# Disjunctive rule ordering in finite state morphology

Robert Malouf
San Diego State University

ABSTRACT

Paradigm Function Morphology (PFM; Stump, 2001) is an elaborated realization-based theory of inflectional morphology which is notable for its empirical scope and formal precision. As Karttunen (2003) shows, most of the apparatus of PFM can be straightforwardly mapped onto regular expressions or finite state machines (FSMs). However, Karttunen's implementation simplifies Stump's theory slightly by assuming that at most one rule per block may be compatible with any given form. This allows rule blocks to be compiled into FSMs simply by composing the FSMs which implement the individual realization rules. However, this precludes the case where more than one potentially applicable rules competes to apply to a particular form. In what Stump argues is a crucial feature of PFM, such rule competition should be resolved according to *Pāṇini's principle*: within each rule block, only the applicable rule with the narrowest domain is applied. In this talk, we will describe an alternative implementation of PFM as FSMs using van Noord and Gerdemann's (2001) FSA Utilities. This implementation, while otherwise similar in many respects to Karttunen's, uses Pāṇini's principle to resolve rule competition and so is more faithful to Stump's version of PFM.

First some background: in PFM, a *paradigm function* maps a pair $\langle X, \sigma \rangle$, where $X$ is a root of some lexeme $L$ and $\sigma$ is a set of morphosyntactic properties, to the pair $\langle Y, \sigma \rangle$, where $Y$ is the expression of the $\sigma$ cell of $L$'s paradigm. The paradigm function in turn is made up of *realization rules* of the form $\mathrm{RR}_{n,\tau,C}(\langle X, \sigma \rangle) = \langle Y, \sigma \rangle$, where $X$ is a form, $C$ is the word class which this rule is part of the paradigm of, $\sigma$ is a set of morphosyntactic features, and $Y$ is the form which realizes the morphosyntactic features in the property-set index $\tau$. A realization rule applies to $\langle X, \sigma \rangle$ just in case $\sigma$ is compatible with $\tau$. Within a paradigm function, realization rules are organized in *blocks* (the index $n$ indicates which block a rule is a member of). These blocks follow Stump's 'Paninian well-formedness condition' (p. 23), which requires that for every $\langle X, \sigma \rangle$, either (a) at most one rule is compatible with $\sigma$, or, (b) if more than one rule is compatible, the there is a narrowest applicable rule. Together with Pāṇini's principle, this ensures that at most one rule within each block may apply to a form.

To compile rule blocks into FSMs we represent sets of morphological features as finite state recognizers and realization rules as finite state transducers using the directed replacement operator (Karttunen, 1996; Gerdemann and van Noord, 1999). If we compile a rule block by composing the individual rules, following Karttunen (2003), the result will be an FSM which applies every applicable rule within a block to each input form. So long as condition (a) above is met, this will give the correct result. If condition (b) is met, however, it will not. Therefore, before combining rules into blocks by composition, our compiler first transforms the rules according to Pāṇini's principle so that condition (a) is met. This transformation takes place in two stages. First, we perform a topological sort of the rules in a block according to the 'narrower than' relation. Suppose we have two rules $\mathrm{RR}_{n,\tau_1,C}(\langle X,\sigma\rangle) = \langle Y_1,\sigma\rangle$ and $\mathrm{RR}_{n,\tau_2,C}(\langle X,\sigma\rangle) = \langle Y_2,\sigma\rangle$ in a block that satisfies Stump's well-formedness condition, where $\tau_1$ and $\tau_2$ are recognizers. Then the first rule is *narrower than* the second just in case the difference $\tau_1 - \tau_2$ is $\emptyset$, the empty language. Once we have sorted the rules in order of decreasing narrowness, then for each rule we change the property set index $\tau$ to $\tau - \cup\tau_i$, where $\cup\tau_i$ is the union of the property set indices of the preceding rules in the current block (cf. Erjavec 1994). The result of this transformation is a set of rules which satisfies condition (a), so that the rules in each block can be composed in any order to form a single transducer.

This compilation method (the prolog implementation will be given in the full paper and the complete source code will be made available on the web) allows linguists working in PFM to express their analyses more directly as finite state machines, with all the well-known benefits that implies. In the other direction, PFM offers language engineers working with finite state morphological systems a restrictive, cross-linguistically motivated formalism for organizing and expressing analyses of complex paradigms.

# References


Erjavec, T. 1994. Formalizing realizational morphology in typed feature structures. In G. Bouma and G. van Noord, eds., *Papers from the Fourth CLIN Meeting*, pages 47–58. BCN/University of Groningen.

Gerdemann, D. and G. van Noord. 1999. Transducers from rewrite rules with backreferences. In *EACL 99*. Bergen, Norway.

Karttunen, L. 1996. Directed replacement. In *Proceedings of the 34rd Annual Meeting of the Association for Computational Linguistics. ACL-96*. Santa Cruz, CA.

Karttunen, L. 2003. Computing with realizational morphology. In A. Gelbukh, ed., *Computational*

*Linguistics and Intelligent Text Processing*, vol. 2588 of *Lecture Notes in Computer Science*, pages 205–216. Heidelberg: Springer-Verlag.

Stump, G. T. 2001. *Inflectional Morphology*. Cambridge University Press.

van Noord, G. and D. Gerdemann. 2001. An extendible regular expression compiler for finite-state approaches in natural language processing. In O. Boldt and H. Juergensen, eds., *Automata Implementation*, vol. 2214 of *Lecture Notes in Computer Science*, pages 122–139. Springer-Verlag.