LING 306: Introduction to Computational Linguistics

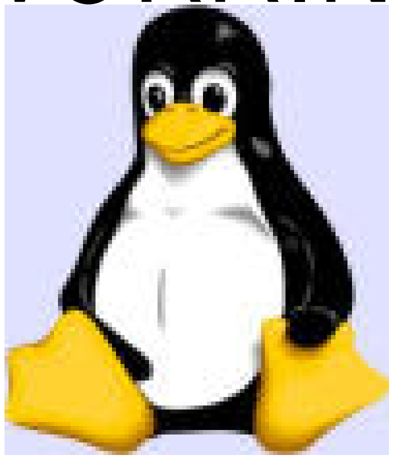
Richard Sproat
rws@uiuc.edu
http://www.staff.uiuc.edu/~rws/

URL for this course:
http://www.staff.uiuc.edu/~rws/Courses/L306

Lecture 6: Phonology and Computational Phonology

September 8, 2003

IF YOU CAN READ THIS THEN ACROBAT IS WORKING TODAY



Introduction

- Phonology is the study of sound alternations in language; computational phonology relates to computational models of those alternations
- Most languages have some amount of phonological alternation (sometimes quite a lot) when words are put together out of morphemes, so doing computational morphology invariably involves doing computational phonology too
- Since most morphological analyzers deal with text, what counts as computational phonology is really “computational orthography”

Orthography Versus Phonology

- Some languages/writing systems exhibit a very close relation between spelling and pronunciation. E.g. Spanish, Serbocroatian, Finnish, Turkish. In these languages, modeling spelling alternations amounts to more or less to the same thing as modeling phonlogical alternations.

  But even here things are not that simple: the c/z alternation we saw last time in the Spanish examples *co*$\underline{c}$*er* 'cook', *cue*$\underline{z}$*o* 'I cook', is purely a spelling rule and has nothing to do with the phonology.

- In other languages (English, French, Gaelic), the spelling is relatively far removed from the pronunciation. In English many alternations one must untie in a morphological analyzer are spelling alternations.

This can both help and hurt:

⋆ Phonological alternations can be obscured by the spelling:

Newt$\underline{\text{o}}$n	Newt$\underline{\text{o}}$nian
Par$\underline{\text{o}}$s	Par$\underline{\text{o}}$sian
m$\underline{\text{a}}$n$\underline{\text{ia}}$c	m$\underline{\text{a}}$n$\underline{\text{ia}}$cal
electri$\underline{\text{c}}$	electri$\underline{\text{c}}$ity

⋆ Or the spelling may introduce alternations that have no counterpart in the phonology:

innovat$\underline{\text{e}}$	innovation
picnic	picnic$\underline{\text{k}}$ing
happ$\underline{\text{y}}$	happ$\underline{\text{i}}$est
goo$\underline{\text{ey}}$	goo$\underline{\text{i}}$est

One instance of a Phonological Alternation: Finnish Harmony

Nominative	Partitive	Gloss
taivas	taivas+ta	'sky'
puhelin	puhelin+ta	'telephone'
lakeus	lakeut+ta	'plain'
syy	syy+tä	'reason'
lyhyt	lyhyt+tä	'short'
ystävällinen	ystävällinen+tä	'friendly'

talossansakaanko	'not in his house either?'
kynässänsäkäänkö	'not in his pen either?'

Note that i,e are said to be *neutral* with respect to harmony.

