CrossMark

Phonology **34** (2017) 407–435. © Cambridge University Press 2017
doi:10.1017/S0952675717000203

*Formal and computational verification of phonological analyses**

Mans Hulden
University of Colorado

This article presents a selection of methods to analyse, compare, verify and formally prove properties about phonological generalisations. Drawing from both well-known and recent results in the domains of model checking and automata theory, a useful methodology for automating the task of comparing analyses and inventing counterexamples is explored. The methods are illustrated by practical case studies that are intended to both resolve concrete issues and be representative of typical techniques and results.

1 Introduction

Phonological argumentation often relies on the notion of equivalence. Perhaps an analysis is pursued in a new formalism that claims to achieve the same results as previous analyses, but with more elegant or parsimonious guiding principles. If equivalence with previous generalisations could be established with mathematical certainty, the phonologist could rest assured that no possible counterexample could later surface to mar the beauty of the system.

This article focuses on the ability to formally investigate properties of input–output mappings defined by various phonological approaches. At the core of the methods discussed is the idea of using finite-state automata, which model regular languages, and finite-state transducers (FSTs), which model regular relations, as a universal exchange language for models of phonotactics and phonological mappings. Techniques from automata theory and the field of formal verification are then employed to enable rigorous, all-encompassing analyses of the phenomena and grammars in question.[1]

* E-mail: MANS.HULDEN@COLORADO.EDU.
Thanks to Dale Gerdemann, Mike Hammond and Lauri Karttunen for useful insights and discussion around various versions of this work. Thanks also to the editors and anonymous reviewers for many helpful comments and pointers. All errors are my own.

[1] All program code used in the examples in this paper is available as online supplementary materials at https://doi.org/10.1017/S0952675717000203 and also (May 2017) at https://fomafst.github.io/fomaphonology.

https://doi.org/10.1017/S0952675717000203 Published online by Cambridge University Press

Although computational finite-state methods are normally associated with segmental models, the mappings in question do not necessarily have to be segmental, or string-based. Many techniques exist for producing string representations of structured information such as autosegmental models (Bird 1995, Yli-Jyrä 2013, Hulden 2015), hence what is said below will apply to a number of domains, although the focus is on models such as Optimality Theory (OT), Lexical Phonology and Morphology and general rewrite-rule models, which can be interpreted as bidirectional string-to-string translations.

The purpose of this article is primarily practical. First, I review some known theoretical results, without extensive proof or analysis. I then develop a detailed analysis of two rule-based accounts and two optimality-theoretic accounts of the same phenomena, to show how formal verification techniques can be employed to develop and rigorously prove properties of phonological analyses.

2 Model checking

In the computational sciences, model checking generally refers to a set of methods used to verify, with mathematically rigorous techniques, that the behaviour of a complex system is guaranteed to be correct. Such approaches are commonly preferred over the more traditional develop-and-debug paradigms when malfunctions and erroneous behaviour would be especially costly (Baier & Katoen 2008).

In general, auditions of system behaviour can roughly be broken down into three distinct approaches, often with increasing empirical power to catch errors: (i) PEER REVIEW, (ii) TESTING and (iii) FORMAL ANALYSIS. The first of these, peer review, generally coincides with the feedback cycle found in the phonological literature – a solution to a particular dataset is presented, subjected to criticism and refined accordingly. A testing procedure whereby the system is subjected to various inputs, and where the outputs are checked for correctness, is often found in computational approaches to phonology, which do not differ greatly from the standard debugging of computer programs. In such cases, a corpus of inputs and expected corresponding outputs is prepared, and automated tests are performed to ensure that the expected outputs are indeed produced (Boersma 1999, Beesley & Karttunen 2003, Hayes *et al*. 2013). The third category, formal analysis, is less often seen in the domain of phonology. This entails exploring all the possible states of a computational system and reporting or flagging possible undesirable situations that may arise. Also integral to the process is the capacity to generate possible counterexamples to a system specification – situations where a system's output would deviate from its specification. This last circumstance is attractive for the phonologist, since counterexamples to proposed analyses may not always be easily conceived or even found in a corpus. If, by contrast, a holistic analysis could be pursued using formal techniques, the linguist

https://doi.org/10.1017/S0952675717000203 Published online by Cambridge University Press

could be freed of the burden of having to think of a potential counterexample or locating a wide-coverage corpus to test the analysis against.

With regard to formal analysis, models of phonology have an advantage *vis-à-vis* general computational models, in that for most practical purposes, phonological mappings can be assumed to be restricted to the class of REGULAR RELATIONS (Johnson 1972, Langendoen 1981, Kaplan & Kay 1994). This means that it is possible to draw on the properties of finite-state automata, which characterise languages, and finite-state transducers, which characterise mappings between languages.[2]

3 Finite-state machines in phonology

Finite-state machines have been widely used in computational morphophonology, as they have a number of attractive features. The two main properties are (i) the ability to convert rewrite-rule specifications into transducers in various modalities, and (ii) the ability to compose multiple such transducers together into one monolithic transducer that performs the mapping that a battery of rewrite rules would do step by step. Regular and subregular languages also have attractive learnability properties, and serve to restrict phonology in many ways, something that has been the focus of much recent interest (e.g. Heinz 2009, Heinz & Idsardi 2011, Rogers & Pullum 2011, Chandlee *et al*. 2014). Over the years, many models of phonology have been shown to be no more powerful in expressive capacity than finite-state models; for example, grammars of two-level phonology, rewrite models of phonology, and (partially) Optimality Theory, and other formalisms have all been shown to be convertible into finite-state models (Koskenniemi 1983, Kaplan & Kay 1994, Karttunen 1998).

4 Properties of finite-state machines

Here I briefly review some of the central properties of finite-state machines – automata and transducers – with an eye on linguistic applications. Natural questions to ask from the point of view of formal verification include the following. Are two mappings equivalent (equivalence testing of grammars)? Is one mapping a subset of the other? Does a mapping ever produce more than one output for an input (grammar analysis)? For which input words do two mappings produce different outputs? Is a mapping a function (does the grammar admit variation)? What is the set of input words that a mapping simply repeats without alteration (grammar analysis)?

[2] The term MODEL CHECKING often implies working with a logical specification of some system, such as a sequence of rewrite rules. In this work, such a specification is implicit in the way phonological rewrite rules are in practice first converted into statements in first-order logic (Hulden 2009c), and subsequently into transducers. This step, however, is hidden from the user of the tool employed here, and I will not dwell on this point.

https://doi.org/10.1017/S0952675717000203 Published online by Cambridge University Press

4.1 Automata

Finite-state automata enjoy almost every imaginable closure and decidability property we might be interested in: 'an often stated rule of thumb [is] that everything concerning finite automata is decidable' (Salomaa 1993: 572). Automata are closed under all Boolean operations, and other operations such as reversals and homomorphisms; two automata can be tested for equivalence; two automata can be tested to see if one represents a subset of the other; an automaton can be tested for emptiness and universality (whether it accepts all strings over the alphabet). Automata also have a canonical minimal representation, which is efficient to compute (from a deterministic automaton) (Hopcroft 1971). Automata can also be constructed from various flavours of regular expression formalisms (Kaplan & Kay 1994, Karttunen 1995, Mohri & Sproat 1996, Vaillette 2004, Yli-Jyrä 2007, Hulden 2009a). These are mostly classical results dating back to Moore (1956), Myhill (1957) and Rabin & Scott (1959), and most of these properties can also be decided through very efficient algorithms (Hopcroft & Ullman 1979).

4.2 Transducers

In contrast to finite automata, transducers are more delicate in their properties. Transducers are closed under concatenation and union, but not complementation and intersection, or subtraction. Transducers also cannot, in the general case, be tested for equivalence (Griffiths 1968) – i.e. the problem is undecidable by algorithm. From the point of view of phonology, these properties imply that in order to perform model checking on transducer implementations of phonological mappings, we must be careful to stay within the bounds of algorithmic capacity. Below, I outline the main stumbling blocks that transducers pose for formal verification, and also point to methods to get around the limitations in most cases.

The first two well-known theorems which I will refer to are given in (1).

(1) a. It is undecidable whether two transducers represent the same relation (Griffiths 1968).
    b. It is decidable whether two transducers that represent functions represent the same function, and an efficient algorithm exists for the purpose (Culik II 1978, Hulden 2009a).

The consequence of the two theorems is that whenever two phonological mappings or grammars are represented as transducers and at least one of the mappings represents a function (i.e. admits no variation), the two grammars can be tested for equivalence. If this is not the case, equivalence testing may not be available. It should be noted that two grammars that admit variation may very well end up being represented by structurally identical transducers, in which case we can immediately conclude their equivalence. The general problem here is that two mappings may be

https://doi.org/10.1017/S0952675717000203 Published online by Cambridge University Press

modelled by two structurally different transducers that still represent the same relation; Fig. 1 gives a simple example.



