THE ARCHITECTURE AND CONSTRUCTION OF AN OPTIMAL GRAMMAR

Lect. Dr.Abbas Fadhil Albayati *

تأريخ التقديم: ٢٠١٢/١١/٧ تأريخ القبول: ٢٠١٣/١/٢

1. Introduction

The Optimality Theory is a relatively new theory of language that was originally proposed by the linguists Alan Prince and Paul Smolensky (1993/2002), and was later expanded by Alan Prince and John McCarthy (see, e.g., McCarthy, 2004). It is a major assumption of the theory that 'there are no fixed bounds on language' (see Aitchison, 2003, p. 32) .

Although much of the interest in optimality theory has been associated with its use in phonology, the area to which optimality theory was first applied, the theory is also applicable to other subfields of linguistics, e.g. syntax and semantics.

Optimality theory is usually considered a development of generative grammar, which shares its focus on the investigation of universal principles, linguistic typology, and language acquisition.

The optimality theory grammar is an input–output mechanism that pairs an output form to an input form in a way that each input has precisely one output (Kager, 2004, p.18). To accomplish this function, the grammar contains a component which matches the input with an infinite set of candidate output forms, and another component that evaluates the candidate output forms by a set of ranked constraints, and selects the optimal output among these. These two components are known as **Generator** and **Evaluator**, respectively. Generator is a function that, when applied to some input, produces a set of candidates, all of which are logically possible analyses of this input. Similarly, Evaluator is a function that, when applied to a set of output candidates, produces an output, the optimal analysis of the input (cf. Vogel, 2004, p. 211). In addition to Generator and Evaluator, the

***Dept. of English / College of Languages / Univesity of Salahaddin.**

grammar contains a **Lexicon** storing all lexical forms that are input to Generator.

2. Components of the Optimality Theory Grammar

The model of grammar provided by the Optimality Theory consists of the following:

2.1 Lexicon

Lexicon contains lexical representations (or underlying forms) of morphemes, which form the input. In other words, it contains all contrastive properties of morphemes (roots, stems, and affixes) of a language, including phonological, morphological, syntactic, and semantic properties. The Lexicon provides the input specifications which are to be submitted to the Generator. In this connection, perhaps the most striking property of the Lexicon, as conceived of in the Optimality Theory, is that no specific property can be stated at the level of underlying representations. So no constraints hold at the level of underlying forms. This is called **Richness of the Base** (see Kager, 2004, pp. 19-20).

The Optimality Theory thus abandons the Morpheme Structure Constraints (MSCs), which in classical generative phonology (see, e.g., Chomsky & Halle, 1968) account for prohibitions against specific types of structure at the level of the morpheme, in specific languages. MSCs were used, for example, to express prohibitions against front rounded vowels, or sequences of three or more consonants, or two labial consonants occurring within a morpheme. In the early 1970s, MSCs were argued to be theoretically problematic in the sense that they duplicate information which is, independently, expressed by phonological rewrite rules, or that they globally guide the application of rules, a property called 'structure-preservingness'. By locating the burden of explanation of the lack of specific kinds of structure at the level of the output, the Optimality Theory, at least in principle, circumvents this duplication problem (also see Chomsky, 1995, p. 52).

2.2 The Generator

The Generator generates output candidates for some input, and then submits these for evaluation. The essential property of the Generator is that it is free to generate any conceivable output candidate for some input. This property is called **Freedom of Analysis**. Accordingly, any amount of structure may be posited. The only true restriction imposed on all output

candidates generated by Generator is that these are made up of licit elements from the universal vocabularies of linguistic representation, such as segmental structure (features and their grouping below the level of the segment), prosodic structure (mora, syllable, foot, prosodic word, etc.), and morphology (root, stem, word, affix, etc.), and syntax (X-bar structure, heads, complements, specifiers, etc). Within these limits, 'anything goes', as Ito and Mester (2004, p. 559) put it. The Generator contains information about the representational primitives and their universally irrevocable relations. For example, in case of syllable structure in phonology, the Generator tells that the nucleus may dominate an onset or a coda, but never vice versa.

Since Generator generates all logically possible candidate analyses of a given input, an optimal grammar needs no rewrite rules to map inputs onto outputs. All structural changes are applied in one step, in parallel. The evaluation of these candidate analyses is the function of the Evaluator, the component of ranked constraints, which is discussed in the next section.

