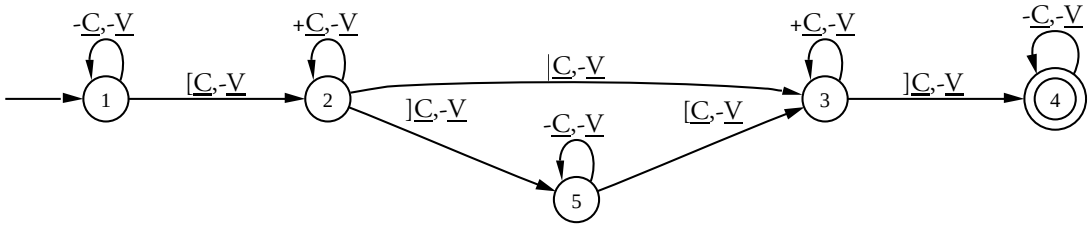
Notes on Primitive Optimality Theory (part 2)

Daniel M. Albro

May 13, 1999

1 Review of last time

- In the OTP conception, GEN is a function that takes an input (a single string) and produces a FSM that accepts all valid OTP representations that include the input (which is on a separate set of tiers from the output).
- There are some problems with syncope and epenthesis for the OTP representation.
- The GEN function for standard OTP produces something like the following for CC:



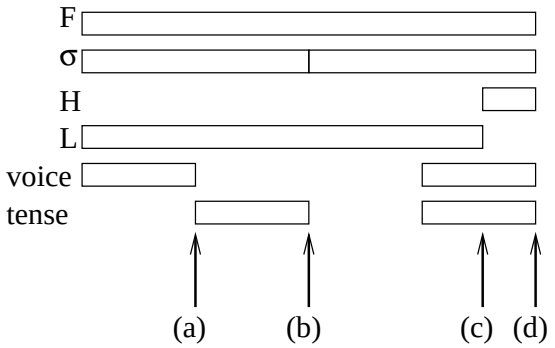
2 Constraints

It's too much to ask, perhaps, that constraints be written directly as WFSMs. Instead, something like the two-level rule notation or the (rather amorphous) one-level phonology notation is needed. Eisner claims that most reasonable constraints have the formulation $\forall x, \exists y$ or $\forall x, \nexists y$ or just $\nexists y$. I think this is based mainly on Generalized Alignment and Correspondence Theory. In GA, we have the family Align(thing1, Edge1, thing2, Edge2), which says something like for every Edge1 of a thing1, there must exist an Edge2 of thing2 such that Edge1 is aligned with Edge2, and in Correspondence Theory constraints say "for every so and so in such and such level of representation, there must exist a corresponding thingee." For these sorts of things, OTP provides a family of constraints called "implication," which say, basically, "for every x, there exists a y such that x and y overlap on the gestural score. Here

x is a conjunction with one or more elements, of which the elements can be interiors, left edges, or right edges, thus "C", "[C", or "]C". Here "C" refers to any part or the entirety of the interior of any "C" constituent, "[C" refers to the left edge (thus it matches "[" and "|"), and "]C" refers to the right edge. Similarly, y is a disjunction with one or more elements, of which the elements can be interiors, left edge, or right edges, as well. See (Eisner 1997) for lots of examples of this. The other basic kind of constraint is the "*" constraint, where we say something isn't permitted. In OTP this is called a "clash" constraint, for some reason, and is noted by the symbol $\perp$. (Don't ask me why... Personally, I would prefer something like "$\oslash$" or "†.") These basically take a single conjunction of the element types given above, and forbid mutual coöccurrence.

2.1 Implementation

Notation: implication is $A \rightarrow B$, where A and B are lists of tier descriptors (interior, left edge, or right edge, of some constituent type), A is conceived as a conjunction, B is conceived as a disjunction. Clash is $A_1 \perp A_2$ where A_1 and A_2 are just parts of a single conjunction (might as well say $\perp A$, the infix notation is for convenience). The different element types are implemented as +x, [|x, or]|x. The implementation makes use of a fancy expression simplifier/disambiguator which takes notations involving the operators **and**, **or**, and **not**, plus "tier descriptors" such as "+[|back," and converts them into edges. In the following examples, A refers to the elements to the left of the implication arrow, or all of the elements in a clash, and B refers to the elements to the right of the implication arrow. A is joined together with the operator **and**, and B is joined together with the operator **or**. Another operator used in the examples is **end** (a prefix operator), which is used only when A is comprised entirely of interiors ("+") — "**end** A" is equivalent to "**and**$(\{+]|\text{tier}\,|\,\text{tier} \in \text{tiersof}(A)\})$ **and not** A." In other words, "**end** A" describes a time-slice such that for each tier in A, the time-slice mentions either the interior of an interval on that tier or a right-edge boundary of an interval, and for at least one tier in A, the time-slice mentions a right-edge boundary. Thus, it describes a situation where a simultaneous interval of all of the members of A has just ended. Example:

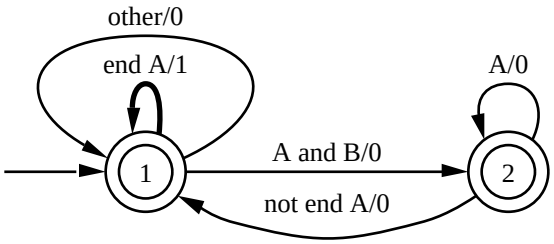


In the above, (a) would match **end** A if A was "F and σ and L and voice" (remember a tier by itself is construed as referring to an interior in these things), "voice," or any other such expression containing

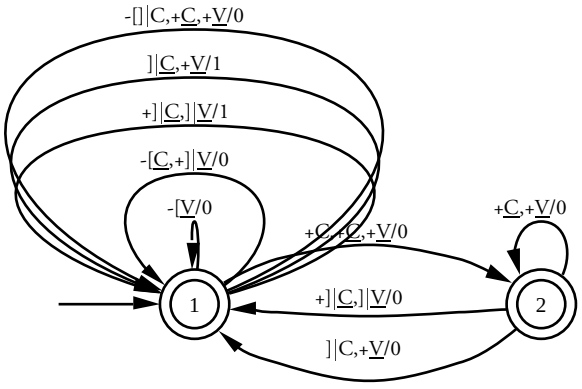
the tier "voice" and not containing the tiers "H" or "tense." (B) would match **end** *A* if *A* contained either "tense" or "σ" but not "voice" or "H", (c) would match **end** *A* if *A* contained "L" and not "H," and (d) would match **end** *A* if *A* contained anything except *L*.

In the original version of OTP, the constraints translate into four different cases. Below are the weighted finite state machines that correspond to the different cases (using the algebra defined above):

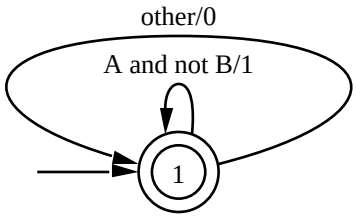
Implication where *A* is made up entirely of interiors Here we want to say that for every instance of an interval in which all the elements of *A* are present at the same time, there must be some member of *B* present during that interval, else a weighted edge must be traversed. Thus:



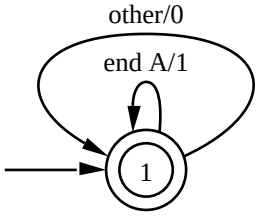
An example is "$\underline{C}$ and $\underline{V} \rightarrow C$":



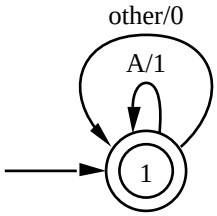
Implication where *A* has edges in it Here we want to say that any edge on which all of *A* appears, but not any of *B*, should have a weight. Note that "other" has its standard meaning of "all possible labels other than those currently specified leading out of this state."



Clash where *A* is all intervals Here we want to say that every instance of an interval in which all elements of *A* are simultaneously present should go through a single weighted edge:

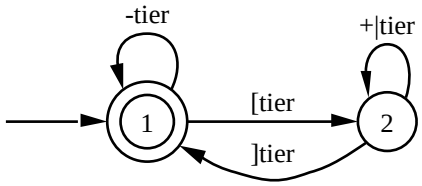


Clash where *A* has edges Here we want to say that every edge where all of *A* is present should be weighted:



2.1.1 Tier FSMs

One more detail: a part of GEN that's implemented as a FSM rather than as a procedure is the enforcement of bracket matching. For each tier there's an FSM like this one:



For each non-underlying tier, the corresponding tier FSM is intersected with the "candidates" FSM just before introducing the first constraint that mentions it.

