Optimality Theory is not computable[*]

Andrew Lamont

University College London

1. Introduction

This paper demonstrates that Optimality Theory (OT; Prince and Smolensky 1993/2004) is not computable. This means that it is impossible to write a computer program that determines the output of a given underlying representation, a set of constraints, and a GEN function, and does so in a finite amount of time. Not only is OT not computable in general, but I ground the result in the specific version used to model natural language. The practical consequences of this result should give us pause as linguists, casting even more doubt (e.g., Karttunen 2006) on analyses couched in OT. The empirical consequence of this result is that OT sacrifices any claim to being restrictive. While it may not be possible to model opaque interactions directly (Buccola 2013), it is possible to simulate the source code of XFST (Beesley and Karttunen 2003) in an OT grammar and use it to compile a rule-based analysis.

I derive this result by reducing the Post correspondence problem (PCP; Post 1946) to the generation problem in OT, i.e., determining the output of a given grammar for a given input. The PCP asks whether, given a finite set of domino types as in (1), a finite sequence of domino tokens exists such that the string running along the top is the same as the string running along the bottom. For this example, there are three solutions with the string *1 1 1* along the top and bottom. As I review in section 3, the Post correspondence problem is not computable: it is impossible to write a computer program that determines whether an arbitrary set of dominos has a solution. I construct two OT grammars in this paper whose output depends on whether a given instance of the PCP has a solution. Because this is impossible to determine, it is impossible to determine their output, and it follows that it is impossible to determine the output of an arbitrary OT grammar.

[*]For his patience and comments on this work at various stages of development, I am grateful to Neil Immerman. I also want to thank Julien Bradfield, Guy Emerson, Sophie Hao, Jeff Heinz, Shay Hucklebridge, Brandon Prickett, Geoffrey Pullum, Nicholas Rolle, Rachel Walker, and audiences at the Atelier de phonology at CNRS and the Formal Language Theory in Morphology and Phonology workshop at LSA 2024 for discussion. All remaining errors are of course my own.

(1) *An instance of the Post correspondence problem*

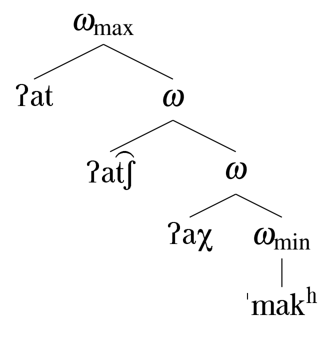
$$\left\{ \left[\frac{\quad}{1\;1\;1}\right], \left[\frac{1\;1}{\quad}\right], \left[\frac{1}{\quad}\right] \right\}$$

The result that OT is not computable is trivially true if the components of OT are entirely unrestricted (Eisner 2000:§3.3 makes a related point). Consider, for example, a GEN function that interprets inputs as a programming language and attempts to execute their instructions. Until a phonologist proposes that GEN function, I dismiss such a result as disingenuous and irrelevant to the work done by practicing phonologists. However, the result of this paper implies that OT, as it is used now, is not different from one that employs a compiler as its GEN function.

The extreme computational complexity of OT stands in stark relief to rule-based models of phonology such as Chomsky and Halle (1965) which are known to be finite-state (Johnson 1972, Koskenniemi 1983, Kaplan and Kay 1994). Insofar as these models are empirically adequate, natural language phonology also appears to be finite-state (see Heinz 2018 for an overview). Among other things, this means that it is impossible to write a rule-based grammar that takes URs of any length and returns their mirror image, e.g., mapping /fənalod͡ʒi/ onto [.i.d͡ʒo.lə.nof.], because that is not a finite-state function (Behrenfeldt 2009, Wang and Hunter 2023). While some variants of OT are finite-state (Ellison 1994, Karttunen 1998, 2006, Eisner 2000, 2002, Frank and Satta 1998, Riggle 2004, Hulden 2017), most are strictly more powerful including the frameworks most commonly used by practicing phonologists (Eisner 1997b, 2000, Frank and Satta 1998, Gerdemann and Hulden 2012).

As an illustration, I briefly present an OT grammar that copies inputs and reverses them. Suppose there is a language, call it *Zaqchikel*, with recursive prosodic words like those in Kaqchikel (cak; Mayan), as illustrated in (2) by [.ʔat.ʔat͡ʃ.ʔaχ.ˈmakʰ.] ‘you are an accomplice’ Patal Majzul (2007) *apud* (Bennett 2018:10). Following Itô and Mester (2007), the prosodic word dominating the root [makʰ] is a minimal prosodic word ω_{min} because it does not dominate another prosodic word, and the topmost prosodic word is a maximal prosodic word ω_{max} because it is undominated.

(2) *Prosodic word recursion in Kaqchikel (Bennett 2018:10)*



Suppose further that *Zaqchikel* has a process of total reduplication which preserves the recursive prosodic structure of the base in the reduplicant, while minimizing violations of the constraint ALIGN(ω_{min}, L, ω_{min}, R, σ) (3), which penalizes syllables that intervene between minimal prosodic words.

(3) ALIGN(ω_{min}, L, ω_{min}, R, σ) (following McCarthy 2003b:78)

For every minimal prosodic word ω_{min}, assign as many violations as there are syllables σ separating its left edge from the nearest right edge of a following minimal prosodic word ω_{min}.

As the tableau in (4) illustrates, this constraint prefers the minimal prosodic words of the base and reduplicant to meet at the base-reduplicant juncture (4h). Assuming other constraints force corresponding syllables to be equally embedded (e.g., by matching hierarchical structure in the syntax; Selkirk 2011), this forces the syllables in the reduplicant to mirror the order of those in the base.

(4) *Palindromic reduplication in* Zaqchikel

/$\sigma_1\sigma_2\sigma_3\sigma_4$/	ALIGN(ω_{min}, L, ω_{min}, R, σ)
a. $[\sigma_1[\sigma_2[\sigma_3[\sigma_4]]]]$-$[\sigma_1[\sigma_2[\sigma_3[\sigma_4]]]]$	W 3
b. $[\sigma_1[\sigma_2[\sigma_3[\sigma_4]]]]$-$[\sigma_1[\sigma_2[[\sigma_4]\sigma_3]]]$	W 2
c. $[\sigma_1[\sigma_2[\sigma_3[\sigma_4]]]]$-$[\sigma_1[[\sigma_3[\sigma_4]]\sigma_2]]$	W 2
d. $[\sigma_1[\sigma_2[\sigma_3[\sigma_4]]]]$-$[[\sigma_2[\sigma_3[\sigma_4]]]\sigma_1]$	W 2
e. $[\sigma_1[\sigma_2[\sigma_3[\sigma_4]]]]$-$[\sigma_1[[[\sigma_4]\sigma_3]\sigma_2]]$	W 1
f. $[\sigma_1[\sigma_2[\sigma_3[\sigma_4]]]]$-$[[\sigma_2[[\sigma_4]\sigma_3]]\sigma_1]$	W 1
g. $[\sigma_1[\sigma_2[\sigma_3[\sigma_4]]]]$-$[[[\sigma_3[\sigma_4]]\sigma_2]\sigma_1]$	W 1
☞ h. $[\sigma_1[\sigma_2[\sigma_3[\sigma_4]]]]$-$[[[[\sigma_4]\sigma_3]\sigma_2]\sigma_1]$	

The output of this grammar are strings of the form s-s^R, where s^R is the reverse of s (see Stemberger 1996 for another OT grammar that reverses strings). This is not a finite-state language, and therefore the mapping that produces it is not finite-state. Notably, the complexity of this mapping does not entirely depend on the recursive structure: Lamont (2021, 2022b) presents more complex OT grammars that entirely factor out representational complexity, employing only logically simple structures and constraints (Potts and Pullum 2002, Rogers and Pullum 2011). Grammars like these reflect OT's enormous capacity to generate complex mappings with simple constraints.

As this and examples like it illustrate, OT is more expressive than rule-based models, and by hypothesis, therefore more expressive than necessary as a model of natural language phonology. This complexity implies that it is also less time-efficient. Given an input of length n, a finite-state mapping produces an output within n steps, processing the input one segment at a time. Previous work characterizing non-finite-state OT frameworks along these lines has shown them to be too slow to be practical (Eisner 1997a, Idsardi 2006, Hao 2024; see Heinz et al. 2009 for discussion). For example, Idsardi's (2006) construction relies on generating all permutations of an input, taking at least $n!$ steps for an input with n segments. While this value explodes as n grows, it is finite and therefore computable within a finite amount of time: it is possible to write a computer program that takes an input and returns every permutation of it in a finite, albeit very large, amount of time. The result in this paper implies that it is not always possible to find the optimal candidate in a finite amount of time.

In general, OT being not computable means that it is impossible to write a computer program that determines the output for an arbitrary OT grammar in a finite amount of time. Similar results have been shown for a handful of other linguistic frameworks (Kaplan and Bresnan 1982, Johnson 1988, Smolensky 1992, Kepser 2004, Hale and Smolensky 2006, Francez and Wintner 2012, Hampe 2022, 2024, Kaplan and Wedekind 2023, Przepiórkowski 2023), two very similar to OT. The most relevant result to this paper is Smolensky's (1992) construction of complex Harmonic Grammars (Legendre et al. 1990a,b). This work demonstrates that it is possible to build a constraint set for a given formal grammar such that the strings it generates have harmony scores of at least zero and strings it does not generate have negative harmony. Given an unrestricted formal grammar, this determination is not computable. The construction relies on allowing constraints to both add to and subtract from harmony, which is not standardly assumed in phonological work (Potts et al. 2010, Pater 2009b, 2016), and allows languages to have entirely disjoint constraint sets, which also contradicts standard assumptions (Prince and Smolensky 1993/2004:6). In Appendix 1, I present a worked example of this construction. Hampe (2022, 2024) demonstrates that it is possible to model arbitrary computations in directional Harmonic Serialism (Lamont 2022a), implying that that framework is also not computable. Harmonic Serialism (Prince and Smolensky 1993/2004, McCarthy 2000, 2006, 2007, 2016) is a constraint-based framework that models input-output mappings recursively, supplanting representational complexity with derivational complexity. The results in this paper depend only on parallel derivations, controlling for the power of recursion – I return to this point in the following section.

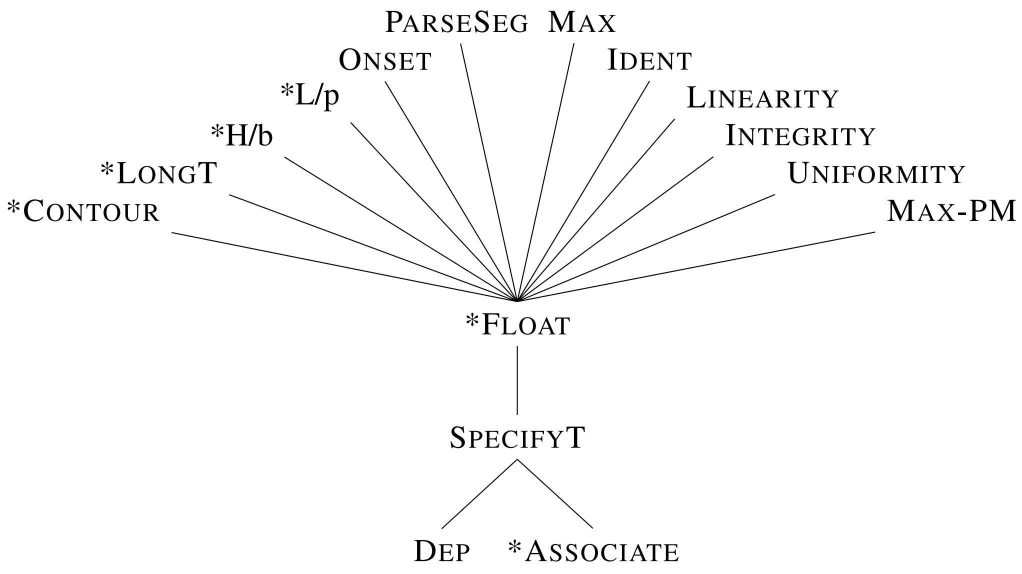
Given the venue, I assume readers have some basic knowledge of OT, but not necessarily any knowledge of computational theory. The main result is split between the following two sections. Section 2 constructs an OT grammar that attempts to create a one-to-one match between tones and syllables by inserting or copying morphemes. Its output depends only on whether such a match exists. Section 3 illustrates that it is impossible to determine its existence, proving the main result. In that section, I build up the result gradually after introducing enough computational theory for it to make sense to a general audience. Section 4 discusses the result and concludes.

2. The grammar

This section presents an Optimality Theoretic grammar that is not computable. In brief, it prefers outputs that have a one-to-one match between tones and syllables and inserts morphemes from the lexicon to achieve wellformedness. This section's goal is to present the grammar and argue that this is in fact what it does. An alternative construction involving reduplication is given in Appendix 2. The formal consequences of its existence are discussed in section 3.

The Hasse diagram in (5) summarizes the ranking of the constraints in this grammar. They are defined below.

(5) *Constraint ranking*



Collectively, the phonotactic constraints *CONTOUR (6), *LONGT (7), SPECIFYT (8), and *FLOAT (9) prefer one-to-one associations between tonal melodies and strings of syllables, the tone-bearing unit. To simplify the discussion below, SPECIFYT and *FLOAT are defined as binary constraints (Frank and Satta 1998): they assign at most one violation to any candidate.

(6) *CONTOUR (Yip 2002:83)

Assign one violation to every syllable linked to more than one tone.

(7) *LONGT (Yip 2002:83)

Assign one violation to every tone linked to more than one syllable.

(8) SPECIFYT (Yip 2002:83)

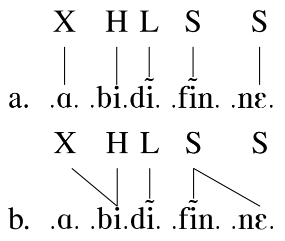
Assign one violation if there is a syllable not linked to a tone.

(9) *FLOAT (Yip 2002:83)

Assign one violation if there is a tone not linked to a syllable.

As a concrete illustration, the Seenku (sos; Niger-Congo) phrase meaning ‘these two goats’ (McPherson 2020:178) satisfies all four constraints (10a): each of its five syllables is linked to exactly one tone, and each of its five tones is linked to exactly one syllable. By contrast, (10b) violates all four constraints: the syllable [.bi.] is linked to two tones, violating *CONTOUR, the penultimate superhigh tone is linked to two syllables, violating *LONGT, the initial syllable [.ɑ.] is toneless, violating SPECIFYT, and the final superhigh tone is not linked to any syllables, violating *FLOAT. As in this example, I allow structures to violate the Obligatory Contour Principle (Leben 1973, McCarthy 1986).

(10) *A well-formed structure (a) and an ill-formed structure (b)*



Further restricting tone-syllable association are the constraints *H/b (11) and *L/p (12). The former penalizes high toned syllables with voiced or breathy onsets such as [l̤] and the latter low toned syllables with voiceless onsets such as [h].

(11) *H/b (Lee 2008:77)

Assign one violation to every high tone associated to a syllable with a voiced or breathy onset.

(12) *L/p (Lee 2008:77)

Assign one violation to every low tone linked to a syllable with a voiceless onset.

As a mnemonic aid, I restrict the set of tones to H and L and the set of syllables to [.ha.] and [.l̤a.]. Accordingly, phonotactically well-formed structures are those consisting only of the syllables [.há.] (*H* ↔ *h*) and [.l̤à.] (*L* ↔ *l̤*).

To ensure that segments are parsed into .CV. syllables, the constraint ONSET (13) penalizes onsetless syllables and the constraint PARSESEG (14) penalizes unsyllabified segments. These constraints prevent structures like [<h>.à.] where a violation of *L/p is avoided by underparsing the fricative.

(13) ONSET (Prince and Smolensky 1993/2004:20)

Assign one violation to every syllable without an onset.

(14) PARSESEG (Prince and Smolensky 1993/2004:243)

Assign one violation to every segment not parsed into a syllable.

The faithfulness constraints MAX (15), IDENT (16), LINEARITY (17), INTEGRITY (18), and UNIFORMITY (19) are standard Correspondence Theoretic (McCarthy and Prince 1994, 1995, 1999) constraints used here to prevent any modifications of lexical forms. With these faithfulness constraints ranked above SPECIFYT and *FLOAT, it is not optimal to delete, modify, reorder, fission, or fuse any segments or tones to achieve phonotactic wellformedness. *ASSOCIATE (20) penalizes the insertion of autosegmental associations; it must be ranked below SPECIFYT and *FLOAT to allow them to motivate linking.

(15) MAX (McCarthy and Prince 1994:342)

Assign one violation to every segment/tone in the input without a correspondent in the output.

(16) IDENT (McCarthy and Prince 1994:340)

Assign one violation to every segment/tone in the input that differs in its featural specifications from its correspondent in the output.

(17) LINEARITY (McCarthy and Prince 1994:340)

Assign one violation to every pair of segments/pair of tones in the input whose order differs from the order of their correspondents in the output.

(18) INTEGRITY (McCarthy and Prince 1999:296)

Assign one violation to every segment/tone in the input with multiple correspondents in the output.

(19) UNIFORMITY (McCarthy and Prince 1999:296)

Assign one violation to every segment/tone in the output with multiple correspondents in the input.

(20) *ASSOCIATE (Yip 2002:79)

Assign one violation to every unassociated segment and tone in the input with associated correspondents in the output.

The tableau in (21) illustrates the blocking effects of seven of the undominated constraints; this and following tableaux are presented in a comparative format recording candidates' violations and their comparison to the winner (Prince 2002, Brasoveanu and Prince 2011). The input consists of the segmental string /l̤a ha/ and a string of two floating tones /Ⓗ Ⓛ/. It surfaces faithfully with one violation each of SPECIFYT and *FLOAT (21a); recall that both constraints assign at most one violation. Associating the high tone to the first syllable and the low tone to the second is ruled out by *H/b and *L/p, which regulate which tones may associate to which syllables (21b). Underparsing the consonants trades these violations for fatal violations of ONSET and PARSESEG (21c). Deleting one tone and two segments achieves phonotactic wellformedness, but is ruled out by MAX (21d). Likewise, changing the tones to match the syllables is ruled out by IDENT (21e) and reordering them is ruled out by LINEARITY (21f).

(21) H L → H L
l̤a ha → .l̤a.ha.