Figure 1
Two transducers that model the same mapping but are structurally distinct.

Note that in this article, a phonological grammar that produces the same UR/SR mappings as some other grammar with a different *alignment* of individual input–output symbol pairs will be considered equivalent. In other words, the question of exactly how input–output symbols align in the final UR/SR mapping is ignored.

The theorem in (2) (Blattner & Head 1977, Culik II 1978) tells us, from a phonological point of view, that if a grammar is represented as a transducer, we can decide algorithmically whether it admits alternative outputs.

(2) It is decidable whether a transducer represents a function.

Here it is worth noting that there is no algorithm for separating the ambiguous mappings (i.e. those inputs that admit free variation) from the unambiguous ones (Hulden 2009a). The equivalence of two transducers is thus not decidable, even for those parts that have unambiguous input–output pairs.

However, there is a technically subtle point that is important here: while we cannot separate the ambiguous mappings in a transducer (i.e. $x \mapsto \{y, z\}$) from the unambiguous ones ($x \mapsto y$), we can extract the set of input strings that induce two or more successful paths in a transducer (Hulden 2009a: 150–156), as in (3). Some of those paths may very well result in the same output (i.e. map $x \mapsto y$ with different alignments of symbols). In other words, we cannot disentangle those ambiguous paths in a transducer that actually contain ambiguous mappings from those that do not.[3] However, if we know *a priori* from the way a transducer was constructed that every ambiguous input path through the transducer also produces an ambiguous output, we can separate out the parts that induce variation, and perform equivalence testing on those parts that do not – this is indeed the case with most rewrite-rule grammars built through composition of single rules.

(3) a. It is decidable whether a transducer maps an input to an output using multiple paths.
b. The set of words that induce an ambiguous path through a transducer is a regular set, and there exists an effective algorithm for extracting this set.

[3] This problem is known to be undecidable by algorithm in the general case.

https://doi.org/10.1017/S0952675717000203 Published online by Cambridge University Press

An important tool in the formal verification of phonological grammars is the ability to extract the set of inputs and outputs from an FST. Since these can be represented as automata, the results of input/output (domain and range) extraction from a relation often allow us to investigate properties of a grammar in greater depth, by virtue of the strong closure and decision properties that automata enjoy.

Many of the above algorithms are available in standard finite-state toolkits like xfst (Beesley & Karttunen 2003) and Kleene (Beesley 2012). In the following, I will assume the use of the foma-toolkit (Hulden 2009b), which contains implementations of all the algorithms discussed in this paper.

4.3 Proofs

Judicious combinations of the above techniques allow us to achieve something very desirable in phonology – formal proof of equivalence between models. Such proofs could be model-internal: one may want to prove that two different OT analyses are equivalent, despite employing different constraints, or prove that two rule-based accounts yield the same results. Alternatively, and probably more commonly, one may want to show equivalence of seemingly different analyses in different formalisms, say a rule-based account and an OT account, or an autosegmental model and a rule-based model. But we need not restrict ourselves to proof of equivalence only. Finite-state techniques also allow us to analyse, in detail, where two superficially dissimilar accounts contrast in their predictions.

5 Rule-based models

It is well-known that most phonological notions of 'rewrite rule' can be converted into a transducer representation, assuming rules are not allowed to apply to their own outputs more than a finite number of times. The first observation of this dates back to Johnson (1972), and actual implementations include Kaplan & Kay (1994), Karttunen (1995), Mohri & Sproat (1996), Vaillette (2004) and Hulden (2009a). By compiling individual rules into FSTs and combining the FSTs by composition, the action of extrinsically ordered rewrite rules can be effected.

In what follows, I will assume the availability of a number of types of rewriting modalities; these are based on results and suggestions developed over the years by various authors. In actuality, these replacement rules do not invoke a procedural notion of systematic replacement, but rather dictate what kinds of input–output correspondences are permitted and disallowed. The now-standard type of FST-based rewrite rule in (4) (Kaplan & Kay 1994, Karttunen 1995) in essence dictates that any string drawn from the regular set A may and must correspond to a string in regular

https://doi.org/10.1017/S0952675717000203 Published online by Cambridge University Press

set B if that string occurs between strings occurring in the regular languages C and D, where C and D are present in the INPUT SIDE of the relation.

(4) $A \rightarrow B \,||\, C __ D$

In order to model the notion of spreading in some rules (where apparently rules do apply multiple times to their output), modifiers to the rules can be added so that the conditioning environment can consist of segments produced as output by a rule (to the left or right), and not only the original input, as in (5).

(5) a. $A \rightarrow B \,\backslash\backslash\, C __ D$
C holds on the input side.
D holds on the output side.

b. $A \rightarrow B \,//\, C __ D$
C holds on the output side.
D holds on the input side.

One particular innovation from Hulden (2009a), which is implemented in at least two toolkits (foma, Kleene), is the TWO-LEVEL rewrite rule, inspired by the two-level phonology of Koskenniemi (1983). Here the contents of triggering contexts may be specified arbitrarily on the input and output sides, as in (6).

(6) $A \rightarrow B \,|||\, C_{in}{:}C_{out} __ D_{in}{:}D_{out}$

Such rules allow for the modelling of apparently cyclic phenomena often argued for in the literature, such as those advocated by Rubach (1984) and Halle & Nevins (2009).

Further useful variations of rewrite rule semantics include so-called longest-match and shortest-match rules (Karttunen 1996). For the practical discussions that follow, Table I gives the regular expression formalism assumed.

5.1 Conspiracies and output goals

An introductory example: an important type of formal analysis is afforded by the ability to represent sequences of rewrite rules as transducers is the ability to verify the presence of generalisations over the output set (Kisseberth 1970).

Having at our disposal a transducer which represents an entire grammar directly permits an analysis of the possible outputs of such a transducer. Extracting the range of outputs from a transducer that represents the effect of a rule or multiple rules yields a finite automaton that encodes the implicit surface phonotactics entailed by the rules. As an illustration, consider a simple hiatus-avoiding rule that deletes consecutive vowels, as in (7).

(7) $[+\text{syll}] \rightarrow \emptyset \,/\, [+\text{syll}] __$

https://doi.org/10.1017/S0952675717000203 Published online by Cambridge University Press

AB	Concatenation
A|B	Union
A*	Kleene star (zero or more)
A+	Kleene plus (one or more)
~A	Complement of language
?	Any symbol in alphabet
0	The empty set
\A	Any single symbol except those in A
$A	The language that contains a string in A (= ?* A ?*)
X/A	X, ignoring intervening instances of A
%	Escape symbol for reserved characters
[and]	Grouping brackets
A:B	Cross product
T1 .o. T2	Composition of T1 and T2
T.u, T.l	Input, output projection of transducer T
T.i	Inverse of T
A → B or A:B →	Rewrite strings from A with a string from B
A → B … C	Insert B and C around instances of A
A @→ B	Leftmost-longest rewriting
A @> B	Leftmost-shortest rewriting
(A), (→)	Optionality in language, optionality in rewriting
[..] → B	Insert (epenthesise) exactly one instance of B
|| C _ D	Input-side context specifier
\\ C _ D	Left (input) right (output) side context
// C _ D	Left (output) right (input) side context
||| C _ D	Two-level context-specification
.#.	End or beginning of string
_ ambpart(T)	Extract the ambiguous part of transducer T
def F(X_1, … , X_n) [regex]	Definition of macro
def X [regex]	Definition of language constant

Table I
Regular expression notation used in this paper.

By recasting this operation as a regular expression [V → 0 // V _].l, we can compile it into an automaton that characterises the set of possible outputs, as in Fig 2.

In this case, an inspection of the result is sufficient to establish that the effect of the rule is such as to preclude output configurations that contain VV. We could also establish this formally by calculating the intersection of this language with the language where every string contains a VV sequence somewhere, as in (8).

https://doi.org/10.1017/S0952675717000203 Published online by Cambridge University Press

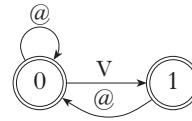


Figure 2
Automaton encoding all the possible outputs (given any input) of a rule that deletes the latter of two consecutive vowels. The @-symbol represents any symbol except V.

(8) $[V \rightarrow \emptyset \,//\, V __].1 \,\&\, \$[V\ V]$

By virtue of this analysis yielding an automaton that represents the empty set, it serves to verify formally that no VV sequences can surface as a consequence of the rule.[4]

In the analysis of more complex interactions of rules, the combined effect of multiple rules that conspire to constrain outputs may not be as easy to identify; casting the results as an automaton allows us to directly analyse the potential outputs of a cascade of rules and the indirect 'goals' that a battery of rules implicitly defines.