2.2 Examples from an analysis of Malagasy...

Some examples, in UCLA OTP constraint-file notation of constraints that form part of an analysis of Malagasy reduplication I've been working on. Explication will be verbal. A constraint file in UCLA OTP begins with a list of tiers, and then gives the constraints.

```
Tiers:

  Word: #w.

  Morphemes:
     _^root

  Others:
     % Prosodic Categories
     wd,   % surface word
     X,      % Primary stress
     x,      % secondary stress
     sigma,  % Syllable
     mw,     % Weak Mora
     ms,     % Strong Mora

     % Surface Features
     C, V,                          % +/- syllabic, X
     cons, vowel,                   % +/- consonantal
     son, obs,                      % +/- sonorant
     voiced, vcl,                   % +/- voice
     cont, stop,                    % +/- continuant
     labial, coronal, dorsal,       % Place
     high, low, front, round,       % Vowel Place
```

```
    nasal, lateral,                 % +nasal, +lateral      dist, apical,
    % Underlying Features
    _x,  % Stress is (somewhat) phonemic.
    _C, _V,
    _cons, _vowel,
    ...
    _dist, _apical.
```

Some constraints:

```
Constraint *Spread(C):
        ]C _|_ _C, C[ _|_ _C,
        ]_C _|_ C, _C[ _|_ C.

Constraint Max(C):
        _C --> C.

Constraint Dep(C):
        C --> _C.

Constraint *(C, V):
        C _|_ V.

Constraint Align(wd, L, seg, L):
        wd[ --> C[ or V[.

Constraint *Word:
        wd _|_ wd.

Constraint *Cons(WordFinal):
    ]C _|_ ]wd.

Constraint Onset:
    ms --> C.

Constraint *BrOns:
    ms _|_ C[ and ]C.

Constraint *Coda:
    mw _|_ mw.
```

```
Constraint *BrCoda:
    mw _|_ C[ and ]C.

Constraint *Clash:
    x[ _|_ ]x.

Constraint *Lapse:
    sigma[ and ]sigma --> x[ or ]x.

Constraint PrimaryStress:
    ]wd --> ]X,
    X[ --> x[,
    X _|_ x[.

Constraint Ident(cons):
        C and _cons --> cons,
        V and _cons --> cons.

Constraint Specify(C, son, obs):
        C --> son or obs.

Constraint *(C, labial, coronal):
        C _|_ labial _|_ coronal.

Constraint Specify3(C, labial, coronal, dorsal):
        C --> labial or coronal or dorsal.

Constraint *(labial, apical):
        labial _|_ apical.

Constraint Segmental(front):
        front[ --> C[ or V[,
        ]front --> ]C or ]V.
```

Okay, no more examples. On to modifications!

3 Insertion and Deletion

As was mentioned last time, there are certain flaws in the representation of places where there is a time difference between representations (that is, areas more or less equivalent to epsilons in a simple

transducer). So, problems:

(1) When multiple underlying segments are deleted (say, multiple V's) on the surface, the underlying representation will be the same ("|" on the V tier) no matter how many have been deleted. This means only one faithfulness violation will be accrued, rather than one per deleted vowel, as I think would be appropriate.

(2) Say you have the following input:

```
    _C: [ + | + ] - [ + ]
    _V: - - - - [ + ] - -
_sonor: [ + ] - [ + ] - -
```

If the two initial consonants get deleted, there's no way to represent the underlying form in such a way as to indicate how many consonants were deleted, or that a single sonorant was deleted, and that the deleted sonorant was not adjacent to the one that was deleted. The most obvious representation in OTP terms would be

```
    _C: | - [ + ]
    _V: [ + ] - -
_sonor: | + ] - -
```

but this is not a proper OTP representation, since "|" can only occur between pluses. It also has the problems mentioned above.

(3) Similar problems occur with inserted material.

A "good" representation would have these properties:

- The contents of any of the levels of representation can be easily reconstructed.
- Constraints referring to a representation can be easily written in OTP terms, and can be easily and efficiently implemented as WFSMs.

