Taking Primitive Optimality Theory Beyond the Finite State

Daniel M. Albro
Linguistics Department
UCLA

Abstract

Primitive Optimality Theory (OTP) (Eisner, 1997a; Albro, 1998), a computational model of Optimality Theory (Prince and Smolensky, 1993), employs a finite state machine to represent the set of active candidates at each stage of an Optimality Theoretic derivation, as well as weighted finite state machines to represent the constraints themselves. For some purposes, however, it would be convenient if the set of candidates were limited by some set of criteria capable of being described only in a higher-level grammar formalism, such as a Context Free Grammar, a Context Sensitive Grammar, or a Multiple Context Free Grammar (Seki et al., 1991). Examples include reduplication and phrasal stress models. Here we introduce a mechanism for OTP-like Optimality Theory in which the constraints remain weighted finite state machines, but sets of candidates are represented by higher-level grammars. In particular, we use multiple context-free grammars to model reduplication in the manner of Correspondence Theory (McCarthy and Prince, 1995), and develop an extended version of the Earley Algorithm (Earley, 1970) to apply the constraints to a reduplicating candidate set.

1 Introduction

The goals of this paper are as follows:

- To show how finite-state models of Optimality Theoretic phonology (such as OTP) can be extended to deal with non-finite state phenomena (such as reduplication) in a principled way.
- To provide an OTP treatment of reduplication using the standard Correspondence Theory account.
- To extend the Earley chart parsing algorithm to multiple context free grammars (MCFGs).

The basic idea of this approach is to begin with a non-finite-state description of the space of acceptable candidates (*e.g.,* candidates with some sort of reduplication inherent in them, or candidates which are the outputs of a syntactic grammar), and to repeatedly intersect the high-level grammar representing those candidates with finite state machines representing constraints. The intersection operation is one of weighted intersection (where only the set of lowest-weighted candidates survive) in order to model Optimality Theory, and will make use of a modified version of the Earley parsing algorithm.

There are at least two alternative approaches to that which we will propose here: to abandon finite state models altogether and move to uniformly higher-level approaches (*e.g.,* Tesar (1996)), or to modify finite state models minimally to allow for (perhaps limited) reduplication (*e.g.,* Walther (2000)). The first of these alternative approaches deals with context free grammars alone, so it would not be able to model reduplicative effects. Besides this, it seems preferable to stick with finite-state approaches as far as possible, because phonological effects beyond the finite state seem quite rare. The second of these approaches seems reasonable in itself, but it is not suited for the type of analyses for which the approach laid out here is designed. In particular, Walther's approach is tied to One-Level Phonology, a theory which limits itself to surface-true generalizations, whereas the approach here is designed to model Optimality Theory—a system with violable constraints—and in particular Correspondence Theory. Tesar's approach as well, while it is a model of Optimality Theory, does not seem suited to Correspondence Theory. A final argument for using this approach, in preference to one similar to Walther's approach, is that it can be extended to cover other non-finite-state areas of phonology, such as phrasal stress patterns, with no modification to the basic model.

2 Quick Overview of OTP

2.1 Optimality Theory

Optimality Theory (OT), of which OTP is a formalized computational model, is structured as follows, with three components:

1. **Gen:** a procedure that produces infinite surface candidates from an underlying representation (UR)

2. **Con:** a set of constraints, defined as functions from representations to integers

3. **Eval:** an evaluation procedure that, in succession, winnows out the candidates produced by **Gen.**

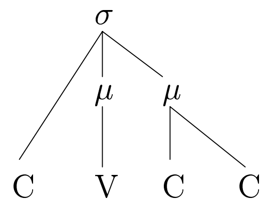
So OT is a theory that deals with potentially infinite sets of phonological representations. The OT framework does not by itself specify the character of these representations, however.

2.2 Primitive Optimality Theory (OTP)

The components of OT, as modeled by OTP (see Eisner (1997a), Eisner (1997b), Albro (1998)):

1. **Gen:** a procedure that produces from an underlying representation a finite state machine that represents all possible surface candidates that contain that UR (always an infinite set)

2. **Con:** a set of constraints definable in a restricted formalism—internally represented as Weighted Deterministic Finite Automata (WDFAs) which accept any string in the representational alphabet. The weights correspond to constraint violations. The weights passed through when accepting a string are the violations incurred by that string.



(a) Conventional

```
σ: [ + + + + + + + ]
μ: - - [ + | + + + ]
C: [ + ] - [ + | + ]
V: - - [ + ] - - - -
C: - - - - [ + | + ]
V: - - [ + ] - - - -
```

(b) OTP

Figure 1: OTP Representation

3. **Eval:** the following procedure, where I represents the input FSM produced by Gen, and M is a machine representing the output set of candidates:

   $M \leftarrow I$
   for all $C_i \in$ **Con**, taken in rank order **do**
   $M \leftarrow$ intersection of M with C_i
   Remove non-optimal paths from M
   Zero out weights in M
   end for

Representations in OTP are gestural scores using symbols from the set $\{-, +, [,], |\}$. See Figure 1 for an example. This figure shows a CVCC syllable in a conventional notation, and also in OTP notation. The OTP notation is slightly more complex, though, in that it also shows an underlying form for the syllable. The overlap relation of the conventional notation's association lines is expressed in the OTP notation by the presence of constituent interiors ("+") in the same vertical slice through the diagram. This same-time-slice-membership relation is also used to show correspondence. Thus we see from this diagram that the surface "CVCC" syllable corresponds to underlying "VCC," and that the initial "C" does not correspond to any underlying segment. Note that tiers with no special marking are used to represent the surface level of representation, and underlined tiers are used to represent the underlying level of representation.