6 Optimality Theory and finite-state machines

Optimality Theory (Prince & Smolensky 1993) grammars are well known to both exceed FSTs in computational power in some respects (Frank & Satta 1998) and fall short in others, namely opaque interactions (see e.g. Idsardi 2000). The fact that an OT evaluation may involve comparison of an unbounded number of violations is the reason why even some very simple OT grammars are not expressible as FSTs. One such grammar is shown in (9). Here, the effect of the grammar is that, given an input of the form a^+b^+, it deletes all *a*'s or all *b*'s, whichever there are fewer of.

(9)

/aaabb/	IDENT	DEP	*ab	MAX
a. aaaaa	*!*			
b. aaacbb		*!		
c. aaabb			*!	
d. aaab			*!	*
e. bb				***!
☞ f. aaa				**

Such pathological cases are easily constructed, but they tend to not surface very often in real-world models. For this reason, it is often

[4] The @-symbol used in the transducer is a metasymbol that represents repetition of any symbol outside the alphabet.

https://doi.org/10.1017/S0952675717000203 Published online by Cambridge University Press

possible to convert many OT grammars into FST representations, which can then be used as a starting point for formal comparison with other models.

6.1 FST implementations of OT grammars

In the following, I draw on and develop further a technique for converting OT specifications into FSTs introduced by Gerdemann & van Noord (2000) and expanded in Gerdemann & Hulden (2012); similar approaches can be seen elsewhere (Eisner 2002), and many other finite-state approaches have been proposed (e.g. Ellison 1994, Hammond 1997, Karttunen 1998, Riggle 2004).

The basic idea is as follows. We first create a Gen-transducer that takes an arbitrary input and modifies it in all the possible ways GEN is assumed to do. An entirely generic GEN would of course arbitrarily insert, delete and epenthesise segments, as well as lay down feet, syllabify, mark stress, etc. Since we later need to identify where constraints are violated, in practice this requires us to preserve the input and keep the output distinct from the input. There are many potential mark-up schemes for doing so, one simple choice being to separate the input and output through mark-up symbols. If, as is the case in all of the concrete grammars presented in this article, we only choose to investigate certain possible deletions, insertions and modifications, it is possible to restrict Gen accordingly. This may be the case when a stress pattern or syllabification is the phenomenon under consideration, and all faithfulness constraints are assumed to outrank markedness constraints for the phenomenon at hand. A Gen-transducer for syllabification, for example, may be designed to only add syllable boundaries to the input arbitrarily. Naturally, it is possible to always work with a maximally generic Gen, at the cost of some complexity. Such a Gen would also require explicit specification of all faithfulness constraints for every grammar.

Having created a Gen-transducer, we can proceed to model the evaluation of constraints, i.e. the EVAL component. This is done by composition with successive transducers that remove the suboptimal candidates in the pool for each constraint, i.e. copy the best candidate(s) from input to output and map everything else to the empty set. In the transducer notation presented above, the overall scheme is as in (10).

(10) Gen .o. EvalC_1 .o. … .o. EvalC_n

Since transducers have a finite number of states and hence cannot 'count' violations without bound, some approximation has to be performed at this step. One possibility is to break down each filtering transducer into a transducer that marks up violations and then removes those candidates with k violations or more, then $k-1$, etc.

https://doi.org/10.1017/S0952675717000203 Published online by Cambridge University Press

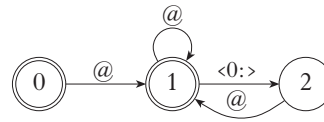


Figure 3
Example of a simple Gen-transducer for syllabification.
The @-symbols represent any symbol outside the alphabet
(i.e. symbols mentioned on other transitions here).

This is the approach taken in Karttunen (2006b). The downside of such an approach is that it will not produce a transducer that is generic in the sense of being able to produce the correct output for an input of any length with any number of violations. There exists another possibility, however, which is to use the set of candidates to produce worse candidates by an auxiliary transducer, and in turn use that transducer to filter out the suboptimal candidates (Gerdemann & van Noord 2000). For example, consider a syllabifying Gen-transducer (as in Fig. 3) and a constraint PEAK, which requires syllables to have exactly one vowel; assume that PEAK is the first constraint to be evaluated. For the input /paka/, the possible outputs of Gen will be [paka, pak.a, pa.ka, pa.k.a, p.aka, p.ak.a, p.a.ka, p.a.k.a].

Now consider a transducer, W, that performs the mappings in (11) for syllabified words.

(11) pa.ka → p.aka, paka, p.a.ka
pak.a → p.aka, paka, p.ak.a, pa.k.a
paka → p.a.k.a
pa.k.a → p.a.k.a
p.aka → p.a.k.a
p.ak.a → p.a.k.a
p.a.ka → p.a.k.a
p.a.k.a → ∅

It can be is easily seen that the set of outputs for this transducer has the following structure: (i) all six suboptimal candidates are in the union of the set of outputs, and (ii) each input maps to only candidates that are strictly worse than the corresponding input. If W fulfils these properties for *any* syllabified input, then we can use this set of outputs to mimic an EVAL-function, as in (12).

(12) Gen .o. ~[[Gen .o. W].l]

That is, we compose Gen with the complement of the set of outputs that occur when Gen is composed with the W-transducer. The result of this will be a transducer that always removes the suboptimal candidates from the current pool. This hinges crucially on our ability to create such a transducer for each constraint.

The conditions for being able to express an OT grammar as a transducer can then be summarised as in (13).

https://doi.org/10.1017/S0952675717000203 Published online by Cambridge University Press

(13) If, for every constraint, a transducer can be constructed that has the set of possible candidates as input and every suboptimal candidate in the pool exists in the output projection of that transducer, then the entire OT grammar is expressible as a transducer.

This is a sufficient condition; whether it is a necessary condition is an open question.[5] More informally, it simply places a restriction on the class of OT grammars that one must be able to describe, by means of a transducer, what the set of worse candidates looks like, given the optimal candidate and some constraint.

A roundabout way of achieving this kind of filter – one that is very convenient for most constraints – is to create it in two steps. First, create a transducer that marks violations of a particular constraint with a symbol, say *, then compose the result with a transducer that adds at least one violation mark to each word, as in (14a), which allows us calculate (14b).

(14) a. def E .#. | %.
def Peak \[V|E]* | \E* V \E* V \E* → … %* || E _ E
def AddViol [?* 0:%* ?*]+
b. Gen .o. Peak .o. ~[Gen.i .o. Gen]/%* .o. AddViol].l

The logic is that out of the marked-up violations produced by the Gen .o. Peak transducer, we want to filter out those that would be in the output if we further added at least a violation marker (AddViol) and undoing and redoing Gen's mark-up arbitrarily. We thus indirectly define the worse candidates as those that have the same violation markers as the optimal ones, and some more besides. The reason for Gen's effect being 'undone' and 'redone' by [Gen.i .o. Gen]/%* is that an optimal candidate, say [pa.ta], will not generate a suboptimal one such as [p*.ata*] by this scheme, because the syllable boundaries do not align in the two. By undoing and redoing Gen's effects arbitrarily we can produce, from [pa.ta], a number of suboptimal candidates, such as [p*.ata*, pata*, p*.a.t*.a]. This approach has the advantage of performing the desired calculation in a generic fashion, and allows us to focus on constructing only a transducer that marks violations with *.

Despite the calculations being somewhat involved, many finite-state toolkits allow one to wrap all of the above into convenient automated macros, freeing the phonologist to simply specify what constitutes a violation, using regular expressions.

[5] To my knowledge, no tight characterisation exists regarding the precise conditions under which an OT grammar can be modelled as a finite-state transducer. The condition above is implied by Gerdemann & van Noord (2000) and Gerdemann & Hulden (2012), but not explicitly stated. It is more general than the condition introduced by Frank & Satta (1998), which says that if GEN is regular and constraint transducers have a finite co-domain (i.e. can only distinguish between a finite number of classes), then the entire system is regular. By contrast, (13) implicitly takes the position of a constraint in a grammar into account by referring to the set of possible candidates given as input.

https://doi.org/10.1017/S0952675717000203 Published online by Cambridge University Press

In the formalism we are working with, we can define generic macros that perform all the work described above for multiple constraints, as in (15a).

(15) a. def AddViol [?* 0:%* ?*]+;
def Worsen [Gen.i .o. Gen]/%. .o. AddViol;
def Eval(X) X .o. ~[[X .o. Worsen].l] .o. {*} → 0;
b. Eval(Eval(Eval(Gen .o. Con1) .o. Con2) .o. Con3)

Here Eval is a macro that, when given an arbitrary transducer argument X, performs exactly the above calculation, and at the end, removes the temporary violation markers. The action of adding violations is defined through a generic transducer Worsen. For example, if we now had an OT grammar with three constraints and the corresponding transducers that mark up violations of those constraints, say Con1, Con2 and Con3, we could simply calculate the expression in (15b) to produce an FST of the entire grammar. Here Gen may be tailored to the grammar, of course. It is worth noting in particular the recursive use of the Eval macro/function, which is calculated inside out in the expression.