3.1 First suggestion

The first proposed solution came from Eisner when I confronted him with these problems. He did not work out any of the details or implement it, however. The idea is to add two non-linguistic tiers: DEL and INS. During any DEL interval, the underlying form has material that does not correspond to anything in the surface form, and during any INS interval, the surface form has material that does not correspond to anything in the underlying form. During the DEL interval, the surface tiers must all be "+" or "-", and during the INS interval, the contents of the underlying tiers must be "+" or "-." Thus, the tier FSMs given above will still work. Examples:

```
  C: [ + ] - [ + ]
  V: - - [ + ] - -
  C: [ + ] - [ + ]
  V: - - - - - - -
INS: - - [ + ] - -
```

In the above representation, a vowel has been inserted between two underlying consonants. Note that to implement constraints on this representation, the constraints have to be such that they will notice the INS element and treat the underlying consonants as being adjacent. Next, we have the case of syncope:

```
  C: [ + ] - [ + ]
  V: - - - - - - -
DEL: - - [ + ] - -
  C: [ + ] - [ + ]
  V: - - [ + ] - -
```

Here the constraints will have to notice the DEL constituent and treat the surface consonants as being adjacent. The deletion of the two consonants with a sonorant we saw before would be represented as follows:

```
    C: - - - - - - [ + ]
    V: - - - - [ + ] - -
sonor: - - - - [ + ] - -
  DEL: [ + + + ] - - - -
    C: [ + | + ] - [ + ]
    V: - - - - [ + ] - -
sonor: [ + ] - [ + ] - -
```

Given this representation, faithfulness constraints have enough information to do their job.

3.1.1 Complications: Constraint Implementation

Unfortunately, this representation makes constraint implementation much more complex. There are now 12 cases:

1. Implication where *A* is composed entirely of interiors — as before.
2. Implication where *A* is composed entirely of surface tiers — here we need to check for the DEL tier and see whether *A* splits across it, and whether *B* is on either side of DEL.
3. Implication where *A* is composed entirely of underlying tiers — similar to (2), except here we have to check for INS.

4. Implication where *A* has underlying and surface tiers, and the surface tiers are all interiors — here we have to do the same sorts of checks as for (3), but also we have to check for places where *B* has some surface component and it's on the other side of a DEL from the place where *A* matches (which could be split across an INS). Some other checks I don't remember, too.

5. Implication where *A* has underlying and surface tiers, and the underlying tiers are all interiors — here we have to do the same sorts of checks as for (2), but also we have to check for *B* on the other side of INS constituents, as in (4). Some other checks I don't remember, too.

6. Implication where *A* has underlying and surface tiers with edges in both — here we have to do every type of check in both (4) and (5). This is quite complicated.

7. Clash where *A* is composed entirely of intervals.

8. Clash where *A* is composed entirely of surface tiers — here we need to check for places where *A* splits across a DEL tier.

9. Clash where *A* is composed entirely of underlying tiers — here we need to check for places where *A* splits across an INS tier.

10. Clash where *A* is composed of underlying and surface tiers, and the underlying tiers are all interiors — here we have to do more or less the same checks as (4), except for the *B* stuff.

11. Clash where *A* is composed of underlying and surface tiers, and the surface tiers are all interiors — here we have to do more or less the same checks as (5), except for the *B* stuff.

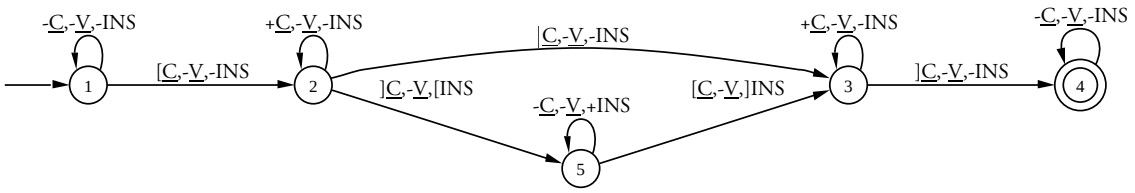
12. Clash where *A* is composed of underlying and surface tiers, and both levels have edges.

These types are shown in a separate handout.

At some point, I tried to extend these to deal with reduplication, and I thought about what would be involved in extending these to deal with paradigm uniformity, where there would be many levels of representation. For each new level of representation you would need two new tiers (*e.g.* RINS, RDEL) to indicate when there is material on it that isn't present in the underlying form, and vice versa. Doing this sort of thing necessitates making the constraints much more complicated than they already are.

