A Users Guide to the Optimality Interpreter: A Software Tool for Optimality Theoretic Analysis[1]

William Raymond and Apollo Hogan
Software by Apollo Hogan
University of Colorado at Boulder

Optimality Theory (Prince & Smolensky 1993) is a constraint-based formalism that has been successfully applied to the explanation of various phenomena of language in the areas of both phonology and syntax. While OT is not inherently computational, it requires some symbol manipulation, and so lends itself to facilitation through software that can aid in the design of optimality theoretic solutions. Such a software application, the Optimality Interpreter, is described in this document. The Interpreter allows for the specification of constraints and competing candidates, as well as valuation of the well-formedness of candidates vis-à-vis the constraints. Constraint orderings can be used to determine the most harmonic candidates, or the Interpreter can determine the existence of a constraint ordering that will force a specified candidate to win. In addition, the interaction of candidate sets can be explored, with implications for the explanation or definition of language typologies. This document describes the functionality of the Optimality Interpreter, provides a detailed tutorial of its application to a problem in syntax from the literature, and discusses the algorithms and data structures underlying the application's implementation. A user's manual is included as an appendix.

1.0 INTRODUCTION. Optimality Theory (OT), as proposed in Prince & Smolensky (1993), is a constraint-based formalism that can be profitably employed to explore how universal principles of well-formedness determine language structure at all levels of the grammar.[2] Competing surface structures ('candidate outputs') for a given underlying form (an 'input') are evaluated against a set of constraints (given 'marks') to determine their well formedness. Each candidate is either well-formed or not with respect to each constraint. By ranking the constraints in a strict dominance hierarchy, the unique candidate output which will be produced as the surface structure for a given input is selected (the 'winning candidate', or 'output'). The winning output violates only lower-ranked or fewer constraints than all its competitors, and is said to be the most 'harmonic' output. By this evaluation procedure a constraint ranking on a set of competing forms defines a grammar.

While OT is not inherently computational, it does require some symbol manipulation, the amount and complexity of which grows in an analysis as the number of constraints and candidate sets increase. To facilitate Optimality Theoretic analyses, a software program, the Optimality Interpreter, has been designed and implemented. The Interpreter is a tool which allows the linguist to create OT tableaux—the basic data structure of an OT analysis—and manipulate their contents by varying the constraints, their ranking, and the number of inputs and corresponding candidate outputs. The well-formedness of each candidate vis-à-vis each constraint can also be indicated and changed. After creating a tableau, the space of allowed output structures and constraint rankings can be readily explored, both for a single input and across multiple inputs and their candidate sets.

This document is an introduction to the use of the Optimality Interpreter software. It is organized as follows. A brief prose overview of the software is first presented. The functions provided by the software are described and tied to the commands that implement them (§2). After this overview, a tutorial is presented, leading the reader through the use of the Interpreter on an example OT application from the literature (§3). For those interested in learning more detail about the workings of the software, a discussion of the major algorithms employed can be found in §4. For ease of future reference, a list of commands, with formats and usage, is provided in the Appendix.

---

[1]The software described herein was developed under NSF grant number IRI-9213894. The documentation was written under NSF grant number DBS-9209265. The authors would like to acknowledge Géraldine Legendre and Paul Smolensky for their sponsorship and thank them for their support and input on this project.

A copy of the software can be obtained on request from Paul Smolensky at the University of Colorado through email to smolensky@cs.colorado.edu. Bug reports and general feedback (including requests for enhancements) can be directed to the author of the software, Apollo Hogan, at hogan@ucsu.colorado.edu. All comments are welcome.

[2]Readers are referred to Prince & Smolensky 1993 for a detailed discussion of Optimality Theory. The basic principles and terminology of the theory are henceforth assumed.

2.0 A MAP THROUGH THE OPTIMALITY INTERPRETER. This section contains a high-level discussion of the functionality of the Optimality Interpreter and is intended to orient the user in its functioning. The commands implementing each function have been indicated in parentheses following the functional descriptions. Command syntax for all commands can be found in the Appendix, to which the reader is referred for usage particulars. We note that commands can be entered by typing the full command name or the command abbreviation. Abbreviations are indicated in this section and elsewhere by the bold underscored letters in the command names.

The Interpreter is invoked by entering 'opti'. A file may be specified on the invocation line. A file entered as a parameter must have been created in a previous Interpreter session (see §2 for more about the file interface). If no file is specified on initiation, the Interpreter begins with a tabula rasa.

There are two modes of operation in the Optimality Interpreter: 'tableau' mode (**t**ableau_mode) and 'interaction' mode (**i**nteraction_mode). Tableau mode is used to create and manipulate the basic representation structure for Optimality Theoretic analyses, tableaus. Interaction mode is used for investigating the complex interaction of constraints across candidate output sets for different inputs under the same constraint ranking. It is useful for generating language typologies through the exploration of the possible concurrent winners for multiple inputs under all constraint rankings. In addition to specifying a mode explicitly (with **t** or **g**) one can toggle between these modes (using **m**ode_change). Note that upon entering the Interpreter, the user begins the session in tableau mode, and the current mode is always indicated via the prompt ('TableauMode--' or 'InteractionMode--', respectively).

TABLEAU MODE. Tableaux are created in tableau mode. Creating a tableau involves specifying the candidate outputs (**a**dd **o**utput, **r**emove **o**utput) and a constraint set (**a**dd **c**onstraint, **r**emove **c**onstraint), and levying marks for each output against those constraints that it violates (**a**dd **m**arks). Output candidate sets are delineated by making competing outputs members of a 'candidate set' corresponding to a single input (**a**dd **se**t, **r**emove **se**t). Candidate sets are used by many other Interpreter functions.