	$H_1 L_2$ l̤a ha	*H/b	*L/p	ONSET	PARSESEG	MAX	IDENT	LINEARITY	*FLOAT	SPECIFYT	*ASSOCIATE
☞ a.	$H_1 L_2$ .l̤a.ha.								1	1	
b.	$H_1 L_2$ | | .l̤a.ha.	W 1	W 1						L	L	W 2
c.	H_1 L_2 | | <l̤>.a.<h>.a.			W 2	W 2				L	L	W 2
d.	L_2 | .l̤a.					W 3			L	L	W 1
e.	$L_1 H_2$ | | .l̤a.ha.						W 2		L	L	W 2
f.	$L_2 H_1$ | | .l̤a.ha.							W 1	L	L	W 2

The blocking effects of four more undominated constraints are illustrated in the tableau in (22), where an input consisting of the segmental string /l̤a ha ha/ and the floating melody /Ⓛ Ⓛ Ⓗ/ surfaces faithfully (22a). Linking both low tones to the initial syllable fatally violates *CONTOUR and linking the high tone to the following two syllables fatally violates *LONGT (22b). Fusing the low tones and fissioning the high tone allows for one-to-one associations, but is ruled out by INTEGRITY and UNIFORMITY (22c). Because SPECIFYT and *FLOAT are defined as binary constraints, they are only satisfied by candidates without any toneless syllables or floating tones. Partially associated structures, such as (22d), violate *Associate without improving on SpecifyT or *Float, and are harmonically bounded.

(22) $\begin{array}{ccc} \text{L L H} & & \text{L L H} \\ \text{l̤a ha ha} & \rightarrow & \text{.l̤a.ha.ha.} \end{array}$

$L_1 L_2 H_3$ l̤a ha ha		*Contour	*LongT	Integrity	Uniformity	*Float	SpecifyT	*Associate
☞	a. $L_1 L_2 H_3$ .l̤a.ha.ha.					1	1	
	b. $L_1 L_2 H_3$ .l̤a.ha.ha.	W 1	W 1			L	L	W 3
	c. $L_{1,2} H_3 H_3$ .l̤a.ha.ha.			W 1	W 1	L	L	W 3
	d. $L_1 L_2 H_3$ .l̤a.ha.ha.					1	1	W 2

The grammar thus cannot optimally achieve phonotactically well-formed structures by applying most phonological operations. However, because *Float dominates Dep (23) and *Associate, it is possible to insert material to provide tones to syllables and segmental hosts to tones.

(23) Dep (McCarthy and Prince 1994:342)

Assign one violation to every segment/tone in the output without a correspondent in the input.

While default tone insertion is common, inserting segmental material to host a floating tone is not. The former is attested in Poko (`rwa`; Skou), where lexically toneless syllables surface with mid tones, as in /nānē bini dō^{H}/ → [.nān. .bīn. .dō.] 'I get a fish' (McPherson and Dryer 2021:13); Yip (2002) cites a number of other examples. The latter is attested in Arapaho (`arp`; Algic), where epenthetic vowels only surface when a floating tone is present, as in /t͡ʃew-Ⓗseː/ → [.t͡ʃe.bí.se.] 'to walk along' (Gleim 2019:2), as well as in Wamey (`cou`, Niger-Congo; Rolle and Merrill 2022), and Ghomala' (`bbj`; Niger-Congo; Rolle to appear).

Following proposals by Xu (2007, 2011), Wolf (2008, 2015), and Rolle (2020), I allow Gen to insert not only arbitrary phonological structures, but also items from the lexicon. The formal result does not depend on this; section 2 presents an alternative grammar that relies instead on reduplication. As a simplification, both types of epenthesis violate Dep. The constraint Max-PM (24) penalizes all material not associated to a morpheme, restricting optimal epenthetic structures to those present in the lexicon.

(24) Max-PM (Walker and Feng 2004:777)

Assign one violation to every segment/tone not associated to a morpheme.

As another simplification, I assume that morphemes can only be inserted faithfully, but this can be derived. Suppose that another corner of the grammar tolerates toneless syllables and floating tones, allowing all inputs to surface faithfully. Given the appropriate output-output correspondence constraints (Benua 1997, Burzio 1998, Albright 2011), it is straightforward to disallow unfaithful morphological forms from surfacing. An alternative approach is exploited in Appendix 2.

To complete the construction, consider a lexicon consisting of a finite number of morphemes. Morphemes are composed of segmental strings containing only /ha/ and /l̤a/ and tonal melodies composed of high and low tones, but without any underlying associations between them. This again is a simplification – it should be possible to allow underlying associations provided that the constraint penalizing their loss, *Disassociate (Yip 2002:79), is dominated by *Float. Morphemes are color coded using the color palette from Okabe and Ito (2008), which is accessible to colorblindness.

As a first illustration, consider a language with the ranking in (5) and the lexicon {/ha ha ha/, /Ⓗ Ⓗ/, /Ⓗ/}. There are three morphemes: the first consisting only of a segmental string /ha ha ha/ and the second two consisting only of strings of floating high tones /Ⓗ Ⓗ/ and /Ⓗ/. Passing any one of these morphemes into the grammar will produce an output [.há.há.há.] with one of three morphemic parses. This is illustrated in the tableau in (25) with the input /Ⓗ/. The faithful candidate (25a) violates *Float because it consists only of a floating high tone. The six unfaithful candidates (25b-g) satisfy *Float by inserting segmental material; as discussed above, applying other operations cannot be optimal. Candidate (25b) inserts a schwa to host the floating tone and a glottal stop to provide a syllable onset, incurring two violations of Dep; here and elsewhere, I invert the colors of epenthetic material that does not belong to a morpheme. However, because this string is not a morpheme in the lexicon, it fatally violates Max-PM. Candidate (25c) avoids that violation by inserting the morpheme /ha ha ha/ and associating the floating tone to it. By doing so, however, it incurs a violation of SpecifyT (recall that this is a binary constraint) because it contains two toneless syllables. The optimal candidates (25d-f) solve this problem by inserting tonal morphemes from the lexicon, achieving phonotactic wellformedness. Because it does not matter which morphemic parse is optimal, I do not include any constraints to break the tie between them. Candidate (25g) likewise is well-formed, but loses because it inserts more material than necessary. The grammar thus finds a one-to-one match between tones and syllables by repeatedly inserting morphemes.

(25) /Ⓗ/ → $\begin{array}{ccc}\text{H}&\text{H}&\text{H}\\|&|&|\\\text{.ha.}&\text{ha.}&\text{ha.}\end{array}$ ∼ $\begin{array}{ccc}\text{H}&\text{H}&\text{H}\\|&|&|\\\text{.ha.}&\text{ha.}&\text{ha.}\end{array}$ ∼ $\begin{array}{ccc}\text{H}&\text{H}&\text{H}\\|&|&|\\\text{.ha.}&\text{ha.}&\text{ha.}\end{array}$

/Ⓗ/	MAX-PM	*FLOAT	SPECIFYT	DEP	*ASSOCIATE
a. Ⓗ		W 1		L	L
b. $\begin{array}{c}\text{H}\\|\\\text{.ʔə.}\end{array}$	W 2			L 2	L 1
c. $\begin{array}{ccc}\text{H}&&\\|&&\\\text{.ha.}&\text{ha.}&\text{ha.}\end{array}$			W 1	L 6	L 1
☞ d. $\begin{array}{ccc}\text{H}&\text{H}&\text{H}\\|&|&|\\\text{.ha.}&\text{ha.}&\text{ha.}\end{array}$				8	3
☞ e. $\begin{array}{ccc}\text{H}&\text{H}&\text{H}\\|&|&|\\\text{.ha.}&\text{ha.}&\text{ha.}\end{array}$				8	3
☞ f. $\begin{array}{ccc}\text{H}&\text{H}&\text{H}\\|&|&|\\\text{.ha.}&\text{ha.}&\text{ha.}\end{array}$				8	3
g. $\begin{array}{cccccc}\text{H}&\text{H}&\text{H}&\text{H}&\text{H}&\text{H}\\|&|&|&|&|&|\\\text{.ha.}&\text{ha.}&\text{ha.}&\text{ha.}&\text{ha.}&\text{ha.}\end{array}$				W 17	W 6

Consider a minimally different lexicon where the segmental string contains breathy laterals instead of voiceless fricatives: {/l̤a l̤a/, /Ⓗ Ⓗ/, /Ⓗ/}. For this language, every input that is composed only of these three morphemes is return faithfully. Because high tones cannot associate to syllables with breathy onsets, no combination will be phonotactically well-formed. The tableau in (26) illustrates this with the input /l̤a l̤a/. The faithful candidate (26a) violates SPECIFYT and is optimal because its satisfaction violates higher ranked constraints. Inserting either tonal string from the lexicon violates *H/b (26b). Deleting the input violates MAX (26c); epsilon ε represents the empty string. Inserting a tonal morpheme and changing the breathy laterals to voiceless fricatives violates IDENT (26d). As above, inserting a non-morphemic string is ruled out by MAX-PM (26e). The behavior of the grammar thus entirely depends on the lexicon – sometimes it finds a one-to-one match and sometimes it does nothing.

(26) /la la/ → [.la.la.]

/lala/	*H/b	Max	Ident	Max-PM	SpecifyT	Dep
☞ a. .la.la.					1	
b. $.\overset{\text{H}}{\text{la}}.\overset{\text{H}}{\text{la}}.$	W 2				L	W 2
c. ε		W 4			L	
d. $.\overset{\text{H}}{\text{ha}}.\overset{\text{H}}{\text{ha}}.$			W 2		L	W 2
e. $.\overset{\text{L}}{\text{la}}.\overset{\text{L}}{\text{la}}.$				W 2	L	W 2

Lastly, consider the lexicon {/la Ⓗ Ⓛ/, /la la la Ⓛ/, /la ha Ⓛ Ⓛ/}; this example comes from Lorentz (2001:215-216). For any morpheme passed in as input, the output will be that in (27); given its length, I do not include a tableau. This is to illustrate that when a one-to-one match is available, it may be surprisingly large. Another example from Lorentz (2001:215) is the lexicon {/la Ⓛ Ⓗ/, /ha ha la Ⓛ Ⓗ/, /ha la la Ⓛ/, /ha la Ⓗ Ⓛ Ⓛ/} which produces an output with seventy-six syllables. See Zhao (2002) for more extreme examples.

(27) $.\overset{\text{L}}{\text{la}}.\overset{\text{L}}{\text{la}}.\overset{\text{L}}{\text{la}}.\overset{\text{L}}{\text{la}}.\overset{\text{H}}{\text{ha}}.\overset{\text{L}}{\text{la}}.\overset{\text{L}}{\text{la}}.\overset{\text{L}}{\text{la}}.\overset{\text{L}}{\text{la}}.\overset{\text{L}}{\text{la}}.\overset{\text{H}}{\text{ha}}.\overset{\text{L}}{\text{la}}.\overset{\text{H}}{\text{ha}}.\overset{\text{L}}{\text{la}}.\overset{\text{L}}{\text{la}}.\overset{\text{L}}{\text{la}}.\overset{\text{H}}{\text{ha}}.\overset{\text{L}}{\text{la}}.$

Finally, consider what this grammar does when passed in an input containing some toneless syllables but no floating tones. If it is possible to create a phonotactically well-formed output by inserting from the lexicon, the grammar will return the shortest such candidate (28b). By doing so, it trades one violation of SpecifyT (again, a binary constraint) for arbitrarily many violations of Dep and *Associate. Because constraints are strictly ranked in OT, the amount of epenthetic structure is irrelevant; one violation of SpecifyT is strictly better than arbitrarily many violations of Dep and *Associate. If it is impossible to create a phonotactically well-formed output by inserting from the lexicon, the grammar returns the faithful candidate as output (28a′). Harmonic improvement is only possible by providing every syllable a tone, and because *Float dominates SpecifyT, doing so cannot create floating tones. What the grammar thus returns depends only on the input and the lexicon.

(28) *The grammar either returns the shortest well-formed candidate (b) or the faithful candidate (a′)*

/x/	*FLOAT	SPECIFYT	DEP	*ASSOCIATE
a. *x*		W 1	L	L
☞ b. *yxz*			$\lvert y\rvert + \lvert z\rvert$	$\frac{\lvert x\rvert + \lvert y\rvert + \lvert z\rvert}{3}$
☞ a′. *x*		1		
b′. *yxz*	(W 1)	(1)	W $\lvert y\rvert + \lvert z\rvert$	W $\frac{\lvert x\rvert + \lvert y\rvert + \lvert z\rvert}{3}$

As a local summary, the OT grammar in this section prefers outputs where every syllable [.ha.] is associated to exactly one high tone, every syllable [.l̤a.] is associated to exactly one low tone, every high tone is associated to exactly one syllable [.ha.], and every low tone is associated to exactly one syllable [.l̤a.]. If the input does not satisfy these criteria, the grammar either returns the input faithfully or returns the shortest phonotactically well-formed candidate that can be derived by inserting morphemes from the lexicon. The grammar's behavior is determined by whether such a candidate exists. However, it is impossible to determine whether such a candidate exists for any given lexicon. This is shown formally in section 3. I note in passing that a simpler construction is possible using Wilson's (2003) recursive definition of targeted constraints (Wilson 2000, 2001, 2013, Baković and Wilson 2000), where SPECIFYT is defined as preferring lexical insertion.

Before moving on, I want to mention that a stronger version of this grammar can be built that returns the faithful candidate or a perfectly matched candidate for all inputs. All this requires is a mechanism to rule out partial matches by penalizing derived violations of SPECIFYT of *FLOAT. One option is to include two conjoined constraints (Smolensky 1993, 2006b, Moreton and Smolensky 2002, Łubowicz 2002, 2003, 2005) that penalize morphemes that violate both DEP and either of these phonotactic constraints. Another option is to include comparative markedness constraints (McCarthy 2002, 2003a) that penalize new violations of SPECIFYT or *FLOAT. Ruling out partial matches for some inputs is not necessary to derive the formal result, but doing so provides a simpler characterization of the grammar for all inputs.

3. Computability

As mentioned in the introduction, the Optimality Theoretic grammar in section 2 models the Post correspondence problem (PCP; Post 1946), which is not computable. This implies that in general, OT is not computable: given an arbitrary input (and lexicon) and constraint ranking, the output cannot be determined algorithmically. This section builds towards the technical details of this result gradually; see Partee et al. (1990), Hopcroft et al. (2008), and Sipser (2013) for book-length introductions. Before discussing the PCP in section 3.3, I introduce Turing machines in section 3.1 and the halting problem, the archetypal non-computable problem, in section 3.2. Readers already familiar with computability should skip ahead to section 3.3. I include all this background to ensure that the result is meaningful to a general linguistics audience.

3.1 Turing machines

Turing machines are abstract computational models equipped with a finite set of instructions and an infinite memory or *tape* (Turing 1937). All Turing machines are defined with six components (29).

(29) *Definition of a Turing machine*

Q	a finite set of states
Σ	a finite set of symbols, the *alphabet*
δ	the *transition function*
$q_{\text{start}} \in Q$	the *start state*
$q_{\text{accept}} \in Q$	the *accept state*
$q_{\text{reject}} \in Q$	the *reject state*

As a concrete example, consider the Turing machine BB2 (Radó 1962) defined in (30).

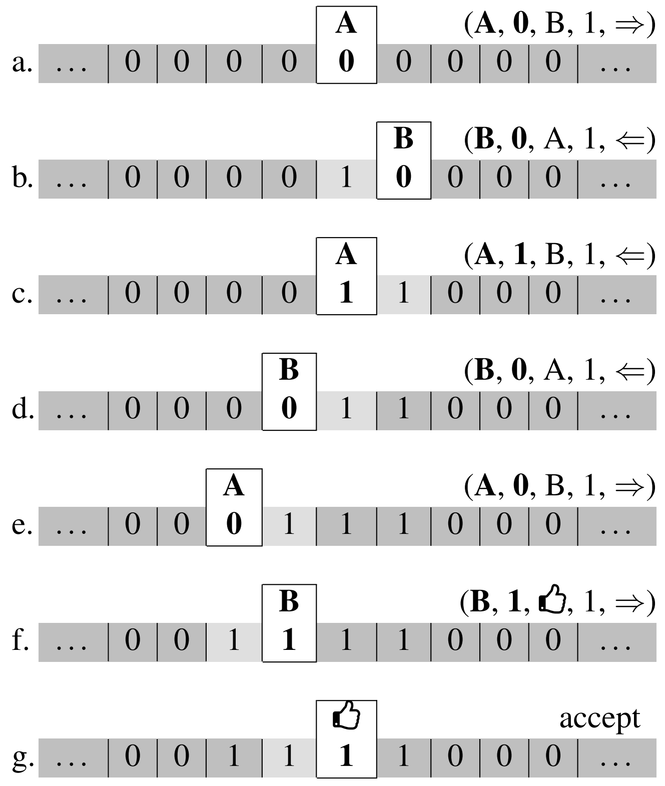
(30) BB2

Q	{A, B, 👍, 👎}
Σ	{0, 1}
δ	{(A, 0, B, 1, $\Rightarrow$), (A, 1, B, 1, $\Leftarrow$), (B, 0, A, 1, $\Leftarrow$), (B, 1, 👍, 1, $\Rightarrow$)}
q_{start}	A
q_{accept}	👍
q_{reject}	👎

BB2 has four states, A, B, 👍, and 👎, and operates over a binary alphabet consisting only of 0s and 1s. The transition function δ is the finite control that instructs BB2 what to do in any possible situation. Each of the four instructions consists of five parts: the current state, the current symbol, the new state, the new symbol, and the direction to move in. For example, the first instruction (A, 0, B, 1, $\Rightarrow$) means that if BB2 is in state A and it reads a 0, it should transition into state B, write a 1, and move one cell to the right. The start state is A, meaning that BB2 begins each computation in state A. The accept state is 👍 and the reject state is 👎; I use these emoji for accept and reject states throughout this section. If BB2 ends its computation in state 👍, it accepts the input, and if it ends its computation in state 👎, it rejects it. Because there is no instruction that puts BB2 in state 👎, it does not reject any inputs.

The behavior of `BB2` on a tape of all 0s is illustrated in (31). In the first step (31a), `BB2` begins in the start state, A, reading a 0. It is instructed to transition into state B, write a 1, and move to the right. In the second step (31b), `BB2` is in state B and reads a 0; it transitions back into state A, writes a 1, and moves to the left. `BB2` oscillates back and forth between states A and B over the next four steps (31c-f), writing two more 1s. In the sixth step (31f), `BB2` is in state B and reads a 1; it transitions into the accept state 👍, writes a 1, and moves to the right. Once `BB2` enters the accept state, it *halts*, terminating the computation and accepting the input.