3 Handling Reduplication: Overview

3.1 Overview

Finite State Machines are useful in phonology because it is possible to take any two finite state machines, each of which represents a set of strings, and perform an *intersection* operation on them. The resulting machine represents the intersection of the two sets of strings. For example, this allows us to use constraints represented as FSMs to limit a candidate set.

Although we would sometimes like to characterize the candidate sets using CFGs or MCFGs, it must be kept in mind that these formalisms do not have the property of being intersectable with each other. Thus, in OTP terms, it would not be possible to represent the constraints as CFGs or MCFGs. However, there is a way out: it is possible to intersect an FSM with a CFG or an MCFG.

Based on the above, an approach to handing reduplication in phonology becomes clear—we start with an MCFG that enforces reduplicative identity, then intersect it with the input FSM (produced by **Gen**), then the constraint FSMs, as before. The hard part, then, is to come up with an efficient FSM-intersection algorithm for MCFGs which also deals correctly with weighted FSMs.

3.2 MCFGs

A grammar formalism that is midway between CFGs and CSGs in expressive power, an MCFG is like a CFG except that categories may rewrite to tuples of strings instead of rewriting to just one string as usual. It should be noted that MCFGs have been shown (van Vugt, 1996) to be equivalent to string-valued attribute grammars with only s-attributes, relational grammars, and top-down tree-to-string transducers, so we could use any one of these grammars to provide a candidate space. As an example of an MCFG, here's a simple MCFG for the language $\{ww|w \in \{0,1\}^+\}$ (the language of total reduplication):

$$\begin{array}{lcl} S & \rightarrow & A_0\,A_1 \\ A & \rightarrow & (1,1) \\ & | & (0,0) \\ & | & (0\,A_0, 0\,A_1) \\ & | & (1\,A_0, 1\,A_1) \end{array}$$

The nonterminals of this grammar are S, which has arity 1, and A, which has arity 2. The right-hand sides of the productions include notations such as A_0, which indicate the placement of each part of the tuple-yield of any category. Here, A_0 and A_1 are the two parts of the single category A, so a rule like $A \rightarrow (0\,A_0, 0\,A_1)$ indicates that A rewrites to $0\;0\;A$, with the actual strings arranged in a tuple with a 0 preceding the first part of A in the first half of the pair, and 0 preceding the second part of A in the second half of the pair.

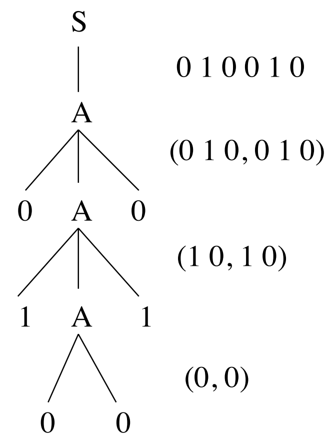


Figure 2: Derivation of "010010"

This grammar is in the normal form required by the algorithms presented here. This normal form can be characterized as follows:

> For any category C of arity greater than 1, the category may appear in the right hand side of a production only if the right hand side refers to each element of C exactly once.

A derivation of the string "010010" in this grammar would go as follows: S rewrites as A_0A_1, that is, to the concatenation of the string-yield of the two parts of A. From here, A_0 and A_1 must both come from the parts of a single one of the four productions for A. A then rewrites to $(0\,A_0, 0\,A_1)$, making, for example, the value of A_0 in the S production be $(0\,A_0)$. A then rewrites to $(1\,A_0, 1\,A_1)$, so S reduces to $01\,A_0\,01\,A_1$. Finally, A rewrites to $(0,0)$, leaving the value of S as 010010. This derivation is illustrated in Figure 2, the left side of which depicts the derivation tree, while its right side shows (from the bottom up) the string-yield of each non-terminal (shown just below and to the right of it).

3.3 Representation of Reduplicative Forms in OTP

OTP constraints are inherently local—they can only refer to overlap or non-overlap of interiors or edges in an instant of time. Therefore, to enforce correspondences between forms, they must be juxtaposed so as to occur in the same time-slices. In OTP, correspondence between the surface and underlying forms is established by using one set of tiers for the surface form (each tier represents either a feature or a type of prosodic constituent) and another corresponding set for the underlying form. For example, the tier *son* might specify the distribution of the surface feature “sonorant”, while the tier $\underline{\textit{son}}$ would specify its underlying correspondent. Elements of those tiers placed in the same time-slice are considered to be in correspondence with one another. In order to create correspondence between two portions of the same surface form, then, we need to somehow have them simultaneously juxtaposed so as to appear in the same segments of time and separated in time as they will be on the surface. This is accomplished by a representational trick: in the example of reduplication, a copy of the reduplicant's surface form is placed in a special set of tiers within the base:

SL:	BASE	RED_2
UL:	UR_1	UR_2
RL:	RED_1	—

— or —

SL:	RED_2	BASE
UL:	UR_2	UR_1
RL:	—	RED_1

In these representations **SL** stands for the surface level of representation, **UL** for the underlying level, and **RL** for the special reduplicant level (the place where a copy of the reduplicant is kept). UR_1 and UR_2 are identical in the input, and RED_1 and RED_2 need to be kept identical by other means. The means chosen here is to use an MCFG enforcing the identity. BASE-RED correspondence constraints operate upon RED_1 while templatic and general surface well-formedness constraints operate upon RED_2. An example of this sort of representation might help here. Suppose that there are two surface tiers, *C* and *V*. Then a form such as [CV+CVC] (with CV prefixing reduplication, assuming that the base is CVC, and with the underlying form RED+/VC/) might be represented as follows:

C:	[+] - - - [+] - [+]
V:	- - [+] - - - [+] - -
$\underline{C}$:	- - - - [+] - - - [+]
$\underline{V}$:	- - [+] - - - [+] - -
$\underline{\underline{C}}$:	- - - - - - [+] - - - -
$\underline{\underline{V}}$:	- - - - - - - - [+] - -
INS:	[+] - - - [+] - - - -
DEL:	- - - - [+] - - - - - -
RDEL:	- - - - - - - - - - [+]
RED:	[+ + + + +] - - - - - -
BASE:	- - - - - - [+ + + + +]

Note here that the special *BASE* and *RED* tiers indicate the portions of the surface forms that are the base and reduplicant, and that the reduplicant level of representation (that is, the level that holds the copy of the reduplicant used for correspondence) is present on the tiers labeled with double underlines. The *INS* tier represents a time-discrepancy between the levels of representation where time does not exist on the underlying level (so the period of time taken up by the initial C in the surface reduplicant and base doesn't correspond to anything in the underlying level), and the *DEL* tier represents time that does not exist on the surface level, so the time taken up by the final C in the underlying form of the reduplicant does not correspond to anything on the surface. The *RDEL* tier is a mirror of the contents of the *DEL* tier in the surface reduplicant, and thus represents time that does not exist in the special reference copy of the reduplicant. This representation allows us to notice that the reduplicant fits a CV template — the left edge of it is aligned with a surface C, the right edge with a surface V, and there are no other segments within it. (The relevant OTP constraints to reinforce this would be “RED[$\rightarrow$ C[,” “]RED $\rightarrow$]V,” “]C $\perp$ C[$\perp$ RED,” and “]C $\perp$ V[$\perp$ RED,” if highly ranked and in that order.)

In terms of translating these representations to finite state machines (or to strings), we use the alphabet $\{-, +, [,], |\}$, so that each FSM edge is labeled with a member of this alphabet. This representation differs from that of earlier accounts of OTP, in that the FSM edges in those accounts represented entire time slices, whereas an edge in this representation represents a single tier in a time slice. As an example, the representation of:

C:	[	$+^*$	]
V:	-	-	-

is as shown in Figure 3, where the "C" and "V" labels are not part of the representation, but just there to ease reading.

3.4 The Grammar Used

The grammar used here is a bit complicated, but the important thing to note about it is that it generates exactly the set of possible OTP output forms in which the special reduplicant reference level of representation contains an exact copy of the surface reduplicant, placed within the time-duration of the base. The grammar for a situation in which there are two surface tiers appears in Figure 4. Extending this grammar to other numbers of tiers is straightforward. The constituents of this grammar are as follows:

S The start symbol.

Non Non-reduplicating material (such as non-reduplicating morphemes) before and/or after the reduplicating material.

SSR The surface tiers in a time-slice.

UR The underlying tiers in a time-slice.

MRD The reduplicant reference-level tiers in a time-slice where the tiers must contain the value $-$ (that is, outside of the base, which is the only place where the reduplicant level is used).

Rd/Rd1/Rd2 The reduplicating part of an utterance.

BDR A right-facing boundary (allows anything to be in the surface tiers during its time-slice, and copies the right-facing half of that material into the reduplicant).

BDL A left-facing boundary (see *BDR*).

B The surface tiers in a time-slice plus identical material in the reduplicant tiers. Thus *B* represents an item in the reduplicant plus its copy in the special reduplicant reference level.

The remaining non-terminals define different values for the INS, DEL, RDEL, RED, and BASE tiers, where INS and DEL are as defined in Albro (1998), RDEL represents time that does not exist in the reduplicant, RED represents the reduplicant (as a morpheme boundary), and BASE represents the base as a morpheme boundary:

NBR represents the state of not being in the base or the reduplicant.

RLE represents the left edge of the reduplicant.

RRE represents the right edge of the reduplicant.

BLE represents the left edge of the base.

BRE represents the right edge of the base.

RB represents a boundary between a reduplicant and a base, where the reduplicant comes first.

BR represents the reverse of *RB*.

RED represents the inside of the reduplicant.

BASE represents the inside of the base.

In this grammar any given time-slice will be defined as *SSR* or the first component of one of the *B* categories, followed by *UR*, followed by *MRD* or the second component of one of the *B* categories, followed by one of the *NBR*, etc., categories.

4 The Earley Algorithm

The Earley algorithm is an efficient chart parsing method. Chart parsing can be seen as a method for taking the intersection of a string or FSM with a CFG (later, an MCFG). Here we take a CFG as a 4-tuple $\langle V, N, P, S\rangle$ where V represents the set of terminals in the grammar, N represents the set of non-terminals, P represents the set of productions, and $S \in N$ is the start symbol. In the definitions to follow, α, β, and γ represent arbitrary members of $(V \cup N)^*$, A and C represent arbitrary members of N, a and b represent arbitrary members of V, p represents an arbitrary member of P, and the indices i, j, and k represent positions within the input string to be parsed, numbered as in Figure 5.