The current state of a tableau can be viewed at any time on request (**d**isplay) or displayed automatically after each command (**p**erpetual_display). Tableaux can be saved in a system file for further work, perhaps in a future Interpreter session (**s**ave), and (re-)loaded for reuse or modification once created and saved (**l**oad). While analysis can only be done on a single tableau at a time, saving and reloading allow multiple tableaux to be explored in series in a single Interpreter session.

Once a tableau has been created, tableau mode is used to evaluate a designated candidate output set against the defined constraints. The software will eliminate 'occulted' outputs (i.e. outputs that cannot possibly win under any constraint ordering) (**o**cculted_remove) and identify the unique winner in a candidate set given the current constraint ranking (**d**isplay **w**inners). By convention, the Interpreter enters constraints from left to right across the top of a tableau. This order implicitly defines a constraint ranking from highest (leftmost) to lowest (rightmost). The constraints can be reranked in three ways: (1) moving constraints all the way to the left, to the top of the hierarchy, so that they 'dominate' all other constraints in the tableau (**f**ront); (2) moving constraints all the way to the right, to the bottom of the hierarchy, so that they are dominated by all other constraints (**b**ack); or (3) interchanging two constraint positions, and hence reversing their rankings and dominance relation (**e**xchange). It is also possible to request that the Interpreter produce a ranking of constraints (or to determine that one does not exist) for a user-designated winning output (**w**inner_ordering).

INTERACTION MODE. In tableau mode, multiple sets of output candidates can be created, each corresponding to a different input. When the sets overlap and are sensitive to the same constraints, possible constraint rankings and winning outputs are further constrained: the ordering of constraints determines the winning output candidate within each set, and this constrains the possible combinations of winners across sets. The interaction of sets can be explored in interaction mode.

The Interpreter performs its manipulations in interaction mode on a set of directed acyclic graphs (DAGs), though this data structure may remain invisible to the user. A DAG is automatically created from the current tableau (if any) when interaction mode is entered (and any previously-created DAGs are deleted). Like tableaux in tableau mode, DAGs can be saved (**s**ave) and subsequently reloaded for further analysis (**l**oad) from within interaction mode.

In order to explore candidate set interactions in interaction mode, the user binds sets which will interact into a 'composite candidate set' while at the same time generating possible winning combinations from each composite set (**c**ross). Composite sets can be modified (**r**emove) and displayed (**d**isplay). The DAGs' information, including

the minimum ordering requirements that apply to the constraints for a given combination of winners, can also be displayed in symbolic form (**d**isplay **f**ull).

At any time, the Interpreter can be terminated (**q**uit) or the current data can be cleared to begin a new analysis (**n**ew_tableau, **n**ew_interactions, depending on the current mode). In either mode, a summary of the commands for that mode can be accessed in the format presented in the Appendix (**h**elp).

3.0 AN OPTIMALITY INTERPRETER TUTORIAL: A CASE TYPOLOGY ANALYSIS. Legendre, Raymond, and Smolensky (1993) provide an application of Optimality Theory to syntax that derives the intransitive case marking typology of Nominative/Accusative, Ergative/Absolutive, and Active/Stative languages from the interaction of a small set of simple constraints. This section assumes their analysis and shows how the Optimality Interpreter might be used to develop it. We will create the necessary tableaux in tableau mode and then generate the typology in interaction mode. By repeating the commands provided in this section, the user is guided through a self-tutorial in the use of the Optimality Interpreter.

The intransitive case system typology analysis can be performed using six constraints acting on two candidate sets. The candidate sets correspond to two possible inputs, Agent and Patient. Inputs are assumed to correspond to thematic roles at some underlying level, while the outputs are viewed as a high-level representation of what representation these roles are mapped onto at surface structure. The analysis assumes that inputs are predicate arguments with thematic role A or P, and outputs are the input NP's marked for abstract Case, which may be recognized by morphological case marking or by some other identification strategy, such as word order, agreement, etc. The analysis assumes three possible Cases in which Agents and Patients can surface, labeled **C1**, **C2**, and **C4**. Cases **C1** and **C2** correspond to the marking of Agents and Patients, respectively, in a simple transitive sentence; **C4** subsumes all other cases.

In the analysis, we will refer to the inputs as **A** and **P**. The six candidate outputs are then the two inputs marked for each of the three possible Cases. The outputs are represented mnemonically as **A1**, **A2**, **A4**, **P1**, **P2**, and **P4**, where **A1** is an Agent (**A**) input appearing on the surface in Case **C1** (a nominative agent, e.g., in a Nominative/Accusative language), etc. These six outputs are divided into two candidate sets: the Agent set {**A1**, **A2**, **A4**} and the Patient set {**P1**, **P2**, **P4**}.

The six constraints pertinent to the intransitive analysis are listed in (3) p. 468 of Legendre et al, and defined in the paper. As an example, the constraint **A->C1** says that an Agent input must be mapped to Case **C1**. We refer the reader to the paper for a complete discussion of the constraints. The six constraints are represented mnemonically as **P->C2**, **P/-> C1**, **alpha-> C2**, **X->C1**, **A->C1**, and **A/-> C2**, and these labels will be used in this Interpreter analysis.

