Functional Morphology
by Markus Forsberg & Aarne Ranta

Otakar Smrž

Institute of Formal and Applied Linguistics
Charles University in Prague

Functional Morphology

- Implementing **morphological models**
  - Programming environment within Haskell
  - Extensible, powerful, language-independent
- Markus Forsberg & Aarne Ranta
  - Chalmers University of Technology
  - September 2004, International Conference on Functional Programming
- Inspired by Gérard Huet's toolkit Zen
  - Computational processing of Sanskrit, 2002

Outline of the Talk

- Little bit of Theory and Research
  - Karttunen, Stump, Buckwalter, Maxwell, Huet
- Finite-state modeling of morphology
  - Regular relations, finite-state transducers
  - Two-level morphology, lexicons and grammars
- Functional Morphology
  - Features, concepts, implementation issues
  - Demo of the system – formats, applications
  - Meeting requirements of different languages

Linguistic Perspective

- Inflectional morphology is understood in various ways (Stump 2001)
- Description of the inflectional processes
  - **Inferential** **rules, paradigms**
  - Lexical decomposition, affixation
- Preferred direction of consideration
  - **Realizational** **forms *reflect* parameters**
  - Incremental morphs *identify* features

Decisive Evidence

- Extended morphological exponence
  - One or more markings of a single property
- Null morphological exponence
  - Composition/decomposition not equivalent
- Non-concatenative inflection
  - Why restrict morphological operations to concatenation?

good < better << best * good|er << good|est

dobr|ý < lep|ší << nej|lep|ší * dobř|ejší << nej|dobř|ejší

Computational Concern

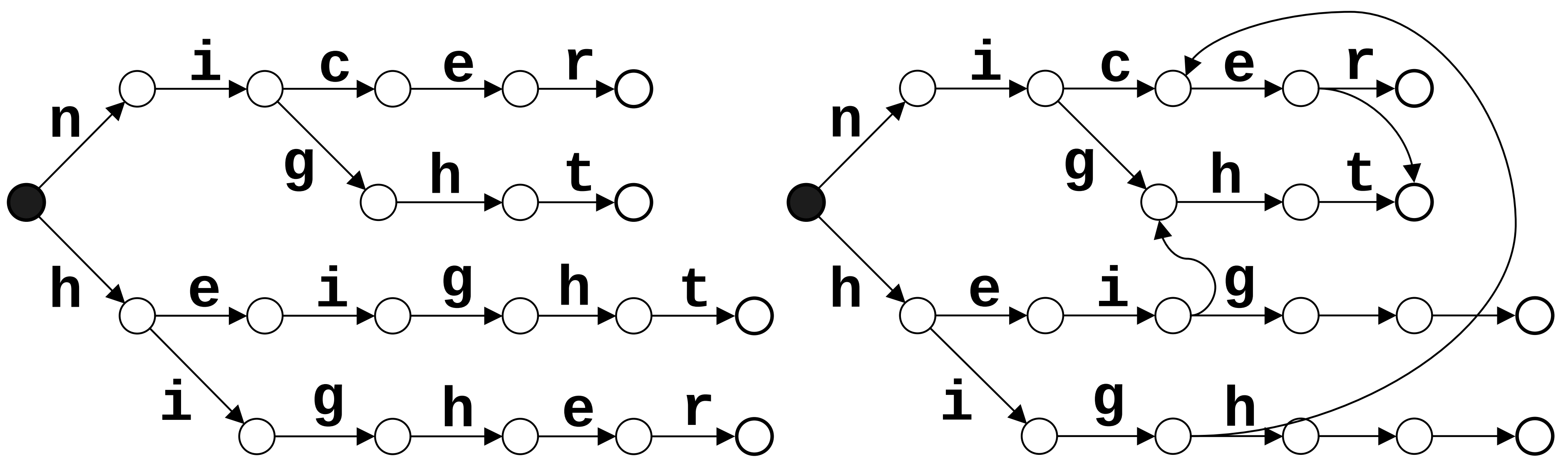
- Morphology can be captured by finite-state networks (Beesley and Karttunen 2003)
  - Implementation — **regular expressions**, right linear grammars
  - Complexity — **linear runtime**, advanced compilation techniques
  - Efficiency — **fast**, but large networks
- Non-regular formalisms might be difficult to implement efficiently enough