Some Approaches to Modeling Phonological Alternations

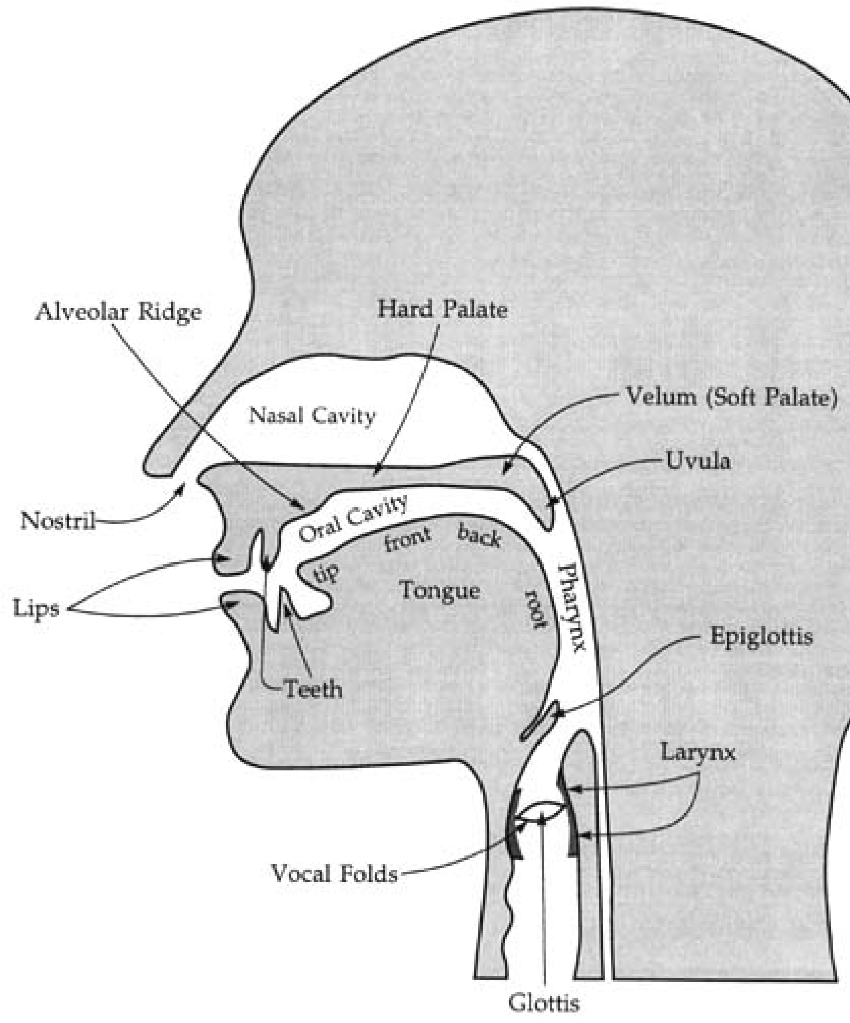
- Rewrite rules (Chomsky and Halle, 1968)
- Autosegmental Phonology.
- Constraint-based Approaches.

Rewrite Rules Applied to the Finnish Case

$$a \rightarrow \text{ä} \ / \ [\text{ä},\text{ö},y]\ C^*\ ([i,e]\ C^*)^*\ \_\_\_\_$$
$$o \rightarrow \text{ö} \ / \ [\text{ä},\text{ö},y]\ C^*\ ([i,e]\ C^*)^*\ \_\_\_\_$$

But we can have a linguistically cleaner description if we make use of features.

Your Standard Midsagittally Bisected Head



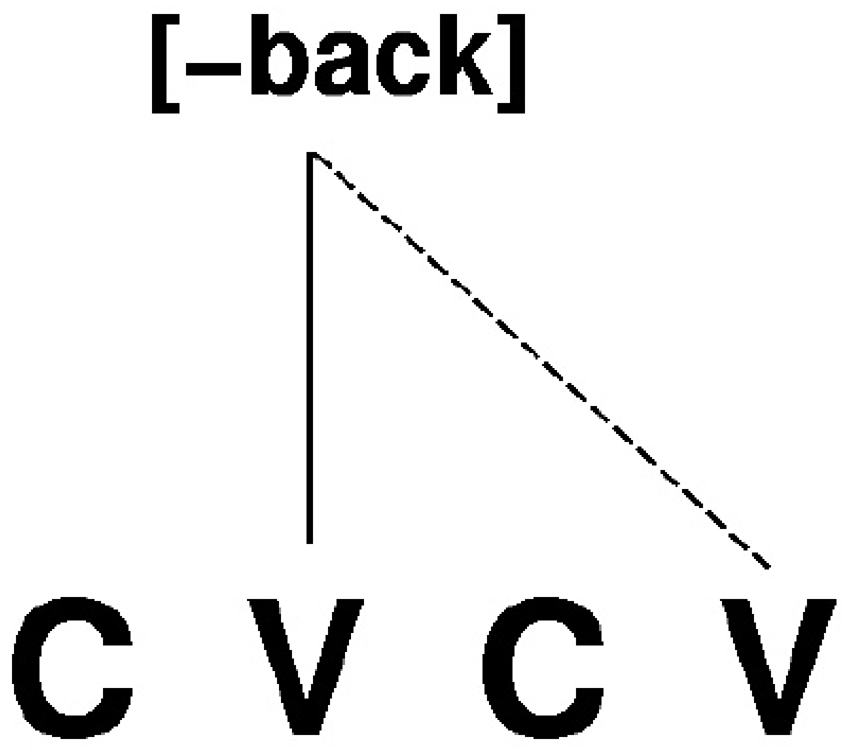
Features of Finnish Vowels

Feature	a	o	u	ä	ö	y	i	e
High	−	−	+	−	−	+	+	−
Low	+	−	−	+	−	−	−	−
Back	+	+	+	−	−	−	−	−
Round	−	+	+	−	+	+	−	−