TABLEAU MODE. The Optimality Interpreter is first invoked from the system prompt by entering

```
opti
```

Once in the Interpreter, we can create the tableaux. Since the constraints apply to both candidate sets, we can create a single tableau containing both sets. First, the constraints are entered using the commands below, the first three with the complete command name, at the prompt, and the last three using the abbreviated command name **a c**. Note that the prompt reminds us that we are in tableau mode.

```
TableauMode-- add constraint P->C2
TableauMode-- add constraint P/->C1
TableauMode-- add constraint alpha->C2
TableauMode-- a c X->C1
TableauMode-- a c A->C1
TableauMode-- a c A/->C2
```

Following the constraint definition, we add the outputs and their marks for each constraint. Underscore represents no mark, and an asterisk indicates that a mark is levied for an output with respect to the corresponding

constraint in the current tableau ranking (here, in order of entry above). Note that mark positions must be separated by white space--i.e. a space or tab (here, tab is used for clarity).

```
TableauMode-- add output A1      _       _       *       _       _       _
TableauMode-- add output A2      _       _       _       *       *       *
TableauMode-- add output A4      _       _       *       *       *
TableauMode-- a o P1             *       *       *                       _
TableauMode-- a o P2                                     _       _       _
TableauMode-- a o P4             _       _       _       *       _       _
                                 *       _       *       *       _       _
```

Finally, we define the two output sets, one for each of the two inputs **A** and **P**. We will name the sets by the inputs to which they corresond.

```
TableauMode-- add set A A1 A2 A4
TableauMode-- a s P P1 P2 P4
```

The tableau as created by the interpreter is then displayed with

```
TableauMode-- d
```

and will appear with the following format:

```
6 constraints
6 output candidates
2 candidate sets

constraints:
            P->C2 P/->C1 alpha->C2 X->C1 A->C1 A/->C2
outputs:
    A1      _       _       *       _       _       _
    A2      _       _       _       *       *       *
    A4      _       _       *       *       *
    P1      *       *       *                       _
    P2                              _       _       _
    P4      _       _       _       *       _       _
            *       _       *       *       _       _
candidate sets:
    A     3 A1    A2    A4
    P     3 P1    P2    P4
```

From the discussion in Legendre et al. (p.468), we know that under no ranking of the constraints can **A4** or **P4** be winning outputs. These occulted outputs can be identified automatically and removed from the tableau in the Interpreter by employing

```
TableauMode-- occulted_remove
```

It is easily seen in this simple example by applying the Cancellation Domination Lemma (Prince & Smolensky 1993) that the current ranking of the constraints (from **P->C2** highest to **A/->C2** lowest) will produce **A2** and **P2** as winners in their respective candidate sets. This result can be produced in the Interpreter by requesting that it

```
TableauMode-- display winners
```

This will create the following tableau display, with '>>' marking the winner in each set:

```
                P->C2 P/->C1 alpha->C2 X->C1 A->C1 A/->C2
A
    A1          _     _      *
                                        _      _      _
    >>A2        _     _      _          *      *      *
P
    P1          *     *      *
                                        _      _      _
    >>P2        _     _      _          *      _      _
```

Having winners **A2** and **P2** means either an Agent or a Patient argument in an intransitive sentence would be mapped to the Case that a Patient appears in in a transitive sentence. Thus, this constraint ranking defines a possible Ergative/Absolutive language. However, it is also possible to get **A1** and **P1** to win, the outcome for a Nominative/Accusative language, where all intransitive arguments appear in the Case of transitive Agents. Let's play with the constraint ranking to produce this result. We will try promoting **X->C1** to the top of the hierarchy. Moving a constraint to the most dominant position in the hierarchy is accomplished using

```
TableauMode-- front X->C1
```

Then,

```
TableauMode-- d w
```

will identify the new winners, which as desired are **A1** and **P1**:

```
                X->C1 P->C2 P/->C1 alpha->C2 A->C1 A/->C2
A
    >A1         _     _      _                 _      _
                                        *
    A2          *     _      _          _      *      *
P
    >P1         _     *      *          *
                                                _      _
    P2          *     _      _          _      _      _
```

Clearly, there are many rankings which will produce the two combinations of winners from the two sets, {**A1**, **P1**} and {**A2**, **P2**}. The above rankings are examples of each outcome. But are the other output combinations possible (viz. {**A1**, **P2**} and {**A2**, **P1**})? Further, is the set of possible rankings constrained in any way? Legendre et al. prove that the only output combinations possible are {**A1**, **P1**}, {**A2**, **P2**}, and {**A1**, **P2**} , and consequently that *no* ranking will concurrently produce outputs {**A2**, **P1**}. This is the intransitive case marking typology. They also provide an answer to the second question, summarized in the conditions in (3): "choose **C2** [for the output of **P**] unless **X->C1** dominates **P->C2**, **P/->C1**, and **alpha->C2**" and "choose **C1** [for the output of A] unless **alpha->C2** dominates **X->C1**, **A->C1**, and **A/->C2**." This result was arrived at by examining all output options with respect to all constraint orderings.

INTERACTION MODE. Fortunately, in the intransitive case, there were few output options. The problem becomes much more complex when transitive outputs are included in the analysis later on in the paper. In general, the questions posed above are difficult to answer by exhaustive hand analysis when the number of candidate outputs and constraints is large. However, they can be explored in the Optimality Interpreter in interaction mode, to which we will now turn by requesting

```
TableauMode-- Interaction_mode
```

To generate the intransitive typology of case marking, we will construct a composite candidate set consisting of the two candidate sets **A** and **P**.

We need to say a word about composite sets. A set consists of members which are outputs of the same input. Thus, our set **A** has members **A1** and **A2**, the outputs of input **A**. A composite set is a set whose members are n-tuples of outputs, one from each of its constituent sets. The input of the composite set is the product of the sets whose interactions we are exploring. In our case, we will join the sets **A** and **P** into a composite set, by convention called **A.P**. The members of this composite set are pairs (2-tuples) of possible simultaneous winners in sets **A** and **P**, which will be a subset of {(**A1**, **P1**), (**A1**, **P2**), (**A2**, **P1**), (**A1**, **P2**)}. the Interpreter can determine what subset is possible.