This method allows us – under most circumstances – to convert an OT grammar specification into an FST that models the grammar exactly for all inputs. If the suboptimal candidates always have violation markers in all the same places as *some* better candidate, the method is implicitly able to distinguish between an unbounded number of violations, despite being finite-state.

Some grammars, however, are going to exhibit cases where the condition on alignments does not hold. Consider a case where only two outputs remain at the point of evaluating a markedness constraint, STRESS-TO-WEIGHT, which militates against non-heavy stressed syllables. The violation markers would be inserted as in (16).

(16) a. ☞ (H́.L).L.(H̀.L).(L̀*.H)
b. (H́.L).(L̀*.H).(L̀*.L).H

The optimal candidate, (a), will not coincide with the suboptimal one, (b), even after shuffling around the foot markers and adding violations, simply because the violations do not align. As such, the above strategy would fail to distinguish between the two candidates, letting the suboptimal one slip through the finite-state sieve. One way to remedy a situation like this is to add a PERMUTATION transducer in the loop that moves violation markers around before adding violations (Gerdemann & van Noord 2000). It is beyond finite-state power to move symbols around some arbitrary distance, but for many purposes the Eval transducer can be sufficiently strengthened to give it the power to distinguish just enough disparity in violation alignment, as in (17).

(17) def Perm [?* ["*":0 \"*"* 0:"*"|0:"*" \"*"* "*":0]* ?*]*;
def EvalP1(X) X .o. ~[[X .o. Worsen .o. Perm].l] .o. {*} → 0;

https://doi.org/10.1017/S0952675717000203 Published online by Cambridge University Press

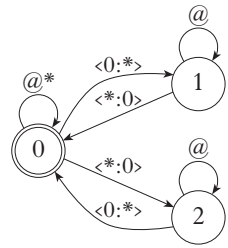


Figure 4
Illustration of a transducer that 'permutes' violation markers. @-symbols repeat everything that is not a *.

The Perm transducer (also shown in Fig. 4) moves markers arbitrarily but maximally to the next violation position in the same word.

For example, given the input (H́.L).L.(H̀.L).(L̀*.H), the Perm transducer would output at least the possibilities in (18).

(18) (H́.L).L.(H̀.L).(L̀.H*) (H́.L).L*.(H̀.L).(L̀.H)
(H́.L).L.(H̀.L).(L̀*.H) (H́.L*).L.(H̀.L).(L̀.H)
(H́.L).L.(H̀.L*).(L̀.H) (H́*.L).L*.(H̀.L).(L̀.H)
(H́.L).L.(H̀*.L).(L̀.H)

After such a manoeuvre, it is clear that (16a) can be made to coincide with (16b) by undoing and redoing Gen – in this case by refooting and restressing, since the role of Gen in this particular application is to lay down feet and stress syllables.

It is possible, of course, to compose the permutation transducer, Perm, indefinitely with itself multiple times to allow for increasingly stricter alignment matching and better Eval functions. For all practical grammar evaluations I have seen, zero or one round of permutation is sufficient. Where one round is not enough, it is generally the case that the grammar itself is not expressible as an FST.

Fortunately for the grammar designer, it is possible to decide algorithmically whether permutation is needed to avoid suboptimal candidates from slipping though the cracks of Eval. It is also possible to decide when a sufficient number of permutations have been performed. This is based on the fact that (as seen above) we can algorithmically decide when a transducer represents a function. To evaluate whether an Eval filter is sufficient, we can first apply the filter that removes suboptimal candidates, retain the violation markers and compose the result with a transducer Rem that removes everything except the markers, as in (19a).

(19) a. def Rem \"*" → 0; b. HLLHLLH → *,**

https://doi.org/10.1017/S0952675717000203 Published online by Cambridge University Press

In this example with the candidates in (16), the resulting transducer would perform the mapping in (19b). That is, some input (HLLHLLH in this case) would map to two different outputs. The entire system, therefore, would not represent a function. This circumstance allows us, after each additional permutation, to test whether the resulting transducer, after retaining the markers and composing with Rem, is a function. If so, we can stop permuting. In short, we can detect the necessity of permutation, and also know when the method has produced an exact FST model of the OT grammar at hand which correctly distinguishes between all optimal and suboptimal candidates.

A grammar can plausibly contain a constraint such that no amount of permutation will suffice to achieve an exact model. In most cases this corresponds to a grammar that simply is not finite-state, such as the one in (9) above.

6.1.1 *Gradient constraints*. Gradient constraints – constraints where the number of violations increase with displacement from some edge – are not implementable using mark-up as described above, for the simple reason that such a mark-up transducer would need to measure an arbitrary distance from an edge, such as the beginning of a word, and output a quantity of segments that corresponds to this distance. However, in many cases this can be remedied by directly creating a transducer that maps a candidate to all and only the worse candidates. A clean example is that of *um*-infixation in Tagalog (McCarthy & Prince 1993). In McCarthy & Prince's analysis, the location of an infixed *um* is determined by a gradient constraint ALIGN-*um*, which calls for a number of violations proportional to the distance (in number of segments) that the morpheme is displaced from the left edge of the stem. This is a case where a transducer model can exhaustively produce all worse candidates by simply moving an *um* morpheme one or more positions rightward, as in (20).

(20) def WorsenUM ?* [u m]:0 ?+ 0:[u m] ?*

This transducer produces the outputs in (21).

(21) umgradwet → gumradwet grumadwet graumdwet grumadwet
gradumwet gradwumet gradweumt gradwetum

A similar technique can be used to create suboptimal candidates for some metrical constraints, such as ALLFEET-L, penalising each foot that is removed from the left edge of the work gradiently. This constraint will be encountered in §7.1.3 in the analysis of Finnish stress.

It is worth noting that an OT grammar implemented as described above will *never* remove information from the input; it will only add symbols in the form of mark-up. This mark-up may represent stress, prosody or suprasegmental information, for example, or it may serve to distinguish the input from the output. If two OT grammars that differ in their constraints and ranking are implemented in this way, the consequence is

https://doi.org/10.1017/S0952675717000203 Published online by Cambridge University Press

that the output side of the resulting FST contains *all the relevant information* about the mapping. Since the output side can be extracted and represented as a regular language, and since regular languages are always testable for equivalence and other properties, the result is that two OT grammars, if encodable as above, can always be tested for equivalence (with the caveat that equivalence entails performing the same input–output mapping in the same way). More often, however, we are interested in checking equivalence without alignments being necessarily equivalent.

6.2 OT variants

Although not illustrated below with practical examples, it is obvious that many variants of OT, such as derivational and stratal models (Rubach 2000, Bermúdez-Otero 2007) can be profitably implemented, insofar as the individual constraints that they employ can be cast into FST form, as above. This is because of the ability to compose individual FSTs together (each of which presumably represents a level in the overall grammar). Other variants, such as comparative markedness models (McCarthy 2003), can also be represented, subject to the general caveats that prevail concerning FST modelling.

7 Practical examples of formal OT analysis

The foremost advantage with the above methods for implementing an OT grammar as an FST is that it yields (whenever possible) an *exact* representation of the grammar in question, which is not limited to handling inputs of only a certain length or a certain number of violations. This exactness allows us to pose universal questions about the resulting FST using the formal equivalence tests developed earlier. This means, for example, that in many cases we do not even need to test the model against specific words, but can investigate the grammatical system's behaviour holistically. Often this involves implementing two allegedly equivalent models representing two different formalisms, and then using finite-state algorithms to verify equivalence – or reveal a discrepancy.

This kind of formal verification is illustrated in what follows, drawing on two well-known examples from the phonological literature: patterns of Finnish secondary stress and Hixkaryana stress. In both cases, a model expressed as rewrite rules is compared to an OT model using the techniques just outlined.

7.1 Comparison between models: a case study of Finnish prosody

7.1.1 *The prosodic pattern in Finnish*. Finnish secondary stress placement has been the subject of many analyses (Elenbaas 1999, Elenbaas & Kager 1999, Kiparsky 2003, Karvonen 2005). The short review of primary and secondary stress placement in (22) outlines the issue in sufficient detail for present purposes.

https://doi.org/10.1017/S0952675717000203 Published online by Cambridge University Press

(22) a. Primary stress is initial, without exceptions.
b. Secondary stress has a trochaic binary pattern, and falls on odd-numbered syllables, except when scanning (from left to right) encounters an odd light syllable followed by a heavy syllable. In such cases, the light syllable is skipped. This can be interpreted as either forming a ternary foot, or leaving the light syllable as extra-metrical. For the current study this difference is irrelevant, since we only examine stress placement, not footing.