(31) *Computation by* `BB2`



a. … 0 0 0 0 A 0 0 0 0 0 … (A, 0, B, 1, ⇒)
b. … 0 0 0 0 1 B 0 0 0 0 … (B, 0, A, 1, ⇐)
c. … 0 0 0 0 A 1 1 0 0 0 … (A, 1, B, 1, ⇐)
d. … 0 0 0 B 0 1 1 0 0 0 … (B, 0, A, 1, ⇐)
e. … 0 0 A 0 1 1 1 0 0 0 … (A, 0, B, 1, ⇒)
f. … 0 0 1 B 1 1 1 0 0 0 … (B, 1, 👍, 1, ⇒)
g. … 0 0 1 1 👍 1 1 0 0 0 … accept

Run on an input of all 0s, `BB2` writes four 1s and then halts. This is the maximum number of 1s that a Turing machine with 2 states (not counting the accept and reject states) can write on an input of all 0s, making `BB2` a *busy beaver* Turing machine (Radó 1962). For a given number of states, a busy beaver Turing machine writes the most 1s onto an input of all 0s before halting.

It is possible to construct a machine that writes more 1s with fewer states, but which does not halt. The one state (again, not counting the accept and reject states) Turing machine `Tsà` defined in (32) does exactly this. `Tsà` is named for the Tłı̨chǫ beaver spirit who teaches productivity.[1] Indeed, as illustrated below, `Tsà` is much more productive than `BB2`.

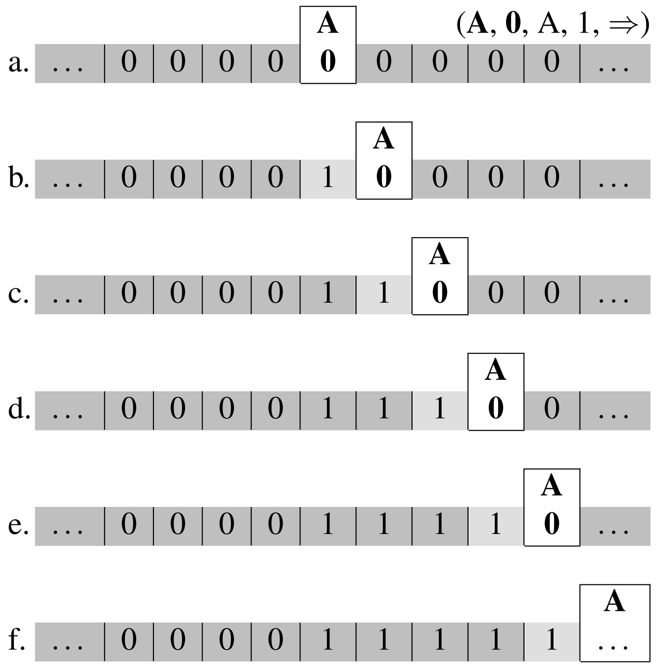
[1]Thanks to Shay Hucklebridge for suggesting the name.

(32) `Tsà`

Q	{A, 👍, 👎}
Σ	$\{0, 1\}$
δ	$\{(\text{A}, 0, \text{A}, 1, \Rightarrow), (\text{A}, 1, \text{A}, 1, \Rightarrow)\}$
q_{start}	A
q_{accept}	👍
q_{reject}	👎

The behavior of `Tsà` on an input of all 0s is illustrated in (33). In the first step (33a), `Tsà` is in state A and reads a 0. Its instruction is to remain in state A, write a 1, and move to the right. Because the input consists of infinitely many 0s, `Tsà` will never transition out of state A. It moves endlessly to the right, leaving behind a wake of arbitrarily many 1s. Unlike `BB2`, `Tsà` does not halt. It therefore cannot be said to be an algorithm and its behavior cannot be said to be a computation, as both terms refer to Turing machines that halt on all inputs.

(33) *Non-computation by* `Tsà`


(**A**, **0**, A, 1, ⇒)
a. … 0 0 0 0 **A 0** 0 0 0 0 …
b. … 0 0 0 0 1 **A 0** 0 0 0 …
c. … 0 0 0 0 1 1 **A 0** 0 0 …
d. … 0 0 0 0 1 1 1 **A 0** 0 …
e. … 0 0 0 0 1 1 1 1 **A 0** …
f. … 0 0 0 0 1 1 1 1 1 **A** …

These examples illustrate that not all Turing machines halt. Further, it is possible to define Turing machines that halt on some inputs, but not others. Suppose that we modified `Tsà` so that it transitioned into the accept state whenever it was in state A and read a 1. That Turing machine would halt on an input of all 1s, but not on an input of all 0s. As the following section demonstrates, it is impossible to determine whether a given Turing machine will halt on a given input.

3.2 The busy beaver problem and the halting problem

The Turing machines in the previous section model decision problems, i.e., binary classifications of strings. If a given Turing machine enters its accept state when run on a given string, it is said to accept the input. If it enters the reject state, it rejects it. A decision problem is said to be *computable* if it can be modeled by a Turing machine that is guaranteed to halt on all inputs; i.e., after a finite number of steps, the Turing machine enters its accept or reject state.

The halting problem is a decision problem that asks whether a given Turing machine halts on a given input. It is not computable (Church 1936, Kleene 1936, Turing 1937). This means that it is impossible to construct a Turing machine that takes as input the definition of a Turing machine and a string and determines in a finite number of steps whether that Turing machine would halt when run on that string.

To understand why the halting problem is not computable, consider first the busy beaver problem, which asks what the maximum number of 1s a Turing machine with a given number of states (ignoring the accept and reject states) can write onto an input of all 0s before halting (Radó 1962). Only five values are known, with some lower bounds established for machines with at least five states (34).

(34) *Known values of the busy beaver problem*

Number of states	Number of 1s	
0	0	Radó (1962)
1	1	Radó (1962)
2	4	Radó (1962)
3	6	Lin and Radó (1965)
4	13	Brady (1983)
5	$\geq 4{,}098$	Marxen and Buntrock (1990)
6	$\geq 3.514 \times 10^{18{,}267}$	Kropitz (2010)
7	$> 10^{10^{10^{10^{18{,}705{,}352}}}}$	Kropitz (2022) *apud* Michel (2022)

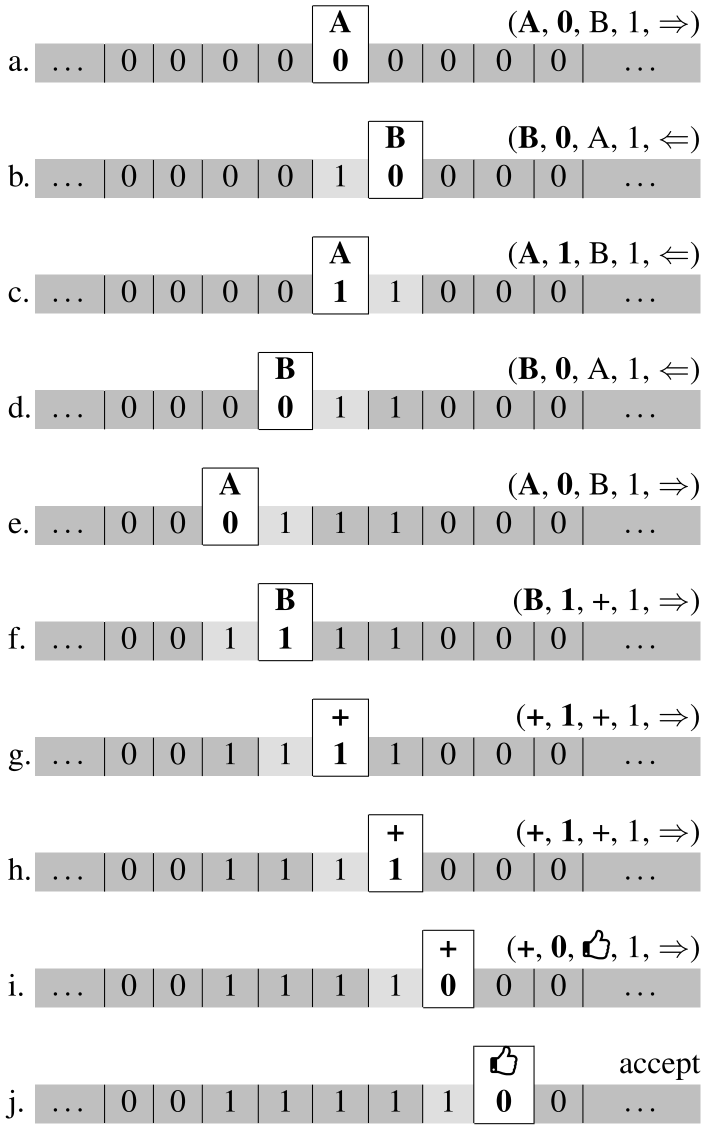
The function BB(n) that relates the number of states to the maximum number of 1s (e.g., BB(0) = 0, BB(2) = 4, etc.) is itself not computable (Radó 1962). BB(n) grows faster than any computable function, so it is impossible to determine it algorithmically (i.e., by a Turing machine that is guaranteed to halt). I reproduce Hopcroft's (1984) proof of this result below.

First, we can establish that BB(n) grows strictly in n, meaning that $\mathrm{BB}(n+1) > \mathrm{BB}(n)$. Given any busy beaver Turing machine, it is possible to add one state that finds the first 0 to the right or left, writes a 1, and transitions into the accept state. `BB2+` (35) illustrates such a modification of `BB2`. When a 1 is read in state B, instead of transitioning into the accept state, `BB2+` transitions into an added state +. In the state +, `BB2` searches for the nearest 0 to the right, writes a 1, and transitions into the accept state, halting. A computation by `BB2+` is shown in (36).

(35) `BB2+`

Q	{A, B, +, 👍, 👎}
Σ	{0, 1}
δ	$\left\{\begin{array}{l}(\text{A}, 0, \text{B}, 1, \Rightarrow), (\text{A}, 1, \text{B}, 1, \Leftarrow), (\text{B}, 0, \text{A}, 1, \Leftarrow), \\ (\text{B}, 1, +, 1, \Rightarrow), (+, 1, +, 1, \Rightarrow), (+, 0, 👍, 1, \Rightarrow)\end{array}\right\}$
q_{start}	A
q_{accept}	👍
q_{reject}	👎

(36) *Computation by* `BB2+`



a. (**A**, **0**, B, 1, ⇒)
b. (**B**, **0**, A, 1, ⇐)
c. (**A**, **1**, B, 1, ⇐)
d. (**B**, **0**, A, 1, ⇐)
e. (**A**, **0**, B, 1, ⇒)
f. (**B**, **1**, +, 1, ⇒)
g. (**+**, **1**, +, 1, ⇒)
h. (**+**, **1**, +, 1, ⇒)
i. (**+**, **0**, 👍, 1, ⇒)
j. accept

Second, for any natural number n, there is a Turing machine with $n + 19$ states (not counting the halting states) that writes n^2 1s onto a tape of all 0s. For example, there is a machine with 119 states that writes $100^2 = 10,000$ 1s on a tape with all 0s, and a machine with 120 states that writes $101^2 = 10,201$ 1s. n states are dedicated to writing n as a unary number. For example, the machine `Four1s` defined in (37) writes four 1s onto a tape of all 0s, one in each of its four states. The remaining 19 states copy the string of 1s, multiply the two unary numbers together, and remove the copies. Appendix 3 provides such a machine with 20 states that is reducible to 19 when combined with a machine like `Four1s`.

(37) `Four1s`

Q	{A, B, C, D, 👍, 👎}
Σ	$\{0, 1\}$
δ	{(A, 0, B, 1, $\Leftarrow$), (B, 0, C, 1, $\Leftarrow$), (C, 0, D, 1, $\Leftarrow$), (D, 0, 👍, 1, $\Leftarrow$)}
q_{start}	A
q_{accept}	👍
q_{reject}	👎

Finally, by way of contradiction, suppose that a Turing machine exists called `BBSolver`. `BBSolver` reads a string of n 1s, writes BB(n) 1s, and halts. For example, `BBSolver` writes six 1s on a tape containing three 1s and thirteen 1s on a tape containing four 1s. `BBSolver` has some number of states, k. It should therefore be possible to build a Turing machine `SquareAndSolve` for any natural number n that writes n^2 1s onto a tape of all 0s and then calls `BBSolver`, halting on a tape containing $\text{BB}(n^2)$ 1s. By definition, $\text{BB}(n^2)$ is the largest number of 1s a Turing machine with n^2 states can write before halting. `SquareAndSolve` also writes $\text{BB}(n^2)$ 1s and does so with only $n+19+k$ states. This contradicts the fact that BB(n) grows strictly in n. Consider the case when $n = 20+k$. `BBSolver` contains $n+19+k = n+19+(n-20) = 2n-1$ states. $2n-1$ is strictly less than n^2-1, when $n > 2$, which must be true because n is at least 20. Because BB(n) is strictly increasing, $\text{BB}(n^2-1)$ must be larger than $\text{BB}(2n-1)$. However, we have shown that $\text{BB}(2n-1) \geq \text{BB}(n^2)$ because `SquareAndSolve` is able to write as many 1s as a machine with n^2 states. Putting things together we get $\text{BB}(n^2-1) > \text{BB}(2n-1) \geq \text{BB}(n^2)$, implying $\text{BB}(n^2-1) > \text{BB}(n^2)$, which contradicts the fact that BB(n) is strictly increasing. Therefore, we conclude that `BBSolver` does not exist, and that BB(n) is not computable.

It immediately follows that the halting problem is not computable. Over the course of writing a large number of 1s to the tape, any given busy beaver Turing machine must take at least as many steps before halting. For example, `BB2` takes seven steps to write four 1s (31). Because we cannot compute the function BB(n), we also cannot compute the related function S(n) which asks how many total steps a Turing machine with n states can take before halting. Such a function would imply a solution to the halting problem – let a machine with n states run on an input until the S(n) + 1th step, at which point you can terminate the run knowing that it will never halt – but because it is not computable, the halting problem is also not computable.

If a formal problem can be shown to imply a solution to the busy beaver problem or the halting problem, it must be the case that it is not computable. The OT construction in this paper models a well known problem that is not computable: the Post correspondence problem (Post 1946). As the following section discusses, a general solution to the Post correspondence problem would imply a solution to the halting problem.

3.3 The Post correspondence problem

An instance of the Post correspondence problem (PCP) defines a finite set of domino types and asks whether a finite sequence of domino tokens can be concatenated such that there is a one-to-one match between upper and lower symbols (Post 1946).[2] Dominos are objects with two strings of symbols (or the empty string), one in the upper half and one in the lower half.

As an illustration, `PCP1` (38) provides a set of three domino types; astute readers may recognize this set from section 2. The colors are not meaningful, they are just a useful visual cue. Concatenating two dominos produces a third domino whose upper string is the concatenation of the two upper strings and its lower string is the concatenation of the two lower strings (compare Jardine's 2016 definition of autosegmental graph concatenation). For example, concatenating the blue and green dominos results in a domino with two 1s on top and 3 1s on the bottom (39a), and concatenating the orange domino to itself results in a domino with two 1s on the top and nothing on the bottom (39b). As in these examples, I only draw the square brackets around the underived domino types.

(38) `PCP1`

$$\left\{ \left[\frac{}{1\;1\;1}\right], \left[\frac{1\;1}{}\right], \left[\frac{1}{}\right] \right\}$$

(39) *Domino concatenation*

a. $\left[\frac{}{1\;1\;1}\right] + \left[\frac{1\;1}{}\right] = \frac{1\;1}{1\;1\;1}$

b. $\left[\frac{1}{}\right] + \left[\frac{1}{}\right] = \frac{1\;1}{}$

There are three solutions to `PCP1` with only three symbols in the upper and lower strings (40). In all three, a 1 in the upper string is matched to a 1 in the lower string. Except for the green and orange dominos, the relative order of concatenation does not matter.

(40) *The shortest solutions to* `PCP1`

a. $\frac{1\;1\;1}{1\;1\;1} = \left[\frac{}{1\;1\;1}\right] + \left[\frac{1\;1}{}\right] + \left[\frac{1}{}\right]$

b. $\frac{1\;1\;1}{1\;1\;1} = \left[\frac{}{1\;1\;1}\right] + \left[\frac{1}{}\right] + \left[\frac{1\;1}{}\right]$

c. $\frac{1\;1\;1}{1\;1\;1} = \left[\frac{}{1\;1\;1}\right] + \left[\frac{1}{}\right] + \left[\frac{1}{}\right] + \left[\frac{1}{}\right]$

[2] I am grateful to Cerek Hillen for introducing me to the Post correspondence problem.

By contrast, there is no solution to PCP2 (41). The upper strings only contain 1s and the lower strings only contain 0s.

(41) PCP2

$$\left\{ \left[\frac{}{0\ 0}\right], \left[\frac{1\ 1}{}\right], \left[\frac{1}{}\right] \right\}$$

The PCP is not computable (Post 1946); it is impossible to write an algorithm that determines whether an arbitrary set of domino types has a solution. Sipser (2013:§5.2) provides a proof that reduces the PCP to the halting problem. Essentially, any Turing machine and input can be translated into a set of domino types such that the set has a solution if and only if the Turing machine halts on that input. I reproduce this proof below.

Consider again the machine BB2 (42) and an input containing all 0s.

(42) BB2 *repeated from (30)*

Q	{A, B, 👍, 👎}
Σ	{0, 1}
δ	{(A, 0, B, 1, $\Rightarrow$), (A, 1, B, 1, $\Leftarrow$), (B, 0, A, 1, $\Leftarrow$), (B, 1, 👍, 1, $\Rightarrow$)}
q_{start}	A
q_{accept}	👍
q_{reject}	👎

The first domino type to add represents the start state and the input (43). In this case, the start state is A and we can represent the input as three 0s (other domino types add 0s as needed). *A 0* represents the machine being in state A and reading the 0 to its right.

(43) *The start state domino*

$$\left\{ \left[\frac{\#}{\#\ 0\ 0\ \mathrm{A}\ 0\ \#}\right] \right\}$$

Then we add dominos that encode the transition function (44). The first two dominos encode transitions where the machine moves to the right. For example, if BB2 reads a 0 in state A, it writes a 1, transitions into state B, and moves to the right. The next four dominoes encode transitions where the machine moves to the left. For example, if BB2 reads a 1 in state A, it writes a 1, transitions into state B, and moves to the left.

(44) *The transition function dominos*

$$\left\{ \left[\frac{\mathrm{A}\ 0}{1\ \mathrm{B}}\right], \left[\frac{\mathrm{B}\ 1}{1\ 👍}\right], \left[\frac{0\ \mathrm{A}\ 1}{\mathrm{B}\ 0\ 1}\right], \left[\frac{1\ \mathrm{A}\ 1}{\mathrm{B}\ 1\ 1}\right], \left[\frac{0\ \mathrm{B}\ 0}{\mathrm{A}\ 0\ 1}\right], \left[\frac{1\ \mathrm{B}\ 0}{\mathrm{A}\ 1\ 1}\right] \right\}$$

We then add matching dominos for each symbol in the alphabet as well as the padding symbol #, and a domino that adds a 0 to the tape (45).