A composite set can be created using **c**ross command. We will leave the name of the composite set unspecified in the command, so the name generated for the cross of inputs **A** and **P** will be **A.P** by default:

```
InteractionMode-- cross A P
InteractionMode-- d
```

After the **d**isplay command, we are presented with the following information in the Interpreter:

```
---Original candidate sets---
6 constraints: X->C1 P->C2 P/->C1 alpha->C2 A->C1 A/->C2
2 candidate sets
        A:      A2(1)   A1(3)
        P:      P1(1)   P2(3)
---Composite candidate sets---
6 constraints: X->C1 P->C2 P/->C1 alpha->C2 A->C1 A/->C2
1 composite candidate sets
        A.P:    A2.P2(1)        A1.P1(1)        A1.P2(8)
```

This shows us our two sets **A** and **P**, with members {**A1**, **A2**} and {**P1**, **P2**}, respectively, and our composite set **A.P**. The intransitive typology can be read from the last line, where we find the members of the composite set. The three candidate outputs for composite set **A.P** correspond to mapping both **A** and **P** to Case **C1**--(**A1.P1**)--or to Case **C2**--(**A2.P2**)--or mapping **A** to Case **C1** and **P** to Case **C2**--(**A1.P2**). These outputs can all be produced by some ranking, as indicated by their appearance on the last line of the display. The final possible member of **A.P**--(**A2**,**P1**)--cannot be produced by any constraint ranking, and so does not appear in the composite set members. The number in parentheses following each possible output indicates the number of ranking conditions which will produce that output. There will be more about this below.

We we said, the Interpreter fails to find a constraint ordering that will produce the output **A2.P1** in composite candidate set **A.P**, as we know because this output does not appear in the list of possible output candidates for the composite set. The software thus verifies the result of the analysis in Legendre et. al predicting that cross-linguistically the mapping of Agent to Case **C2** and Patient to Case **C1** will not exist as a voice strategy.

FURTHER APPLICATIONS. We might ask whether the candidate set can be reduced and still produce the desired typology. We may suspect that simplification is possible, given the apparent redundancy of the constraints; however, reduction is no simple matter when doing an analysis by hand. Such a question can be readily explored with the Optimality Interpreter, as we will now see.

It is relatively easy to remove or add constraints, though before we modify our tableau, we may wish to return to tableau mode and save it for future use, since we know it produces the desired analysis. We will save it in file 'SixConst':

```
InteractionMode-- t
TableauMode-- save SixConst
```

Let's experiment by removing some constraints intuition tells us may be superfluous. For example, there is possible redundancy in the constraint pairs **A->C1**, **P/->C1** and **P->C2**, **A/->C2**. We don't need to specify both that

one input gets mapped to a particular Case and that the other input does not. We'll get rid of one of each of these pairs, the negative ones:

```
TableauMode-- r c A/->C2
TableauMode-- r c P/->C1
```

There is perhaps further redundancy in **alpha->C2** and **P->C2**. Let's remove one of these as well, and then see if we've gone too far by re-crossing the sets in interaction mode. Note that in removing the constraint **P->C2** we use the wildcard character '*'. The command will remove all constraints beginning "**P**"; since there is only one, this is equivalen to "r c P->C2".

```
TableauMode-- r c P*
TableauMode-- i
InteractionMode-- c A P
InteractionMode-- d
```

We are shown

```
---Original candidate sets---
3 constraints: X->C1 alpha->C2 A->C1
2 candidates
 A:      A2(1)       A1(2)
 P:      P1(1)       P2(1)
---Composite candidate sets---
3 constraints: X->C1 alpha->C2 A->C1
1 composite candidate sets
 A.P:    A2.P2(1)     A1.P1(1)     A1.P2(1)
```

The possible output combinations in composite set **A.P** are the same as before. We conclude that these three constraints are sufficient to generate the intransitive typology. The reader can experiment with further or different changes to the candidate set and verify that three constraints is the minimum to which the set can be reduced, though not necessarily these three.

We now know that orderings exist for the three composite candidate set output combinations **A1.P1**, **A2.P2**, and **A1.P2** (and indeed for the simple set outputs **A1**, **A2**, **P1**, and **P2**). As noted, the number in parentheses after each output tells us how many different orderings conditions will produce these outputs. But what are the conditions? This information is stored in the **c**ross_output DAG created in crossing **A** and **P** and can be viewed via

```
InteractionMode-- display full
```

Before seeing what the **c**ross_output looks like in this example, we will discuss the representation of DAGs employed by the Interpreter in some detail.

A **c**ross_output display is an encoding of the minimal conditions for a set of candidate outputs (each chosen from a separate candidate set for a separate input) to be the winners in a composite candidate set. The conditions are represented as a set of directed acyclic graphs (DAGs). The set represents disjunction and the graphs conjunction of the conditions. The nodes on the graphs represent constraints and the edges represent required dominations. Any non-specified constraints can be in any position in the ranking. These sets are the minimal sets, meaning that all orderings that give the specified outputs will satisfy one of the graphs, and conversely that all orderings that satisfy one of the graphs will give the specified outputs.

As an example, the output from crossing output sets **One** and **Two**, containing two graphs, is shown below in Interpreter representation, produced as an excerpt of **d**isplay **f**ull.

```
…
candidate set One.Two:
    candidate Alpha.Bravo:
```

```
            [ A>>B  C>>D ]
            [ B>>C ]
…
```

The Interpreter's convention is that square brackets represent conjunction and separate lines indicate disjunction. Thus, this display tells us that for **Alpha** and **Bravo** to both be winning outputs from their respective candidate sets, these conditions must hold on the constraints **A**, **B**, **C**, and **D**:

A must dominate **B** AND **C** must dominate **D**, OR **B** must dominate **C**.