2.3 The Evaluator

Evaluator is the set of ranked constraints, which evaluates output candidates as to their harmonic values, and selects the optimal candidate. The Evaluator is undoubtedly the central component of the grammar since it is burdened with the responsibility of accounting for all observable regularities of surface forms. Although any candidate output can be posited by the Generator, the crucial role of Evaluator is to assess the harmony of outputs with respect to a given ranking of constraints.

The Evaluator is structured as a language-specific hierarchy of universal constraints, plus devices for evaluation. The latter include the means to assess violation marks on candidate outputs for every constraint, and the means to rank an infinite set of candidate outputs for harmony with respect to the hierarchy of constraints, and select the most harmonic one of these as **optimal** – the actual output of the grammar. Let us now take a closer look at each of these devices:

2.3.1 The Constraint Hierarchy

The constraint hierarchy contains all universal constraints, which are ranked in a language-specific way. We tentatively assume that all constraints are ranked with respect to each other, so as to exclude variable and undetermined rankings. Within the hierarchy, dominance relations are transitive. So there is **Transitivity of Ranking**, e.g.

- If Constraint 1 dominates Constraint 2 and Constraint 2 dominates Constraint 3, then Constraint 1 dominates Constraint 3.

As a simplified example of constraints and their working, let us consider the manifestation of the English plural /z/:

Input	Output
/kæt + z/	[kæts]
/dɒg + z/	[dɒgz]
/diʃ + z/	[diʃɪz]

We also need to consider the following constraint set, in descending order of domination (M: Markedness, F: Faithfulness):

1. M: *SS: Sibilant-Sibilant clusters are ungrammatical: One violation for every pair of adjacent sibilants in the output.

2. M: Agree(Voi): Agree in specification of [voi]: One violation for every pair of adjacent obstruents in the output which disagree in voicing.

3. F: Ident(Voi): Maintain the identity of the [voi] specification: One violation for each segment that differs in voicing between the input and output.

4. F: Max: Maximize all input segments in the output: One violation for each segment in the input that does not appear in the output. This constraint prevents deletion.

5. F: Dep: Output segments are dependent on having an input correspondent: One violation for each segment in the output that does not appear in the input. This constraint prevents insertion.

The following tableau shows the hierarchy of these constraints in the case of 'cats' in the English language:

/kæt + z/	*SS	Agree	Max	Dep	Ident
Kætɪz				*!	
Kætɪs				*!	*
Kætz		*!			
Kæt			*!		
☞ kæts					*

No matter how the constraints are re-ordered, the 'ɪs' allomorph will always lose to 'ɪz'. This is called 'harmonic bounding' (Prince & Smolensky, 2002, p. 193). The violations incured by the candidate 'dɒgɪz' are a subset of the violations incured by 'dɒgɪs'; specifically, if you epenthesize a vowel, changing the voicing of the morpheme is gratuitous violation of constraints. In the 'dɒg + z' tableau, there is a candidate 'dɒgz' which incurs no violations whatsoever. Within the constraint set of the problem, 'dɒgz' harmonically bounds all other possible candidates. This shows that a candidate does not need to be a winner in order to harmonically bound another candidate. This property of ranking will allow us to construct ranking arguments, as we shall see below:

2.3.2 Marking of Violations

With respect to violation marks, we assume that each output candidate is provided with as many marks (asterisks) as it has violations for a constraint. This number of marks potentially ranges from zero until

infinite. However, for purposes of determining optimal outputs, an infinite number of marks is never practically relevant. The essence of minimal violation of constraints is that every violation of a constraint serves a purpose, specifically to avoid a violation of some higher-ranked constraint. Prince and Smolensky (2002, p. 27) call this property **Economy**. It maintains that banned options are available only to avoid violations of higher-ranked constraints and can only be banned minimally. For example, the Generator component is free to submit any kind of analysis of the English word /bed/ that is couched within the universal alphabet of representational options, including excessively unfaithful candidates such as [pɪlow] and [mætrəs]. But these candidates will be ruled out regardless of constraint ranking, since they violate faithfulness constraints without compensation from reductions in markedness.