3.1.2 The GEN function

The GEN function for this representation is a bit complicated, but not much different than the previous one, as long as we're only talking about two levels of representation. Basically, we allow DEL constituents to go anywhere and wherever there's an edge, you build a "pyramid" of the sort shown for standard OTP in the first section, except that the pyramid is an INS constituent:



The GEN function gets much more complicated when a new level of representation is added (such as *reduplicant*) — essentially, the "pyramid" thing has three added states instead of just one, where one state represents surface but not UR and not reduplicant (+INS,-DEL,-RINS,-RDEL), one state represents reduplicant but not UR or surface (-INS,-DEL,+RINS,-RDEL), and one state represents surface and reduplicant but not UR (+INS,-DEL,+RINS,-RDEL). The three states are reachable from each other.

3.2 An improvement

The previous representation system was too complicated for it to be possible for new levels to be added. If you look at the WFSMs that represent the constraints for that system, you will see that most of the complication arises from uncertainty when looking at one side of a DEL or INS constituent about what is on the other side. For example, if "]tier" is on the left side of DEL, then either "[tier" or "-tier" could be on the right side:

```
  T: [ + ] - [ + ]      vs.       T: [ + ] - - - -
DEL: - - [ + ] - -                DEL: - - [ + ] - -
```

The solution is to look at the level as it would be if divorced from the other levels. There will be a single symbol where the inserted space would be. Simply repeat that symbol over the inserted space:

```
  T: [ + | | | + ]      vs.       T: [ + ] ] ] - -
DEL: - - [ + ] - -                DEL: - - [ + ] - -
```

This makes it obvious what's going on, without having to look on both sides of DEL/INS.

3.2.1 Implementation

The cases for the new representation are as follows:

1. Implication where *A* consists entirely of interiors — same as before
2. Implication where *A* and *B* are all on the same level — equivalent to edge implication from before (but some transformations to *A* such that *A* doesn't match during +] for the time that level doesn't exist.

3. Implication where *A* is on a single level, but some of *B* is on another level — need to check if left of period where *A*'s level doesn't exist matches (*A* **and not** *B*) whether *B* matches on the other side.

4. Implication where *A* is on two levels, where one level has only interiors and the other has an edge — like (3) more or less, but need to deal with the situation where *A* matches on one side of the area where the edged level doesn't exist, and during that time *A* on the interior level breaks and restarts, then matches on the other side (2 violations), versus when a continuous interval exists (1 violation):

```
A-intv: [ + + | + + ]     vs   A-intv: [ + + + + + ]
A-edge: [ + ] ] ] - -          A-edge: [ + ] ] ] - -
   INS: - - [ + ] - -             INS: - - [ + ] - -
```

5. Implication where *A* is on two levels, where both levels have edges — like (2) except using the transformation used for (3–4), which I haven't mentioned, but it's that *A* matches everywhere except during + for the time that some level of *A* doesn't exist (so it does match at [DEL and]DEL, for example, if *A* contains surface).