Example rankings that would give **Alpha** and **Bravo** as winners would then be:

{**A**,**B**,**C**,**D**} or {**A**,**C**,**B**,**D**} or {**A**,**C**,**D**,**B**} ,

satisfying **A**>>**B** and **C**>>**D**, or

{**B**,**C**,**A**,**D**} or {**A**,**D**,**B**,**C**} or {**B**,**A**,**D**,**C**}

satisfying **B**>>**C**. Indeed, any ordering that satisfied the above logical conditions would give **Alpha** and **Bravo** as winners in their sets.

We can now return to our example and view the full **c**ross_output display for sets **A** and **P** and composite set **A.P:**

```
---Original candidate sets---
3 constraints:  X->C1 alpha->C2 A->C1
candidate set A
     candidate A2
[  alpha->C2 >> {X->C1,A->C1}  ]
     candidate A1
[  A->C1 >> alpha->C2  ]
[  X->C1 >> alpha->C2  ]
candidate set P
     candidate P1
[  X->C1 >> alpha->C2  ]
     candidate P2
[  alpha->C2 >> X->C1  ]
---Composite candidate sets---
3 constraints:  X->C1 alpha->C2 A->C1
composite candidate set A.P
     composite candidate A2.P2
[  alpha->C2 >> {X->C1,A->C1}  ]
     composite candidate A1.P1
[  X->C1 >> alpha->C2  ]
     composite candidate A1.P2
[  alpha->C2 >> X->C1   A->C1 >> alpha->C2 ]
```

The conditions on mappings of Agents and Patients to their output options are given under "Original candidate sets". The interaction of these two output sets is found under "Composite candidate sets". Here we see the conditions producing each of the language types in the intransitive typology can now be read below "composite candidate set A.P", interpreted here:

Language type	Output Constraint conditions
Erg/Abs (**A2**.**P2**)	**alpha->C2** dominates both **X->C2** and **A->C1**

Nom/Acc (**A1**.**P1**) **X->C1** dominates **alpha->C2**
Act/Stat (**A1**.**P2**) **alpha->C2** dominates **X->C1**,
and **A->C1** dominates **alpha->C2**

This completes the tutorial. We exit the program and terminate our Interpreter session:

```
InteractionMode-- quit
```

4.0 A DESCRIPTION OF THE MAJOR SOFTWARE ALGORITHMS. Those wishing to understand the data structures and algorithms underlying the functionality described in this document will find this section informative. However, this section is not necessary reading to allow one to use the Optimality Interpreter.

4.1 OCCULTED REMOVAL. To find and remove all the occulted outputs, the interpreter compares every pair of candidates within each candidate set and removes occulted candidates from the set. By definition, a candidate **A** is occulted by another candidate **B** if and only if the number of marks against **A** is greater than or equal to the number of marks against **B** for every constraint **c** in the tableau. Symbolically,

$$\mathbf{A} \text{ occults } \mathbf{B} \Leftrightarrow \forall\ \mathbf{c}\text{: } m_{\mathbf{c}}(\mathbf{A}) \geq m_{\mathbf{c}}(\mathbf{B}),$$

where $m_C(\mathbf{A})$ is the number of marks incurred against candidate **A** by constraint **c.**

Optimality theory does not specify what result should obtain if the marks against two candidates are entirely equal, (i.e. $\forall\ \mathbf{c}\text{: } m_{\mathbf{c}}(\mathbf{A}) = m_{\mathbf{c}}(\mathbf{B})$), but these candidates would clearly be redundant. The software will arbitralily eliminate one of the candidates, **A** or **B**.

4.2 WINNER SPECIFICATION. To determine the most harmonic of a set of candidate outputs, it is sufficient to compare the candidates pair-wise. The more harmonic candidate of a pair is determined by comparing the number of marks the two candidates incur for each constraint, beginning with the highest ranked constraint. As long as the numbers of marks against the candidates are equal, the comparison proceeds down the heirarchy. The first constraint against which the candidates have incurred an unequal number of marks terminates the algorithm: the candidate with the least number of marks is the most harmonic of the pair.

Using this pair-wise comparison, the program begins by comparing the first two candidates. The winner of this is compared with the next candidate, and so on. The final winner is the one that wins the last comparison, and is the most harmonic candidate of the candidate set--the winning candidate.

4.3 CONSTRUCTION OF DOMINATION GRAPHS. A "domination graph" is a representation of a set of conjoined constraint orderings. Each node in a graph represents a constraint $\mathbf{c_i}$, and the (directed) edge from $\mathbf{c_i}$ to $\mathbf{c_k}$ indicates that constraint $\mathbf{c_i}$ must be more highly ranked than $\mathbf{c_k}$ in the constraint ordering, represented symbolically in OT as $\mathbf{c_i}$>>$\mathbf{c_k}$ (read "$\mathbf{c_i}$ dominates $\mathbf{c_k}$"). For example, the graph

$$\mathcal{g}\text{: } [\ \mathbf{c_i}>>\mathbf{c_j}>>\mathbf{c_k} \wedge \mathbf{c_m}>>\mathbf{c_n}\ ]$$

requires that $\mathbf{c_i}$>>$\mathbf{c_j}$, $\mathbf{c_j}$>>$\mathbf{c_k}$, and $\mathbf{c_m}$>>$\mathbf{c_n}$.