In the standard definition, a member of the chart is a 3-tuple $(i, C \to \alpha \bullet \beta, j)$, where i represents the position at the beginning of the input

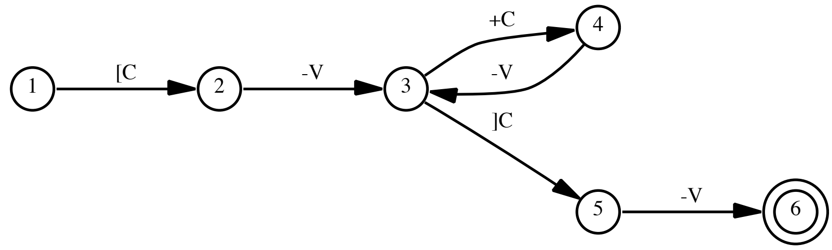


Figure 3: FSM Representation Used Here

string covered by α and j represents the position at the end of the covered portion of the string. The parsing operation in the standard definition, which parses a single input string, is defined as a closure via the following three inference rules of a chart initially consisting of $(0, S \to \bullet\alpha, 0)$:

predict: $\frac{(i,C\to\alpha\bullet A\beta,j)}{(j,A\to\bullet\gamma,j)}$ if $A \to \gamma \in P$ (if γ begins with a terminal, that terminal must be the symbol at position j in the input string)

scan: $\frac{(i,C\to\alpha\bullet a\beta,j)}{(i,C\to\alpha a\bullet\beta,j+1)}$ if a is the symbol after j

complete: $\frac{(i,C\to\alpha\bullet A\beta,j)\ (j,A\to\gamma\bullet,k)}{(i,C\to\alpha A\bullet\beta,k)}$

The input string is recognized if the chart contains an element $(0, S \to \alpha\bullet, n)$, where n is the final position of the input string.

5 Extending Earley

The algorithm presented so far just checks to see whether a particular string exists in a grammar. In order for it to be useful for our purposes, the following extensions must be made:

1. Intersection with an FSM, not just a string
2. Recovery of intersection grammar
3. Weights (intersection should allow lowest-weight derivations only)
4. MCFGs

5.1 Intersection with an FSM

To modify the algorithm to intersect a grammar with an FSM, we replace the input string with an FSM, and change our definition of a chart entry. Now, a chart entry is a 3-tuple $(i, C \to \alpha \bullet \beta, j)$, where i represents the first FSM state covered by α and j represents the last FSM state covered. We define an FSM here as a 5-tuple $\langle Q, \Sigma, s, F, M\rangle$, where Q is the set of states in the FSM, Σ is the label alphabet for the FSM (for our purposes Σ is always the same as V for all grammars in use), $s \in Q$ is the start state, $F \subseteq Q$ is the set of final states of the FSM, and M is a set of 3-tuples (i, a, j), which represent transitions from state i to state j with label a. Given these redefinitions we can then just modify the scan rule:

scan: $\frac{(i,C\to\alpha\bullet a\beta,j)}{(i,C\to\alpha a\bullet\beta,j+1)}$ if $(j, a, k) \in M$, where M is the input FSM.

and the predict rule in the obvious way:

predict: $\frac{(i,C\to\alpha\bullet A\beta,j)}{(j,A\to\bullet\gamma,j)}$ if $A \to \gamma \in P$ (if γ is of the form $a\ \gamma'$, $(j, a, k) \in M$ must hold as well)

Note that the initial entry in the chart is now $(s, S \to \bullet\alpha, s)$.

5.2 Grammar Recovery

It is possible to recover the output of intersection by increasing slightly what is in the chart. In particular, for every item on the chart, we note how it got there (just the last step). Each item on the chart may be referred to by its column number C and its position N within that column. We annotate only items produced by scan and complete steps, as follows:

- sC/N
- $cC_1/N_1; C_2/N_2$

where C_1/N_1 refers to the $(j, A \to \gamma\bullet, k)$ item from the complete step, and C_2/N_2 refers to the

$$
\begin{array}{rcl}
S & \rightarrow & \textit{Non Rd Non} \\
 & | & \textit{Rd Non} \\
 & | & \textit{Non Rd} \\
 & | & \textit{Rd} \\
\textit{Non} & \rightarrow & \textit{SSR UR MRD NBR} \\
 & | & \textit{Non SSR UR MRD NBR} \\
\textit{SSR} & \rightarrow & A\,A \\
\textit{UR} & \rightarrow & A\,A \\
\textit{MRD} & \rightarrow & -\,- \\
\textit{Rd} & \rightarrow & \textit{Rd1}_0\ \textit{Rd1}_1 \\
\textit{BDR} & \rightarrow & \begin{pmatrix} \textit{BDR0}_0\ \textit{BDR1}_0, \\ \textit{BDR0}_1\ \textit{BDR1}_1 \end{pmatrix} \\
\textit{BDL} & \rightarrow & \begin{pmatrix} \textit{BDL0}_0\ \textit{BDL1}_0, \\ \textit{BDL0}_1\ \textit{BDL1}_1 \end{pmatrix} \\
B & \rightarrow & \begin{pmatrix} \textit{B0}_0\ \textit{B1}_0, \\ \textit{B0}_1\ \textit{B1}_1 \end{pmatrix} \\
A & \rightarrow & - \mid + \mid [\mid] \mid | \\
\textit{BDR}_n & \rightarrow & \left(\begin{array}{c|c|c|c|c} -, & +, & [, &], & |, \\ - & + & [& - & [\end{array} \right) \\
\textit{BDL}_n & \rightarrow & \left(\begin{array}{c|c|c|c|c} -, & +, & [, &], & |, \\ - & + & - &] &] \end{array} \right) \\
B_n & \rightarrow & \left(\begin{array}{c|c|c|c|c} -, & +, & [, &], & |, \\ - & + & [&] & | \end{array} \right)
\end{array}
$$