(45) *Dominos encoding the alphabet*

$$\left\{\left[\frac{0}{0}\right], \left[\frac{1}{1}\right], \left[\frac{\#}{\#}\right], \left[\frac{\#}{0\,\#}\right]\right\}$$

Finally, we add dominos that encode the accept state (46). All of these have larger upper strings than lower strings and are used to allow the upper string to catch up to the lower string once the machine enters the accept state.

(46) *Dominos encoding the accept state*

$$\left\{\left[\frac{0\,\text{👍}}{\text{👍}}\right], \left[\frac{\text{👍}\,0}{\text{👍}}\right], \left[\frac{1\,\text{👍}}{\text{👍}}\right], \left[\frac{\text{👍}\,1}{\text{👍}}\right], \left[\frac{\text{👍}\,\#\,\#}{\#}\right]\right\}$$

The full set of dominos is given in (47), with dominos color-coded by function to avoid generating a palette with nineteen colors. As Sipser (2013:205) points out, this construction has three solutions of length one corresponding to the first three dominos encoding the alphabet. He provides a way to force solutions to contain the start state domino, which I skip to streamline the presentation. Instead, we will concern ourselves with the modified Post correspondence problem, which asks whether a set of domino types has a solution that starts with a specified domino token, in this case, the start state domino. Unsurprisingly, the modified Post correspondence problem is also not computable; see Harrison (1978:8.3) for discussion.

(47) *The domino encoding of* `BB2`

$$\left\{\begin{array}{l}
\left[\frac{\#}{\#\,0\,0\,A\,0\,\#}\right], \\
\left[\frac{A\,0}{1\,B}\right], \left[\frac{B\,1}{1\,\text{👍}}\right], \left[\frac{0\,A\,1}{B\,0\,1}\right], \left[\frac{1\,A\,1}{B\,1\,1}\right], \left[\frac{0\,B\,0}{A\,0\,1}\right], \left[\frac{1\,B\,0}{A\,1\,1}\right], \\
\left[\frac{0}{0}\right], \left[\frac{1}{1}\right], \left[\frac{\#}{\#}\right], \left[\frac{\#}{0\,\#}\right], \\
\left[\frac{0\,\text{👍}}{\text{👍}}\right], \left[\frac{\text{👍}\,0}{\text{👍}}\right], \left[\frac{1\,\text{👍}}{\text{👍}}\right], \left[\frac{\text{👍}\,1}{\text{👍}}\right], \left[\frac{\text{👍}\,\#\,\#}{\#}\right]
\end{array}\right\}$$

The shortest solution to this instance of the modified Post correspondence problem is shown in (48), split across two lines. The upper/lower string replicates the computation of `BB2` on a tape of all 0s (cf. 31). It begins in state A reading a 0, writes four 1s over five steps, and transitions into the accept state and halts. After a domino with the accept state enters the match, the remaining dominos clean up the structure, gradually allowing the upper string to catch up to the lower string.

(48) *The shortest solution to the domino encoding of* BB2 *that begins with the start state domino*

$$\frac{\#\,0\,0\,A\,0\,\#\,0\,0\,1\,B\,0\,\#\,0\,0\,A\,1\,1\,\#\,0\,B\,0\,1\,1\,\#\,A\,0\,1\,1\,1\,\#\,1\,B\,1\,1\,1\,\#}{\underbrace{\#\,0\,0\,A\,0}_{\text{input}}\,\#\,\underbrace{0\,0\,1\,B\,0}_{\text{step 1}}\,\#\,\underbrace{0\,0\,A\,1\,1}_{\text{step 2}}\,\#\,\underbrace{0\,B\,0\,1\,1}_{\text{step 3}}\,\#\,\underbrace{A\,0\,1\,1\,1}_{\text{step 4}}\,\#\,\underbrace{1\,B\,1\,1\,1}_{\text{step 5}}\,\#}\cdots$$

$$\cdots\frac{1\,1\,\text{👍}\,1\,1\,\#\,1\,1\,\text{👍}\,1\,\#\,1\,\text{👍}\,1\,\#\,\#\,\text{👍}\,1\,\#\,\text{👍}\,\#\,\#}{\underbrace{1\,1\,\text{👍}\,1\,1}_{\text{accept}}\,\underbrace{\#\,1\,1\,\text{👍}\,1\,\#\,1\,\text{👍}\,1\,\#\,\#\,\text{👍}\,1\,\#\,\text{👍}\,\#\,\#}_{\text{clean up}}}$$

Compare the domino encoding of BB2 to that of Tsà (32), the machine that moves endlessly to the right writing 1s (49). Except for the transition function, the sets of dominos are identical. It is clear that there is no solution to this instance of the Post correspondence problem that begins with the start state domino. That domino creates an imbalance of As, adding one to the lower string but none to the upper string. There is no domino with the opposite asymmetry – only the transition function dominos contain the symbol A and they both have one on the top and one on the bottom. Tsà does not halt when run on an input of all 0s and therefore its corresponding domino encoding has no solution.

(49) *The domino encoding of* Tsà

$$\left\{\begin{array}{l}\left[\frac{\#}{\#\,0\,0\,A\,0\,\#}\right],\\ \left[\frac{A\,0}{1\,A}\right],\left[\frac{A\,1}{1\,A}\right],\\ \left[\frac{0}{0}\right],\left[\frac{1}{1}\right],\left[\frac{\#}{\#}\right],\left[\frac{\#}{0\,\#}\right],\\ \left[\frac{0\,\text{👍}}{\text{👍}}\right],\left[\frac{\text{👍}\,0}{\text{👍}}\right],\left[\frac{1\,\text{👍}}{\text{👍}}\right],\left[\frac{\text{👍}\,1}{\text{👍}}\right],\left[\frac{\text{👍}\,\#\,\#}{\#}\right]\end{array}\right\}$$

Because it is possible to translate any Turing machine and input into an instance of the Post correspondence problem, it cannot be computable. If there were an algorithm to solve the Post correspondence problem, you could use it to solve the halting problem simply by generating the domino set representing the machine and input you were interested in.

3.4 Putting things together

This section presented three formal problems – the busy beaver problem, the halting problem, and the Post correspondence problem – and proved that they are not computable. The busy beaver problem is not computable because the number of 1s a Turing machine with *n* states can write before halting grows faster than any computable function. Over the course of a computation to write some number of 1s, a Turing machine must take at least as many steps. Thus, the number of steps a machine with *n* states can take before halting is also not computable. This implies that the halting problem is not computable because given a machine with *n* states, it is impossible to determine a maximum step threshold for halting. Finally, the Post correspondence problem is not computable because some instances of it have solutions if and only if corresponding Turing machines halt on a given input.

For a function to be computable, it must be the case that it can be modeled by a Turing machine that halts on all inputs. This does not imply that it is impossible to classify some inputs. For example, if a given instance of the Post correspondence problem contains only one or two dominos, there is an algorithm to determine whether it is solvable (Ehrenfeucht et al. 1982). However, there is no algorithm that determines whether an arbitrary instance of the problem has a solution. This is exactly because some instances encode the computation of Turing machines.

With this background established, I finally return to the eponymous result of this paper, that Optimality Theory is not computable. The construction in section 2 is simply a phonological encoding of the Post correspondence problem over a binary alphabet: one symbol corresponds to high tones and [.ha.], the other to low tones and [.l̤a.]. Note that any finite set of symbols can be translated into a binary representation. For example, the six symbols used in the domino encoding of `BB2` (47) can be rewritten using strings of three symbols each, as in (50); symbols in the upper string of a domino are replaced with a string of tones and those in the lower string with a segmental string.

(50) *Phonological encoding of symbols used in the domino encoding of* `BB2`

#	L L L	l̤a l̤a l̤a
0	L L H	l̤a l̤a ha
1	L H H	l̤a ha ha
A	H H H	ha ha ha
B	H H L	ha ha l̤a
👍	H L L	ha l̤a l̤a

Translating the domino encoding of `BB2` yields a lexicon similar to that in (51), which I have minimally modified so that the three low tones encoding the # on the upper string of the start state domino are associated to the three /l̤a/s that encode the # on the lower string.

(51) *The phonological encoding of the domino encoding of* `BB2`

{
/l̤à l̤à l̤à l̤a l̤a ha l̤a l̤a ha ha ha ha l̤a l̤a ha l̤a l̤a l̤a/,
/l̤a ha ha ha ha l̤a ⒽⒽⒽⓁⓁⒽ/,
/l̤a ha ha ha l̤a l̤a ⒽⒽⓁⓁⒽⒽ/,
/ha ha l̤a l̤a l̤a ha l̤a ha ha ⓁⓁⒽⒽⒽⒽⓁⒽⒽ/,
/ha ha l̤a l̤a ha ha l̤a ha ha ⓁⒽⒽⒽⒽⒽⓁⒽⒽ/,
/ha ha ha l̤a l̤a ha l̤a ha ha ⓁⓁⒽⒽⒽⓁⓁⓁⒽ/,
/ha ha ha l̤a ha ha l̤a ha ha ⓁⒽⒽⒽⒽⓁⓁⓁⒽ/,
/l̤a l̤a ha ⓁⓁⒽ/,
/l̤a ha ha ⓁⒽⒽ/,
/l̤a l̤a l̤a ⓁⓁⓁ/,
/l̤a l̤a ha l̤a l̤a l̤a ⓁⓁⓁ/,
/ha l̤a l̤a ⓁⓁⒽⒽⓁⓁ/,
/ha l̤a l̤a ⒽⓁⓁⓁⓁⒽ/,
/ha l̤a l̤a ⓁⒽⒽⒽⓁⓁ/,
/ha l̤a l̤a ⒽⓁⓁⓁⒽⒽ/,
/l̤a l̤a l̤a ⒽⓁⓁⓁⓁⓁⓁⓁⓁ/
}

Recall that the grammar in section 2 has the following behavior: when passed in an input that contains toneless syllables but no floating tones, it either returns the input faithfully or finds the shortest phonotactically well-formed candidate by inserting from the lexicon. All well-formed structures are isomorphic to a solution of the PCP. If I pass the morpheme encoding the start state domino /l̤à l̤à l̤à l̤a l̤a ha l̤a l̤a ha ha ha ha l̤a l̤a ha l̤a l̤a l̤a/ into the grammar with this lexicon, the output will be (52), the phonological equivalent of (48).

(52) *The surface form of* /l̤à l̤à l̤à l̤a l̤a ha l̤a l̤a ha ha ha ha l̤a l̤a ha l̤a l̤a l̤a/

[.l̤à.l̤à.l̤à.l̤à.l̤à.há.l̤à.l̤à.há.há.há.há.l̤à.l̤à.há.l̤à.l̤à.l̤à.l̤à.l̤à.há.l̤à.l̤à.há.l̤à.há.há.há.há. . . .
.l̤à.l̤à.l̤à.há.l̤à.l̤à.l̤à.l̤à.l̤à.há.l̤à.l̤à.há.há.há.há.l̤à.há.há.l̤à.há.há.l̤à.l̤à.l̤à.l̤à.l̤à.há.há. . . .
.há.l̤à.l̤à.l̤à.há.l̤à.há.há.l̤à.há.há.l̤à.l̤à.l̤à.há.há.há.l̤à.l̤à.há.l̤à.há.há.l̤à.há.há.l̤à.há. . . .
.há.l̤à.l̤à.l̤à.l̤à.há.há.há.há.l̤à.l̤à.há.há.l̤à.há.há.l̤à.há.há.l̤à.l̤à.l̤à.l̤à.há.há.l̤à.há.há. . . .
.há.l̤à.l̤à.l̤à.há.há.l̤à.há.há.l̤à.l̤à.l̤à.l̤à.há.há.l̤à.há.há.há.l̤à.l̤à.l̤à.há.há.l̤à.l̤à.l̤à.l̤à.há. . . .
.há.há.l̤à.l̤à.l̤à.há.há.l̤à.l̤à.l̤à.l̤à.l̤à.l̤à.há.l̤à.l̤à.l̤à.há.há.l̤à.l̤à.l̤à.há.l̤à.l̤à.l̤à.l̤à.l̤à.l̤à.l̤à.l̤à.]

If I change the lexicon to instead correspond to the domino encoding of `Tsà` (49), then the same input will be returned faithfully because it is impossible to build a phonotactically well-formed structure.

In general then, these lexicons encode instances of the Post correspondence problem, and their behavior with these grammars depends on whether a given instance has a solution. Having established earlier that the Post correspondence problem is not computable, we therefore conclude that OT is not computable. In other words, it is impossible to write an algorithm that takes an arbitrary OT grammar and input and determines its output.

4. Conclusion

This paper demonstrates that Optimality Theory, as used by phonologists, is not computable, joining the likes of many other constraint-based grammatical formalisms (Kaplan and Bresnan 1982, Johnson 1988, Kepser 2004, Francez and Wintner 2012, Kaplan and Wedekind 2023, Przepiórkowski 2023). This means that it is impossible for a computer program to determine the output for an arbitrary grammar. The practical consequences of this result should give us pause as phonologists, casting even more doubt (e.g., Karttunen 2006) on analyses couched in OT. The empirical consequence of this result is that OT sacrifices any claim to being restrictive. While it may not be possible to model opaque interactions directly (Buccola 2013), it is possible instead to encode a rule-based grammar in a set of dominos and model their computation indirectly (see `https://davidlazar.github.io/PCPL/` for a programming language defined over dominos).

Given that the field has to date largely disregarded the fact that OT is more powerful than necessary as a model of natural language phonology, this paper's impact may be minimal. However, it is important to characterize exactly what OT does, given its complexity (Prince 2007). An easy solution would be to adopt directional constraints (Eisner 2000, 2002, Lamont 2022a) in the phonological literature, preserving the benefits of OT (factoring complex problems into simple constraints, typology, acquisition, etc.) in an appropriately restrictive framework. With Guy Emerson, I am also actively pursuing a proof that Harmonic Grammar is finite-state given very weak restrictions, presenting another mainstream alternative that already enjoys ample empirical support. Adopting a finite-state framework makes it possible to translate between rule-based models, automata-based models, finite-state OT/HG, and all other mathematically equivalent theories, thus enabling conversations between superficially distinct models in the past, present, and future. We owe it to the field at large to work towards an underlyingly universal model that can survive the fickle fashions of formal analysis.

References

Akinbo, Samuel. 2023. Reduplication, repetition and sound symbolism in Fungwa. *Phonological Data and Analysis* 5. `https://doi.org/10.3765/pda.v5art2.72`.

Albright, Adam. 2011. Paradigms. In *The Blackwell companion to phonology*, ed. by Marc van Oostendorp, Colin Ewen, Elizabeth Hume, and Keren Rice, 1972–2001. Malden, MA: Wiley-Blackwell.

Alderete, John. 1995. Faithfulness to prosodic heads. Unpublished manuscript, University of Massachusetts Amherst. Available at `http://www.sfu.ca/~alderete/pubs/alderete1995_prosfaith.PDF`.

Baković, Eric, and Colin Wilson. 2000. Transparency, strict locality, and targeted constraints. In *WCCFL 19: Proceedings of the 19th West Coast Conference on Formal Linguistics*, ed. by Roger Billerey and Brook Danielle Lillehaugen, 43–56. Somerville, MA: Cascadilla Press.

Beesley, Kenneth R., and Lauri Karttunen. 2003. *Finite state morphology*. CSLI Studies in Computational Linguistics. Stanford, CA: CSLI Publications.

Behrenfeldt, Johan. 2009. A linguist's survey of pumping lemmata. Master's thesis, University of Gothenburg.

Bennett, Ryan. 2018. Recursive prosodic words in Kaqchikel (Mayan). *Glossa: a journal of general linguistics* 3. `https://doi.org/10.5334/gjgl.550`.

Benua, Laura. 1997. Transderivational identity: Phonological relations between words. Doctoral dissertation, University of Massachusetts Amherst.

Brady, Allen H. 1983. The determination of the value of Rado's noncomputable function $\sigma(k)$ for four-state Turing machines. *Mathematics of Computation* 40:647–665.

Brasoveanu, Adrian, and Alan Prince. 2011. Ranking and necessity: the Fusional Reduction Algorithm. *Natural Language & Linguistic Theory* 29:3–70.

Buccola, Brian. 2013. On the expressivity of Optimality Theory versus ordered rewrite rules. In *Formal Grammar: 17th and 18th International Conferences*, ed. by Glyn Morrill and Mark-Jan Nederhof, 142–158. Berlin, Heidelberg: Springer Berlin Heidelberg.

Burzio, Luigi. 1998. Multiple correspondence. *Lingua* 104:79–109.

Chen, Tsung-Ying. 2010. Some remarks on Contour Tone Units. *Journal of East Asian Linguistics* 19:103–135.

Chomsky, Noam. 1956. Three models for the description of language. *IRE Transactions on Information Theory* 2:113–124.

Chomsky, Noam. 1959. On certain formal properties of grammars. *Information and Control* 2:137–167.

Chomsky, Noam, and Morris Halle. 1965. Some controversial questions in phonological theory. *Journal of Linguistics* 1:97–138.

Church, Alonzo. 1936. An unsolvable problem of elementary number theory. *American Journal of Mathematics* 58:345–363.

Cooper, Adam. 2013. Constraint indexation, locality and epenthesis in Vedic Sanskrit. In *NELS 40: Proceedings of the fortieth annual meeting of the north east lingusitic society*, ed. by Seda Kan, Claire Moore-Cantwell, and Robert Staubs, 119–132. Amherst, MA: Graduate Linguistics Students Association.

Ehrenfeucht, A., J. Karhumäki, and G. Rozenberg. 1982. The (generalized) post correspondence problem with lists consisting of two words is decidable. *Theoretical Computer Science* 21:119–144.

Eisner, Jason. 1997a. Efficient generation in Primitive Optimality Theory. In *Proceedings of the 35th Annual ACL and 8th European ACL*, ed. by Philip R. Cohen and Wolfgang Wahlster, 313–320. Association for Computational Linguistics.

Eisner, Jason. 1997b. What constraints should OT allow? Paper presented at LSA 71. Available at `http://roa.rutgers.edu/article/view/215`.

Eisner, Jason. 2000. Directional constraint evaluation in Optimality Theory. In *Proceedings of the 18th International Conference on Computational Linguistics*, 257–263. The Association for Computational Linguistics.

Eisner, Jason. 2002. Comprehension and compilation in Optimality Theory. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, ed. by

Pierre Isabelle, Eugene Charniak, and Dekang Lin, 56–63. The Association for Computational Linguistics.