Given two sets of graphs $\mathcal{g}'$ and $\mathcal{g}''$, the "product" of the sets is formed by taking the union of each graph of one set with each graph of the other. Thus, if $\mathcal{g}' = \{\mathcal{g}_1,\mathcal{g}_2\}$ and $\mathcal{g}'' = \{\mathcal{g}_3,\mathcal{g}_4\}$,

$$\mathcal{g}' * \mathcal{g}'' = \{\mathcal{g}_1,\mathcal{g}_2\} * \{\mathcal{g}_3,\mathcal{g}_4\} = \{\mathcal{g}_1\cup\mathcal{g}_3,\ \mathcal{g}_1\cup\mathcal{g}_4,\ \mathcal{g}_2\cup\mathcal{g}_3,\ \mathcal{g}_2\cup\mathcal{g}_4\}$$

The product represents the intersection of the two sets of domination graphs, and it can be seen that a constraint ordering will satisfy the product of two sets if and only if it satisfies both sets. In forming the product, the union of two graphs may produce inconsistent domination graphs, those containing a cycle that can never be satisfied by any linear ordering of the constraints. These cyclic graphs are discarded.

Each candidate output **A** (or composite candidate) can be associated with a set of domination graphs G_A such that satisfying any $g \in G_A$ will allow **A** to be the winner in its candidate set, and no ordering not satisfying some $g \in G_A$ will allow **A** to win.[3] Note that it is sufficient that the ordering requirements of only *one* of the graphs in the set be satisfied for cadidate **A** to be the most harmonic output.

The disjunctive domination graph set G_A for a candidate output **A** is constructed by first computing pair-wise domination graph sets for **A** over each other candidate. For example, given candidates **A**, **B**, and **C**, we construct the graph set $g_{A>B}$ that will make **A** more harmonic than **B**, and the graph set $g_{A>C}$ that will make **A** more harmonic than **C**. These sets can then be used to form G_A by finding the product of the two graph sets, $g_{A>B} * g_{A>C}$. The product will give us exactly those orderings that satisfy both sets, and hence make **A** more harmonic than **B** *and* more harmonic than **C**.

To construct the pair-wise domination graph set $g_{A>B}$ describing the orderings for which candidate **A** is more harmonic than candidate **B**, we first note that, as a necessary and sufficient condition for this to obtain, a constraint giving a mark to **B** must dominate all constraints giving a mark to **A**. Thus, the pair-wise domination graph is simply the set of orderings, one for each mark of **B**, where the mark-giving constraint dominates all constraints giving any marks to **A**. For example, given the following tableau

	c_1	c_2	c_3	c_4	c_5
A	*		*		*
B		*		*	*
C		*	*		

the pair-wise domination graph set $g_{A>B}$ allowing **A** to be more harmonic than **B** would consist of the graph

$$[\ c_2 >> c_1 \wedge c_2 >> c_3],$$

indicating that c_2 dominates c_1 and c_3, and the graph

$$[c_4 >> c_1 \wedge c_4 >> c_3].$$

The satisfaction of either graph will make **A** more harmonic than **B**. Note that the constraint c_5 is ignored, as the two candidates both incur the same mark against it. Similarly, the pair-wise domination graph set $g_{A>C}$ would be

$$g_{A>C} = \{\ [\ c_2 >> c_1 \wedge c_2 >> c_5] \vee [c_3 >> c_1 \wedge c_3 >> c_5]\ \}.$$

We construct the domination graph set for **A** by taking the product of the pair-wise sets, as discussed:

$$\begin{aligned} G_A = \{&[\ c_2 >> c_1 \wedge c_2 >> c_3 \wedge c_2 >> c_5\ ] \vee \\ &[\ c_2 >> c_1 \wedge c_2 >> c_3 \wedge c_3 >> c_1 \wedge c_3 >> c_5\ ] \vee \\ &[\ c_4 >> c_1 \wedge c_4 >> c_3 \wedge c_2 >> c_1 \wedge c_2 >> c_5\ ] \vee \\ &[\ c_4 >> c_1 \wedge c_4 >> c_3 \wedge c_3 >> c_1 \wedge c_3 >> c_5\ ]\ \} \end{aligned}$$

Similarly, given another candidate set of **D**, **E**, and **F** related through the following tableau with the same constraints,

[3]The domination graphs $g \in G_A$ are not necessarily disjoint. A linear order of constraints may satisfy more than one graph g.

	c_1	c_2	c_3	c_4	c_5
D			*		
E		*			
F				*	*

the set of disjunct domination graphs G_D would be:

$$G_D = \{ [c_2 >> c_3 \wedge c_4 >> c_3] \vee [c_2 >> c_3 \wedge c_5 >> c_3] \}$$

For interaction mode, the crossing of candidate sets involves the product of sets of domination graphs, with the elimination of inconsistent orderings. Thus,

$$\begin{aligned} A \times D = G_A * G_D = \{ & \\ & [\, c_2 >> c_3 \wedge c_4 >> c_3 \wedge c_2 >> c_1 \wedge c_2 >> c_5 \,] \vee \\ & [\, c_2 >> c_3 \wedge c_5 >> c_3 \wedge c_2 >> c_1 \wedge c_2 >> c_5 \,] \vee \\ & [\, c_4 >> c_3 \wedge c_2 >> c_1 \wedge c_2 >> c_3 \wedge c_3 >> c_1 \wedge c_3 >> c_5] \vee \\ & [\, c_2 >> c_3 \wedge c_4 >> c_3 \wedge c_4 >> c_1 \wedge c_2 >> c_1 \wedge c_2 >> c_5] \vee \\ & [\, c_2 >> c_3 \wedge c_5 >> c_3 \wedge c_4 >> c_1 \wedge c_4 >> c_3 \wedge c_2 >> c_1 \wedge c_2 >> c_5 \,] \vee \\ & [\, c_2 >> c_3 \wedge c_4 >> c_3 \wedge c_4 >> c_1 \wedge c_3 >> c_1 \wedge c_3 >> c_5] \ \} \end{aligned}$$

