Taking Primitive Optimality Theory Beyond the Finite State

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 Goals

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).

2 Quick Overview of OTP

2.1 Optimality Theory

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

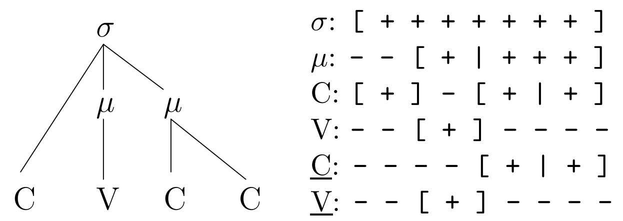
1. **Gen:** a procedure that produces infinite surface candidates from an underlying representation (UR)
2. **Con:** a set of constraints, defined as functions from representations to integers
3. **Eval:** an evaluation procedure that, in succession, winnows out the candidates produced by **Gen.**

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 (WDFA's) which accept any string in the representational alphabet. The weights correspond to constraint violations (the weights passed through when ac-



(a) Conventional

(b) OTP

Figure 1: OTP Representation

cepting a string are the violations incurred by that string).

3. **Eval:** the following procedure, where I represents the input FSM produced by Gen, and M is 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.

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. As an example, here's a simple MCFG for the language $\{ww | w \in \{0,1\}^*\}$ (the language of total reduplication):

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

Here the category S has arity 1, whereas A has arity 2. This grammar is in the normal form the algorithms presented here:

> 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.

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. This is accomplished for reduplication by placing a copy of the reduplicant's surface form 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, 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.

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 2.

3.4 The Grammar Used

A 2-surface-tier grammar will appear as in Figure 3, where S is the start symbol; *Non* represents non-reduplicating material (such as non-reduplicating morphemes); *SSR* represents the surface tiers in a time-slice; *UR* represents the underlying tiers in a time-slice; *MRD* represents the reduplicant tiers in a time-slice where the tiers must contain the value $-$; *Rd*, *Rd1*, and *Rd2* represent the reduplicating part of an utterance; *BDR* represents a right-facing boundary (it allows anything on the surface in its time-slice, and copies the right-facing half into the reduplicant); *BDL* represents a left-facing boundary; B represents the surface tiers in a time-slice plus corresponding reduplicant tiers. 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. Among these, *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 precedes; *BR* represents a boundary between a base and a reduplicant, where the base precedes; *RED* represents the state of being inside the reduplicant; and *BASE* represents the state of being inside the base. Thus, in this grammar any given timeslice 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). The standard definition, which parses a single string, is defined as a closure via the following three inference rules of a chart initially consisting of $(0, S \rightarrow \bullet\alpha, 0)$, where V represents the set of terminals in a grammar, N represents the set of terminals, P represents the set of productions, S is the start symbol, and $\alpha, \beta, \gamma \in (V \cup N)^*$, $A, C \in N, a \in V$):

predict: $\frac{(i,C\rightarrow\alpha\bullet A\beta,j)}{(j,A\rightarrow\bullet\gamma,j)}$ if $A \rightarrow \gamma \in P$ (if γ begins with a terminal, it must be the next symbol)

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

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

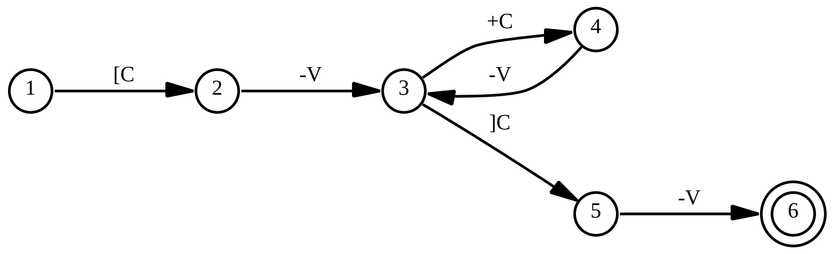


Figure 2: FSM Representation Used Here

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

(1) is pretty simple; 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 we start at column 1 instead of 0, if our FSMs use 1 as their start state.

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 $(i, C \to \alpha \bullet A\beta, j)$ item.