continuing with

$$
\begin{array}{rcllllll}
\textit{NBR} & \rightarrow & A & A & - & - & - \\
\textit{RLE} & \rightarrow & A & A & - & [& - \\
\textit{RRE} & \rightarrow & A & A & - &] & - \\
\textit{BLE} & \rightarrow & A & A & A & - & [\\
\textit{BRE} & \rightarrow & A & A & A & - &] \\
\textit{RB} & \rightarrow & A & A & A &] & [\\
\textit{BR} & \rightarrow & A & A & A & [&] \\
\textit{RED} & \rightarrow & A & A & - & + & - \\
\textit{BAS} & \rightarrow & A & A & A & - & +
\end{array}
$$

In cases where the reduplicant precedes the base, the reduplication rules will appear as follows:

$$
\begin{array}{rcl}
\textit{Rd1} & \rightarrow & \begin{pmatrix} \textit{BDR}_0 & \textit{UR} & \textit{MRD} & \textit{RLE} & \textit{Rd2}_0, \\ \textit{BDL}_0 & \textit{UR} & \textit{BDR}_1 & \textit{RB} & \textit{Rd2}_1 \\ \textit{SSR} & \textit{UR} & \textit{BDL}_1 & \textit{BRE} & \end{pmatrix} \\
\textit{Rd2} & \rightarrow & \begin{pmatrix} B_0 & \textit{UR} & \textit{MRD} & \textit{RED}, \\ \textit{SSR} & \textit{UR} & B_1 & \textit{BAS} \end{pmatrix} \\
 & | & \begin{pmatrix} \textit{Rd2}_0 & B_0 & \textit{UR} & \textit{MRD} & \textit{RED}, \\ \textit{Rd2}_1 & \textit{SSR} & \textit{UR} & B_1 & \textit{BAS} \end{pmatrix}
\end{array}
$$

Otherwise, where the base precedes the reduplicant, the rules will appear as follows:

$$
\begin{array}{rcl}
\textit{Rd1} & \rightarrow & \begin{pmatrix} \textit{SSR} & \textit{UR} & \textit{BDR}_1 & \textit{BLE} & \textit{Rd2}_0, \\ \textit{BDR}_0 & \textit{UR} & \textit{BDL}_1 & \textit{BR} & \textit{Rd2}_1 \\ \textit{BDL}_0 & \textit{UR} & \textit{MRD} & \textit{RRE} & \end{pmatrix} \\
\textit{Rd2} & \rightarrow & \begin{pmatrix} \textit{SSR} & \textit{UR} & B_1 & \textit{BAS}, \\ B_0 & \textit{UR} & \textit{MRD} & \textit{RED} \end{pmatrix} \\
 & | & \begin{pmatrix} \textit{Rd2}_0 & \textit{SSR} & \textit{UR} & B_1 & \textit{BAS}, \\ \textit{Rd2}_1 & B_0 & \textit{UR} & \textit{MRD} & \textit{RED} \end{pmatrix}
\end{array}
$$

Figure 4: Reduplication Grammar

$$
\begin{array}{ccccccc}
 & \text{a} & & \text{b} & & \text{c} & \\
\uparrow & & \uparrow & & \uparrow & & \uparrow \\
0 & & 1 & & 2 & & 3
\end{array}
$$

Figure 5: Numbering of string positions in the string "abc"

$(i, C \rightarrow \alpha \bullet A\beta, j)$ item. A chart item is thus now a 4-tuple $(i, C \rightarrow \alpha \bullet \beta, j, H)$, where H is a set of history items of the type described here, one for each scan or complete step that put the item there.

Recovery of a grammar then starts from the "success items," that is items in the chart that begin in state 1 and end with a final state and represent a production from the start symbol of the grammar, with the Earley position dot at the end of the production. We then move from right to left within those productions, filling in the state pairs for each constituent we pass, and tracing through their productions as well. Whenever we get to the left side of a production, we output it. The exact algorithm is as follows:

GrammarRecovery(*chart*)

> *queue* $\leftarrow$ []
> **for all** *success items* $(s, S \rightarrow \gamma\bullet, f \in F, H_0)$ at (C, N) **do**
>   queue up (C, N) onto *queue*
>   **while** *queue* not empty **do**
> $(C, N) \leftarrow$ dequeue from *queue*
>   *item* $\leftarrow$ item at (C, N): $(i, A \rightarrow \alpha\bullet, j, H_1)$
>   *pos* $\leftarrow$ pos. of $\bullet$ in *item*
>   *RHSs* $\leftarrow$ **GetRHSs**([[]], *item, pos, queue*)
>   **for all** *RHS* $\in$ *RHSs* **do**
>   output "$A(i,j) \rightarrow \textit{RHS}$"
>   **end for**
>   **end while**
> **end for**