Elfner, Emily, and Wendell Kimper. 2008. Reduplication without RED: Evidence from *diddly*-infixation. In *Proceedings of the 27th West Coast Conference on Formal Linguistics*, ed. by Natasha Abner and Jason Bishop, 150–158. Somerville, MA: Cascadilla Proceedings Project.

Ellison, T. Mark. 1994. Phonological derivation in Optimality Theory. In *COLING 1994 Volume 2: The 15th International Conference on Computational Linguistics*, 1007–1013. New Brunswick, NJ: Association for Computational Linguistics.

Fitzgerald, Colleen M. 2000. Vowel hiatus and faithfulness in Tohono O'odham reduplication. *Linguistic Inquiry* 31:713–722.

Francez, Nissim, and Shuly Wintner. 2012. *Unification grammars*. Cambridge: Cambridge University Press.

Frank, Robert, and Giorgia Satta. 1998. Optimality Theory and the generative complexity of constraint violability. *Computational Linguistics* 24:307–315.

Gafos, Diamandis. 1998. Eliminating long-distance consonantal spreading. *Natural Language & Linguistic Theory* 16:223–278.

Gerdemann, Dale, and Mans Hulden. 2012. Practical finite state Optimality Theory. In *Proceedings of the 10th International Workshop on Finite State Methods and Natural Language Processing*, ed. by Iñaki Alegria and Mans Hulden, 10–19. The Association for Computational Linguistics.

Gleim, Daniel. 2019. A feeding Duke-of-York interaction of tone and epenthesis in Arapaho. *Glossa: a journal of general linguistics* 4:97.

Gourley, René. 1995. Harmony networks do not work. In *Advances in Neural Information Processing Systems 8*, ed. by D. Touretzky, M.C. Mozer, and M. Hasselmo, 31–37. Cambridge, MA: MIT Press.

Hale, John, and Paul Smolensky. 2006. Harmonic grammars and harmonic parsers for formal languages. In *The harmonic mind: From neural completeness to Optimality-Theoretic grammar*, ed. by Paul Smolensky and Géraldine Legendre, volume I, 393–415. Cambridge, MA: MIT Press.

Hampe, Anton. 2022. The generative capacity of Harmonic Serialism. Bachelor's thesis, Universität Leipzig.

Hampe, Anton. 2024. Simulating circular Post machines with directional Harmonic Serialism. Unpublised manuscript. Universität Leipzig.

Hao, Sophie. 2024. Universal generation for Optimality Theory is PSPACE-Complete. *Computational Linguistics* 50:83–117.

Harrison, Michael A. 1978. *Introduction to formal language theory*. Addison-Wesley Series in Computer Science. Reading, MA and Menlo Park, CA and London and Amsterdam and Don Mills, Ontario and Sydney: Addison-Wesley Publishing Company.

Haugen, Jason D. 2009. What is the base for reduplication? *Linguistic Inquiry* 40:505–514.

Heinz, Jeffrey. 2018. The computational nature of phonological generalizations. In *Phonological typology*, ed. by Larry M. Hyman and Frans Plank, 126–195. Berlin: De Gruyter Mouton.

Heinz, Jeffrey, Gregory M. Kobele, and Jason Riggle. 2009. Evaluating the complexity of Optimality Theory. *Linguistic Inquiry* 40:277–288.

Hopcroft, John E. 1984. Turing machines. *Scientific American* 250:86–98.

Hopcroft, John E., Rajeev Motwani, and Jeffrey D. Ullman. 2008. *Introduction to automata theory, languages, and computation*. Pearson, 3 edition.

Hou, Jinyi. 1983. Changzhi fangyan jilu [Notes on the Changzhi dialect]. *Fangyan* 1983:260–274.

Hulden, Mans. 2017. Formal and computational verification of phonological analyses. *Phonology* 34:407–435.

Idsardi, William J. 2006. A simple proof that Optimality Theory is computationally intractable. *Linguistic Inquiry* 37:271–275.

Itô, Junko, and Armin Mester. 2007. Prosodic adjunction in Japanese compounds. In *Formal approaches to Japanese linguistics (FAJL 4)*, ed. by Yoichi Miyamoto and Masao Ochi, 97–111. Cambridge, MA: MIT Working Papers in Linguistics.

Jardine, Adam. 2016. Locality and non-linear representations in tonal phonology. Doctoral dissertation, University of Delaware.

Johnson, C. Douglas. 1972. *Formal aspects of phonological description*. The Hague: Mouton.

Johnson, Mark. 1988. *Attribute-value logic and the theory of grammar*, volume 16 of *CSLI Lecture Notes*. Stanford, CA: Center for the Study of Language and Information.

Kaplan, Ronald M., and Joan Bresnan. 1982. Lexical-Functional Grammar: A formal system for grammatical representation. In *The mental representation of grammatical representation*, ed. by Joan Bresnan, 173–281. Cambridge, MA: The MIT Press.

Kaplan, Ronald M., and Martin Kay. 1994. Regular models of phonological rule systems. *Computational Linguistics* 20:331–378.

Kaplan, Ronald M., and Jürgen Wedekind. 2023. Formal and computational properties of LFG. In *Handbook of Lexical Functional Grammar*, ed. by Mary Dalrymple, volume 13 of *Empirically Oriented Theoretical Morphology and Syntax*, 1035–1082. Berlin: Language Science Press.

Karttunen, Lauri. 1998. The proper treatment of optimality in computational phonology. In *Proceedings of the International Workshop on Finite State Methods in Natural Language Processing*, ed. by Lauri Karttunen and Kemal Oflazer, 1–12. Ankara, Turkey: Bilkent University.

Karttunen, Lauri. 2006. The insufficiency of paper-and-pencil linguistics: the case of Finnish prosody. In *Intelligent linguistic architectures: Variations on themes by Ronald M. Kaplan*, ed. by Miriam Butt, Mary Dalrymple, and Tracy Holloway King, volume 179 of *CSLI Lecture Notes*, 287–300. Stanford, CA: CSLI Publications.

Kawahara, Shigeto. 2004. Locality in echo epenthesis: Comparison with reduplication. In *Proceedings of NELS 34*, ed. by Keir Moulton and Matthew Wolf, volume 2, 295–310. Amherst, MA: Graduate Linguistics Students Association.

Kawu, Ahmadu Ndanusa. 2000. Structural markedness and nonreduplicative copying. In *Proceedings of NELS 30*, ed. by Masako Hirotani, Andries Coetzee, Nancy Hall, and Ji-yung Kim, volume 1, 377–388. Amherst, MA: Graduate Linguistics Students Association.

Kepser, Stephan. 2004. On the complexity of RSRL. *Electronic Notes in Theoretical Computer Science* 53:146–162.

Kleene, Stephen C. 1936. λ-definability and recursiveness. *Duke Mathematical Journal* 2:340–353.

Koskenniemi, Kimmo. 1983. *Two-level morphology: A general computational model for word-form recognition and production*, volume 11 of *Publications*. Helsinki: University of Helsinki.

Kropitz, Pavel. 2010. Problém busy beaver. Bachelor's thesis, Univerzita Karlova v Praze.

Kurisu, Kazutaka. 2001. The phonology of morpheme realization. Doctoral dissertation, University of California, Santa Cruz.

Lamont, Andrew. 2021. Optimizing over subsequences generates context-sensitive languages. *Transactions of the Association for Computational Linguistics* 9:528–537.

Lamont, Andrew. 2022a. Directional Harmonic Serialism. Doctoral dissertation, University of Massachusetts Amherst.

Lamont, Andrew. 2022b. Optimality theory implements complex functions with simple constraints. *Phonology* 38:729–740.

Lamont, Andrew. 2024. Myopic spreading with weighted constraints and gradient activity. Talk at Myopia in Grammar. Available at `https://aphonologist.github.io/presentations`.

Leben, William R. 1973. Suprasegmental phonology. Doctoral dissertation, Massachusetts Institute of Technology.

Lee, Seunghun Julio. 2008. Consonant-tone interaction in Optimality Theory. Doctoral dissertation, Rutgers, The State University of New Jersey.

Legendre, Géraldine, Yoshiro Miyata, and Paul Smolensky. 1990a. Harmonic Grammar: A formal multi-level connectionist theory of linguistic well-formedness: An application. In *The Twelfth Annual Conference of the Cognitive Science Society*, 884–891. Hillsdale, NJ: Lawrence Erlbaum Associates, Publishers.

Legendre, Géraldine, Yoshiro Miyata, and Paul Smolensky. 1990b. Harmonic Grammar: A formal multi-level connectionist theory of linguistic well-formedness: Theoretical foundations. In *The Twelfth Annual Conference of the Cognitive Science Society*, 388–395. Hillsdale, NJ: Lawrence Erlbaum Associates, Publishers.

Lin, Shen, and Tibor Radó. 1965. Computer studies of Turing machine problems. *Journal of the Association for Computing Machinery* 12:196–212.

Lorentz, Richard J. 2001. Creating difficult instances of the Post Correspondence Problem. In *Computers and games*, ed. by Tony Marsland and Ian Frank, volume 2063 of *Lecture Notes in Computer Science*, 214–228. Berlin: Springer.

Łubowicz, Anna. 2002. Derived environment effects in Optimality Theory. *Lingua* 112:243–280.

Łubowicz, Anna. 2003. Local conjunction and comparative markedness. *Theoretical Linguistics* 29:101–112.

Łubowicz, Anna. 2005. Locality of conjunction. In *Proceedings of the 24th West Coast Conference on Formal Linguistics*, ed. by John Alderete, Chung hye Han, and Alexei Kochetov, 254–262. Somerville, MA: Cascadilla Press.

Marantz, Alec. 1982. Re reduplication. *Linguistic Inquiry* 13:435–482.

Marxen, Heiner, and Jürgen Buntrock. 1990. Attacking the busy beaver 5. *Bulletin of the European Association for Theoretical Computer Science* 40:247–251.

McCarthy, John J. 1986. OCP effects: Gemination and antigemination. *Linguistic Inquiry* 17:207–263.

McCarthy, John J. 2000. Harmonic Serialism and Parallelism. In *Proceedings of NELS 30*, ed. by Masako Hirotani, Andries Coetzee, Nancy Hall, and Ji-yung Kim, volume 2, 501–524. Amherst, MA: Graduate Linguistics Students Association.

McCarthy, John J. 2002. Comparative markedness. In *Papers in Optimality Theory II*, ed. by Angela Carpenter, Andries Coetzee, and Paul de Lacy, 171–246. Amherst, MA: Graduate Linguistics Students Association.

McCarthy, John J. 2003a. Comparative markedness. *Theoretical Linguistics* 29:1–51.

McCarthy, John J. 2003b. OT constraints are categorical. *Phonology* 20:75–138.

McCarthy, John J. 2006. Restraint of analysis. In *Wondering at the natural fecundity of things: Essays in honor of Alan Prince*, ed. by Eric Baković, Junko Itô, and John J. McCarthy, 195–219. Santa Cruz, CA: Linguistics Research Center.

McCarthy, John J. 2007. Restraint of analysis. In *Freedom of analysis?*, ed. by Sylvia Blaho, Patrik Bye, and Martin Krämer, volume 95 of *Studies in Generative Grammar*, 203–231. Berlin and New York: Mouton de Gruyter.

McCarthy, John J. 2016. The theory and practice of Harmonic Serialism. In *Harmonic Grammar and Harmonic Serialism*, ed. by John J. McCarthy and Joe Pater, 47–87. Sheffield: Equinox Publishing.

McCarthy, John J., and Alan Prince. 1994. The emergence of the unmarked: Optimality in prosodic morphology. In *Proceedings of NELS 24*, ed. by Mercè Gonzàlez, volume 2, 333–379. Amherst, MA: Graduate Linguistics Students Association.

McCarthy, John J., and Alan Prince. 1995. Faithfulness and reduplicative identity. In *Papers in Optimality Theory*, ed. by Jill Beckman, Suzanne Urbanczyk, and Laura Walsh Dickey, 249–384. Amherst, MA: Graduate Linguistics Students Association.

McCarthy, John J., and Alan Prince. 1999. Faithfulness and identity in prosodic morphology. In *The prosody-morphology interface*, ed. by René Kager, Harry van der Hulst, and Wim Zonneveld, 218–309. Cambridge: Cambridge University Press.

McPherson, Laura. 2020. *A grammar of Seenku*, volume 83 of *Mouton Grammar Library*. Berlin/Boston: De Gruyter Mouton.

McPherson, Laura, and Matthew S. Dryer. 2021. The tone system of Poko-Rawo (Skou). *Phonological Data & Analysis* 3. `https://doi.org/10.3765/pda.v3art1.54`.

Michel, Pascal. 2022. The Busy Beaver Competition: a historical survey. Unpublished manuscript, Université Paris Diderot and Université de Cergy-Pontoise. Available at `https://doi.org/10.48550/arXiv.0906.3749`.

Moreton, Elliott, and Paul Smolensky. 2002. Typological consequences of local constraint conjunction. In *Proceedings of WCCFL 21*, ed. by Line Mikkelsen and Christopher Potts, 306–319. Somerville, MA: Cascadilla Press.

O'Hara, Charlie. 2016. Harmony in Harmonic Grammar by reevaluating faithfulness. In *Proceedings of NELS 46*, ed. by Christopher Hammerly and Brandon Prickett, volume 3, 71–84. Amherst, MA: Graduate Linguistics Students Association.

Okabe, Masataka, and Kei Ito. 2008. Color Universal Design (CUD) - how to make figures and presentations that are friendly to Colorblind people. `https://jfly.uni-koeln.de/color/#select`.

Partee, Barbara H., Alice ter Meulen, and Robert E. Wall. 1990. *Mathematical methods in linguistics*, volume 30 of *Studies in Linguistics and Philosophy*. Dordrecht / Boston / London: Kluwer Academic Publishers.

Patal Majzul, Lolmay Filiberto. 2007. *Rusoltzij ri Kaqchikel: Diccionario estándar bilingüe Kaqchikel-Español*. Ciudad de Guatemala, Guatemala: Cholsamaj.

Pater, Joe. 2007. The locus of exceptionality: Morpheme-specific phonology as constraint indexation. In *Papers in Optimality Theory III*, ed. by Leah Bateman, Michael O'Keefe, Ehren Reilly, and Adam Werle, University of Massachusetts Occasional Papers, 259–296. Amherst, MA: Graduate Linguistics Students Association.

Pater, Joe. 2009a. Morpheme-specific phonology: Constraint indexation and inconsistency resolution. In *Phonological argumentation: Essays on evidence and motivation*, ed. by Steve Parker, 123–154. London: Equinox Publishing.

Pater, Joe. 2009b. Weighted constraints in generative linguistics. *Cognitive Science* 33:999–1035.

Pater, Joe. 2016. Universal grammar with weighted constraints. In *Harmonic Grammar and Harmonic Serialism*, ed. by John J. McCarthy and Joe Pater, 1–46. Sheffield: Equinox Publishing.

Post, Emil L. 1946. A variant of a recursively unsolvable problem. *Bulletin of the American Mathematical Society* 52:264–268.

Potts, Christopher, Joe Pater, Karen Jesney, Rajesh Bhatt, and Michael Becker. 2010. Harmonic Grammar with linear programming: from linear systems to linguistic typology. *Phonology* 27:77–117.

Potts, Christopher, and Geoffrey K. Pullum. 2002. Model theory and the content of OT constraints. *Phonology* 19:361–393.

Prince, Alan. 2002. Arguing optimality. In *Papers in Optimality Theory II*, ed. by Angela Carpenter, Andries Coetzee, and Paul de Lacy, 269–304. Amherst, MA: Graduate Linguistics Students Association.

Prince, Alan. 2007. The pursuit of theory. In *The Cambridge handbook of phonology*, ed. by Paul de Lacy, 33–60. Cambridge: Cambridge University Press.

Prince, Alan, and Paul Smolensky. 1993/2004. *Optimality Theory: Constraint interaction in generative grammar*. Malden, MA: Blackwell Publishing.

Przepiórkowski, Adam. 2023. LFG and Head-Driven Phrase Structure Grammar. In *Handbook of Lexical Functional Grammar*, ed. by Mary Dalrymple, volume 13 of *Empirically Oriented Theoretical Morphology and Syntax*, 1861–1918. Berlin: Language Science Press.

Radó, Tibor. 1962. On non-computable functions. *Bell System Technical Journal* 41:877–884.

Riggle, Jason. 2004. Generation, recognition, and learning in finite state Optimality Theory. Doctoral dissertation, University of California Los Angeles.

Rogers, James, and Geoffrey K. Pullum. 2011. Aural pattern recognition experiments and the subregular hierarchy. *Journal of Logic, Language and Information* 20:329–342.

Rolle, Nicholas. 2020. In support of an OT-DM model: Evidence from clitic distribution in Degema serial verb constructions. *Natural Language & Linguistic Theory* 38:201–259.

Rolle, Nicholas. to appear. A tonological rarity: A tone-driven epenthesis in Ghomala'. In *Rarities in phonetics and phonology: Structural, typological, evolutionary, and social dimensions*, ed. by Natalia Kuznetsova, Cormac Anderson, and Shelece Easterday. Berlin: Language Science Press.

Rolle, Nicholas, and John T. M. Merrill. 2022. Tone-driven epenthesis in Wamey. *Phonology* 39:113–158.

Selkirk, Elisabeth. 1995. The prosodic structure of function words. In *Papers in Optimality Theory*, ed. by Jill Beckman, Laura Walsh Dickey, and Suzanne Urbanczyk, 439–470. Amherst, MA: Graduate Linguistics Students Association.

Selkirk, Elisabeth. 1996. The prosodic structure of function words. In *Signal to syntax: Bootstrapping from speech to grammar in early acquisition*, ed. by James L. Morgan and Katherine Demuth, 187–213. Mahwah, NJ: Lawrence Erlbaum Associates, Publishers.

Selkirk, Elisabeth. 2011. The syntax-phonology interface. In *The handbook of phonological theory*, ed. by John A. Goldsmith, Jason Riggle, and Alan C. L. Yu, 435–484. Oxford: Blackwell Publishing, 2 edition.

Shaw, Patricia A. 2005. Non-adjacency in reduplication. In *Studies on reduplication*, ed. by Bernhard Hurch, Empirical Approaches to Language Typology, 161–210. Berlin and New York: Mouton de Gruyter.

Sipser, Michael. 2013. *Introduction to the theory of computation*. Boston, MA: Cengage Learning, 3 edition.