The two unions conjoining $c_3 >> c_5$ and $c_5 >> c_3$ (the second and fourth members--i.e. rows--of G_A with the second member of G_D) have been eliminated as inconsistent

The domination graphs are additionally reduced to a minimal state in two steps. First, redundant edges of each domination graphs are removed. An edge is redundant if the required domination it represents is implicit in the other edges. For example, in the above set G_A, the second graph contains the redundant edge $c_2 >> c_1$. The redundancy arises because the pair of edges $c_2 >> c_3$ and $c_3 >> c_1$ already entail $c_2 >> c_1$. (Similarly, in the fourth graph we can remove .$c_4 >> c_1$) Thus, the reduced set of domination graphs for **A** would be

$$\begin{aligned} G_A = \{ & [\, c_2 >> c_1 \wedge c_2 >> c_3 \wedge c_2 >> c_5 \,] \vee \\ & [\, c_2 >> c_3 \wedge c_3 >> c_1 \wedge c_3 >> c_5 \,] \vee \\ & [\, c_4 >> c_1 \wedge c_4 >> c_3 \wedge c_2 >> c_1 \wedge c_2 >> c_5 \,] \vee \\ & [\, c_4 >> c_3 \wedge c_3 >> c_1 \wedge c_3 >> c_5 \,] \} \end{aligned}$$

The reduced set of domination graphs for **D** would remain the same. Finally, the reduced set of domination graphs for $A \times D$ would be

$$\begin{aligned} A \times D = \{ & [\, c_2 >> c_3 \wedge c_4 >> c_3 \wedge c_2 >> c_1 \wedge c_2 >> c_5 \,] \vee \\ & [\, c_5 >> c_3 \wedge c_2 >> c_1 \wedge c_2 >> c_5 \,] \vee \\ & [\, c_4 >> c_3 \wedge c_2 >> c_3 \wedge c_3 >> c_1 \wedge c_3 >> c_5] \vee \\ & [\, c_2 >> c_3 \wedge c_4 >> c_3 \wedge c_4 >> c_1 \wedge c_2 >> c_1 \wedge c_2 >> c_5] \vee \\ & [\, c_5 >> c_3 \wedge c_4 >> c_1 \wedge c_4 >> c_3 \wedge c_2 >> c_1 \wedge c_2 >> c_5] \vee \\ & [\, c_2 >> c_3 \wedge \ c_4 >> c_3 \wedge c_3 >> c_1 \wedge c_3 >> c_5] \ \} \end{aligned}$$

The second reduction applied to the domination graph sets is removal of redundant graphs from the set. A redundant graph is one whose set of edges entirely contains the edges of another graph.[4] This smaller graph accepts all orderings of that the larger graph does and additionally accepts other orderings. We thus accept the same set of orderings without the larger graph, and it can safely be removed. By this process we can eliminate from $\mathbf{A} \times \mathbf{D}$ the domination graphs alternates (i.e. lines, in the above display) three, four, and six, all made redundant by alternate one, and also alternate five, contained within alternate two. This gives us the following minimum set of domination graphs for $\mathbf{A} \times \mathbf{D}$:

$$\mathbf{A} \times \mathbf{D} = \{ [\ \mathbf{c_2}{>>}\mathbf{c_3} \wedge\ \mathbf{c_2}{>>}\mathbf{c_1} \wedge\ \mathbf{c_2}{>>}\mathbf{c_5} \wedge\ \mathbf{c_4}{>>}\mathbf{c_3}\ ] \vee [\ \mathbf{c_2}{>>}\mathbf{c_1} \wedge\ \mathbf{c_2}{>>}\mathbf{c_5} \wedge\ \mathbf{c_5}{>>}\mathbf{c_3}\ ] \}$$

[4]The set of edges we must examine is the transitive closure or probability expansion of the two graphs. The closure of graph 1 in $\mathbf{A} \times \mathbf{D}$ is [$\mathbf{c_2}{>>}\mathbf{c_3} \wedge \mathbf{c_4}{>>}\mathbf{c_3} \wedge \mathbf{c_2}{>>}\mathbf{c_1} \wedge \mathbf{c_2}{>>}\mathbf{c_5}$] the same as in the cross product; the closure of graphs 3, however, is [$\mathbf{c_4}{>>}\mathbf{c_3} \wedge \mathbf{c_2}{>>}\mathbf{c_3} \wedge \mathbf{c_3}{>>}\mathbf{c_1} \wedge \mathbf{c_3}{>>}\mathbf{c_5}\ \mathbf{c_4}{>>}\mathbf{c_1} \wedge \mathbf{c_2}{>>}\mathbf{c_1} \wedge \mathbf{c_2}{>>}\mathbf{c_5} \wedge \mathbf{c_4}{>>}\mathbf{c_5}$]. The edges in the closure of 1 are clearly a subset of those in 3.

APPENDIX

Command Summary for the Optimality Interpreter

Below is a summary of the commands available in the Optimality Interpreter. The commands are entered by typing the complete name or by using the initial-letter abbreviations (indicated in **u**nderscored **b**old) of each separate word of the command name not joined by an underscore.

Command parameters are indicated by CAPITALs. Optional parameters are enclosed in brackets ([]). Curly braces ({}) indicate zero or more repetitions of a parameter. The parameters formats are:

TABLEAU_FILE, GRAPH_FILE: a legal file name for your system

SET, OUTPUT, CONSTRAINT, CROSS_OUTPUT: a string of any non-whitespace (i.e. anything but space and tab) characters except these five: .?*_+

MARK: a string of marks, with * or + indicating a mark is incurred and _ indicating no mark. (N.B.: MARKs for consecutive constraints or outputs must be separated by whitespace, since multiple MARKs are allowed for any constraint/output combination.)