2.3.3 Harmony Evaluation

We have not yet precisely formulated in which way the evaluation of output candidates by ranked constraints proceeds. Evaluator determines the harmonic status of output candidates, and eventually the most harmonic or optimal candidate. To this end, it uses a process by which the set of candidates is reduced until the point is reached at which one output remains. This is a multi-step process.

The major property of this evaluation process is that it applies from one state to another without looking ahead to following steps. That is, the elimination of candidate outputs by a constraint (C_X) is never affected by a lower-ranked constraint (C_Y) stated in a non-serial manner. This is called **Strict Domination**, meaning that violation of higher-ranked constraints cannot be compensated for by satisfaction of lower-ranked constraints (cf. Dresher, 1996, p. 8).

Optimality does not involve any kind of compromise between constraints of different ranks. No smaller amount of violations can compensate for ranking of constraints. Domination is strict, i.e. any candidate that incurs a violation of some higher-ranked constraint (on which another candidate incurs no violations) is mercilessly excluded, regardless of its relative well-formedness with respect to any lower-ranked constraints (for further details, also see Legendre, 2001, pp. 1-27).

There is yet another sense in which domination is strict: Constraint violations are never added for different constraints. The added violations of two lower-ranked constraints (Constraint 2 and Constraint 3) are not able to

cancel out a single violation of a higher-ranked constraint (Constraint 1). That is, lower-ranked constraints cannot team up against a higher-ranked constraint.

Not all interactions of constraints are of this relatively simple kind, i.e. where an optimal candidate satisfies a high-ranked constraint that is violated by all competitors. Actually, most interactions involve some degree of violation in the optimal candidate. Violation of a constraint is, by itself, an insufficient ground for ungrammaticality. It is necessary to recall that the goal of evaluation is to single out one unique form as the most harmonic one. Elimination of all candidates in the set under consideration is therefore not allowed. This must be avoided. Hence for a violation of some constraint to be fatal, at least one other form must occur in the candidate set that satisfies it, without being less harmonic on higher-ranked constraints, of course. Constraint 1 should not be a no-pass filter. If no such form can be found, some violation must be taken for granted (Prince & Smolensky, 2002, p. 73).

In such a situation, in which all remaining candidate outputs violate a constraint (due to higher-ranked constraints), the seriousness of violation must be taken into account for each individual form. That is, forms with fewer violation marks of Constraint 1 are preferred to forms with more violation marks for Constraint 1. This situation may still produce a ranking argument for Constraint 1 and Constraint 2. Here the amount of violation is decisive.

Finally, if multiple candidates have the same number of violations for Constraint 1 (and this equals the minimal violation in the set), then all survive and are passed on for evaluation by the next constraint down the hierarchy, Constraint 2. Here lower-ranking constraint will be decisive. This situation can be represented as an all-pass filter Constraint 1. Of course, ties between candidates may also arise between forms that have no violations at all, or between forms that have two, three, or any number of violations.

The above discussion emphasizes that lower-ranked constraints are not rendered inactive, or switched off by higher-ranked constraints, but that their violation is only avoided with less priority. Lower-ranked constraints may be violated by the optimal output, but their violation must be minimal. Given the chance, any constraint (regardless of its position in the hierarchy)

will be active in determining the optimal output (activity of a dominated constraint).

The final property of Evaluator is **Parallelism**, which means that all constraints pertaining to some type of structure interact in a single hierarchy (see Prince & Smolensky, 2004, pp. 27-29). In a trivial sense, it is parallelism which predicts that faithfulness constraints may interact with markedness constraints in a single hierarchy. But at a higher level of sophistication, parallelism is also the basis of explanation of phenomena involving interface properties. In particular, many examples show that morphological and phonological properties of an output form are mutually dependent. The most spectacular cases will come from the area of prosodic morphology, i.e. types of morphology that depend on aspects of syllabification and metrical structure such as reduplication, infixation, and truncation. According to Kager (2004, p. 25), it is parallelism that makes information flow back and forth between morphological and prosodic aspects in such cases.