Sloos, Marjoleine, and Aone van Engelenhoven. 2011. Input-Reduplicant correspondence in Leti. In *Linguistics in the Netherlands 2011*, ed. by Rick Nouwen and Marion Elenbaas, volume 28 of *AVT Publications*, 112–124. Amsterdam: John Benjamins Publishing Company.

Smolensky, Paul. 1992. Harmonic Grammars for formal languages. In *Advances in Neural Information Processing Systems 5*, ed. by Stephen José Hanson, Jack D. Cowan, and C. Lee Giles, 847–854. San Francisco: Morgan Kaufmann Publishers Inc.

Smolensky, Paul. 1993. Harmony, markedness, and phonological activity. Paper presented at Rutgers Optimality Workshop 1. Available at `http://roa.rutgers.edu/article/view/88`.

Smolensky, Paul. 2006a. Harmony in linguistic cognition. *Cognitive Science* 30:779–801.

Smolensky, Paul. 2006b. Optimality in phonology II: Harmonic completeness, local constraint conjunction, and feature-domain markedness. In *The harmonic mind: From neural completeness to Optimality-Theoretic grammar*, ed. by Paul Smolensky and Géraldine Legendre, volume II, 27–160. Cambridge, MA: MIT Press.

Stemberger, Joseph Paul. 1996. The scope of the theory: Where does "beyond" lie? In *CLS 32: Papers from the Parasession on Theory and Data in Linguistics*, ed. by Audra Dainora, Rachel Hemphill, Barbara Luka, Barbara Need, and Sheri Pargman, 139–164. Chicago, IL: Chicago Linguistics Society.

Turing, A. M. 1937. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society* s2-42:230–265.

Walker, Rachel, and Bella Feng. 2004. A ternary model of morphology-phonology correspondence. In *Proceedings of WCCFL 23*, ed. by Vineeta Chand, Ann Kelleher, Angelo J. Rodríguez, and Benjamin Schmeiser, 787–800. Somerville, MA: Cascadilla Press.

Wang, Yang, and Tim Hunter. 2023. On regular copying languages. *Journal of Language Modelling* 11:1–66.

Wilson, Colin. 2000. Targeted constraints: An approach to contextual neutralization in Optimality Theory. Doctoral dissertation, The Johns Hopkins University.

Wilson, Colin. 2001. Consonant cluster neutralisation and targeted constraints. *Phonology* 18:147–197.

Wilson, Colin. 2003. Analyzing unbounded spreading with constraints: marks, targets and derivations. Unpublished manuscript. UCLA.

Wilson, Colin. 2013. A targeted spreading imperative for nasal place assimilation. In *Proceedings of NELS 41*, ed. by Yelena Fainleib, Nicholas LaCara, and Yangsook Park, volume 2, 261–272. Amherst, MA: GLSA.

Wolf, Matthew. 2008. Optimal interleaving: Serial phonology-morphology interaction in a constraint-based model. Doctoral dissertation, University of Massachusetts Amherst.

Wolf, Matthew. 2015. Lexical insertion occurs in the phonological component. In *Understanding allomorphy: Perspectives from optimality theory*, ed. by Eulàlia Bonet, Maria-Rosa Lloret, and Joan Mascaró Altimiras, 361–407. Sheffield: Equinox.

Xu, Zheng. 2007. Inflectional morphology in Optimality Theory. Doctoral dissertation, Stony Brook University.

Xu, Zheng. 2011. Optimality Theory and morphology. *Language and Linguistics Compass* 5:424–508.

Yip, Moira. 2002. *Tone*. Cambridge Textbooks in Linguistics. Cambridge: Cambridge University Press.

Yu, Alan C. L. 2003. Reduplication in English Homeric infixation. In *Proceedings of NELS 34*, ed. by Keir Moulton and Matthew Wolf, volume 2, 619–633. Amherst, MA: Graduate Linguistics Students Association.

Yu, Alan C. L. 2005. Toward a typology of compensatory reduplication. In *Proceedings of the 24th West Coast Conference on Formal Linguistics*, ed. by John Alderete, Chung-hye Han, and Alexei Kochetov, 397–405. Somerville, MA: Cascadilla Proceedings Project.

Yu, Alan C. L. 2007. *A natural history of infixation*. Oxford Studies in Theoretical Linguistics. Oxford: Oxford University Press.

Zhao, Ling. 2002. Solving and creating difficult instances of Post's correspondence problem. Master's thesis, University of Alberta.

Andrew Lamont
`andrew.lamont@ucl.ac.uk`

1. Appendix: Harmonic Grammar with positive and negative constraints

Smolensky (1992) (see also Hale and Smolensky 2006 and Smolensky 2006a) demonstrates that for any formal language, it is possible to construct a Harmonic Grammar (HG; Legendre et al. 1990a,b) such that members of the language have harmony $\mathscr{H} \geq 0$ and non-members have harmony $\mathscr{H} < 0$ (see Gourley 1995 for critical discussion). This result is clearly related to the main result of the paper, but it deviates from the standard assumption that constraints in HG can only subtract from harmony (Potts et al. 2010, Pater 2009b, 2016). I replicate Smolensky's construction below, but I strongly suspect that it cannot be reproduced with constraints that only subtract from harmony. As such, its relevance for phonologists is unclear.

A context-free grammar (CFG; Chomsky 1956) is one composed of rules of the shape $A \rightarrow B\ C$, where A, B, and C are non-terminals, and $A \rightarrow a$, where a is a terminal. All CFGs can be written in this binary branching format, or *Chomsky Normal Form* (Chomsky 1959). An example CFG is given in (53); it generates strings like *the cats walk the dogs* and *dogs sleep*. A CFG's formal language is the set of strings it generates.

(53) *A context-free grammar in Chomsky Normal Form*

S	→	NP VP
NP	→	D N | cats | dogs
VP	→	V NP | walk | sleep
D	→	the
N	→	cats | dogs
V	→	love | tolerate

Smolensky (1992) defines a Harmonic Normal Form for CFGs which imposes an additional restriction that branching rules are unique. In other words, given that *NP* can expand to *D N*, there cannot be any other branching rules with NP on the left hand side. Converting a CFG into Harmonic Normal Form merely requires labeling and enumerating the possible expansions so that every left hand side is unique. The grammar in (54) translates the grammar in (53) into Harmonic Normal Form.

(54) *A context-free grammar in Harmonic Normal Form*

S	→	C[1]	C[5]	→	NP[2] VP[2]
S	→	C[2]	C[6]	→	NP[2] VP[3]
S	→	C[3]	NP[1]	→	D N
S	→	C[4]	NP[2]	→	cats | dogs
S	→	C[5]	VP[1]	→	V NP[1]
S	→	C[6]	VP[2]	→	V NP[2]
C[1]	→	NP[1] VP[1]	VP[3]	→	walk | sleep
C[2]	→	NP[1] VP[2]	D	→	the
C[3]	→	NP[1] VP[3]	N	→	cats | dogs
C[4]	→	NP[2] VP[1]	V	→	love | tolerate

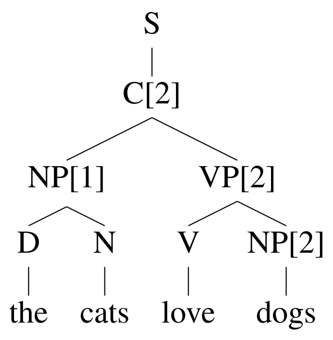
Given a grammar in Harmonic Normal Form, Smolensky provides an algorithm to construct an HG such that members of the language have harmony $\mathscr{H} \geq 0$ and all other strings have harmony $\mathscr{H} < 0$. The algorithm for doing so is given in (55).

(55) *The Harmonic Grammar (Smolensky 1992:850)*

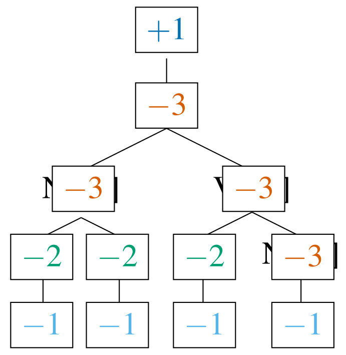
a	If a (e.g., cats) is at any node, add -1 to $\mathscr{H}$
A	If A (e.g., N) is at any node, add -2 to $\mathscr{H}$
$A[i]$	If $A[i]$ (e.g., C[1]) is at any node, add -3 to $\mathscr{H}$
S	If S is at the root, add $+1$ to $\mathscr{H}$
$A \rightarrow a/A[i]$	If $a/A[i]$ is a left child of A (e.g., D $\rightarrow$ the), add $+2$ to $\mathscr{H}$
$A[i] \rightarrow B\ C$	If B is a left child of $A[i]$ (e.g., C[1] $\rightarrow$ $\underline{\text{NP[1]}}$ VP[1]), add $+2$ to $\mathscr{H}$
	If C is a right child of $A[i]$ (e.g., C[1] $\rightarrow$ NP[1] $\underline{\text{VP[1]}}$), add $+2$ to $\mathscr{H}$

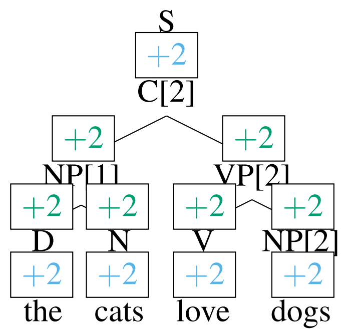
Seven types of constraints are built that either add to or subtract from the harmony. Constraints of the type a, A, and $A[i]$ penalize nodes in the parse tree, and rules of the type S, $A \rightarrow a$, $A \rightarrow A[i]$, $A[i] \rightarrow B\ C$ reward node expansions. For example, every terminal node like *cats*, *dogs*, or *sleep* subtracts 1 from the harmony, while every rule that produces a terminal node like $D \rightarrow$ *the* adds 2 to the harmony.

To illustrate, consider the sentence in (56), which is generated by the grammar.

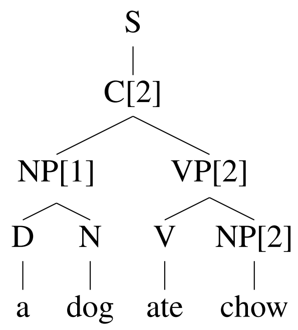
(56)


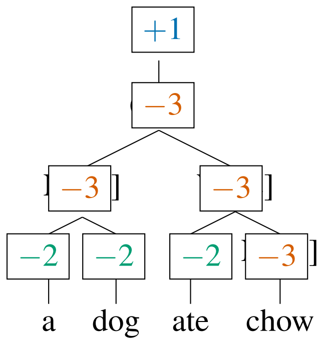
The a rules contribute -4 to the harmony, the A rules -6, the $A[i]$ rules -12, and the S rule $+1$, as in (57). The $A \rightarrow \alpha$ rules add $+10$ to the harmony and the $A[i] \rightarrow B\ C$ rules add another $+12$, as in (58) resulting in a total harmony of $+1$. Because this is at least 0, the grammar produces *the cats love dogs*.

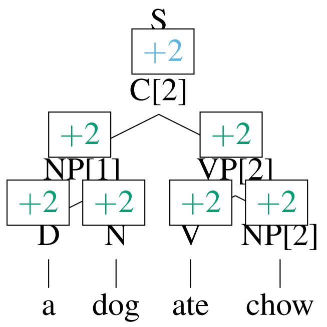
(57)


(58)


The string *a dog ate chow* is not generated by the grammar; but a plausible parse tree is given in (59). Its harmony score is −3, below the threshold of membership. The a rules contribute nothing because none of the terminals belong to the grammar, the A rules −6, the $A[i]$ rules −12, and the S rule +1 60. The $A \rightarrow \alpha$ rules add +2 and the $A[i] \rightarrow B\ C$ add +12 (61), but this does not cancel out the negative contributions.

(59)


(60)


(61)


Smolensky (1992:852) argues that there is a similar construction for the languages defined by Turing machines. This immediately introduces non-computable properties, because asking whether a given string can be assigned a parse tree with harmony $\mathscr{H} \geq 0$ can be easily translated into whether a given Turing Machine halts on that string as input.

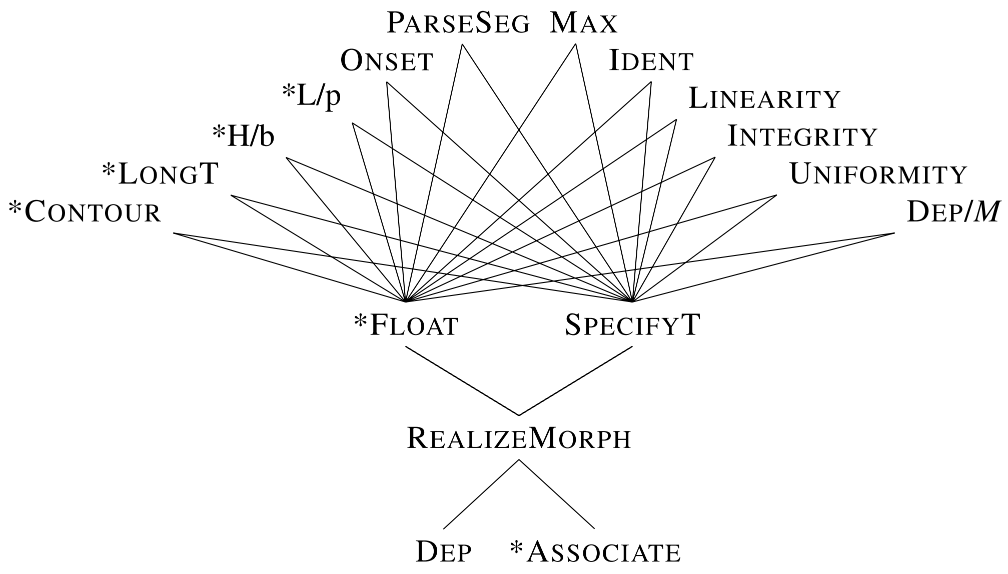
These constructions work by balancing out the rewards gained from expanding non-terminal nodes with the costs of producing terminals. This admits harmonies that may be above or below 0. If constraints are only able to subtract from harmony, this construction would fail. Furthermore, there is a well-known property of HG that finite structures can only motivate finite processes (O'Hara 2016, Lamont 2024). In a CFG, the presence of the root node S should only be able to produce trees up to a given depth, and in the Turing machine construction, only computations up to a certain length should be possible. The limitations imposed by assuming a phonologically grounded HG thus lead me to doubt that this construction applies to single signed constraints. I aim to derive this formally in the near future.

2. Appendix: A reduplicative alternative

The construction in section 2 depends on GEN being able to insert freely from the lexicon, which may be controversial. To avoid addressing this question, this appendix presents an alternative construction that depends only on reduplication.

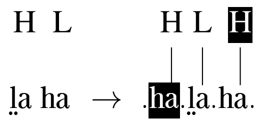
The constraint ranking (62) is very similar to the other grammar's. It imposes the same phonotactic restrictions on outputs, but enforces them using a different mechanism. I discuss the differences below. All constraints not defined in this appendix are defined in section 2.

(62) *Partial constraint ranking*



By removing (or demoting to an inactive position in the ranking) MAX-PM, this grammar is free to create phonotactically well-formed strings with phonological epenthesis. This is illustrated in the tableau in (63) cf. (25). The faithful candidate (63a) violates the binary constraints *FLOAT and SPECIFYT and loses to a candidate that satisfies these constraints by inserting a high tone and the string [ha] (63b). Exactly what tonal melody or segmental string is inserted is irrelevant to the construction, and with the constraints in (62), there are many more optimal outputs such as [.ʔə́.l̤à.hā.].

(63)



H L l̤a ha	*FLOAT	SPECIFYT	DEP	*ASSOCIATE
a. H L .l̤a.ha.	W 1	W 1	L	L
☞ b. H L H .ha.l̤a.ha.			3	3

However, epenthesis is not possible in all contexts; the lexically indexed (Pater 2007, 2009a) constraint DEP/*M* (64) penalizes epenthetic segments and tones that surface within any domain bearing the index *M* (see Alderete 1995 for formally similar constraints on epenthesis in stressed syllables and Cooper 2013 for discussion of the opposite situation, where epenthesis is only tolerated in given morphological contexts).

(64) DEP/*M*

Assign one violation to every segment/tone in the output without a correspondent in the input that surfaces within a domain bearing the index *M*.

Following Marantz (1982), I treat reduplicative morphemes as underlying prosodic templates, and introduce a morpheme indexed *M* that is underlyingly a phonological phrase /φ_M/. This morpheme does not contain any tones or segments, and violates REALIZE-MORPH (65) when it surfaces faithfully (alternately, constraints on headedness may be invoked; Selkirk 1995, 1996).

(65) REALIZEMORPH (simplifying Kurisu 2001:39)

Assign one violation to every morpheme in the input without an overt phonological realization in the output.

Ranked below *FLOAT and SPECIFYT, REALIZEMORPH can only motivate reduplication when it would result in a phonotactically well-formed output. To ground this empirically, non-reduplicative copying is well-attested (Gafos 1998, Kawu 2000, Yu 2003, 2005, 2007, Kawahara 2004, Elfner and Kimper 2008) as is iterative copying (Akinbo 2023). Yoruba (yor; Niger-Congo) presents an instance of tone-driven segmental copying, where a consonant is copied to avoid a high-toned syllable without an onset, as in /í-só/ → [.sí.só.]

'farting' (Kawu 2000:378). The clearest case of segmental-driven tone copying I am aware of comes from the Changzhi dialect of Mandarin (`cmn`; Sino-Tibetan), where the adjectival suffix copies the tone of the root if it is a contour tone as in /swã̀ŋ-tı̃́/ → [.swã̀ŋ.tı̃̀.] 'sour-ADJ' (Hou 1983 *apud* Chen 2010:124); the diacritics represent a high-mid-high contour and a mid-low-mid contour.

The tableau in (66) illustrates the effect of REALIZEMORPH. The faithful candidate (66a) violates all three phonotactic constraints: its tones are floating, its syllables are toneless, and the phonological phrase (represented with angled brackets ⟨ ⟩) does not contain any tones or segments. Associating the tonal melody to the segmental string satisfies *FLOAT and SPECIFYT, and copying both into the phonological phrase satisfies REALIZEMORPH (66b).