GetRHSs(*rhss, item, pos, queue*)

> **if** *pos* $= 0$ **then**
>   return *rhss*
> **end if**
> *new_rhss* $\leftarrow$ []
> **for all** history path components *hitem* of *item* **do**
>   *rhss*$'$ $\leftarrow$ copy *rhss*
>   **extend**(*rhss*$'$, *hitem, pos, queue*)

```
        add rhss' to new_rhss
    end for
    return new_rhss
```

extend(*rhss, hitem, pos, queue*)

```
    if hitem = s(C, N) then
        prepend scanned symbol to each rhs
        ∈ rhss
        prev ← item at (C, N)
    else if hitem = c(C1/N1; C2/N2) then
        (i, A → γ•, j, H) ← item at (C1, N1)
        prepend A(i, j) to each rhs ∈ rhss
        enter (C1, N1) into queue
        prev ← item at (C2, N2)
    end if
    return GetRHSs(rhss, item, pos−1,
    queue)
```

5.3 Weights

The basic idea for handling weights is an adaption from the Viterbi algorithm, as used for chart parsing of probabilistic grammars. Basically, we reduce the grammar to allow only the lowest-weight derivations from each new category.

Implementation: Each chart item has an associated weight, computed as follows:

predict: weight of the predicted rule $A \to \gamma$

scan: sum of the weight of the item scanned from and the weight of the FSM edge scanned across.

complete: sum of the weights of the two items involved

We build new chart items whenever permitted by the rules given in previous sections, assigning weights to them by the above considerations. If no equivalent item (equivalence ignores weight and path to the item) is in the chart, we add the item. If an equivalent item is in the chart, there are three possible actions, according to the weight of the new item:

1. Higher than the old item: do nothing (don't add the new path).
2. Lower than the old item: remove all other paths to the item, add this path to the item. Adjust weights of all items built from this one downward.
3. Same as the old item: add the new path to the item.

A chart item is thus now a 5-tuple $(w, i, C \to \alpha \bullet \beta, j, H)$, where w represents a weight, and all the other items are as before.

5.4 MCFGs

To extend the Earley algorithm to MCFGs, we first reduce the chart-building part of the Earley algorithm for MCFGs to the already-worked out algorithm for CFGs by converting the MCFG into a (not-equivalent) CFG. We then modify the grammar-recovery step to convert the CFG produced into an MCFG, verifying that the MCFG produced is a proper one.

5.4.1 Adjustments to the Chart-Building Algorithm:

First, we treat each part of the rule as a separate rule, and use the regular algorithm. Thus, $B \to \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ becomes $B_0 \to 0$ and $B_1 \to 1$. Having separated a single rule such as $C \to (\alpha, \beta)$ into two parts $C_0 \to \alpha$ and $C_1 \to \beta$, we need to keep track, when building the chart and after, of which rule in the associated MCFG each chart item refers to. These annotations will be useful in Grammar Recovery (something like $C_0 \to \alpha$ can only be combined with $C_1 \to \beta$ if they both come from the same MCFG rule). Thus, a chart item is a 6-tuple $(r, w, i, C \to \alpha \bullet \beta, j, H)$, where r is the rule number from the original MCFG to which the production $C \to \alpha \bullet \beta$ corresponds, and all the others are as before.

5.4.2 Adjustments to Grammar Recovery

As before, followed by a final combinatory and checking step:

```
for all non-terminals A with arity n do
    for all possible combinations A0(i, j) →
    γ0, A1(k, l) → γ1, ..., An(m, n) → γn do
        if the MCFG condition applies to the
        combination then
            output   A(i, j)(k, l) ... (m, n)   →
            (γ0, γ1, ..., γn)
        end if
    end for
end for
```

where the MCFG condition is as follows:

> All γ_i on the right hand side of the combination must be derived from the

same rule in the original set of rules and their yields must not overlap each other in the FSM.

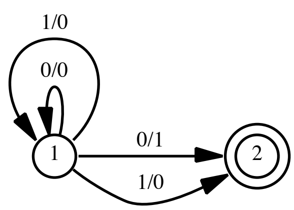
Given the way the chart-parsing and recovery algorithms work, the MCFG condition will be satisfied if we simply check that all the elements of the combination come from the same rule in the original MCFG. This will result in some invalid rules in the output grammar, but this simple check guarantees that these rules will be such that they will be unable to participate in derivations, since their right-hand sides will refer to categories that do not head any productions.

5.5 Example

As an example, let's take the simple reduplication grammar from before:

$$\begin{array}{llll} (1) & S & \rightarrow & A_0\,A_1 \\ (2) & A & \rightarrow & (1,1) \\ (3) & & | & (0,0) \\ (4) & & | & (0\,A_0, 0\,A_1) \\ (5) & & | & (1\,A_0, 1\,A_1) \end{array}$$

and intersect it with the machine



This machine generates the set of strings $\{0|1\}^+$, but weights all strings ending with 0.