Given two candidates, A and B, A is better than B on a constraint if A incurs fewer violations than B. Candidate A is better than B on an entire constraint hierarchy if A incurs fewer violations of the highest-ranked constraint distinguishing A and B. A is optimal in its candidate set if it is better on the constraint hierarchy than all other candidates. For example, given constraints C1, C2, and C3, where C1 dominates C2, which dominates C3 (C1 >> C2 >> C3), A is optimal if it does better than B on the highest ranking constraint which assigns them a different number of violations. If A and B tie on C1, but A does better than B on C2, A is optimal, even if A has 100 more violations of C3 than B. This comparison is often illustrated with a tableau. As noted earlier, the pointing finger marks the optimal candidate, and each cell displays the number of violations for a given candidate and constraint. Once a candidate does worse than another candidate on the highest ranking constraint distinguishing them, it incurs a crucial or fatal violation, marked in the tableau by an exclamation mark. Once a candidate incurs a crucial violation, there is no way for it to be optimal, even if it outperforms the other candidates on the rest of constraints. The following tableau summarizes the above discussion:

A violation tableau

	C1	C2	C3
☞A	*	*	***
B	*	**!	

3. Infinity and Optimality Theory

Freedom of Analysis may seem to pose an overwhelming computational problem for the basic function of a grammar, which is to provide a mapping between input and output. Perhaps the most apparent fear is that an infinite candidate space is computationally intractable. Reactions to this point focus on the nature of candidate space, on evaluation strategies which assure a more efficient processing, and on computational results booked so far in modelling Optimality Theory (for more details, see Prince & Smolensky, 2002).

Firstly, it is a well-accepted assumption among linguists that there is a distinction between the grammar (competence) and its implementation (performance). This distinction is assumed in most formal theories of grammar, and particularly in generative linguistics (Chomsky, 1965). Therefore a model of grammar is adequate to the extent that it explains observed systematicities in natural languages, and the grammatical judgements of speakers. Explaining the actual processing of linguistic knowledge by the human mind is not the goal of the formal theory of grammar, but that of linguistic disciplines such as psycholinguistics, neurolinguistics, cognitive linguistics, and computational linguistics. The central point is that a grammatical model should not be equated with its computational implementation.

Secondly, turning now to computational plausibility, the fact that candidate space is infinite does not imply that the problem is logically unsolvable. We may convince ourselves of this by thinking of arithmetic or any kind of numerical problem. For example, there is a unique solution to the equation $3n^2 - 3 = 45$: $(n = 4)$, which you will be able to find after a

moment's thought, even though the candidate set is infinite, as it involves all integers. From a computational viewpoint, the decisive factor is that a guaranteed method (an algorithm) exists that will certainly produce a solution for any input. Therefore, no simple argument against the Optimality Theory as being computationally intractable can be based on the observation that candidate space is infinite (see McCarthy, 2001, p. 14).

Thirdly, smart computational strategies may eliminate suboptimal candidates by classes, rather than on a one-by-one basis. As soon as a candidate has been excluded due to its violation of some constraint, the evaluation process can immediately eliminate all other candidates that violate this constraint more severely. This leads us to yet another property of candidate space that might be put to use in computational evaluation models. By far the great majority of candidates proposed by Generator can never be selected as optimal, under any possible ranking of constraints. Such intrinsically suboptimal candidates can be readily identified as follows: They share with another candidate (of the same input) some set of violation marks, but have at least one additional violation of some other constraint. Sophisticated evaluation strategies may capitalize on this. Since the identification of intrinsically suboptimal candidates involves no ranked constraints, infinite candidate space may be drastically reduced by eliminating the worst-of-the-worst of candidates by preprocessing prior to the evaluation by ranked constraints. Since this preprocessing would eliminate the great majority of candidates, the ultimately relevant remaining part of candidate space may well have quite manageable proportions, and perhaps even reduce to a finite set (Hammond, 1997; Prince, 1994).

4. Summary and Conclusions

The Optimality theory is a linguistic model proposing that the observed forms of language arise from the interaction between conflicting constraints. It aims to present grammars that are based on universal constraints that are essentially violable. A different set of constraints may apply in different Languages, but they are all selected from the same pool of universal constraints. The same constraints may apply in a different order, which results in changing the output that appears on the surface. There are three basic components of the theory:

1. Generator, which generates the list of possible outputs or candidates. These are taken from the lexicon that contains the lexical representations (underlying forms) of the morphemes and supplies the input for the Generator. The phonological form of the morphemes is language-specific. The Generator produces a potentially infinite number of output candidates and passes them to the Evaluator. The Input is different from one language to another because the underlying forms of the lexicon comprise the input
2. Constraint, which provides the criteria, violable constraints, used to decide between candidates. If two candidates both comply with several constraints, there must be further (lower-order) constraints which differentiate between the two and select one candidate. If two candidates cannot be differentiated, they are identical.
3. Evaluator, which chooses the optimal candidate based on the constraints. It consists of a set of ordered constraints, and evaluates the output candidates with regard to their harmony values, i.e. the degree to which they comply with the constraints. It selects the optimal candidate. The selection is unique in the sense that there is one optimal candidate as output.

