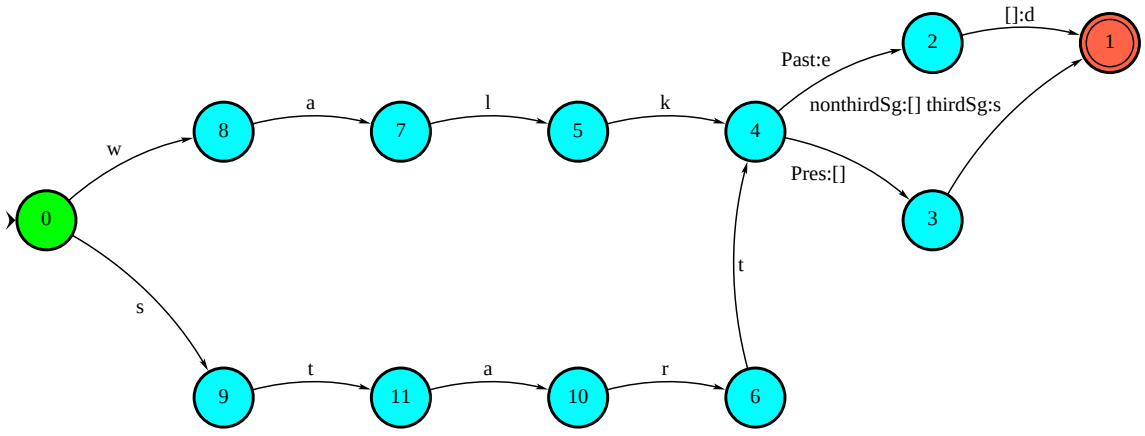
Automaton:



Application samples:

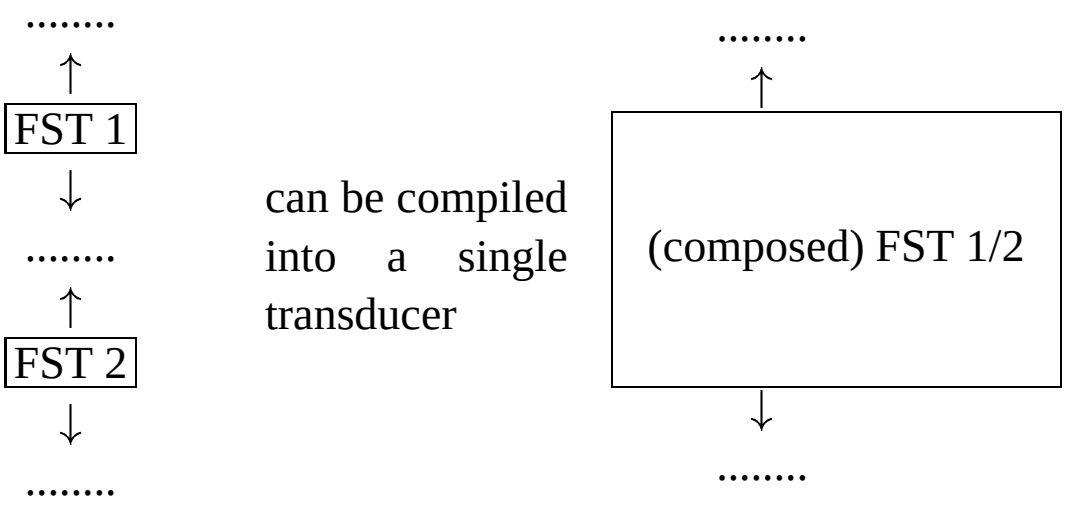
table ↔ CVCCV
car ↔ CVC

```
[{ [w,a,l,k] x [w,a,l,k],
   [s,t,a,r,t] x [s,t,a,r,t] },
 { Past x [e,d],
   [ Pres x [],
   { thirdSg x [s],
     nonthirdSg x [] } ] }      ]
```