6. Clash where *A* consists entirely of interiors — same as before.

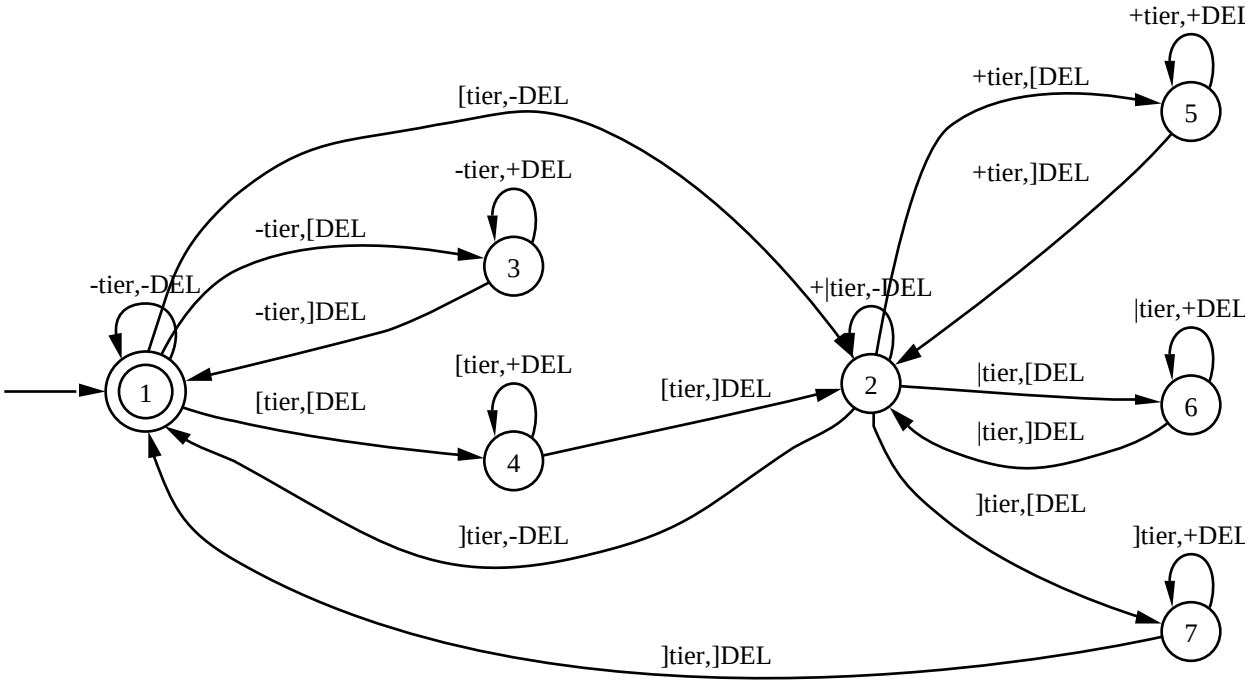
7. Clash where *A* is on a single level — equivalent to the standard edge clash, but using the transformation of (2).

8. Clash where *A* is on two levels and one of the levels has an interval — same checks as for (4).

9. Other clashes — equivalent to standard edge clash from before, with the transformation of (3–5, 8).

In a sense there's one more case than before, but the implementations are much simpler (no more than three states are ever needed and no edge ever has a weight greater than 1), and the procedures are now generalized to ignore what the particular levels are (thus now matter how many more levels you add, the machines shouldn't get any more complicated).

There's a new wrinkle, though: the tier FSMs get more complicated; we have to make sure that the labels stay in a steady state (a violation of bracket matching) during non-existent time. A tier FSM for a surface tier follows:

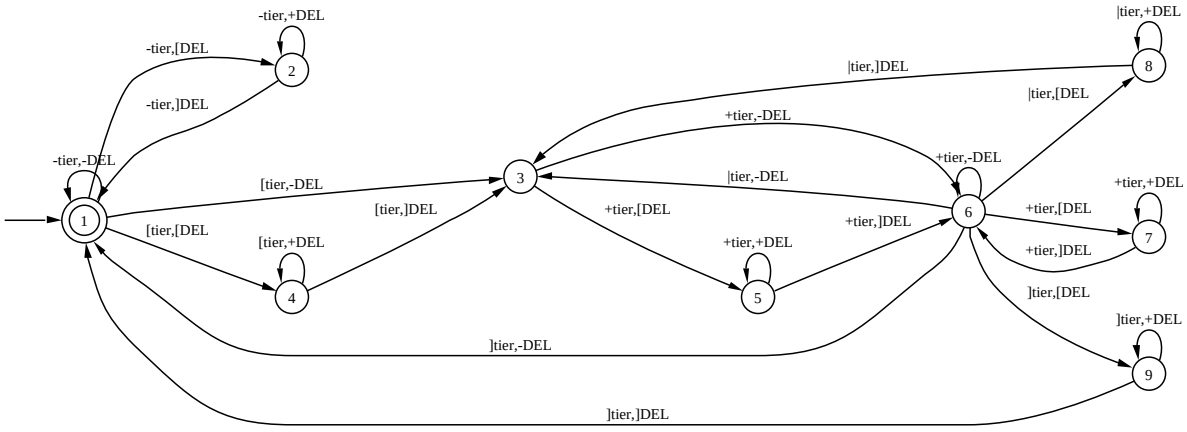


3.3 A final improvement

As matters stand up to this point, for every new level of representation it is necessary to add two new tiers — an "INS" and a "DEL" tier. However, if you interpret "INS" as meaning "the material on the UR doesn't count," and "DEL" as meaning "the material on the SR doesn't count," then you can just add one tier per level, each new tier meaning that the material on that level doesn't count. With this change, GEN for a three-level representation has the three state-pyramid structure for insertion, but the representations are: +INS,+DEL,-RDEL for reduplicant and no surface or UR; +INS, -DEL, +RDEL for surface and no reduplicant or UR; and +INS, -DEL, -RDEL for surface and reduplicant, but not UR. Drawback: you can't have insertion without also being about to use the DEL and RDEL components.