Examples of this alternation are given in (23).

(23) a. ('pe.rus).(ˌtel.la) 'to justify (INF)'
('pe.rus).(ˌte.li).sin 'to justify (1SG PRES COND)'
b. ('ka.me).le.(ˌont.ti) 'chameleon (NOM SG)'
('kom.mu).ni.(ˌkoin.nis).sa 'communication (INESS SG)'

(23a) illustrates the standard trochaic pattern. In (b) we see stress shift to the following heavy syllable.[6] The analysis below focuses on whether two specific OT analyses can capture this traditional pattern, as claimed.

7.1.2 *A rule-based description*. The rule-based description can be modelled as a composition of finite-state transducers built from replacement rules, as is done in Karttunen (2006a). The implementation is straightforward: inputs are syllabified by a syllabification transducer and composed with transducers that lay down ternary and binary feet, after which stress is assigned to the head of each foot. Part of this design is illustrated in (24). The result of the composition of these transducers is a single transducer that assigns primary and secondary stress to an arbitrary input sequence of Finnish segments.

(24) opiskelija
syllabify → o.pis.ke.li.ja
ternary → o.pis.(ke.li.ja)
binary → (o.pis).(ke.li.ja)
stress → ('o.pis).(ˌke.li.ja)

[6] Many unresolved cases exist, particularly with respect to CVC.CVV sequences and whether stress shifts happens there. Hakulinen *et al*. (2004) attributes the occasional shift to morphological causes, as in ('ar.gu).men.(ˌtoi.da) 'debate (INF)', associating the stress shift with an /-Oi-/ derivational morpheme which is claimed to attract stress (where /O/ is an archiphoneme representing either /o/ or /ø/). In my view, this is supported by similar non-shifting examples such as ('ka.lus).(ˌtet.tui).na 'furnished (PL ESS)'. However, Karvonen (2005) cites examples such as ('ho.ri).son.(ˌtaa.li) 'horizontal (NOM SG)'. While such questions could be profitably investigated using the techniques under discussion, I will leave these aside here. Two further complications will be ignored in the subsequent discussion: a final heavy syllable can also be optionally stressed, as in ('os.tin).pas ~ ('os.tin).ˌpas 'I bought', and word stress interacts in complex ways with allomorph selection (Anttila 1997, Kiparsky 2003).

https://doi.org/10.1017/S0952675717000203 Published online by Cambridge University Press

This analysis and the transducer that is produced by the composition of the individual rewrite-rule transducers will be called 'Traditional'. The next step is then to implement two OT analyses which are intended to capture the facts of Finnish secondary stress, and formally compare them against Traditional.

7.1.3 *OT models of Finnish word stress.* Both Elenbaas (1999) and Kiparsky (2003) develop OT analyses of Finnish word stress, implemented in Karttunen (2006b) as a transducer using a method for approximating OT grammars with transducers (Karttunen 1998). In what follows, instead of using an approximation (which cannot be universally compared with an exact rule-based transducer), I will implement the same analyses as an exact transducer using the methods described above.

The constraints proposed in Elenbaas (1999) and Kiparsky (2003) are given in (25), shown in order of ranking, and described together with rewrite rules that indicate how to mark up violations.[7]

(25) a. MainStress: Primary stress falls on the first syllable.
[B MSS ~$MSS]
(Absolute constraint, implemented simply as a filter automaton.)

b. Clash: Adjacent syllables can't be stressed.
SS → … %* || SS B __

c. Align-L: A stressed syllable must be foot-initial.
SV → … %* || .#. ~[?* "(" C*] __

d. FtBin: Feet have minimally two moras and maximally two syllables.
["(" Light ")" | "(" S ["." S]^ > 1] → … %*

e. Lapse: Every unstressed syllable must be adjacent to a stressed syllable or the word edge.
US → … %* || [B US B] __ [B US B]

f. Non-fin: The final syllable must not be stressed.
SS → … %* || __ ~$S .#.

g. i. Stress-to-Weight: Stressed syllables must be heavy.
[SS & Light] → … %* || ")" | E

ii. *(ÌLH): No feet of the form (ÌLH).
def LHfoot "(" [Light & SS] "." Heavy ")"
[..] → … %* || LHfoot __

h. License-σ: Syllables must be parsed into feet.
S → … %* || E __ E

i. AllFeet-L: Every foot coincides with the left edge of a prosodic word.
(Needs separate W-transducer; see below.)

[7] Some simple auxiliary transducers and automata are defined as follows: B (any boundary), C (consonant), V (vowel), E (syllable or word edge), S (syllable), SS (stressed syllable), US (unstressed syllable), Light (light syllable), Heavy (heavy syllable).

https://doi.org/10.1017/S0952675717000203 Published online by Cambridge University Press

The two constraints (g.i) and (g.ii) reflect the main differences between Kiparsky (2003) and Elenbaas (1999). I will consider both variants, but initially describe the implementation of (g.i), STRESS-TO-WEIGHT.

The highest-ranked constraint, MAINSTRESS, is considered categorical and inviolable, and there is no need to mark up violations. Instead, we assume that we can simply filter out all candidates that violate it after Gen has marked up feet, syllable boundaries and stresses. The transducer for Gen is built as in (26) (cf. Karttunen 2006b).

(26) def Gen [C|USV]* .o. MarkNonDiphthongs .o. Syllabify .o. Stress .o. Scan;

The component transducers MarkNonDiphthongs and Syllabify take care of Finnish syllabification, which is not handled by the OT constraints here (although one could do so with constraints ranked higher than those under discussion). Stress arbitrarily marks stress on vowels, and Scan arbitrarily divides words into feet. The very first transducer in the composition, [C|USV]*, simply acts as a filter constraining all inputs to only consist of Finnish consonants or unstressed vowels. For formal comparison, it is important to prevent Gen's mark-up from being allowed in the input and to force the rule-based transducer and the resulting OT transducer to have the same domain. Hence, the Gen-transducer maps the single word [pata] to the outputs in (27).

(27)

ˌpa.ˌta	ˌpa.ˈta	ˌpa.ta	ˌpa.(ˌta)	ˌpa.(ˈta)	ˈpa.ˌta
ˈpa.ˈta	ˈpa.ta	ˈpa.(ˌta)	ˈpa.(ˈta)	pa.ˌta	pa.ˈta
pa.ta	pa.(ˌta)	pa.(ˈta)	(ˌpa.ˌta)	(ˌpa.ˈta)	(ˌpa.ta)
(ˌpa).ˌta	(ˌpa).ˈta	(ˌpa).ta	(ˌpa).(ˌta)	(ˌpa).(ˈta)	(ˈpa.ˌta)
(ˈpa.ˈta)	(ˈpa.ta)	(ˈpa).ˌta	(ˈpa).ˈta	(ˈpa).ta	(ˈpa).(ˌta)
(ˈpa).(ˈta)	(pa.ˌta)	(pa.ˈta)			

This is a relatively small number of candidates. However, the number grows rapidly with the length of the input word. The longest non-compound word in Finnish is often cited as being *epäjärjestelmällistyttämättömyydelläänsäkäänköhän*.[8] This input word yields more than 636 trillion candidates, illustrating the need for formal analyses.

From the constraint CLASH onward we can proceed to filter out suboptimal candidates with the Eval function developed earlier. It turns out that all of the constraints except STRESS-TO-WEIGHT can be implemented exactly, without any permutation of violation markers. After one composition with the permutation transducer, STRESS-TO-WEIGHT becomes exact.

Putting the entire grammar together produces a transducer that has 168 states and 3129 arcs, and exactly represents the OT grammar in question.

[8] This opaque word can be analysed as a derivation from the root noun *järjestelmä* 'system', glossed as 'I wonder if – even with his/her/its quality of not having been made unsystematised'.

https://doi.org/10.1017/S0952675717000203 Published online by Cambridge University Press

It assigns a syllabified, footed and stress-marked output for any combination of Finnish characters, no matter how long the input word and no matter how many violations it incurs for some constraint.

Except for the last gradient constraint, ALLFEET-L, which is implemented with the function EvalAllFeet-L, the actual expression to compile the grammar using the Eval function is as in (28).

(28) def Kiparsky EvalAllFeet-L(Eval(EvalP1(Eval(Eval(Eval(Eval
(Eval(Gen .o. MainStress .o. Clash) .o. Align-L) .o.
FtBin) .o. Lapse) .o. Non-fin) .o. Stress-to-Weight)
.o. Parse));

Before further analysis, let us return to the point about the uncertainty of exactly capturing gradient constraints raised in §6.1.1. How do we know that the transducer implementation of ALLFEET-L indeed manages to filter out all suboptimal candidates, and that none slip through the cracks? As indicated before, this needs to be settled on a case-by-case basis. Let us begin by noting that in this particular ranking the gradient constraint is the *last* in the sequence of constraints. Now, the final FST can be tested for FUNCTIONALITY, i.e. whether every input maps to exactly one output, as discussed in §4. In this case, the resulting FST *is* functional, and so we know *a fortiori* that it cannot rank two different candidates equally in the last step.