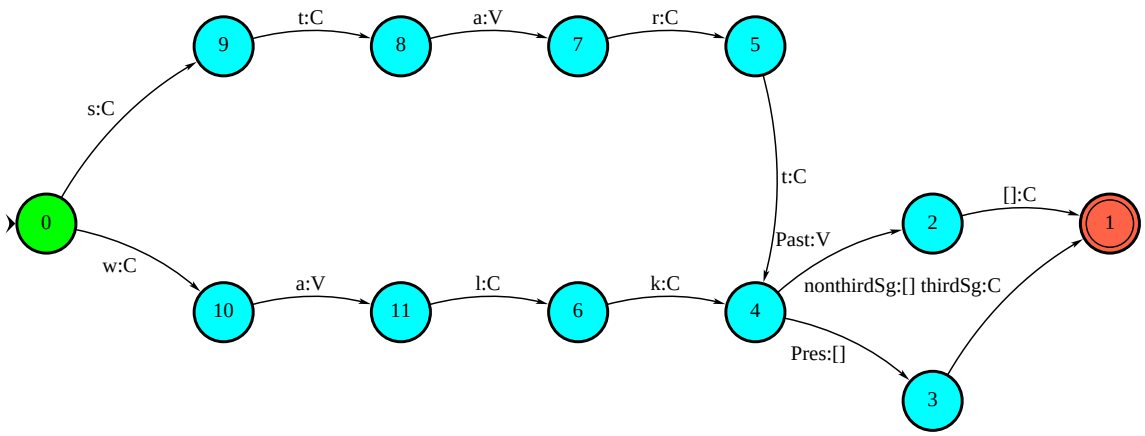


walk Past ↔ walked
start Pres thirdSg ↔ starts
start Pres nonthirdSg ↔ start

Composition of FST's (Notation: `FST1 .o. FST2`)



FST 1/2 maps u to w iff there is some v s.th. FST 1 maps u to v and FST 2 maps v to w.



walk Past	$\leftrightarrow$	CVCCVC
start Pres thirdSg	$\leftrightarrow$	CCVCCC
start Pres nonthirdSg	$\leftrightarrow$	CCVCC

Web demos of finite state transducers

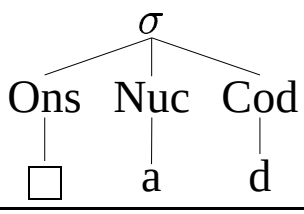
Gertjan van Noord's FSA Utilities (based on Prolog – freely available):
`http://odur.let.rug.nl/~vannoord/fsademo/`

Xerox Finite-State Compiler:
`http://www.rxrc.xerox.com/research/mltt/fst/fsinput.html`

3 Computational OT based on FST's

(Frank and Satta 1998, Karttunen 1998)

Candidate generation: *Gen* – *Syllabification example*

(input /ast/)	Notation (Karttunen 1998):
	`O[ ] N[ a ] D[ d ] X[ b ]`

- *Gen* can be defined as a transducer:
  - upper side: OT input (underlying form)
  - lower side: all possible syllabifications (including faithfulness violations)

Simplified part of the specification expression:

```
[{[ []:'O[', {b:b,c:c,d:d ... },    []:']' ],
    []},
  [ []:'N[', {a:a,e:e,i:i,o:o,u:u}, []:']' ]
]*
```

ba	↔	a.	N[]	D[b]		N[a]
		b.	N[]	O[b]		N[a]
		c.	N[]	X[b]		N[a]
		d.		O[b]	N[]	N[a]
		e.		O[b]		N[a]
		f.		O[b]		N[a] N[]
		g.		O[b]		N[a] D[]
		h.	O[]	X[b]		N[a]
		i.		X[b]	N[]	N[a]
		j.		X[b]		N[a]
		k.		X[b]		N[a] N[]
		l.		X[b]		N[a] D[]
		m.		X[b]	O[]	N[a]
		n.	...			

Formalizing the constraints

Each constraint is expressed as a regular language

NoCoda	the language that does not contain 'D[' `~[ ?* , 'D[', ?* ]`
Onset	the language in which 'N[' is always preceded by 'O[' ...']' `[ [?-'N[']* {['O[',{?,[]},']' 'N['],[]} [?-'N[']*]*`

Faithfulness:

Max-IO	(No deletion.) the language that does not contain 'X[' `~[ ?* , 'X[', ?* ]`
Dep-IO	(No epenthesis.) the language in which 'O[', 'N[' and 'D[' never have ']' immediately following `~[ ?* , [{'O[','N[','D['},']'] , ?* ]`

- Each simple finite-state automaton can be interpreted as a transducer (with upper and lower side identical)
- What happens if we compose *Gen* and a constraint?