Recovery then follows the following algorithm:

```
GrammarRecovery(chart)
    queue ← []
    for all sucess items $(1, S \to \gamma\bullet, N)$ at $(C, N)$ do
      queue up $(C, N)$ onto queue
      while queue not empty do
        $(C, N)$ ← dequeue from queue
        item ← item at $(C, N)$: $(i, A \to \alpha\bullet, j)$
        pos ← pos. of $\bullet$ in item
        RHSs ← GetRHSs([[]], item, pos, queue)
        for all RHS ∈ RHSs do
          output "$A(i, j) \to$ RHS"
        end for
      end while
    end for

GetRHSs(rhss, item, pos, queue)

    if pos = 0 then
      return rhss
    end if
    new_rhss ← []
    for all history path components hitem of item do
      rhss′ ← copy rhss
```

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

continuing with

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

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

$$\begin{array}{rcl}
\mathit{Rd1} & \rightarrow & \left(\begin{array}{lllll} \mathit{BDR}_0 & \mathit{UR} & \mathit{MRD} & \mathit{RLE} & \mathit{Rd2}_0, \\ \mathit{BDL}_0 & \mathit{UR} & \mathit{BDR}_1 & \mathit{RB} & \mathit{Rd2}_1 \\ \mathit{SSR} & \mathit{UR} & \mathit{BDL}_1 & \mathit{BRE} & \end{array}\right) \\
\mathit{Rd2} & \rightarrow & \left(\begin{array}{llll} B_0 & \mathit{UR} & \mathit{MRD} & \mathit{RED}, \\ \mathit{SSR} & \mathit{UR} & B_1 & \mathit{BAS} \end{array}\right) \\
 & | & \left(\begin{array}{lllll} \mathit{Rd2}_0 & B_0 & \mathit{UR} & \mathit{MRD} & \mathit{RED}, \\ \mathit{Rd2}_1 & \mathit{SSR} & \mathit{UR} & B_1 & \mathit{BAS} \end{array}\right)
\end{array}$$

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

$$\begin{array}{rcl}
\mathit{Rd1} & \rightarrow & \left(\begin{array}{lllll} \mathit{SSR} & \mathit{UR} & \mathit{BDR}_1 & \mathit{BLE} & \mathit{Rd2}_0, \\ \mathit{BDR}_0 & \mathit{UR} & \mathit{BDL}_1 & \mathit{BR} & \mathit{Rd2}_1 \\ \mathit{BDL}_0 & \mathit{UR} & \mathit{MRD} & \mathit{RRE} & \end{array}\right) \\
\mathit{Rd2} & \rightarrow & \left(\begin{array}{llll} \mathit{SSR} & \mathit{UR} & B_1 & \mathit{BAS}, \\ B_0 & \mathit{UR} & \mathit{MRD} & \mathit{RED} \end{array}\right) \\
 & | & \left(\begin{array}{lllll} \mathit{Rd2}_0 & \mathit{SSR} & \mathit{UR} & B_1 & \mathit{BAS}, \\ \mathit{Rd2}_1 & B_0 & \mathit{UR} & \mathit{MRD} & \mathit{RED} \end{array}\right)
\end{array}$$

Figure 3: Reduplication Grammar

```
        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(C_1, N_1; C_2, N_2) then
        (i, A → γ•, j) ← item at (C_1, N_1)
        prepend A(i, j) to each rhs ∈ rhss
        enter (C_1, N_1) into queue
        prev ← item at (C_2, N_2)
    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 $A \rightarrow \gamma$

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)

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.

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, $\mathrm{B}\rightarrow\begin{pmatrix}0\\1\end{pmatrix}$ becomes $\mathrm{B}_0\rightarrow 0$ and $\mathrm{B}_1\rightarrow 1$.

Note: some further modifications to the chart-building algorithm would be helpful to maintain the Earley algorithm's efficiency.

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 $A_0(i,j) \rightarrow \gamma_0, A_1(k,l) \rightarrow \gamma_1, \ldots, A_n(m,n) \rightarrow \gamma_n$ do
    if the MCFG condition applies to the combination then
      output $A(i,j)(k,l)\ldots(m,n) \rightarrow (\gamma_0, \gamma_1, \ldots, \gamma_n)$
    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 (that is, the states from i to j non-inclusive must not contain any of the states from k to l, and so on).

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.