Efficiency vs. Expressivity

- Xerox Finite-State Tools like **xfst**, **lexc**
  - Languages of Europe, Arabic, Korean, Malay
- AT&T, Inxight, …, open-source FS tools
- Hybrid systems – Buckwalter's Analyzer
- DATR/KATR, MORPHE, Hermit Crab, …
- Functional Morphology in Haskell, Zen in Objective Caml – **compiled** into **tries**

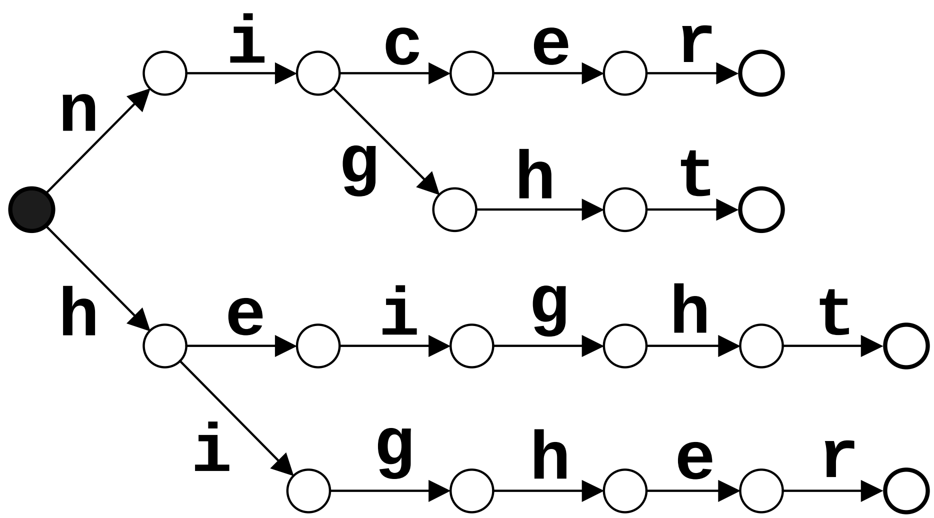
Languages as Networks

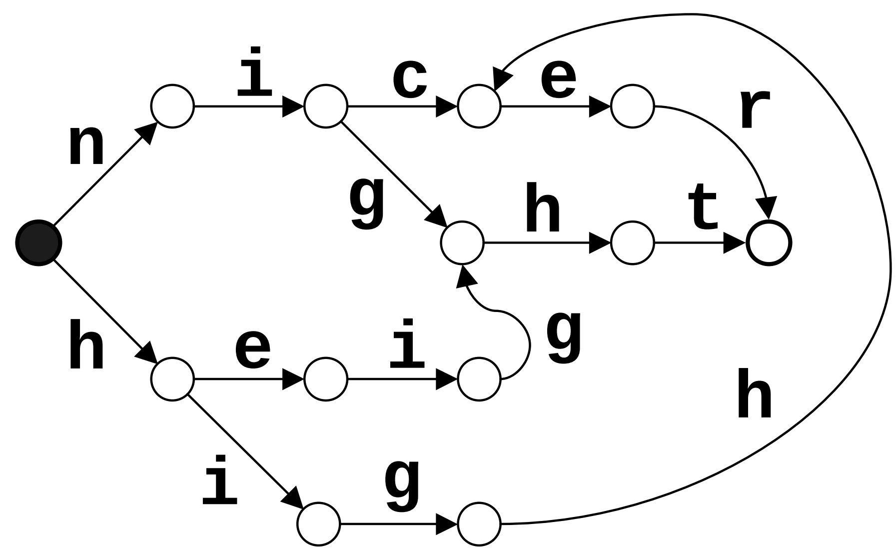
- ✧Languages are **sets** of sequences of symbols
- ✧Networks with limited number of **states**
- ✧Sequences of **symbols** recorded in **arcs**