The corresponding CFG-grammar used for the chart-building step is as follows:

$$\begin{array}{llll} (1) & S & \rightarrow & A_0\,A_1 \\ (2) & A_0 & \rightarrow & 1 \\ & A_1 & \rightarrow & 1 \\ (3) & A_0 & \rightarrow & 0 \\ & A_1 & \rightarrow & 0 \\ (4) & A_0 & \rightarrow & 0\,A_0 \\ & A_1 & \rightarrow & 0\,A_1 \\ (5) & A_0 & \rightarrow & 1\,A_0 \\ & A_1 & \rightarrow & 1\,A_1 \end{array}$$

The chart produced by the chart-building part of the algorithm is as follows:

Column 1 ($j = 1$, $i = 1$)

#	r	w	$\in P$	H
0	1	0	$S \rightarrow \bullet A_0\,A_1$	$\emptyset$
1	2	0	$A_0 \rightarrow \bullet 1$	$\emptyset$
2	3	0	$A_0 \rightarrow \bullet 0$	$\emptyset$
3	4	0	$A_0 \rightarrow \bullet 0\,A_0$	$\emptyset$
4	5	0	$A_0 \rightarrow \bullet 1\,A_0$	$\emptyset$
5	5	0	$A_0 \rightarrow 1 \bullet A_0$	$\{s1/4\}$
6	4	0	$A_0 \rightarrow 0 \bullet A_0$	$\{s1/3\}$
7	3	0	$A_0 \rightarrow 0\bullet$	$\{s1/2\}$
8	1	0	$S \rightarrow A_0 \bullet A_1$	$\{c1/7;1/0, c1/10;1/0, c1/9;1/0, c1/22;1/0\}$
9	5	0	$A_0 \rightarrow 1\,A_0\bullet$	$\{c1/7;1/5, c1/10;1/5, c1/9;1/5, c1/22;1/5\}$
10	4	0	$A_0 \rightarrow 0\,A_0\bullet$	$\{c1/7;1/6, c1/10;1/6, c1/9;1/6, c1/22;1/6\}$
11	2	0	$A_1 \rightarrow \bullet 1$	$\emptyset$
12	3	0	$A_1 \rightarrow \bullet 0$	$\emptyset$
13	4	0	$A_1 \rightarrow \bullet 0\,A_1$	$\emptyset$
14	5	0	$A_1 \rightarrow \bullet 1\,A_1$	$\emptyset$
15	5	0	$A_1 \rightarrow 1 \bullet A_1$	$\{s1/14\}$
16	4	0	$A_1 \rightarrow 0 \bullet A_1$	$\{s1/13\}$
17	3	0	$A_1 \rightarrow 0\bullet$	$\{s1/12\}$
18	1	0	$S \rightarrow A_0\,A_1\bullet$	$\{c1/17;1/8, c1/20;1/8, c1/19;1/8, c1/21;1/8\}$
19	5	0	$A_1 \rightarrow 1\,A_1\bullet$	$\{c1/17;1/14, c1/20;1/14, c1/19;1/14, c1/21;1/14\}$
20	4	0	$A_1 \rightarrow 0\,A_1\bullet$	$\{c1/17;1/13, c1/20;1/13, c1/19;1/13, c1/21;1/13\}$
21	2	0	$A_1 \rightarrow 1\bullet$	$\{s1/11\}$
22	2	0	$A_0 \rightarrow 1\bullet$	$\{s1/1\}$

Column 2 ($j = 2$, $i = 1$)

#	r	w	$\in P$	H
0	5	0	$A_0 \rightarrow 1 \bullet A_0$	$\{s1/4\}$
1	4	1	$A_0 \rightarrow 0 \bullet A_0$	$\{s1/3\}$
2	3	1	$A_0 \rightarrow 0\bullet$	$\{s1/2\}$
3	1	0	$S \rightarrow A_0 \bullet A_1$	$\{c2/13;1/0, c2/5;1/0, c2/4;1/0\}$
4	5	0	$A_0 \rightarrow 1\,A_0\bullet$	$\{c2/13;1/5, c2/5;1/5, c2/4;1/5\}$
5	4	0	$A_0 \rightarrow 0\,A_0\bullet$	$\{c2/13;1/6, c2/5;1/6, c2/4;1/6\}$
6	5	0	$A_1 \rightarrow 1 \bullet A_1$	$\{s1/14\}$
7	4	1	$A_1 \rightarrow 0 \bullet A_1$	$\{s1/13\}$
8	3	1	$A_1 \rightarrow 0\bullet$	$\{s1/12\}$
9	1	0	$S \rightarrow A_0\,A_1\bullet$	$\{c2/12;1/8, c2/11;1/8, c2/10;1/8\}$
10	5	0	$A_1 \rightarrow 1\,A_1\bullet$	$\{c2/12;1/15, c2/11;1/15, c2/10;1/15\}$
11	4	0	$A_1 \rightarrow 0\,A_1\bullet$	$\{c2/12;1/16, c2/11;1/16, c2/10;1/16\}$
12	2	0	$A_1 \rightarrow 1\bullet$	$\{s1/11\}$
13	2	0	$A_0 \rightarrow 1\bullet$	$\{s1/1\}$

In this chart the items with an empty history list were entered by prediction steps. The "success item" for this grammar is then item (2,9): $(r = 1, w = 0, i = 1, p = S \rightarrow A_0\,A_1\bullet, j =$