With two FSTs in hand, Traditional and Kiparsky, we can proceed to test the two for equivalence, as described in §4. The two are not equivalent. This is expected, since the Traditional transducer contains ternary feet of the form (*σσσ*), which the OT transducer does not. Composing each with a transducer that removes the parentheses marking off feet, they are still not equivalent – which means they assign stress in different ways. In a similar fashion, we can also compile the analysis in Elenbaas (1999), which replaces STRESS-TO-WEIGHT with the constraint *(ÌLH). It also differs from Traditional.

For a formal analysis it is clearly unsatisfactory to simply conclude that two grammars behave differently. One would also like to know precisely for which inputs this happens. The first step toward a deeper analysis is to transform the three transducers, Traditional, Kiparsky and Elenbaas, to produce a more generic output, one that simply outputs sequences of Ls and Hs, with possible stress markings. This can be achieved by composing the stress-marking transducers with a transducer that converts the syllables into sequences over symbols H, Ĥ, L, Ĺ, as in (29a).

(29) a. def Collapse [Heavy & $SV] → Ĥ || E __ E .o.
[Heavy & ~$SV] → H || E __ E .o.
[Light & $SV] → Ĺ || E __ E .o.
[Light & ~$SV] → L || E __ E .o. %. → 0

b. matematiikassani → ĹLLĤHĹL

https://doi.org/10.1017/S0952675717000203 Published online by Cambridge University Press

That is, we simply map all syllables to one-letter descriptions. The resulting FSTs now map word inputs into HL sequences, as in (29b).

The output projections of these transducers are regular languages over four symbols. These can be extracted, producing automata that implicitly encode how the grammars in question assign stress. This is because if, for example, an ĹLLĹHHĹL appears in the language, we know that this is how the grammar stresses any LLLHHLL input. I will call the resulting automata TraditionalC, KiparskyC and ElenbaasC. Since we are now dealing with regular languages, i.e. automata as opposed to relations, we are free to use all Boolean operations on the automata. We can, for example, calculate the set of words for which a pair of the three automata give different predictions, say KiparskyC and TraditionalC, as shown in (30).[9]

(30) def DiffKT
[KiparskyC – TraditionalC] | [TraditionalC – KiparskyC]

We now have an automaton that models all the differences between the two grammars. One useful question to ask is: what is the shortest string where the two differ? This is a simple calculation; there are 32 strings of length 6 with different outputs, one of them being ĹHLĹLL. This corresponds to, for example, the real word *('ka.las).te.(ˌle.mi).nen (Kiparsky) and ('ka.las).(ˌte.le).(ˌmi.nen) (Traditional). Apart from these, there are an infinite number of input patterns where the two grammars give diverging results, as shown in Fig. 5.

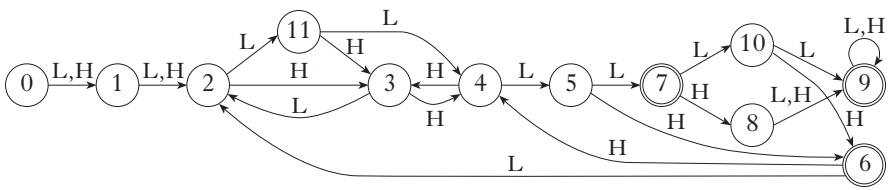


Figure 5
An FSA that represents the complete set of input patterns in which the Stress-to-Weight OT analysis (Kiparsky) differs from the traditional one.

To better analyse the differences, it is convenient to collapse the stress distinctions in this 'difference automaton', since our primary concern is to see which inputs produce different outputs. Hence we compose DiffKT with a transducer that removes the distinction between Ĺ and L, as well as between Ĥ and H, as in (31).

(31) def CollapseAccents(X) [X .o. Ĥ → H , Ĺ → L].l

[9] This is done by calculating the symmetric difference of the two sets; if two sets are identical their symmetric difference yields the empty set, otherwise the result contains all differing elements.

https://doi.org/10.1017/S0952675717000203 Published online by Cambridge University Press

The result of this is what is seen in Fig. 5, giving a compact overview of the shortcomings of the OT analysis. Figure 6 shows the corresponding differences between TraditionalC and ElenbaasC, i.e. the Elenbaas analysis.

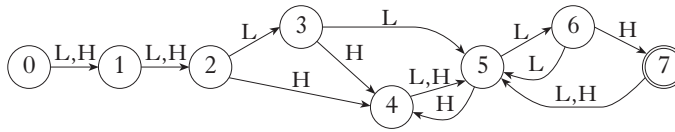


Figure 6
An FSA that represents the complete set of patterns in which the *(ÌLH) OT analysis (Elenbaas) differs from the traditional one.

7.1.4 *A provably correct OT model for Finnish secondary stress.* The tools at our disposal can not only be used to verify correctness, but also as a strong aid in debugging analyses. As Fig. 6 reveals, the analysis in Elenbaas (1999) only fails under very specific circumstances: when certain words of six syllables or more end in LH. This allows us to pinpoint the problem, which lies in the *(ÌLH) constraint. It unnecessarily penalises an LH foot even at the end of a word, when there is nothing to be gained by skipping it. That is, at the end of a word a new H-headed foot cannot be formed by skipping the light syllable. This is illustrated by errors such as those in (32).

(32) a. *('kes.kus).te.('le.mi).nen
b. ('kes.kus).('te.le).('mi.nen*)

Here, the *(ÌLH) constraint militates against the (mi.nen) foot, although a better foot cannot be formed, producing an incorrect winner. This hints at a possible solution (among many): *(ÌLH) could be modified to not penalise final LH feet, as in (33).

(33) *(LH): Disprefer non-final LH feet.
def LH2 [..] → {*} || LHfoot ? __ ;

As can be seen, the change is very minor: the FST implementation of the proposed solution only differs in one symbol, '?', from the original.

Does this produce a correct solution? Compiling the grammar with this modified constraint produces a transducer that is also functional, like the Traditional grammar. Given that two functional transducers can be tested for equivalence (see §4), we can proceed to do so. The result is that they are indeed equivalent, and therefore their analyses of the stress patterns are the same.

https://doi.org/10.1017/S0952675717000203 Published online by Cambridge University Press

7.2 Partial analyses and internal comparison: Hixkaryana stress

One perhaps troubling aspect about Optimality Theory is the prevalence of partial analyses in the literature.[10] These are analyses where only a few constraints are investigated through tableaux, and any candidates not examined are implicitly assumed to be removed by higher-ranked constraints. This makes formal analysis difficult, and any potential errors can only be assumed to be 'real' errors if they stem from a miscounting of violations for the candidates that are scrutinised. Any candidate omitted from the discussion can always be chalked up to the effects of a higher-ranked constraint.

The analysis of Hixkaryana stress exemplifies a number of these issues. First, in Kager's (1999) analysis, we find that a number of constraints are simply omitted from the analysis. In particular, these constraints are tacitly assumed to outrank the constraints discussed, and so the analysis proceeds without considering a large number of candidates that under other circumstances would be in contention.

Even so, there are ways to approach such partial analyses. We can attempt to encode the grammar in two ways: with and without the implicit assumptions included. Upon implementation, we find that the constraints give incorrect winners. This result was also arrived at in Halle & Idsardi (2000); however, the approach proposed here allows us to verify formally that the analysis is flawed, and also helps in correcting the analysis in a much simpler way than is done in Halle & Idsardi (2000).

7.2.1 *Kager's OT analysis*. Hixkaryana stress is analysed in two different ways in Kager (1999). First, a rule-based account is given, followed by a set of OT constraints which are claimed to have an equivalent effect. The rules are as in (34), given here with a finite-state implementation.