(66)
$$\begin{array}{ccc} \text{L H} & & \text{L H} \quad \text{L H} \\ & & |\;\;\; | \quad\;\; |\;\;\; | \\ \text{l̤a ha } \varphi_M & \rightarrow & \text{.l̤a.ha. } \langle\text{.l̤a.ha.}\rangle_M \end{array}$$

$\begin{array}{c}\text{L H}\\ \text{l̤a ha } \varphi_M\end{array}$	*FLOAT	SPECIFYT	REALIZEMORPH	*ASSOCIATE
a. $\begin{array}{c}\text{L H}\\ \text{l̤a ha } \langle\rangle_M\end{array}$	W 1	W 1	W 1	L
☞ b. $\begin{array}{c}\text{L H} \quad \text{L H}\\ |\;\;\; | \quad\;\; |\;\;\; |\\ \text{.l̤a.ha. } \langle\text{.l̤a.ha.}\rangle_M\end{array}$				4

When it is impossible to create a phonotactically well-formed structure by copying from the input, the phonological phrase is not overtly realized. The input to the tableau in (67) is identical to that in (66), except that its tonal melody has been reversed. As in (63), phonotactic well-formedness is achieved outside the phonological phrase by inserting tones and segments (67b). However, DEP/*M* blocks insertion into the phonological phrase (67e), and the phonological phrase goes unrealized. Copying the segmental string (67c) or the tonal string (67d) would satisfy REALIZEMORPH, but violate *FLOAT or SPECIFYT, respectively, and there is no way to build a well-formed output by making more copies.

(67) H L l̤a ha φ_M $\rightarrow$ H L H .ha.l̤a.ha. ⟨⟩$_M$

H L l̤a ha φ_M	DEP/*M*	*FLOAT	SPECIFYT	REALIZEMORPH	DEP	*ASSOCIATE
a. H L l̤a ha ⟨⟩$_M$		W 1	W 1	1	L	L
☞ b. H L H .ha.l̤a.ha. ⟨⟩$_M$				1	3	3
c. H L H .ha.l̤a.ha. ⟨.l̤a.ha.⟩$_M$			W 1	L	3	3
d. H L H H L .ha.l̤a.ha. ⟨ ⟩$_M$		W 1		L	3	3
e. H L H L H .ha.l̤a.ha. ⟨.l̤a.ha.⟩$_M$	W 2			L	W 5	W 5

A handful of additional constraints are needed to guarantee fully faithful copying. First, as in the previous construction, consider only inputs defined over whole morphemes. One option is to duplicate all the faithfulness constraints defined over the input-output dimension and define the copies over the input-reduplicant dimension (McCarthy and Prince 1999, Fitzgerald 2000, Sloos and van Engelenhoven 2011). Another is to require that the base of reduplication is a whole morpheme (Shaw 2005, Haugen 2009), and duplicate the faithfulness constraints over the base-reduplicant dimension. Either approach restricts the phonological phrase to copying entire morphemes, where the available set consists of those that happen to be passed in as input, rather than the entire lexicon.

This grammar has a very similar behavior to the lexical insertion grammar. Given an input containing some number of morphemes consisting of floating tones and segmental strings as well as /φ_M/, the grammar either (a) returns an output where the phonological phrase is empty or (b) returns an output where the phonological phrase contains the smallest phonotactically well-formed structure that can be built by copying morphemes from the input. Like the preceding construction, this behavior crucially depends on whether such a structure exists. The following section demonstrates why this cannot be computed.

3. Appendix: A Turing machine that squares unary numbers

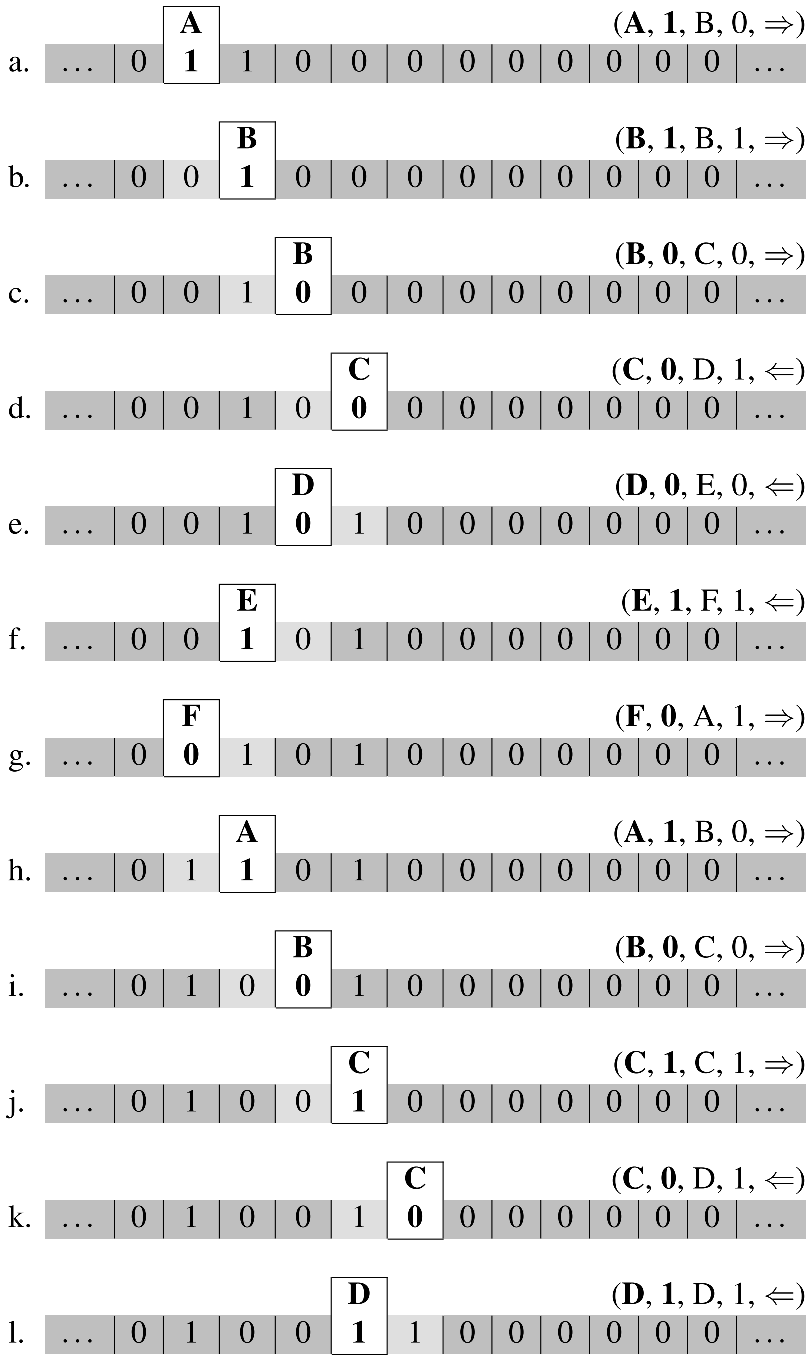
The machine `Squarer` (68) reads an input consisting of n 1s and all 0s otherwise, writes n^2 1s, and halts. It has 20 states not counting the accept and reject states and can be reduced to 19 in a larger machine that first writes a string of 1s per the discussion in section 3.2. A Python implementation is available at `https://github.com/aphonologist/Squarer`.

(68) `Squarer`

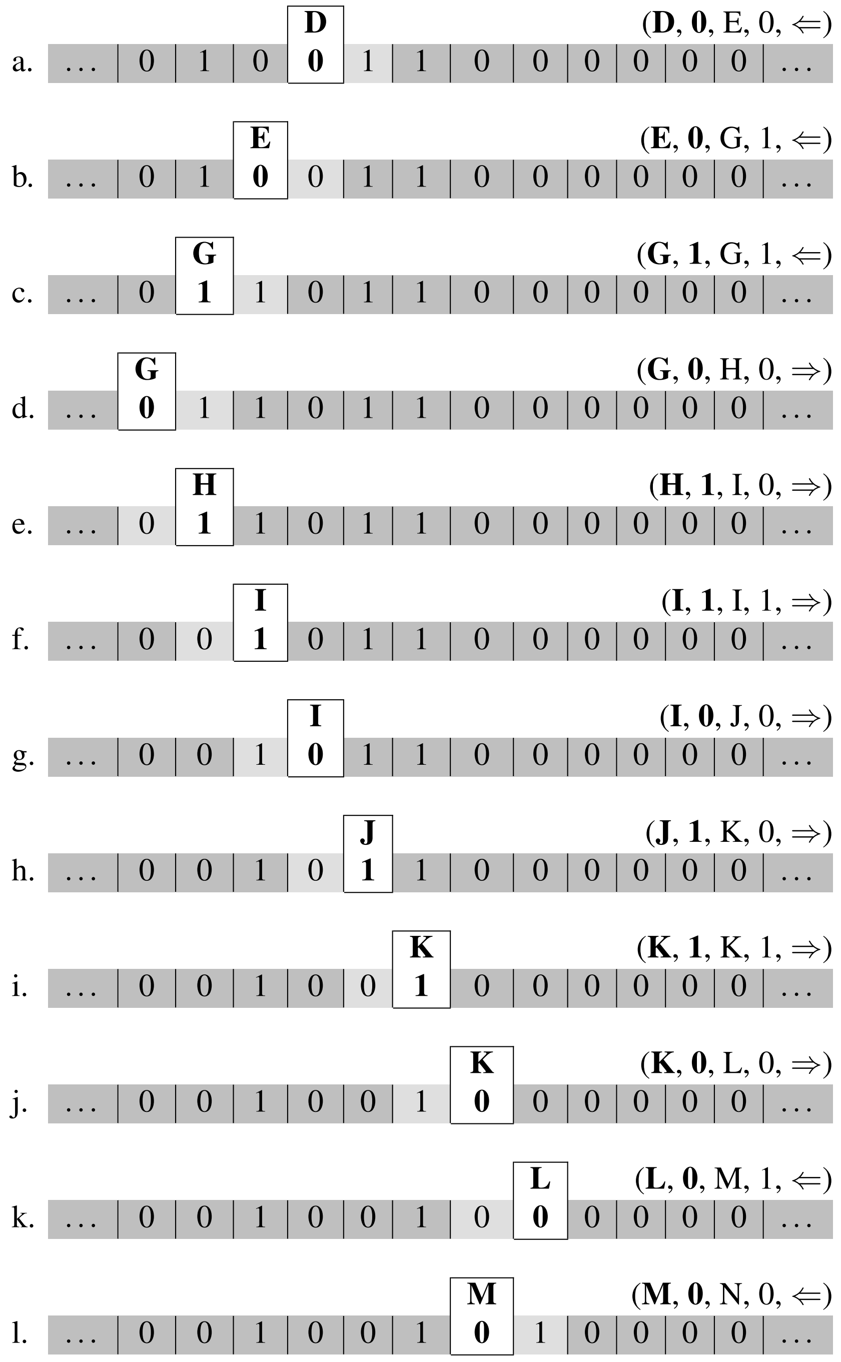
Q	{A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, 👍, 👎}
Σ	{0, 1}
δ	{ (A, 0, A, 0, $\Rightarrow$), (A, 1, B, 0, $\Rightarrow$), (B, 0, C, 0, $\Rightarrow$), (B, 1, B, 1, $\Rightarrow$), (C, 0, D, 1, $\Leftarrow$), (C, 1, C, 1, $\Rightarrow$), (D, 0, E, 0, $\Leftarrow$), (D, 1, D, 1, $\Leftarrow$), (E, 0, G, 1, $\Leftarrow$), (E, 1, F, 1, $\Leftarrow$), (F, 0, A, 1, $\Rightarrow$), (F, 1, F, 1, $\Leftarrow$), (G, 0, H, 0, $\Rightarrow$), (G, 1, G, 1, $\Leftarrow$), (H, 0, H, 0, $\Rightarrow$), (H, 1, I, 0, $\Rightarrow$), (I, 0, J, 0, $\Rightarrow$), (I, 1, I, 1, $\Rightarrow$), (J, 0, J, 0, $\Rightarrow$), (J, 1, K, 0, $\Rightarrow$), (K, 0, L, 0, $\Rightarrow$), (K, 1, K, 1, $\Rightarrow$), (L, 0, M, 1, $\Leftarrow$), (L, 1, L, 1, $\Rightarrow$), (M, 0, N, 0, $\Leftarrow$), (M, 1, M, 1, $\Leftarrow$), (N, 0, P, 1, $\Leftarrow$), (N, 1, O, 1, $\Leftarrow$), (O, 0, J, 1, $\Rightarrow$), (O, 1, O, 1, $\Leftarrow$), (P, 0, Q, 0, $\Leftarrow$), (P, 1, P, 1, $\Leftarrow$), (Q, 0, S, 0, $\Leftarrow$), (Q, 1, R, 1, $\Leftarrow$), (R, 0, H, 0, $\Leftarrow$), (R, 1, R, 1, $\Leftarrow$), (S, 0, S, 0, $\Rightarrow$), (S, 1, T, 0, $\Rightarrow$), (T, 0, 👍, 0, $\Rightarrow$), (T, 1, T, 0, $\Rightarrow$) }
q_{start}	A
q_{accept}	👍
q_{reject}	👎

The full computation of $2^2 = 4$ by `Squarer` is presented in (69-74).

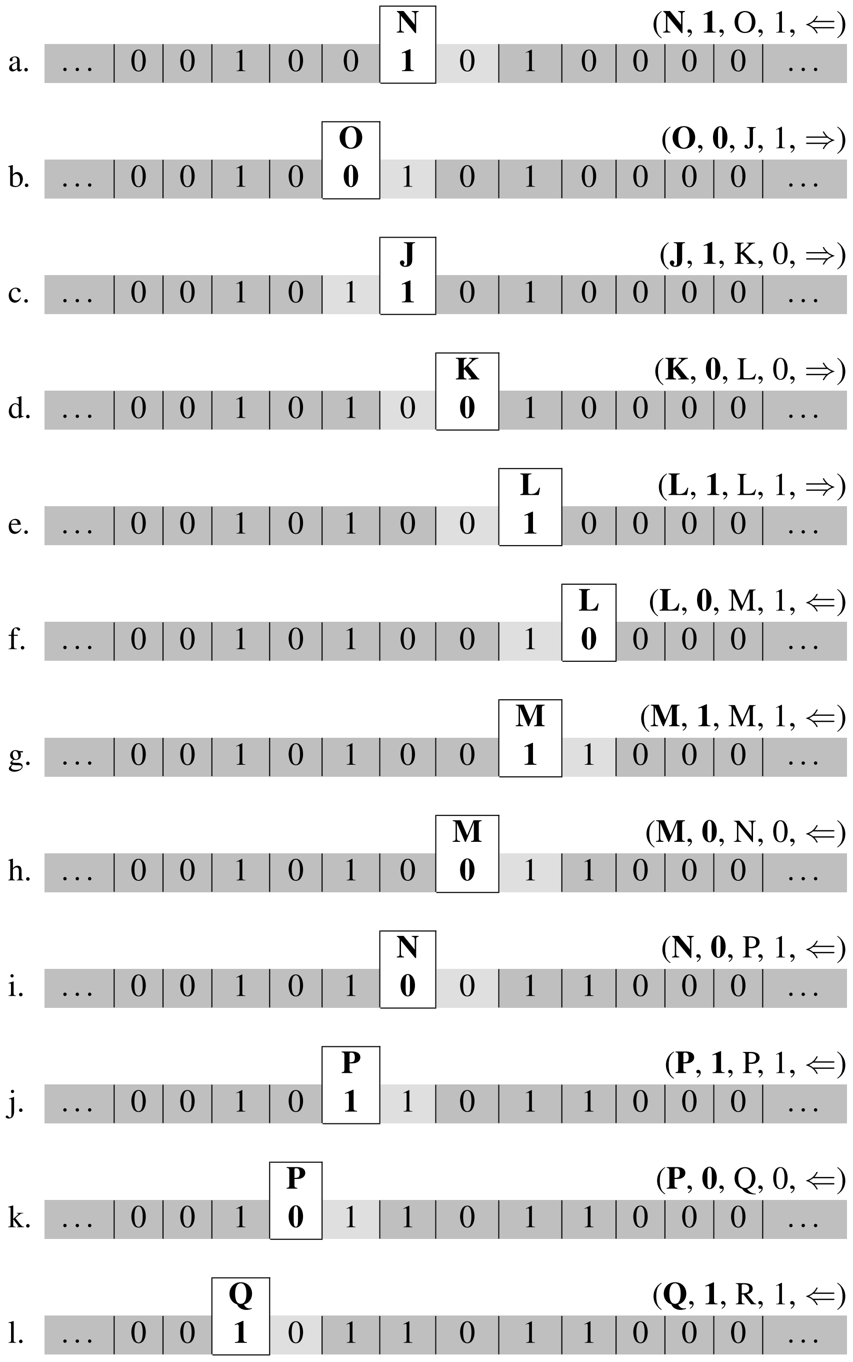
(69) *Computation by* `Squarer` *(1/6)*


a. … 0 **A** **1** 1 0 0 0 0 0 0 0 0 0 … (**A**, **1**, B, 0, ⇒)
b. … 0 0 **B** **1** 0 0 0 0 0 0 0 0 0 … (**B**, **1**, B, 1, ⇒)
c. … 0 0 1 **B** **0** 0 0 0 0 0 0 0 0 … (**B**, **0**, C, 0, ⇒)
d. … 0 0 1 0 **C** **0** 0 0 0 0 0 0 0 … (**C**, **0**, D, 1, ⇐)
e. … 0 0 1 **D** **0** 1 0 0 0 0 0 0 0 … (**D**, **0**, E, 0, ⇐)
f. … 0 0 **E** **1** 0 1 0 0 0 0 0 0 0 … (**E**, **1**, F, 1, ⇐)
g. … 0 **F** **0** 1 0 1 0 0 0 0 0 0 0 … (**F**, **0**, A, 1, ⇒)
h. … 0 1 **A** **1** 0 1 0 0 0 0 0 0 0 … (**A**, **1**, B, 0, ⇒)
i. … 0 1 0 **B** **0** 1 0 0 0 0 0 0 0 … (**B**, **0**, C, 0, ⇒)
j. … 0 1 0 0 **C** **1** 0 0 0 0 0 0 0 … (**C**, **1**, C, 1, ⇒)
k. … 0 1 0 0 1 **C** **0** 0 0 0 0 0 0 … (**C**, **0**, D, 1, ⇐)
l. … 0 1 0 0 **D** **1** 1 0 0 0 0 0 0 … (**D**, **1**, D, 1, ⇐)