3.4 Okay, one more

With the addition of these INS/DEL tiers, it's no longer necessary to allow intervals with no +'s in them. Thus, we modify GEN such that the pluses must occur at least once, and we modify the tier FSMs correspondingly. The final version of the tier FSM is as follows:



4 Fancy Stuff, Often Involving Gen

Some comments on how to do non-obvious stuff:

4.1 Non-ranked Constraints

If you want to have two constraints which are mutually unranked with respect to one another, form the weighted product of the two WFSMs, adding the weights of intersecting edges:

$$M_1 \wedge M_2 = \langle Q_1 \times Q_2, \Sigma_1 \cap \Sigma_2, \delta', I_1 \times I_2, F_1 \times F_2 \rangle$$
$$\delta' = \{\langle (q_1, q_2), a, w, (q_3, q_4) \rangle | \langle q_1, a, w_1, q_3 \rangle \in \delta_1 \wedge \langle q_2, a, w_2, q_4 \rangle \in \delta_2 \wedge (w = w_1 + w_2)\}$$

4.2 Constraint Conjunction

To get constraint conjunction (a constraint which is satisfied only if both of its component constraints is satisfied, or alternatively, a constraint which outputs a violation whenever either of its component constraints does), form the weighted product of the two WFSMs, generating the weight of intersected edges as the disjunction of the weights of the edges:

$$M_1 \wedge M_2 = \langle Q_1 \times Q_2, \Sigma_1 \cap \Sigma_2, \delta', I_1 \times I_2, F_1 \times F_2 \rangle$$
$$\delta' = \{\langle (q_1, q_2), a, w, (q_3, q_4) \rangle | \langle q_1, a, w_1, q_3 \rangle \in \delta_1 \wedge \langle q_2, a, w_2, q_4 \rangle \in \delta_2 \wedge (w = w_1 \vee w_2)\}$$

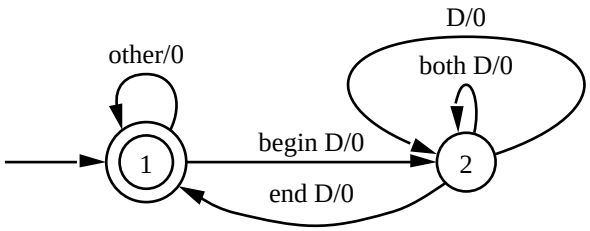
4.3 Local Disjunction (Local Weakening)

To get things like "bad clash" where a constraint c is defined in terms of another constraint c_1 such that c is violated whenever c_1 is violated n times within a particular domain, first *weaken* c_1 n times. To weaken c_1 the first time:

1. Copy $c_1 = \langle Q, \Sigma, \delta, s, F \rangle$ so that you have two copies of it, call the copy $c_1' = \langle Q', \Sigma, \delta', s', F' \rangle$.
2. Renumber the copy so no two states have the same numbering.
3. 1-weaken(c_1,c_1')$= \langle Q \cup Q', \Sigma, \delta_w, s, F \cup F' \rangle$, where $\delta_w = \delta' \cup \{\langle q_1, a, w, q_2 \rangle \in \delta | w = 0\} \cup \{\langle q_1, a, w - 1, s' \rangle | \exists q \in Q, \langle q_1, a, w, q_2 \rangle \in \delta \wedge w > 0\}$.
4. n-weaken(c_1,c_1')=1-weaken(($n-1$)-weaken(c_1,c_1'), c_1').