(34) a. Syllabify (open syllables are light; closed syllables are heavy).
def Syllabify C* V+ C* @> … %. || __ (C) V

b. Mark the final syllable of each word as extrametrical.
def MarkFinal \E* → %< … %> || E __ .#.

c. Assign iambs {(LH), (LL), (H)} iteratively from left to right.
def Stress [a:ˈa|e:ˈe|i:ˈi|o:ˈo|u:ˈu]
def Iambs (L %.) H | L %. L @→ %(… %) || E __ E .o.
Stress → || __ \V* %)

d. When the entire metrical domain is a single light syllable, assign (L) to it.
def OnlyL 0:%(C* Stress 0:%) → || .#. __ [%. %< | .#.]

e. Lengthen the vowel of each strong open syllable.
def Lengthen [..] → {:} || [ˈa|ˈe|ˈi|ˈo|ˈu] __ %)

[10] Albro (2005) is perhaps the only work that aims for a complete constraint description, providing a large-scale analysis of Malagasy.

https://doi.org/10.1017/S0952675717000203 Published online by Cambridge University Press

These are straightforward to implement as a composition of various types of replacement rules, as above. A few of them may merit comment. The syllabification rule uses a leftmost-shortest (@>) chunking rule (indicated by …) whereby sequences of C* V+ C* have a syllable boundary added to them on the right, if preceded by an optional consonant and a vowel. The leftmost-shortest regimen is often employed in syllabification rules such as this one, since it minimises the coda in case of ambiguity concerning consonant clusters, resulting in classic onset-maximisation behaviour (Hulden 2006). Since Kager does not specify the syllable patterns of Hixkaryana, this can safely be assumed to be the case, and indeed turns out to be true. By contrast, the Iambs rule uses the leftmost-longest regimen (@→) when laying down the feet, this particular rewriting modality being common in precisely this circumstance. The Lengthen rule uses an epenthetic insertion of the ':' symbol after stressed vowels and before the end of feet.

All the expected predictions in Kager (1999) are produced as the output; some are given in (35).

(35) kʷaja → (ˈkʷa).ːja 'red and green macaw'
atʃowowo → (a.ˈtʃoː).wo.wo 'wind'
tohkurʲehona → (ˈtoh).(ku.ˈrʲeː).ho.na 'to Tohkurye'
kʰananɨhno → (kʰa.ˈnaː).(ˈnɨh).no 'I taught you'

These effects are couched in an OT framework, using the eight constraints in (36).

(36) NON-FIN, GRWD=PRWD, FTBIN, WSP ≫ UNEVENIAMB, PARSE-σ, ALLFEET-L, DEP-IO(μ)

NON-FINALITY militates against final feet; GRWD=PRWD penalises non-footed words; FTBIN calls for feet that contain either two moras or two syllables; WSP is the Weight-to-Stress principle violated by heavy, unstressed syllables; UNEVENIAMB prefers (LH) feet over (LL) and (H); PARSE-σ calls for a violation of each unfooted syllable; ALLFEET-L is a gradient constraint which counts the number of syllables by which the beginning of a foot is displaced from word-initial position; DEP-IO(μ) penalises lengthening of vowels.

This analysis is incomplete, in that there is no mechanism by which stress assignment is controlled. Implementing the OT constraints directly yields a transducer that produces the type of ambiguous mappings in (37).

(37) kʰananɨhno → (ˈka.ˈnaː).ˈnɨh.ˈno (ˈka.ˈnaː).ˈnɨh.no (ˈkaː.ˈna).ˈnɨh.ˈno
(ˈkaː.ˈna).ˈnɨh.no (ˈkaː.na).ˈnɨh.ˈno (ˈkaː.na).ˈnɨh.no
(ka.ˈnaː).ˈnɨh.ˈno (ka.ˈnaː).ˈnɨh.no

This anomaly can be remedied by inspecting the original analysis. It is obvious that all the candidates considered in Kager (1999) carry stress only

https://doi.org/10.1017/S0952675717000203 Published online by Cambridge University Press

on the rightmost syllable in each foot. We can constrain the Gen-transducer to only assign stress in such positions (assuming that the analysis is not in violation of richness of the base, but is simply omitting high-ranked constraints that rule out stressed syllables elsewhere). This produces a transducer very close to the rule-based one, which we can analyse for correctness. Using techniques similar to the previous section, we end up with an expression like (38).

(38) def Hixkaryana Eval(EvalAllFeet-L(Eval(Eval(Eval(Eval(Eval(Eval (Gen .o. Non-fin) .o. GrWd=PrWd) .o. FtBin) .o. WSP) .o. UnevenIamb) .o. Parse-σ)) .o. Dep-IO(μ))

The OT transducer, it turns out, is not equivalent to the rule-based one, and yields different outputs for LLH-initial words such as /kʰananɨhno/ in (35), as shown in (39).

(39)

/kʰananɨhno/	Non-fin	GrWd=PrWd	Ft Bin	WSP	Uneven Iamb	Parse-σ	All Ft-L	Dep-IO (μ)
a. (kʰ a.ˈna:).(ˈnɨh).no					*!	*	**	*
☞ b. kʰa.(na.ˈnɨh).no						**	*	

This inconsistency is also noted by Halle & Idsardi (2000), who develop a lengthy critique of OT in general that hinges on the difficulty in fixing the Hixkaryana analysis. They write: 'in order to repair these inadequacies in the analysis, the number of constraints actively involved must be radically increased' (2000: 216). However, we will see that, if instead of adding constraints, we are content to modify the existing ones, we can produce an OT model that can be proved to be equivalent to the rule-based generalisation.

The troublemaker in Kager's proposed constraints is obviously the UnevenIamb constraint, which penalises (LL) and (H)-structured feet. The foot of concern in (39) is specifically a heavy lone (CVC) foot. We can indeed analyse the transducer representing this incorrect model to produce an exhaustive listing of the error patterns and reach the conclusion that *all* the unintentionally inferior candidates exhibit this particular structure. By modifying the transducer representation of UnevenIamb so that it does not consider CVC syllables heavy, we can create an alternative grammar that gives the correct predictions for LLH-initial words.

Formal comparison of the rule-based transducer and this modified OT transducer shows that they are equivalent in structure. Both yield a transducer of 25 states and 398 transitions, and are identical, thus always producing the same outputs for all inputs. This is a special case, where we do not need to resort to transducer-equivalence techniques (as in the Finnish example), since it so happens that the rule-based and OT transducers exhibit exactly the same structure, establishing certainty of equivalence for any input word whatsoever. As noted above, we could end up

https://doi.org/10.1017/S0952675717000203 Published online by Cambridge University Press

with two structurally different transducers that could still be equivalent in the mappings they define. In the present case, no special further techniques are needed to prove that the rule-based and the OT analyses are identical, and produce the same output for any input.

8 Conclusion

In this article I have advocated a specific methodology to formally verify the correctness or equivalence of phonological generalisations. The crucial component in such model checking is the use of finite-state transducers and automata as intermediaries, into which grammars in other formalisms are cast, and which are then algorithmically compared. The more extensive analyses of Finnish and Hixkaryana illustrate the strength of the approach – to prove equivalence, in many cases the phonologist need not even evoke a single word of the language that is being analysed. Rather, appropriately used finite-state algorithms can be put to use in exhaustively analysing the potential alternations and mappings that grammars give rise to. An internet search of the phrase 'an OT analysis of …' produces thousands of articles, almost all of which in some form claim to have reanalysed generalisations that have already been made in some other formalism. The methods shown in this paper allow for watertight proofs (or refutations, as the case may be) of such claims.

REFERENCES

Albro, Daniel M. (2005). *Studies in computational Optimality Theory, with special reference to the phonological system of Malagasy*. PhD dissertation, University of California, Los Angeles.

Anttila, Arto (1997). Deriving variation from grammar. In Frans Hinskens, Roeland van Hout & W. Leo Wetzels (eds.) *Variation, change and phonological theory*. Amsterdam & Philadelphia: Benjamins. 35–68.

Baier, Christel & Joost-Pieter Katoen (2008). *Principles of model checking*. Cambridge, Mass.: MIT Press.

Beesley, Kenneth R. (2012). Kleene, a free and open-source language for finite-state programming. In *Proceedings of the 10th International Workshop on Finite State Methods and Natural Language Processing*. Association for Computational Linguistics. 50–54.

Beesley, Kenneth R. & Lauri Karttunen (2003). *Finite state morphology*. Stanford: CSLI.

Bermúdez-Otero, Ricardo (2007). Diachronic phonology. In Paul de Lacy (ed.) *The Cambridge handbook of phonology*. Cambridge: Cambridge University Press. 497–517.

Bird, Steven (1995). *Computational phonology: a constraint-based approach*. Cambridge: Cambridge University Press.

Blattner, Meera & Tom Head (1977). Single-valued *a*-transducers. *Journal of Computer and System Sciences* **15**. 310–327.