GEN .o. NoCoda

ba ↔	b.	N[]	O[b]		N[a]
	c.	N[]	X[b]		N[a]
	d.		O[b]	N[]	N[a]
	e.		O[b]		N[a]
	f.		O[b]		N[a] N[]
	h.	O[]	X[b]		N[a]
	i.		X[b]	N[]	N[a]
	j.		X[b]		N[a]
	k.		X[b]		N[a] N[]
	m.		X[b]	O[]	N[a]

GEN .o. DepIO

ba ↔	e.	O[b]	N[a]
	j.	X[b]	N[a]

GEN .o. Onset:

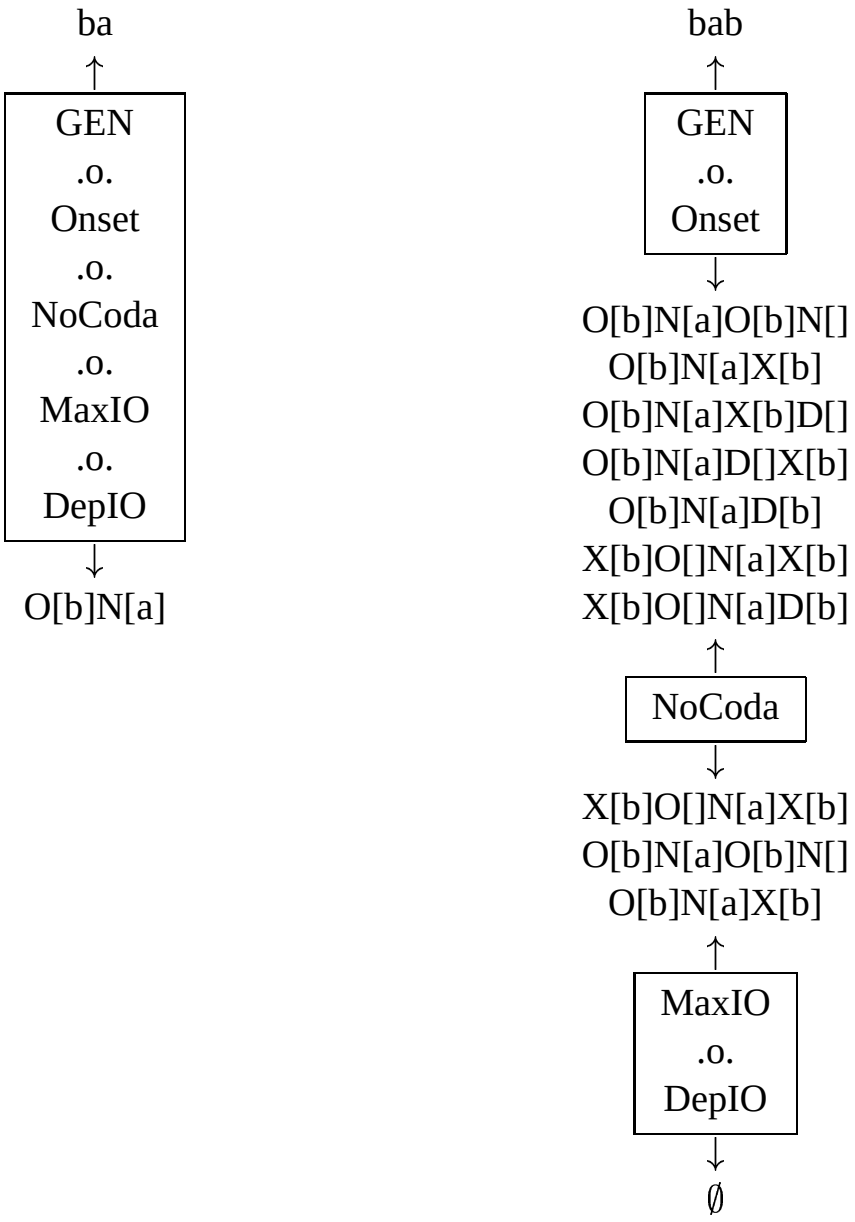
ba ↔	e.	O[b]		N[a]
	g.	O[b]		N[a] D[]
	m.	X[b]	O[]	N[a]

GEN .o. MaxIO:

ba ↔	a.	N[]	D[b]		N[a]
	b.	N[]	O[b]		N[a]
	d.		O[b]	N[]	N[a]
	e.		O[b]		N[a]
	f.		O[b]		N[a] N[]
	g.		O[b]		N[a] D[]

Composing *Gen* and a constraint has the effect that all candidates violating the constraint are filtered out

- Could we compose a cascade of all the constraints to implement an OT system?
  (assumed ranking: ONSET ≫ NOCODA ≫ MAXIO ≫ DEPIO)



- Problem: Does not account for violability of constraints
  - Only perfect candidates will go through
  - Constraints should only be applied when at least one candidate satisfies them!