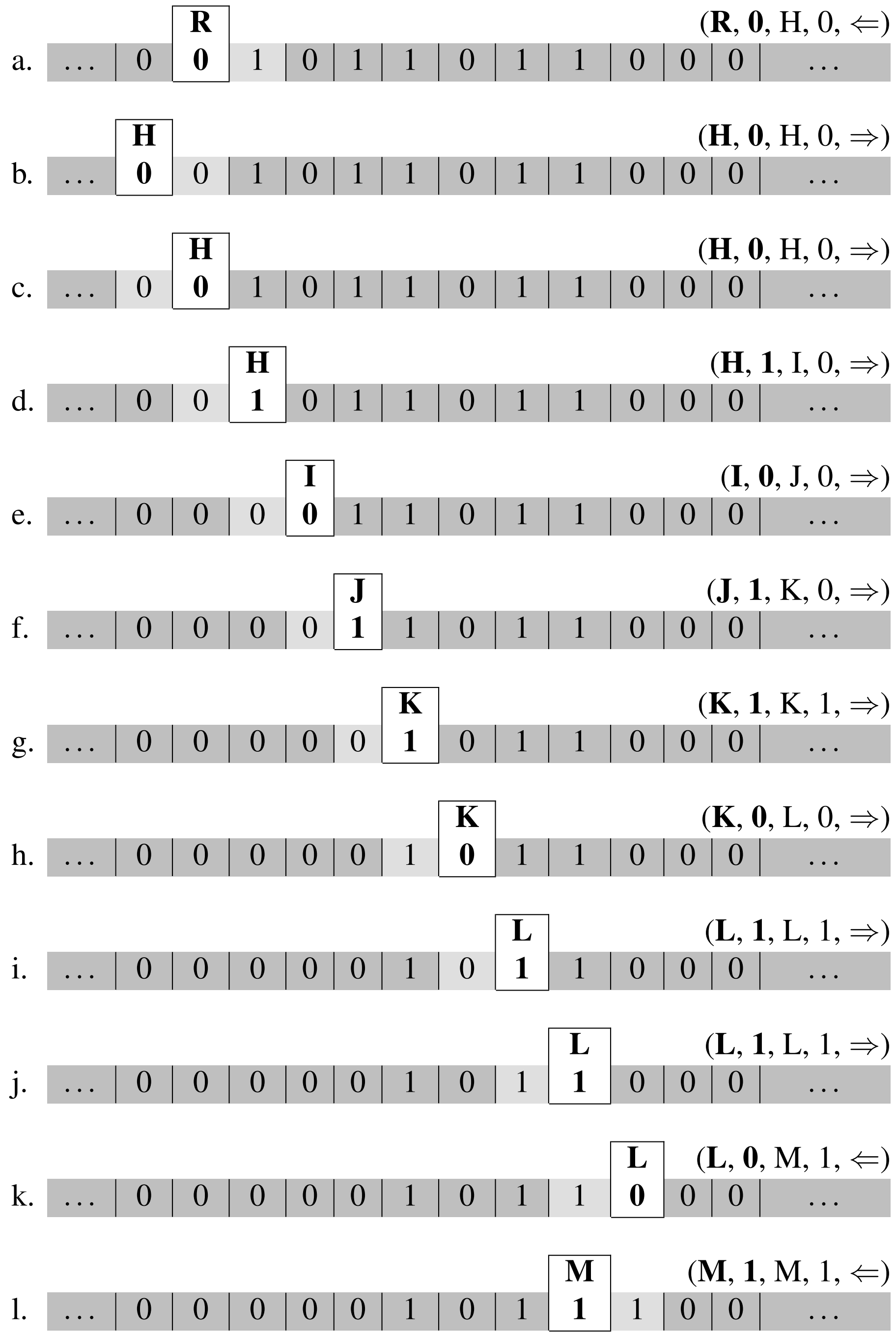
(70) *Computation by* `Squarer` *(2/6)*


a. … 0 1 0 D 0 1 1 0 0 0 0 0 0 … (D, 0, E, 0, ⇐)
b. … 0 1 E 0 0 1 1 0 0 0 0 0 0 … (E, 0, G, 1, ⇐)
c. … 0 G 1 1 0 1 1 0 0 0 0 0 0 … (G, 1, G, 1, ⇐)
d. … G 0 1 1 0 1 1 0 0 0 0 0 0 … (G, 0, H, 0, ⇒)
e. … 0 H 1 1 0 1 1 0 0 0 0 0 0 … (H, 1, I, 0, ⇒)
f. … 0 0 I 1 0 1 1 0 0 0 0 0 0 … (I, 1, I, 1, ⇒)
g. … 0 0 1 I 0 1 1 0 0 0 0 0 0 … (I, 0, J, 0, ⇒)
h. … 0 0 1 0 J 1 1 0 0 0 0 0 0 … (J, 1, K, 0, ⇒)
i. … 0 0 1 0 0 K 1 0 0 0 0 0 0 … (K, 1, K, 1, ⇒)
j. … 0 0 1 0 0 1 K 0 0 0 0 0 0 … (K, 0, L, 0, ⇒)
k. … 0 0 1 0 0 1 0 L 0 0 0 0 0 … (L, 0, M, 1, ⇐)
l. … 0 0 1 0 0 1 M 0 1 0 0 0 0 … (M, 0, N, 0, ⇐)

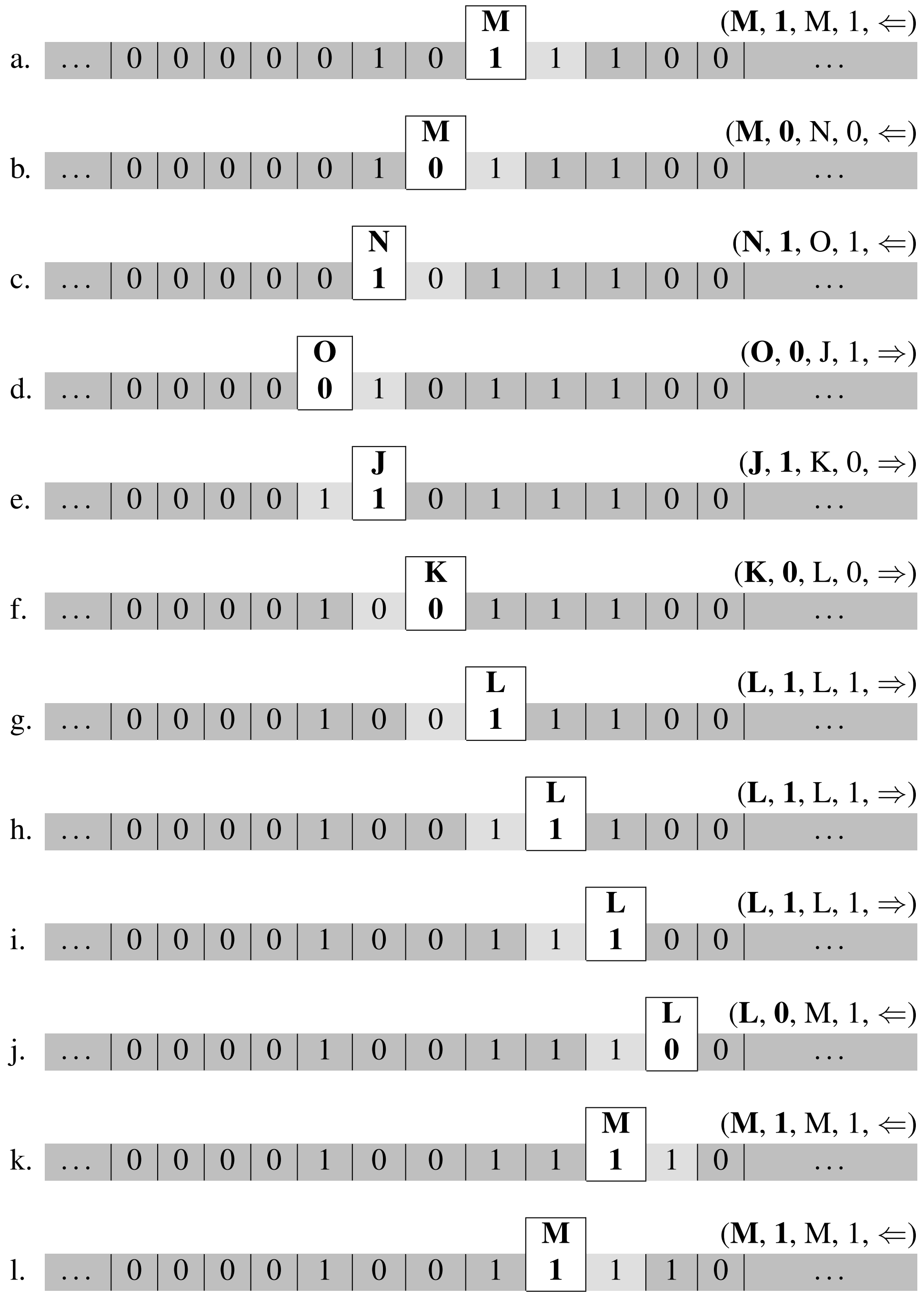
(71) *Computation by* `Squarer` *(3/6)*


a. … 0 0 1 0 0 N 1 0 1 0 0 0 0 … (**N**, **1**, O, 1, ⇐)
b. … 0 0 1 0 O 0 1 0 1 0 0 0 0 … (**O**, **0**, J, 1, ⇒)
c. … 0 0 1 0 1 J 1 0 1 0 0 0 0 … (**J**, **1**, K, 0, ⇒)
d. … 0 0 1 0 1 0 K 0 1 0 0 0 0 … (**K**, **0**, L, 0, ⇒)
e. … 0 0 1 0 1 0 0 L 1 0 0 0 0 … (**L**, **1**, L, 1, ⇒)
f. … 0 0 1 0 1 0 0 1 L 0 0 0 0 … (**L**, **0**, M, 1, ⇐)
g. … 0 0 1 0 1 0 0 M 1 1 0 0 0 … (**M**, **1**, M, 1, ⇐)
h. … 0 0 1 0 1 0 M 0 1 1 0 0 0 … (**M**, **0**, N, 0, ⇐)
i. … 0 0 1 0 1 N 0 0 1 1 0 0 0 … (**N**, **0**, P, 1, ⇐)
j. … 0 0 1 0 P 1 1 0 1 1 0 0 0 … (**P**, **1**, P, 1, ⇐)
k. … 0 0 1 P 0 1 1 0 1 1 0 0 0 … (**P**, **0**, Q, 0, ⇐)
l. … 0 0 Q 1 0 1 1 0 1 1 0 0 0 … (**Q**, **1**, R, 1, ⇐)

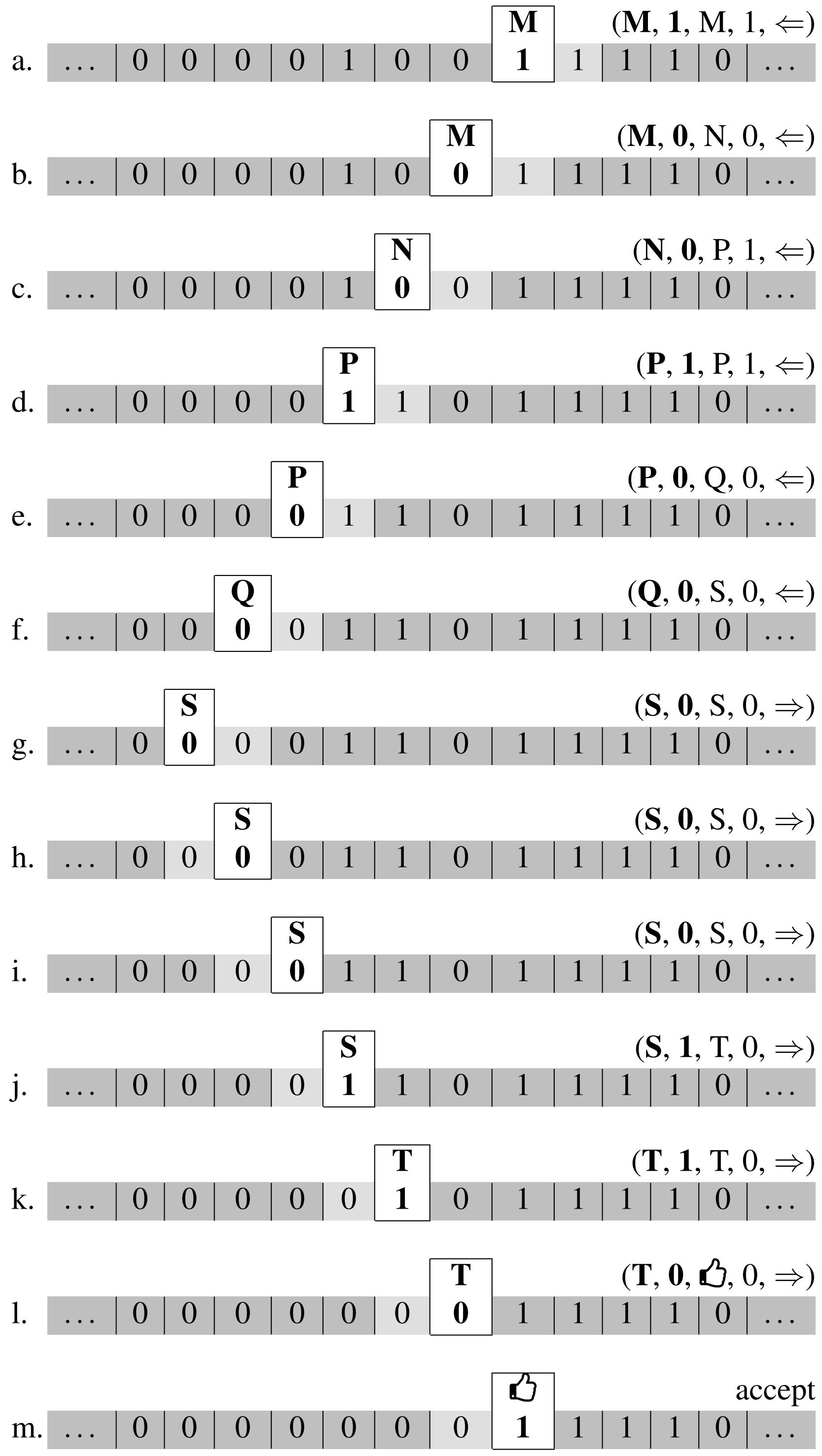
(72) *Computation by* `Squarer` *(4/6)*


a. … 0 **R** **0** 1 0 1 1 0 1 1 0 0 0 … (**R**, **0**, H, 0, ⇐)
b. … **H** **0** 0 1 0 1 1 0 1 1 0 0 0 … (**H**, **0**, H, 0, ⇒)
c. … 0 **H** **0** 1 0 1 1 0 1 1 0 0 0 … (**H**, **0**, H, 0, ⇒)
d. … 0 0 **H** **1** 0 1 1 0 1 1 0 0 0 … (**H**, **1**, I, 0, ⇒)
e. … 0 0 0 **I** **0** 1 1 0 1 1 0 0 0 … (**I**, **0**, J, 0, ⇒)
f. … 0 0 0 0 **J** **1** 1 0 1 1 0 0 0 … (**J**, **1**, K, 0, ⇒)
g. … 0 0 0 0 0 **K** **1** 0 1 1 0 0 0 … (**K**, **1**, K, 1, ⇒)
h. … 0 0 0 0 0 1 **K** **0** 1 1 0 0 0 … (**K**, **0**, L, 0, ⇒)
i. … 0 0 0 0 0 1 0 **L** **1** 1 0 0 0 … (**L**, **1**, L, 1, ⇒)
j. … 0 0 0 0 0 1 0 1 **L** **1** 0 0 0 … (**L**, **1**, L, 1, ⇒)
k. … 0 0 0 0 0 1 0 1 1 **L** **0** 0 0 … (**L**, **0**, M, 1, ⇐)
l. … 0 0 0 0 0 1 0 1 **M** **1** 1 0 0 … (**M**, **1**, M, 1, ⇐)

(73) *Computation by* `Squarer` *(5/6)*


a. … 0 0 0 0 0 1 0 M 1 1 1 0 0 … (M, 1, M, 1, ⇐)
b. … 0 0 0 0 0 1 M 0 1 1 1 0 0 … (M, 0, N, 0, ⇐)
c. … 0 0 0 0 0 N 1 0 1 1 1 0 0 … (N, 1, O, 1, ⇐)
d. … 0 0 0 0 O 0 1 0 1 1 1 0 0 … (O, 0, J, 1, ⇒)
e. … 0 0 0 0 1 J 1 0 1 1 1 0 0 … (J, 1, K, 0, ⇒)
f. … 0 0 0 0 1 0 K 0 1 1 1 0 0 … (K, 0, L, 0, ⇒)
g. … 0 0 0 0 1 0 0 L 1 1 1 0 0 … (L, 1, L, 1, ⇒)
h. … 0 0 0 0 1 0 0 1 L 1 1 0 0 … (L, 1, L, 1, ⇒)
i. … 0 0 0 0 1 0 0 1 1 L 1 0 0 … (L, 1, L, 1, ⇒)
j. … 0 0 0 0 1 0 0 1 1 1 L 0 0 … (L, 0, M, 1, ⇐)
k. … 0 0 0 0 1 0 0 1 1 M 1 1 0 … (M, 1, M, 1, ⇐)
l. … 0 0 0 0 1 0 0 1 M 1 1 1 0 … (M, 1, M, 1, ⇐)

(74) *Computation by* `Squarer` *(6/6)*


(**M**, **1**, M, 1, ⇐)
a. … 0 0 0 0 1 0 0 **M 1** 1 1 1 0 …
(**M**, **0**, N, 0, ⇐)
b. … 0 0 0 0 1 0 **M 0** 1 1 1 1 0 …
(**N**, **0**, P, 1, ⇐)
c. … 0 0 0 0 1 **N 0** 0 1 1 1 1 0 …
(**P**, **1**, P, 1, ⇐)
d. … 0 0 0 0 **P 1** 1 0 1 1 1 1 0 …
(**P**, **0**, Q, 0, ⇐)
e. … 0 0 0 **P 0** 1 1 0 1 1 1 1 0 …
(**Q**, **0**, S, 0, ⇐)
f. … 0 0 **Q 0** 0 1 1 0 1 1 1 1 0 …
(**S**, **0**, S, 0, ⇒)
g. … 0 **S 0** 0 0 1 1 0 1 1 1 1 0 …
(**S**, **0**, S, 0, ⇒)
h. … 0 0 **S 0** 0 1 1 0 1 1 1 1 0 …
(**S**, **0**, S, 0, ⇒)
i. … 0 0 0 **S 0** 1 1 0 1 1 1 1 0 …
(**S**, **1**, T, 0, ⇒)
j. … 0 0 0 0 **S 1** 1 0 1 1 1 1 0 …
(**T**, **1**, T, 0, ⇒)
k. … 0 0 0 0 0 **T 1** 0 1 1 1 1 0 …
(**T**, **0**, 👍, 0, ⇒)
l. … 0 0 0 0 0 0 **T 0** 1 1 1 1 0 …
accept
m. … 0 0 0 0 0 0 0 **👍 1** 1 1 1 0 …