The Optimality Theory assumes that these components are universal. Differences in grammars reflect different rankings of the universal constraint set. Consequently, language acquisition can be described as the process of adjusting the ranking of these constraints.

References

- Aitchison, J. (2003). **Linguistics** (6th ed.). London: Teach Yourself.
- Chomsky, N. (1995). **The Minimalist Program**. Cambridge, MA: The MIT Press.
- Dresher, B. E. (1996). **The Rise of Optimality Theory in First Century Palestine**. *GLOT International 2*, 1/2, January/February, p. 8.
- Ito, J., & Mester, A. (2004). **The Phonological Lexicon**. In J. J. McCarthy (Ed),*Optimality Theory in Phonology* (pp. 552-568). Oxford: Blackwell Publishing.
- Kager, R. (2004). **Optimality Theory**. Cambridge: Cambridge University Press.

- Legendre, G. (2001). **An Introduction to Optimality Theory in Syntax**. In G.Legendre, J. Grimshaw, & S. Vikner (Eds.), *Optimality-Theoretic Syntax*. Cambridge, MA: MIT Press.
- McCarthy, J. (2001). **A Thematic Guide to Optimality Theory**. Cambridge: Cambridge University Press.
- McCarthy, J. (Ed.) (2004). **Optimality Theory in Phonology: A Reader**. Malden, MA: Blackwell publishing
- Prince, A., & Smolensky, P. (2002). **Optimality Theory: Constraint Interaction in Generative Grammar**. N.P: ROA
- Prince, A., & Smolensky, P. (2004). **Optimality Theory: Constraint Interaction in Generative Grammar**. In J. McCarthy (Ed.), *Optimality Theory in Phonology: A Reader* (pp. 3-71). Malden, MA: Blackwell Publishing.
- Vogel, R. (2004). **Remarks on the Architecture of Optimality Theoretic Syntax Grammars**. In R. Blutner & H. Zeevat (Eds), *Optimality Theory and Pragmatics* (pp. 211-227). New York, NY: Palgrave Macmillan.

الخلاصة تصميم القاعدة المثلى وصياغتها

م.د.عباس فاضل البياتي

المستخلص

إنَّ نظرية الأمثلية قائمة على الزعم بأن قواعد اللغة مبنية على عناصر ثلاثة هي: (١) المولد: الذي يتكون من قائمة المخرجات أو المرشحات المحتملة، (٢) المقيدات: والتي تقدم المنهاج لتحديد المرشح الأمثل فضلاً عن المقيدات القابلة للخرق في عملية التحديد، (٣) المقييم: وهو العنصر الذي ينتهي إلى المرشح الأمثل بعد إسقاط المقيدات عليها.

والتعميمات القواعدية في نظرية الأمثلية هي تنافس بين مجموعة من المقيدات على مستوى البنى السطحية لا العميقة، إذ لايوجد أي قيد على البنية العميقة ، وفيه نوعان من المقيدات: (١) المقيدات الواسمة التي تبين المتطلبات الشكلية للمخرجات، (٢) مقيدات التماثل التي تقيّم شكل المخرجات رغم أنها تشير إلى المدخلات في بيان متطلباتها.

فهذه النظرية تفترض عدم وجود مقيدات خاصة بكل لغة فيما يتعلق بالمدخل، وتسمى هذه الظاهرة بـــ"غنى الأساس"، فلكل قاعدة أن تتصدى لأي مدخل مقبول، وحال تواجد المدخل، يستطيع عنصر المولد أن يقدم مجموعة غير متناهية من المخرجات أو التمثيلات الشكلية لذلك المدخل؛ وبهذا تكون أية قاعدة في أية لغة مجرد تسمية المرشح (المخرج) الأمثل من بين باقي المرشحين ليكون ممثلاً عن مدخل ما، وعملية التسمية هذه هي في صلب عمل مكون المقييم.