$2, H = \{c2/12; 1/8, c2/11; 1/8, c2/10; 1/8\})$, so begin there:

$$S(1,2) \to A_0\, A_1$$

We then queue up (2,12), (2,11), and (2,10), noting that for all of these the states for A_1 are (1,2), and we move to item (1,8): $(r = 1, w = 0, i = 1, p = S \to A_0 \bullet A_1, j = 1, H = \{c1/7; 1/0, c1/10; 1/0, c1/9; 1/0, c1/22; 1/0\})$. Here we queue up (1,7), (1,10), (1,9), and (1,22), noting that for all of these the states for A_0 are (1,1). Moving to (1,0), we note that we are done, and we thus output a complete rule:

$$(r1)\ S(1,2) \to A_0(1,1)\, A_1(1,2).$$

We then encounter (2,12) on the queue: $(r = 2, w = 0, i = 1, p = A_1 \to 1\bullet, j = 2, H = \{s1/11\})$, which can be output with no further ado:

$$(r2)\ A_1(1,2) \to 1$$

Moving to item (2,11) $(r = 4, w = 0, i = 1, p = A_1 \to 0\ A_1\bullet, j = 2, H = \{c2/12; 1/16, c2/11; 1/16, c2/10; 1/16\})$ we don't need to queue anything, and we can see that the output will be:

$$(r4)\ A_1(1,2) \to 0\ A_1(1,2)$$

Item (2,10) is $(r = 5, w = 0, i = 1, p = A_1 \to 1\ A_1\bullet, j = 2, H = \{c2/12; 1/15, c2/11; 1/15, c2/10; 1/15\})$, so we output

$$(r5)\ A_1(1,2) \to 1\ A_1(1,2)$$

We now move on to item (1,7): $(r = 3, w = 0, i = 1, p = A_0 \to 0\bullet, j = 1, H = \{s1/2\})$, which we output as

$$(r3)\ A_0(1,1) \to 0.$$

Item (1,10) is $(r = 4, w = 0, i = 1, p = A_0 \to 0\ A_0\bullet, j = 1, H = \{c1/7; 1/6, c1/10; 1/6, c1/9; 1/6, c1/22; 1/6\})$. In dealing with this we need to queue nothing, and we output:

$$(r4)\ A_0(1,1) \to 0\ A_0(1,1)$$

Moving to (1,9), which is $(r = 5, w = 0, i = 1, p = A_0 \to 1\ A_0\bullet, j = 1, H = \{c1/7; 1/5, c1/10; 1/5, c1/9; 1/5, c1/22; 1/5\})$, we queue nothing and output

$$(r5)\ A_0(1,1) \to 1\ A_0(1,1)$$

Finally we get to (1,22): $(r = 2, w = 0, i = 1, p = A_0 \to 1\bullet, j = 1, H = \{s1/1\})$, which gets output as

$$(r2)\ A_0(1,1) \to 1$$

Collecting these together (for category A), we get the following pairings:

$$\left(\begin{array}{c|lcl|lcl} (r2) & A_0(1,1) & \to & 1 & A_1(1,2) & \to & 1 \\ (r3) & A_0(1,1) & \to & 0 & & & \\ (r4) & A_0(1,1) & \to & 0\ A_0(1,1) & A_1(1,2) & \to & 0\ A_1(1,2) \\ (r5) & A_0(1,1) & \to & 1\ A_0(1,1) & A_1(1,2) & \to & 1\ A_1(1,2) \end{array}\right)$$

Note that the "pair" for (r3) has no second member, so nothing will be output for it. Combining the compatible rules, we get the following grammar:

$$\begin{array}{rcl} S(1,2) & \to & A(1,1)(1,2)_0\, A(1,1)(1,2)_1 \\ A(1,1)(1,2) & \to & (1,1) \\ & | & (0\ A_0(1,1)(1,2), 0\ A_1(1,1)(1,2)) \\ & | & (1\ A_0(1,1)(1,2), 1\ A_1(1,1)(1,2)) \end{array}$$

which is equivalent to the grammar:

$$\begin{array}{rcl} S & \to & A_0\, A_1 \\ A & \to & (1,1) \\ & | & (0\ A_0, 0\ A_1) \\ & | & (1\ A_0, 1\ A_1) \end{array}$$

This grammar indeed represents the best outputs from the intersection—all reduplicating forms which end in a 1.

References

Daniel M. Albro. 1998. Evaluation, implementation, and extension of Primitive Optimality Theory. Master's thesis, UCLA.

Jay Earley. 1970. An efficient context-free parsing algorithm. *Comm. of the ACM*, 6(2):451–455.

Jason Eisner. 1997a. Efficient generation in primitive Optimality Theory. In *Proceedings of the ACL*.

Jason Eisner. 1997b. What constraints should OT allow? Handout for talk at LSA, Chicago, January.

John McCarthy and Alan Prince. 1995. Faithfulness and reduplicative identity. In J. Beckman, S. Urbanczyk, and L. Walsh, editors, *Papers in Optimality Theory*, number 18 in University of Massachusetts Occasional Papers, pages 259–384. GLSA, UMass, Amherst.

Alan Prince and Paul Smolensky. 1993. Optimality Theory: Constraint interaction in generative grammar. Technical Report 2, Center for Cognitive Science, Rutgers University.

H. Seki, T. Matsumura, M. Fujii, and T. Kasami. 1991. On multiple context-free grammars. *Theoretical Computer Science*, 88:191–229.

Bruce Tesar. 1996. Computing optimal descriptions for optimality theory grammars with context-free position structures. In *Proceedings of ACL*.

Nikè van Vugt. 1996. Generalized context-free grammars. Master's thesis, Universiteit Leiden. Internal Report 96-12.

Markus Walther. 2000. Finite-state reduplication in one-level prosodic morphology. In *Proceedings of NAACL-2000*, pages 296–302, Seattle, WA.