“Lenient” composition

- it is possible to define a special composition operation within the FST formalism that
  - applies a particular transducer as a filter if the resulting language is non-empty, but
  - else ignores the transducer

Definition of “lenient” composition (Karttunen 1998):

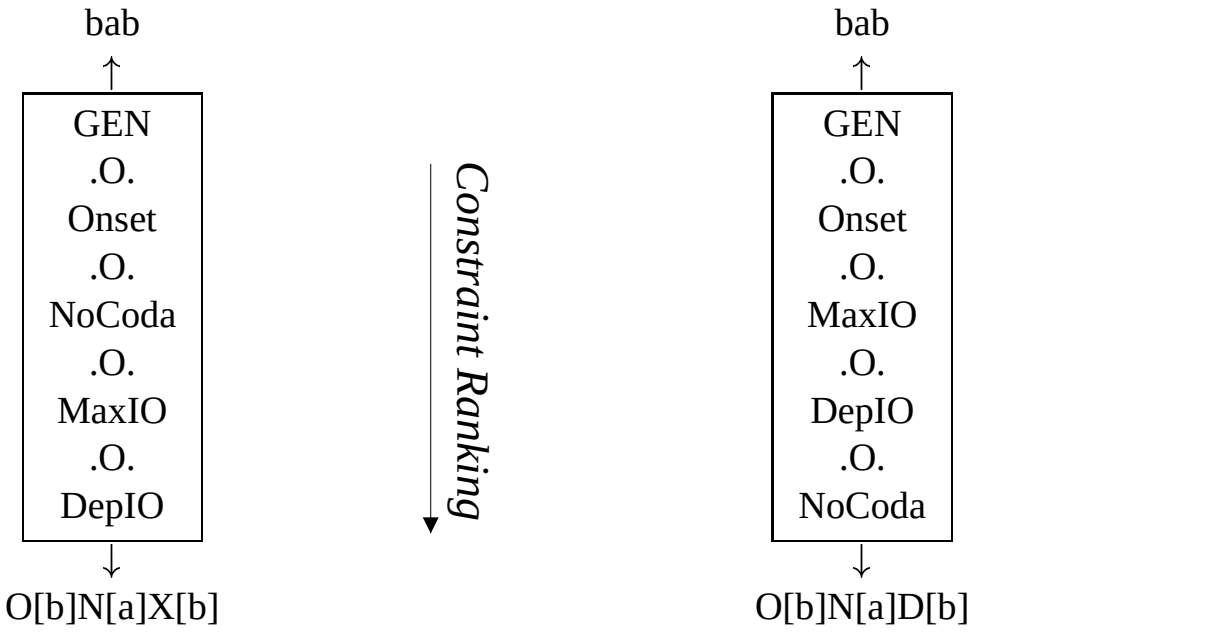
```
R .O. C = [R .o. C] .P. R
```

based on *priority union*: A .P. B (if a string is compatible with the upper side of A, then A is applied, else B is applied)

Advantages

- the entire OT system is precompiled into a single transducer: a *lenient cascade*
- no runtime computation of candidates – very efficient
- compact FST: 66 (or 248) different states (Karttunen 1998 – slightly different system)