Languages as Networks

- ✧ Languages are **sets** of sequences of symbols
- ✧ Networks with limited number of **states**
- ✧ Sequences of **symbols** recorded in **arcs**





REs and RLGs

- ✧Regular expressions describe such networks

  `L =(nicer|night|higher|height)` *listing*

  `=(ni(cer|ght)|h(eight|igher))` *prefix-*

  `=((nic|high)er|(n|he)ight)` *suffix trie*

- ✧Right linear grammars / lexicons do as well

  `ADJ ->{nice,high,happy}{CMP,{}}` *where*

  `CMP ->{+er}` *deriving from* **`L' ->{ADJ,{}}`**

  `L' = {nice,nice+er,happy+er,high,…}`

  or even **`{nice/ADJ+er/CMP,high/ADJ,…}`**

Regular Relations

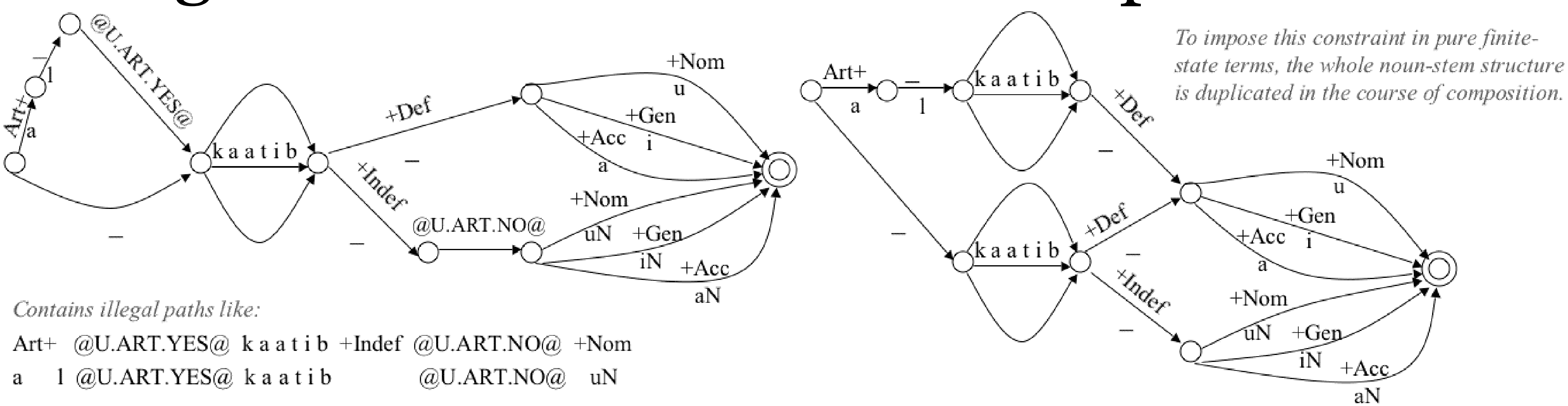
- ✧Networks can convert **input** into **output**
  - ✧Two languages – lexical/upper **:** surface/lower

`L" = {nice/ADJ+er/CMP:nicer,high/ADJ:high, happy/ADJ+er/CMP:happier,…}` *regular relation*

  - ✧Invertible structure, **analysis** iff **synthesis**
  - ✧Networks can be **composed** one over another
- ✧Building relations is not trivial!
  - ✧Two-level **rules** for orthographical alternations
  - ✧Every information merges into untyped **string**

Not Only Finite-State (Beesley)

- ✧ Flag diacritics vs. network multiplication



`~$[ Art%+ ?* %+Indef ] .o.` *the filter in* ***xfst***

- ✧ http://www.stanford.edu/~laurik/fsmbook/lecture-notes/Beesley2004/thupm.html

Burning Issues (Karttunen)

- Non-concatenative phenomena like interdigitation or reduplication
- Non-local dependencies
- Syntax/morphology interface
- http://www.cog.jhu.edu/workshop-03/Handouts/karttunen.ppt