A Featural Specification

[V, +back] $\rightarrow$ [V, -back] / [V, -back] C* ([i,e] C*)*

Autosegmental Phonology



Constraint-Based Approach

- Instead of having explicit rules to model the alternation, you assume that all variants are generated, and that surface constraints serve as a filter to disallow illegal variants.
  - ⋆ So in the Finnish case, suffixes would be generated in all possible forms: say *-ta*, and *-tä* for the partitive.
  - ⋆ Then the *-ta* case would be filtered out after [-back] vowels, and the *-tä* case would be filtered out after [+back] vowels.

- Optimality Theory—currently a popular approach to phonology (see J&M 113–118)—adopts a model very similar to this:
  - ⋆ GEN generates all possible forms.
  - ⋆ A set of rank-ordered (supposedly universal) violable constraints is used to assign violations to each form.

- ⋆ Consider the set of forms and the worst violation assigned to each of them: the form that is has the *least* ranked of these violations wins.

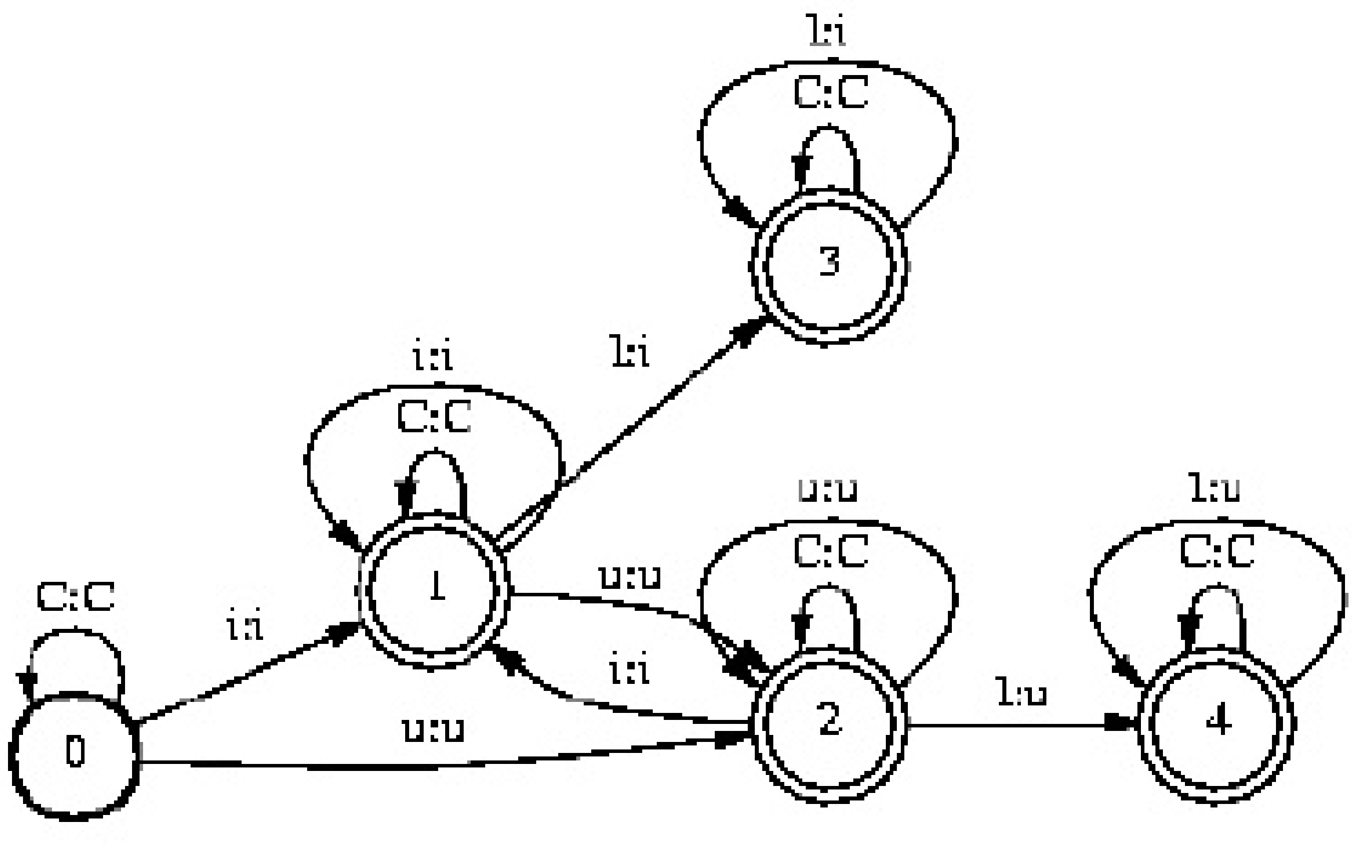
Rule-Based Approach

- Assume baseforms that are constrained as follows:

  ```
  (C*(u|i))+ C* (C*I)* C*
  ```

-
  ```
  I -> i / i C* __
  I -> u / u C* __
  ```

Rule-Based Approach



Constraint-Based Approach

- Assume baseforms that are constrained as follows:

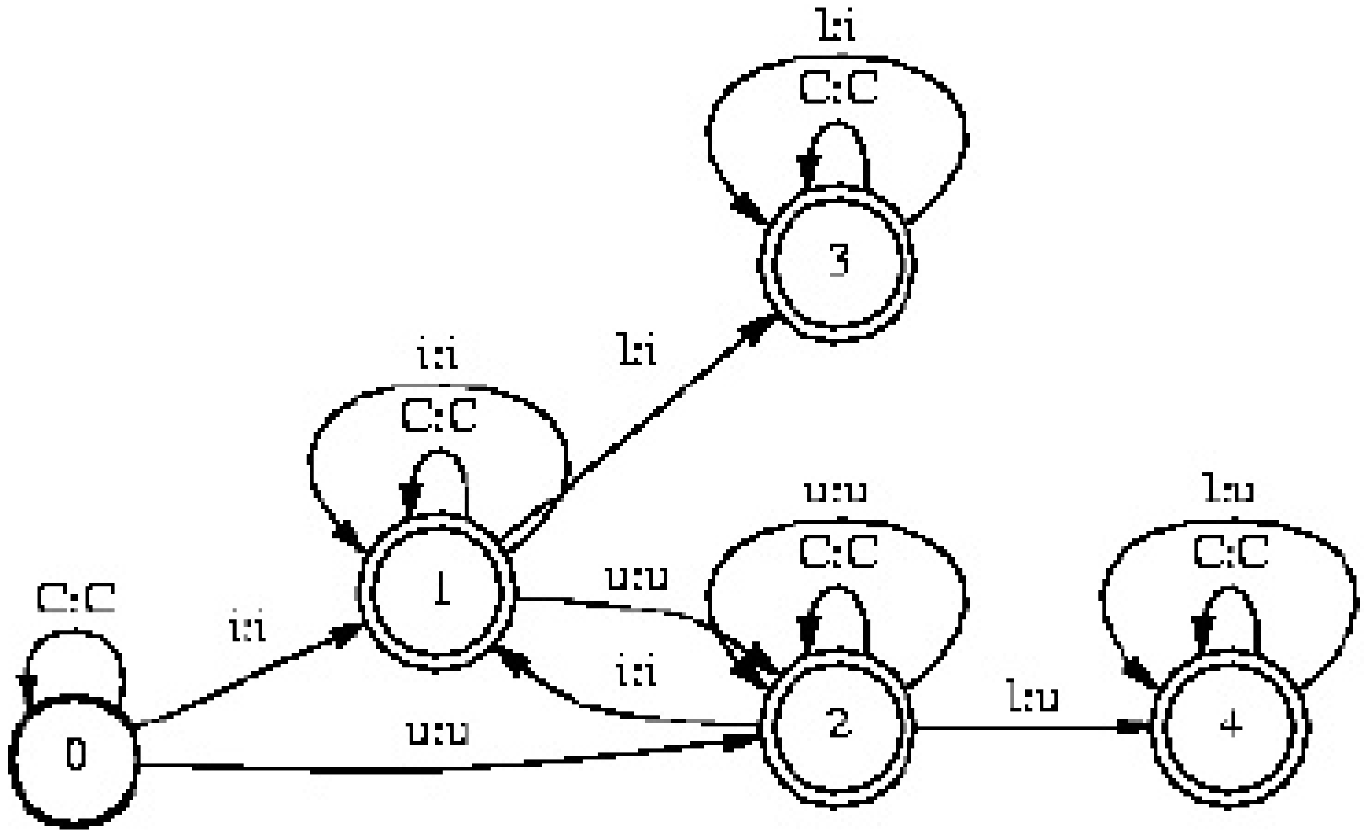
```
(C*(u|i))+ C* (C*I)* C*
```

- Assume that ‘I’ is freely substituted with ‘i’ or ‘u’. But we need to distinguish lexical /i,u/ from harmony /i,u/, so let’s encode the latter as /I,U/.
- Then you can constrain /I,U/:

$$!((\Sigma * (I|i)C * U\Sigma *)|(\Sigma * (U|u)C * I\Sigma *)$$

- Finally, /I,U/ are spelled out as /i,u/

Constraint-Based Approach



Computational Equivalence of Rule-Based and Constraint Based Approaches

- Karttunen's paper "The Proper Treatment of Optimality Theory in Computational Phonology" argues that there is no computational difference between traditional ordered rules and OT.
- Traditional ordered rules can be implemented using composed transducers.
- OT can be implemented using constraints *leniently composed* together.
- It's worth reading this paper.

A Brief History of Computational Phonology

- The theory of phonology, based on rewrite rules, developed in Chomsky and Halle's *Sound Pattern of English* (1968) had a problem: unconstrained rewrite rules were too powerful.

  Johnson (1972) argued: "It is a well-established principle that any mapping whatever that can be computed by a finitely statable, well-defined procedure can be effected by a rewriting system . . . Hence any theory which allows phonological rules to simulate arbitrary rewriting systems is seriously defective, for it asserts next to nothing about the sorts of mappings these rules can perform."

- But in fact the "context sensitive" rewrite rules, as they are invariably used in phonology, were really much weaker, and in fact are equivalent to regular relations.

- The main constraint is that such rules cannot apply arbitrarily to their own output. Otherwise a rule such as

$\epsilon \rightarrow ab/a\_\_\_b$
could produce $a^n b^n$ (and indeed one could do more powerful things than this).

A Brief History of Computational Phonology

- The obvious computational question was: if rewrite rules are implementable as FST's, can one build a compiler that takes a set of these rules and produces an FST.

- This was first attempted by Ron Kaplan and Martin Kay at Xerox PARC starting in the late 1970's.

  They eventually hit upon a solution in the early 1980's, but this was not actually implemented and published until 1994:

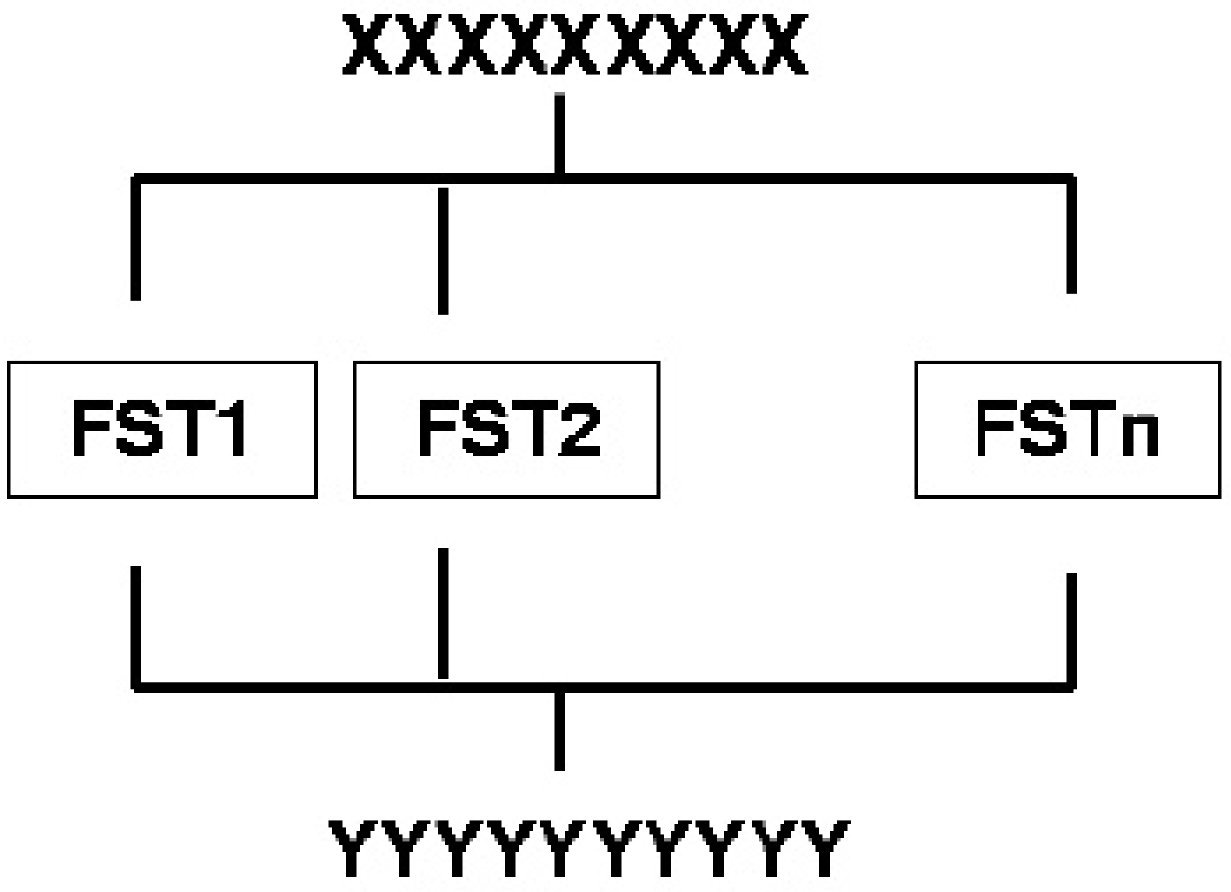
  Ronald M. Kaplan and Martin Kay. Regular model of phonological rule systems. *Computational Linguistics*, 20(3):331-378, September 1994.