Examples of lenient cascades



4 (Bi-)Directionality

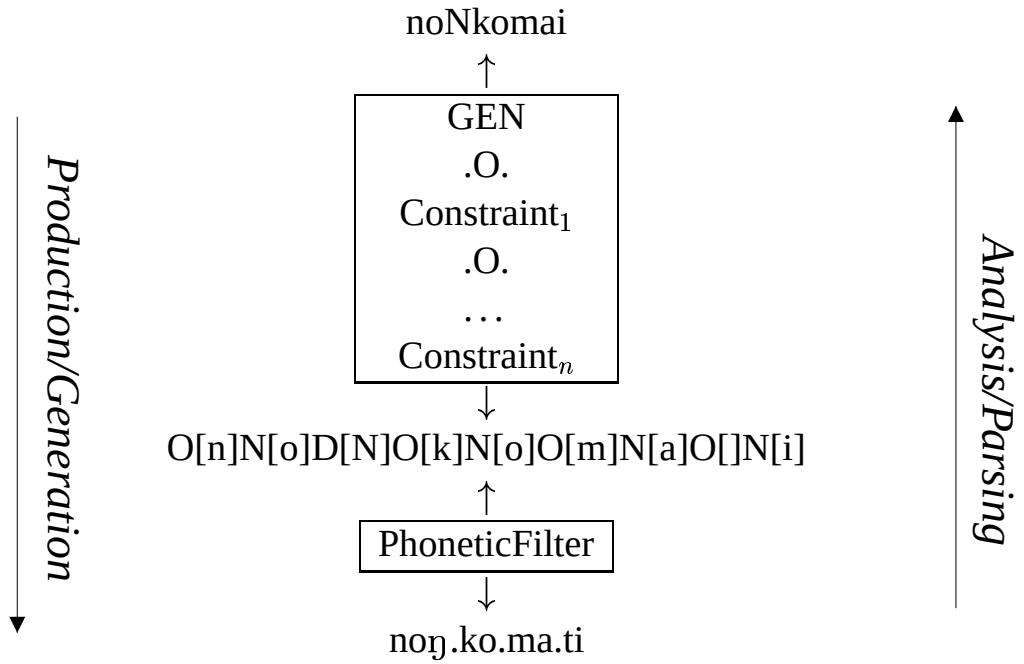
Directionality issue (review): even unidirectional OT models require a kind of "backward" processing

- no problem with FST formalization
- recall that it is possible to swap the upper and lower side of an FST

$\Rightarrow$ lenient cascade can be run from the output side

  - technically, this requires a simple additional transducer at the bottom which models phonetic realization, i.e., which
    - removes all X[...] material
    - maps O[] etc. to some particular phonetic realization
    - filters out all the remaining opening and closing brackets (possibly introducing some marking of syllable boundaries)
  - this transducer can be added with classical (non-OT) composition

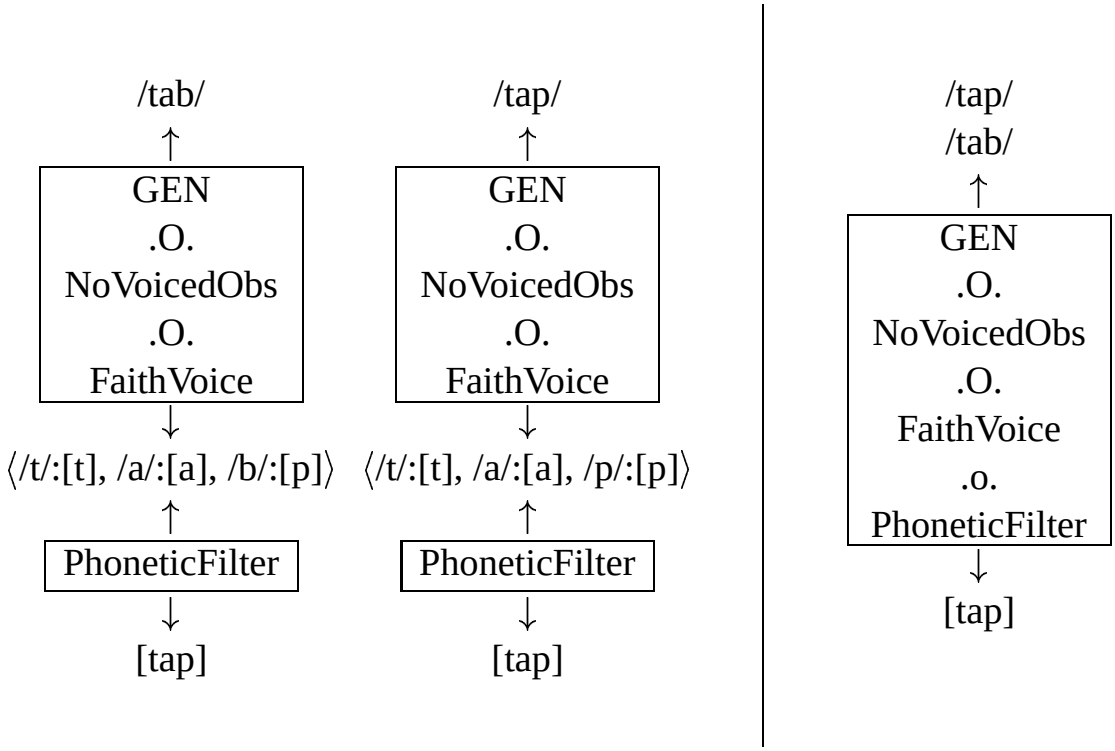
Processing with unidirectional OT model: expressive optimization