More Burning Issues

- Does the direct coding allow to implement one's linguistic abstraction adequately?
  - Correspondence of formulations, expressivity
- Is the model **extensible** and **reusable**?
  - How much will it cost to add a lexical item?
  - Will refinement of information require global re-design, and/or will it cause inconsistencies?
- How can it be integrated into applications?
  - API and GUI interfaces, modularity, openness

Why **Functional**

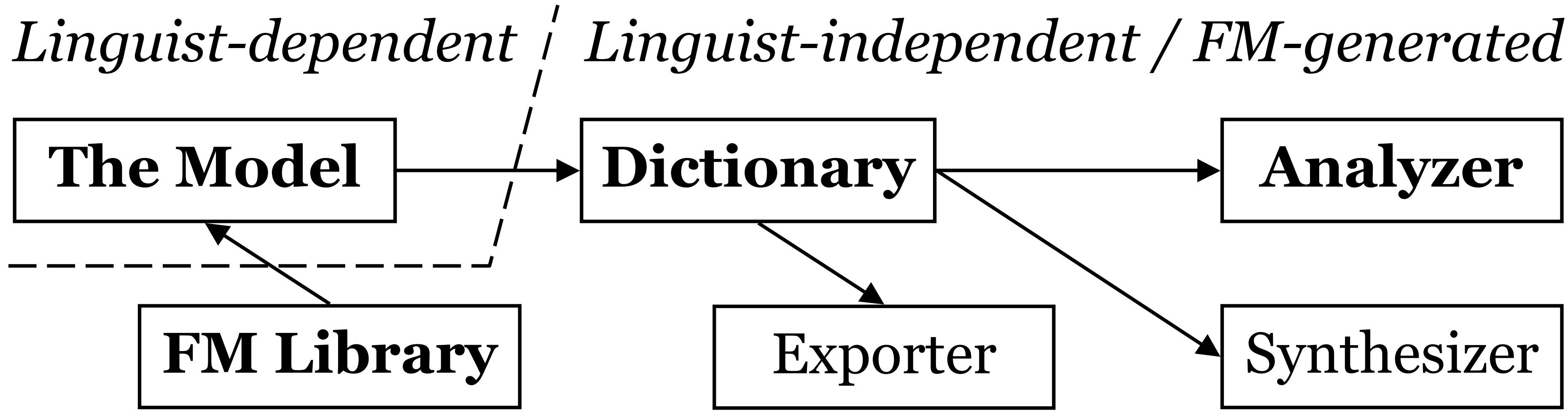
- Purely functional programming language **Haskell**
  - Higher-order functions, type classes, polymorphism
- Linguistic process ~ **function** on **entities** of the given description
  - Distinction between functions and forms in a language
  - Inflectional morphology may extend to derivational
  - Decomposition – phonology, orthography, grammar, …
- Excellent **progressive** functionality
  - FM provides high-level interfaces for concrete models
  - Inferential-realizational generality *& freedom of speech*

Why **Morphology**

- ✧ Methodology for developing similar models
  - ✧ Paradigms, inflectional + inherent parameters
- ✧ Embedded domain-specific language
- ✧ Collection of morphology implementations
  - ✧ Swedish, Spanish, Russian, Italian, Latin

- ✧ The Zen Computational Linguistics Toolkit
- ✧ Grammatical Framework ✧ FST Studio

FM Architecture



- The language model
  - Types meta-information
  - Functions tables/rules
  - *Lexicons* classified units
- Provisions by FM
  - *Dictionary* compilation
  - Runtime applications
  - Data export utilities

Inflection Tables & Parameters

- **Inflection** described by **finite functions**
- Analogy shown on a selected instance of the given group
- Realization of inflectional parameters yields the word form

rosa	**Singular**	**Plural**
Nominative	rosa	rosae
Vocative	rosa	rosae
Accusative	rosam	rosas
Genitive	rosae	rosarum
Dative	rosae	rosis
Ablative	rosa	rosis

Inherent Properties & Classes

- ✧ How do I describe words' non-inflectional properties, i.e. inherent parameters?
- ✧ Design word classes that refine the inflectional groups, and characterize them