A pound sign following a parameter indicates that wildcarding is available for identifying that string. The wildcards are:

* for a string of zero or more characters

? for any single character.

COMMANDS FOR TABLEAU MODE

save TABLEAU_FILE

Save the current tableau in text file TABLEAU_FILE on your system.

load TABLEAU_FILE

Load a tableau from the file TABLEAU_FILE. (N.B.: the current tableau will be overwritten!)

add **s**et SET {OUTPUT#}

Create a set named SET if not already present and add any specified OUTPUTs from the current tableau to SET.

Ex: add set g1 a b c

creates a set of candidate outputs ‘g1’ and adds outputs ‘a’, ‘b’, and ‘c’ to it.

add **o**utput OUTPUT {MARK}

Add a candidate OUTPUT to the current tableau and optionally indicate the marks incurred by OUTPUT as MARK. The ordering of MARKs corresponds to the current constraint ordering, and MARKs for consecutive constraints must be separated by whitespace. See above for the formate of {MARK}.

Ex: a o out1 _ * * _

Enters an output called ‘out1’ in the current tableau and gives it marks against the second and third constraints.

add **c**onstraint CONSTRAINT {MARK}

Add a CONSTRAINT at the bottom of the constraint ranking (i.e. at the right edge of the current tableau, dominated by all other constraints) optionally adding a column of MARKs, one for each output. MARKs for consecutive outputs must be separated by whitespace.

add **m**ark [CONSTRAINT#] OUTPUT# {MARK}

Add a MARK incurred by OUTPUT for CONSTRAINT in the current tableau. If no CONSTRAINT is specified, MARK is incurred against *all* constraints in the current tableau (i.e. a column of MARKs is entered). If MARK is not specified, it is assumed to be a single *, but if MARK is omitted, CONSTRAINT must be specified.

Ex: a m const1 out1 *

Adds a mark for candidate 'out1' against constraint 'const1'.

remove **s**et SET# {OUTPUT#}

Remove OUTPUTs from SET, and remove SET from tableau if the SET is then empty or if OUTPUT is not specified.

remove **o**utput OUTPUT#

Remove OUTPUT from the current tableau.

remove **c**onstraint CONSTRAINT#

Remove CONSTRAINT from the current tableau.

display

Display the current tableau.

display **w**inners

Display current tableau, indicating the winning output candidates in each set for the current constraint ranking with '>>'.

perpetual_display

Toggle between display of the current tableau after each command and no display of the tableau except on request (via **d**isplay).

occulted_remove

Remove candidates that are occulted (i.e. cannot be winners under any constraint ranking) in each candidate output set.

mode _change
interaction_mode

Change mode to interaction mode.

exchange CONSTRAINT# CONSTRAINT#

Change the constraint ranking by interchanging the positions in the current tableau of the specified CONSTRAINTs

front {CONSTRAINT#}

Change the constraint ranking by moving CONSTRAINT(s) to the top of the ranking (i.e. to the left edge of the current tableau, dominating all other constraints).

back {CONSTRAINT#}

Change constraint ranking by moving CONSTRAINT(s) to the bottom of the ranking (i.e. to the right edge of the current tableau, dominated by all other constraints).

winner_ordering {[SET:]OUTPUT#}

Display the minimum constraint ranking(s) necessary to allow the specified OUTPUTs (in SET, followed by a ":") be winners.

new_tableau

Clear the current tableau entirely to begin a new analysis.

quit

Exit the Optimality Interpreter.

COMMANDS FOR INTERACTION MODE

save GRAPH_FILE

Save current graphs in a text file GRAPH_FILE

load GRAPH_FILE

Load a (previously **s**aved) text file GRAPH_FILE for use in interaction mode.

remove SET# {CROSS_OUTPUT#}

Remove CROSS_OUTPUT from composite SET, and remove SET if the composite SET is then empty or if CROSS_OUTPUT is not specified.

cross {SET#{:CROSS_OUTPUT#}} [:SET]

Given two SETs One and Two 'cross One Two' creates a composite set (named by the parameter SET after the ':', or named 'One.Two' if no name is specified) that contains all possible simultaneous winners from the SETs One and Two. This command requires at least two parameters.

display

Display current graphs, showing composite candidate sets with names of cross_outputs.

display **f**ull

Display current graphs, showing full specification of cross_outputs. The conventions of the display are that square brackets represent the conjunction of conditions and separate lines represent the disjunction of conditions. Curly braces associate conjunction.

Ex: [A >> {B, C} B >> D]
[D >> A]

This is to be interpreted as "A dominates B AND A dominates C AND B dominates D, OR D dominates A."

mode_change
tableau_mode

Change current mode to tableau mode.

new_interactions

Clear current graphs entirely.

quit

Exit the Optimality Interpreter.

REFERENCES

Legendre, Géraldine, William Raymond, & Paul Smolensky. 1993. Analytic typology of case marking and grammatical voice. BLS 19.464-478

Prince, Alan & Paul Smolensky. 1993. Optimality Theory: Constraint interaction in generative grammar. Technical Report CU-CS-696-93, Department of Computer Science, University of Colorado at Boulder and Technical Report TR-2 Rutgers Center for Cognitive Science, Rugters University, New Brunswick, NJ. March. To appear in The Linguistic Inquiry Monograph Series; Cambridge, MA: MIT Press.

Trotter, William T. 1992. Cominatorics and partially ordered sets: Dimension Theory. Baltimore, MD: Johns Hopkins University Press.