(Expressive optimization continued)

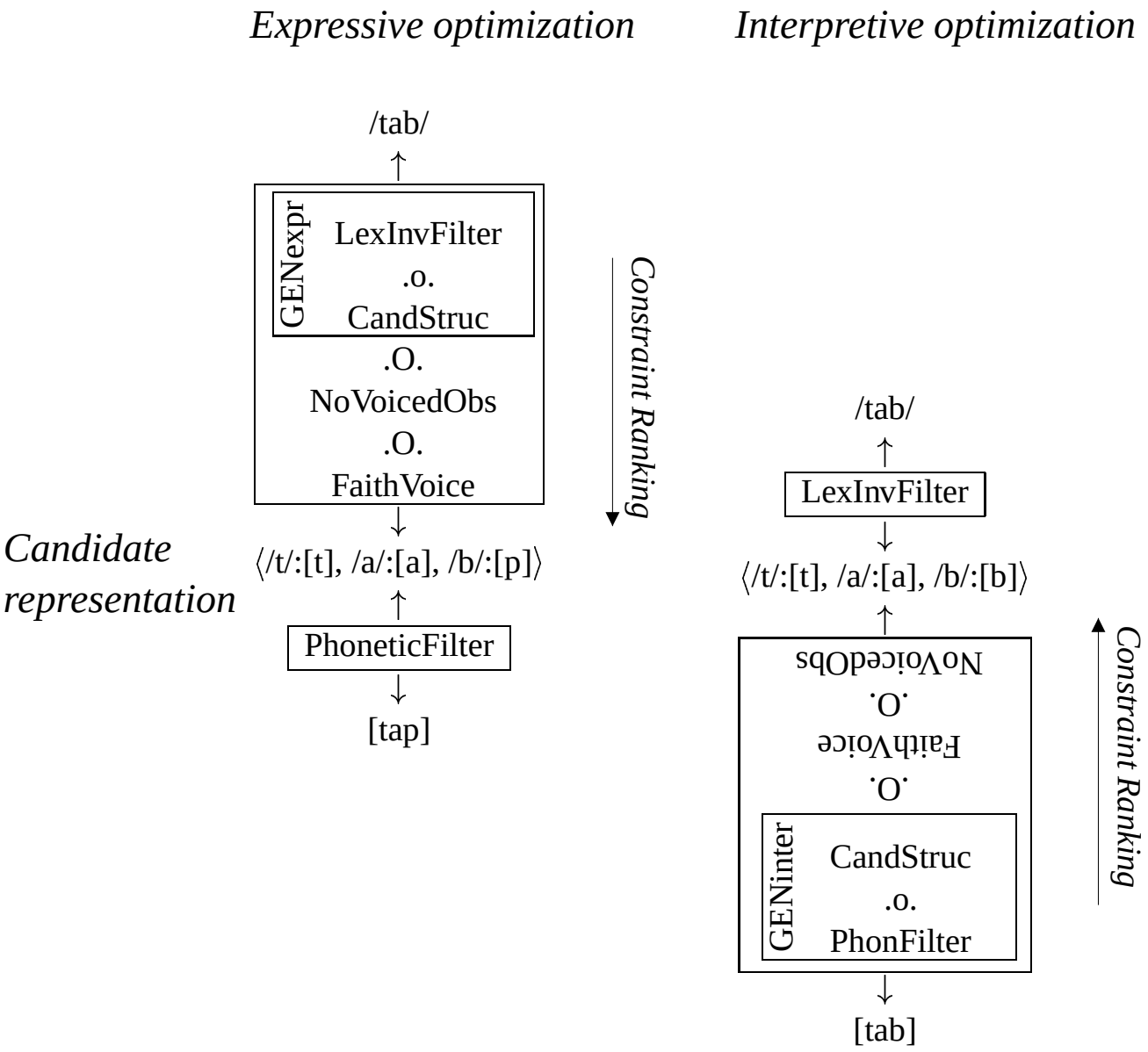
- "backward application" of expressive OT cascade may reveal an ambiguity