Boersma, Paul (1999). Optimality-Theoretic learning in the Praat program. *Proceedings of the Institute of Phonetic Sciences of the University of Amsterdam* **23**. 17–35.

https://doi.org/10.1017/S0952675717000203 Published online by Cambridge University Press

Chandlee, Jane, Rémi Eyraud & Jeffrey Heinz (2014). Learning Strictly Local subsequential functions. *Transactions of the Association for Computational Linguistics* **2**. 491–503.
Culik II, K. (1978). Some decidability results about regular and push down translations. Research Report CS-78-09, University of Waterloo, Ontario. Available (May 2017) at https://cs.uwaterloo.ca/research/tr/1978/CS-78-09.pdf.
Eisner, Jason (2002). Comprehension and compilation in optimality theory. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics. 56–63.
Elenbaas, Nine (1999). *A unified account of binary and ternary stress: considerations from Sentani and Finnish*. PhD dissertation, Utrecht University.
Elenbaas, Nine & René Kager (1999). Ternary rhythm and the lapse constraint. *Phonology* **16**. 273–329.
Ellison, T. Mark (1994). Phonological derivation in Optimality Theory. In *Proceedings of the 15th International Conference on Computational Linguistics* (*COLING*). Association for Computational Linguistics. 1007–1013.
Frank, Robert & Giorgio Satta (1998). Optimality Theory and the generative complexity of constraint violability. *Computational Linguistics* **24**. 307–315.
Gerdemann, Dale & Mans Hulden (2012). Practical finite state Optimality Theory. In *Proceedings of the 10th International Workshop on Finite State Methods and Natural Language Processing*. Association for Computational Linguistics. 10–19.
Gerdemann, Dale & Gertjan van Noord (2000). Approximation and exactness in finite state optimality theory. In Jason Eisner, Lauri Karttunen & Alain Thériault (eds.) *Finite-state phonology: proceedings of the 5th Workshop of the ACL Special Interest Group in Computational Phonology* (*SIGPHON*). Luxemburg. 34–45.
Griffiths, T. V. (1968). The unsolvability of the equivalence problem for Λ-free nondeterministic generalized machines. *Journal of the* ACM **15**. 409–413.
Hakulinen, Auli, Riitta Korhonen, Maria Vilkuna & Vesa Koivisto (2004). *Iso suomen kielioppi*. [*Comprehensive Finnish grammar*.] Helsinki: Suomalaisen kirjallisuuden seura.
Halle, Morris & William J. Idsardi (2000). Stress and length in Hixkaryana. *The Linguistic Review* **17**. 199–218.
Halle, Morris & Andrew Nevins (2009). Rule application in phonology. In Eric Raimy & Charles E. Cairns (eds.) *Contemporary views on architecture and representations in phonology*. Cambridge, Mass.: MIT Press. 355–382.
Hammond, Michael (1997). Parsing syllables: modeling OT computationally. Ms, University of Arizona. Available as ROA-222 from the Rutgers Optimality Archive.
Hayes, Bruce, Bruce Tesar & Kie Zuraw (2013). *OTSoft 2.3.2*. Software package. http://www.linguistics.ucla.edu/people/hayes/otsoft/.
Heinz, Jeffrey (2009). On the role of locality in learning stress patterns. *Phonology* **26**. 303–351.
Heinz, Jeffrey & William J. Idsardi (2011). Sentence and word complexity. *Science* **333**. 295–297.
Hopcroft, John (1971). An n log n algorithm for minimizing states in a finite automaton. Technical Report STAN-CS-71-190, Computer Science Department, Stanford University.
Hopcroft, John E. & Jeffrey D. Ullman (1979). *Introduction to automata theory, languages, and computation*. Boston: Addison-Wesley.
Hulden, Mans (2006). Finite-state syllabification. In Anssi Yli-Jyrä, Lauri Karttunen & Juhani Karhumäki (eds.) *Finite-state methods and natural language processing*. Berlin & Heidelberg: Springer. 86–96.
Hulden, Mans (2009a). *Finite-state machine construction methods and algorithms for phonology and morphology*. PhD dissertation, University of Arizona.

https://doi.org/10.1017/S0952675717000203 Published online by Cambridge University Press

Hulden, Mans (2009b). Foma: a finite-state compiler and library. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics: demonstrations session*. Association for Computational Linguistics. 29–32.
Hulden, Mans (2009c). Regular expressions and predicate logic in finite-state language processing. In Jakub Piskorski, Bruce Watson & Anssi Yli-Jyrä (eds.) *Finite-state methods and natural language processing: post-proceedings of the 7th international workshop FSMNLP 2008*. Amsterdam: IOS. 82–97.
Hulden, Mans (2015). Grammar design with multi-tape automata and composition. In *Proceedings of the 12th International Conference on Finite State Methods and Natural Language Processing*. Association for Computational Linguistics.
Idsardi, William J. (2000). Clarifying opacity. *The Linguistic Review* **17**. 337–350.
Johnson, C. Douglas (1972). *Formal aspects of phonological description*. The Hague: Mouton.
Kager, René (1999). *Optimality Theory*. Cambridge: Cambridge University Press.
Kaplan, Ronald & Martin Kay (1994). Regular models of phonological rule systems. *Computational Linguistics* **20**. 331–378.
Karttunen, Lauri (1995). The replace operator. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics. 16–23.
Karttunen, Lauri (1996). Directed replacement. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics. 108–115.
Karttunen, Lauri (1998). The proper treatment of optimality in computational phonology. In Lauri Karttunen & Kemal Oflazer (eds.) *Proceedings of the International Workshop on Finite State Methods in Natural Language Processing*. Ankara: Bilkent University. 1–12.
Karttunen, Lauri (2006a). A finite-state approximation of optimality theory: the case of Finnish prosody. In Tapio Salakoski, Filip Ginter, Sampo Pyysalo & Tapio Pahikkala (eds.) *Advances in natural language processing*. Berlin & Heidelberg: Springer. 4–15.
Karttunen, Lauri (2006b). The insufficiency of paper-and-pencil linguistics: the case of Finnish prosody. Available as ROA-818 from the Rutgers Optimality Archive.
Karvonen, Daniel Howard (2005). *Word prosody in Finnish*. PhD dissertation, University of California, Santa Cruz.
Kiparsky, Paul (2003). Finnish noun inflection. In Diane Nelson & Satu Manninen (eds.) *Generative approaches to Finnic and Saami linguistics*. Stanford: CSLI. 109–161.
Kisseberth, Charles W. (1970). On the functional unity of phonological rules. *LI* **1**. 291–306.
Koskenniemi, Kimmo (1983). *Two-level morphology: a general computational model for word-form recognition and production*. Helsinki: Department of General Linguistics, University of Helsinki.
Langendoen, D. Terence (1981). The generative capacity of word-formation components. *LI* **12**. 320–322.
McCarthy, John J. (2003). Comparative markedness. *Theoretical Linguistics* **29**. 1–51.
McCarthy, John J. & Alan Prince (1993). Generalized alignment. *Yearbook of Morphology 1993*. 79–153.
Mohri, Mehryar & Richard Sproat (1996). An efficient compiler for weighted rewrite rules. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics. 231–238.
Moore, Edward F. (1956). Gedanken-experiments on sequential machines. In C. E. Shannon & J. McCarthy (eds.) *Automata studies*. Vol. 2. Princeton: Princeton University Press. 129–153.
Myhill, John (1957). Finite automata and the representation of events. Technical Report 57-624, Wright Airport Development Command. 112–137.

https://doi.org/10.1017/S0952675717000203 Published online by Cambridge University Press

Prince, Alan & Paul Smolensky (1993). *Optimality Theory: constraint interaction in generative grammar*. Ms, Rutgers University & University of Colorado, Boulder. Published 2004, Malden, Mass. & Oxford: Blackwell.
Rabin, M. O. & D. Scott (1959). Finite automata and their decision problems. *IBM Journal of Research and Development* **3**. 114–125.
Riggle, Jason (2004). *Generation, recognition, and learning in finite-state Optimality Theory*. PhD dissertation, University of California, Los Angeles.
Rogers, James & Geoffrey K. Pullum (2011). Aural pattern recognition experiments and the subregular hierarchy. *Journal of Logic, Language and Information* **20**. 329–342.
Rubach, Jerzy (1984). *Cyclic and Lexical Phonology: the structure of Polish*. Dordrecht: Foris.
Rubach, Jerzy (2000). Glide and glottal stop insertion in Slavic languages: a DOT analysis. *LI* **31**. 271–317.
Salomaa, Arto (1993). Decidability in finite automata. In G. Rozenberg & A. Salomaa (eds.) *Current trends in theoretical computer science: essays and tutorials*. Singapore: World Scientific. 572–578.
Vaillette, Nathan (2004). *Logical specification of finite-state transductions for natural language processing*. PhD dissertation, Ohio State University.
Yli-Jyrä, Anssi (2013). On finite-state tonology with autosegmental representations. In *Proceedings of the 11th International Conference on Finite State Methods and Natural Language Processing*. Stroudsburg, PA: Association for Computational Linguistics. 90–98.
Yli-Jyrä, Anssi & Kimmo Koskenniemi (2007). A new method for compiling parallel replace rules. In Jan Holub & Jan Žd'árek (eds.) *Implementation and application of automata*. Berlin & Heidelberg: Springer. 320–321.

https://doi.org/10.1017/S0952675717000203 Published online by Cambridge University Press