- ✧ *Lexicon* associates **lemmas** with the classes
- ✧ *Dictionary* lists the **expanded** information

Parameters in FM/Haskell

- Parameters take their distinct type of values
- Values are constructed by symbolic names

```
data Case = Nominative | Genitive |
            Accusative | Ablative |
                Dative | Vocative
data Number = Singular | Plural
data Gender = Feminine | Neuter |
              Masculine
data NounInfl = NounInfl Case Number
```

Paradigm Definition

- Using functions with type signatures

```
ourParadigm :: String -> NounInfl
                     -> String
ourParadigm rosa (NounInfl n c) =
  let rosae = rosa ++ “e”
      rosis = init rosa ++ “is”
  in
    case n of
      Singular -> case c of
                    Accusative -> rosa ++ “m”
                    Genitive   -> rosae
                    Dative     -> rosae
                    _          -> rosa -- next slide
```

-- continued

```
Plural -> case c of
            Nominative -> rosae
            Vocative   -> rosae
            Accusative -> rosa ++ “s”
            Genitive   -> rosa ++ “rum”
            _          -> rosis

   -- where rosis = init rosa ++ “is”
```

- How, when and what does it compute?

ourParadigm “barba” (NounInfl Plural Genitive)

→ “barbarum”

ourParadigm “dea” (NounInfl Plural Dative)

→ “deis” *which is not correct Latin – we misused the paradigm*

FM pre-defined functions

- ✧ Programmer is free to be creative, as long as she keeps to the inferred system of types
- ✧ FM accounts for exceptions, missing/only forms, multiple variants, stem changes, …
- ✧ Each new model can add to this repertoire
- ✧ FM implements the whole mechanism
  - ✧ Tries for efficient analysis/synthesis
  - ✧ Exports to XML, SQL, **xfst**, **lexc**, GF, LaTeX, …

Lexicon Format

- Word **class** identification and the **lemma**
  - Lemma might yet be a function into a database
  - No programming needed – pure lexicography

Dictionary Format

- Class **functions** listing the **information**

```
ourClass :: String -> Entry
type Dictionary = [Entry]
```

Demo of the System

Inflection in Sanskrit

- Computationally pioneered by Huet (2003)
- Challenging issues in Sanskrit
  - Segmentation of compound words/verses
  - Alternation rules – external and internal *sandhi*
  - Phonetical orthography!
- The Zen Toolkit inspired FM greatly

Inflection in Arabic

- Quite structuralist computational models!
- Functional Arabic Morphology
  - Revised description of grammatical parameters
  - Implementation in FM, providing its extensions
- Challenging issues in Arabic
  - Run-on tokens, complex change of parameters
  - Decomposition of phonology and orthography

Summary

- ✧ **Functional Morphology** reconciles linguistic abstraction with computational implementation
- ✧ **Haskell** is a powerful, modern language
- ✧ Development of **morphologies** requires only little initial programming knowledge
- ✧ Development of **lexicons** reduces to natural lexicography

References

- Markus Forsberg and Aarne Ranta. 2004. Functional Morphology. In *Proceedings of the ICFP 2004*, pages 213—223. ACM Press.
- Gérard Huet. 2003. Lexicon-directed Segmentation and Tagging of Sanskrit. In *XIIth World Sanskrit Conference*, pages 307—325, Helsinki, Finland.
- Gregory T. Stump. 2001. *Inflectional Morphology: A Theory of Paradigm Structure*. Cambridge Studies in Linguistics 93. Cambridge University Press.
- Kenneth R. Beesley and Lauri Karttunen. 2003. *Finite State Morphology*. CSLI Studies in Computational Linguistics. CSLI Publications, Stanford, California.

Web Links

- http://www.cs.chalmers.se/~markus/FM/
- http://sanskrit.inria.fr/ZEN/
- http://www.google.com/search?q=AraMorph
- http://www.sil.org/computing/hermitcrab/
- http://www.arabic-morphology.com/
- http://www.fsmbook.com/
- http://www.haskell.org/
- http://www.ocaml.org/