- The reason was that in 1980, computers were not powerful enough to compile sets of rules of any interesting complexity.

Koskenniemi's Two-Level Morphology

- Koskenniemi proposed an alternative approach whereby instead of trying to compile rules into transducers and compose them serially, you would instead have a set of very compact transducers that each related the surface and lexical forms.
- The rules would be interpreted in parallel, which is formally equivalent to intersection.
- What did I say about intersection and transducers?

Koskenniemi's Two-Level Morphology



Koskenniemi's Two-Level Rules

Basic formalism: CorrespondencePair **op** LeftContext — RightContext

Exclusion rule	a:b $/\!\!\Leftarrow$ LC — RC
Context restriction rule	a:b $\Rightarrow$ LC — RC
Surface coercion rule	a:b $\Leftarrow$ LC — RC
Composite rule	a:b $\Leftrightarrow$ LC — RC

Interpretation:

- Exclusion rule: *a* cannot be realized as *b* in the stated context.
- Context restriction rule: *a* can only be realized as *b* in the stated context (and nowhere else)
- Surface coercion rule: *a* must be realized as *b* in the stated context.

- Composite rule: *a* is realized as *b* obligatorily and only in the stated context.

But Koskenniemi didn't have a compiler

T:S $\Leftrightarrow$ V:= — I:=

	I	V	T	T	=
State	=	=	S	T	=
1:	2	2	0	1	1
2:	2	2	4	3	1
3:	0	2	0	1	1
4.	2	0	0	0	0

Notes:

1. "=" means any symbol; "=:=" means any symbol corresponding to itself.

2. ":" notates a final state; "." a nonfinal state

3. "0" is a 'sink' state (certain death)

Koskenniemi's Two-Level Rules: Some notes

- Koskenniemi developed a set of transducers by hand for the entire morphology of Finnish.

- Two-level phonology was more than just a computational model. It was a theory of phonology. It essentially claimed that there was never any need to posit intermediate levels between underlying (abstract) forms and surface forms.

Further Notes

- Lots of morphological analyzers have been built using the Koskenniemi approach, and its offshoots such as PCKIMMO.
- But many systems are not pure two-level: Karttunen and his colleagues at Xerox Research Center Europe (Grenoble) have produced many systems that are based on cascaded two level rules.
- You do not *need* to have two-level rules: sometimes they make the description more convenient, but they are never required.

  Systems of two-level rules and systems of cascaded rules are formally equivalent.

Reading & Homework, Etc.

- Readings:
  1. R&M Chapter 4, sections 4.1–4.4
  2. Karttunen paper: http://www.xrce.xerox.com/publis/mltt/pto/bilkent.html
- PCKIMMO is available at http://www.sil.org/pckimmo/v2/pc-kimmo_v2.html if people want to play with it.
- Miniproject with lextools: http://www.staff.uiuc.edu/~rws/Courses/L306/Homeworks/homework3.html, due Monday, September 22.