  Voiced obstruent example
  (assumed constraint ranking: OBS/*VOICE ≫ FAITH[VOICE])



Expressive vs. interpretive optimization

- interpretive OT (as applied in lexicon optimization) can be modelled by an "inverted" lenient cascade with GEN at the bottom

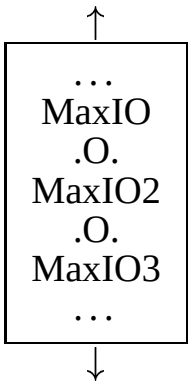


Bidirectional optimization

- (Strong) bidirectional optimization can be implemented based on the individual unidirectional cascades:
  - the regular languages representing the candidates after the application of the lenient cascades can be *intersected*
  - thus, strong bidirectional optimization can be expressed as a single FST (see Jäger 2000 on weak bidirectionality)

5 Limitations of FST-based OT

- Multiple violations of constraints are only possible up to a fixed upper limit (main result of Frank and Satta 1998)
  - for each count of a multiple violation, a special regular language needs to be defined, which is used in the lenient cascade
    MaxIO: `~[ ?*, 'X[', ?* ]`
    MaxIO2: `~[ ?*, 'X[', ?*, 'X[', ?* ]`
    MaxIO3: `~[ ?*, 'X[', ?*, 'X[', ?*, 'X[', ?* ]`



  - this is incompatible with a central theoretical assumption of OT
  - in practical phonological applications, the restriction is presumably irrelevant (see also Gerdemann and van Noord 2000 for a finite-state approximation covering more cases)
  - generally, focusing on local domains for optimization may reduce the problem (then multiple violations would not have to be counted globally)
- All components (*Gen* and constraints) must be modelled as rational relations (or regular languages)
  - reduplication requires expressive power beyond regular languages
    (e.g., Indonesian: wanita *woman* – wanita-wanita *women*)
  - FST-based OT is not applicable for full syntactic systems (note however: Wartena 2000 sketches an extension of the FST-based approach to regular tree languages)

References

Frank, Robert, and Giorgio Satta. 1998. Optimality theory and the generative complexity of constraint violation. *Computational Linguistics* 24:307–316.

Gerdemann, Dale, and Gertjan van Noord. 2000. Approximation and exactness in finite state Optimality Theory. In *SIGPHON 2000, Finite State Phonology. Proceedings of the Fifth Workshop of the ACL Special Interest Group in Computational Phonology,* Luxembourg.

Jäger, Gerhard. 2000. Some notes on the formal properties of bidirectional Optimality Theory. Ms., ZAS Berlin, Rutgers Optimality Archive, `http://ruccs.rutgers.edu/roa.html` (ROA-414-09100).

Karttunen, Lauri. 1998. The proper treatment of optimality in computational phonology. In *Proceedings of the International Workshop on Finite-State Methods in Natural Language Processing, FSMNLP'98*, pp. 1–12. To appear, ROA-258-0498.

Kuhn, Jonas. 2000a. Faithfulness violations and bidirectional optimization. In M. Butt and T. H. King (eds.), *Proceedings of the LFG 2000 Conference, Berkeley, CA*, CSLI Proceedings Online, pp. 161–181.

Kuhn, Jonas. 2000b. Processing Optimality-theoretic syntax by interleaved chart parsing and generation. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics (ACL-2000)*, pp. 360–367, Hongkong.

Kuhn, Jonas. 2001. *Formal and Computational Aspects of Optimality-theoretic Syntax*. PhD thesis, Institut für maschinelle Sprachverarbeitung, Univerität Stuttgart.

Tesar, Bruce. 1995. *Computational Optimality Theory*. PhD thesis, University of Colorado.

Wartena, Christian. 2000. A note on the complexity of optimality systems. Ms. Universität Potsdam, Rutgers Optimality Archive, `http://ruccs.rutgers.edu/roa.html` (ROA-385-03100).