Then make the weakened-constraint $c_w = \langle Q_w, \Sigma, \delta_w, s_w, F_w \rangle$ domain-bound:

1. Create a domain-FSM $d = \langle Q_d, \Sigma, \delta_d, s_d, F_d \rangle$:



2. Use d to limit the evaluation of the constraint c_w to the domain D. To accomplish this, we need to define the behavior at the edges of the D domain. Outside the D domain, violations of c_w will have no effect. At the left edge of the D domain, violations that do not involve the left edge of constituents will have no effect. At the right edge of the D domain, violations that do not involve the right edge of constituents will have no effect. The final WFSM c_{wd} representing the weakened constraint limited to the domain D is produced by intersecting d with c_w, with the following modifications made to the intersection algorithm. Edges from c_w that can be determined to be outside the domain in the sense defined above are assigned a weight of 0, and if their destination within c_w was state s' (from the weakening algorithm), their destination in c_w is treated as having been s_w. This has the effect of limiting the constraint violations of c_w to the domain d. Edges from c_w that are intersected with the edge "**both** D" or **"end** D" from machine d keep their original weight, but are treated as though their destination within c_w was s_w. This has the effect of resetting c_w to zero violations at the beginning of a D domain immediately following another.

4.4 Interspersive Morphology (Planar Segregation)

To be able to make analyses of things like infixes or Arabic morphology where you might want to say that one morpheme is interspersed between the segments of another morpheme, the following method will work. First, build the morphemes as per the usual GEN, but with the difference that where the usual GEN allows insertion of non-UR elements between segments, the new GEN allows insertion of UR-elements there as well. Next, intersect the morphemes together. This will produce an acceptor for all possible precedence-preserving interspersals of the morphemes within one another, and all possible morpheme orderings. The morphemes can now be given ordering preferences that interact with the other constraints:

```
Constraint Leftward(rt):
        ]_^rt --> ]rr,
        _#wd[ --> rr[,
        ]rr --> ]_^rt,
        rr[ --> _#wd[,
        rr _|_ _C,
        rr _|_ _V.

Constraint Rightward(a):
        _^a[ --> aa[,
        ]_#wd --> ]aa,
        aa[ --> _^a[,
        ]aa --> ]_#wd,
        aa _|_ _C,
        aa _|_ _V.
```

4.5 Paradigm Uniformity/Base Correspondence

Put reference forms on new levels, intersect them with the input, write constraints that refer to the new levels. For multiple reference forms, do weighted additive products between the constraints to form constraints that penalize according to the number of reference forms by which a representation differs.

4.6 Sympathy Theory

Reorder the constraints with the sympathy constraint highest (removing the sympathy-candidate-faithfulness constraints), produce an output. Use that output as a reference form as per the previous section.

4.7 Reduplication

Eisner's proposal can't be right. Why? Here's one in the same spirit that seems likely to work (I've gotten it to mostly work, and am working out changes needed). First, copy the UR and mark the base and reduplicant with BASE and RED tiers. Now make a new level of representation called "reduplicant." After each constraint intersection/best paths thing, separate out the base and reduplicant from the rest (but not yet from each other). Form the resulting FSM into a semi-prefix-tree FSM by splitting states with multiple in-arcs. Now split this FSM into base and reduplicant. Take one semi-path at a time from each side and intersect them together. If they intersect, take that path, copy it, flip the copy (swap the surface and reduplicant levels), and concatenate the two together. Keep adding these concatenated things to a new union FSM. When finished, concatenate the pre-base-red stuff, the union FSM, and the post-base-red stuff, and minimize. This new thing has the property that to the greatest extent possible for any given path through it, the reference form of the reduplicant is the same as the actual copy reduplicant, and any restrictions that the constraint adds to one side propagate to the other.

4.8 Long-Distance Metathesis!

In GEN create two UR-like levels of representation. On one, preserve the order of the original input, and on the other allow any scrambling of segments. Intersect these levels together. The scrambled level is used as we would have used the normal UR from before. The unscrambled level can be used to make constraints between the scrambled and unscrambled levels that say contiguity relations and ordering, etc., should be preserved, whereas constraints acting on the surface will tend to favor scramblings that are well-formed.

5 Assignment

Read (Eisner 1997) and the UCLA OTP user's manual (optional), which can be found in the "doc" directory of OTP (I'll put a recent version on the ftp site, there's an old one on my web page). Write a constraint system using OTP constraints for a language in which there's just one tier — "S" (in input and output variants) — and the output is completely faithful to the input. Extra credit: run the thing under OTP and make sure you were right. More extra credit: write the corresponding WFSMs.

References

EISNER, JASON, 1997. What constraints should OT allow? Handout for talk at LSA, Chicago.