

"That man loved Maria"

where (2) is analyzed as the result of topicalizing the subject NP, thus providing a left host for the clitic.

Inkelas and Zec (1990) discuss the phonological constraints on prosodic attachment in more detail. They point out that some lexical words, for example, some prepositions, are not full phonological words, and thus are not suitable hosts for clitic attachment. Thus

(3) *U =je kuci Petar. (Inkelas and Zec 1990)
In aux house Petar.
"Petar is in the house"

They also note that topicalization is limited to constituents which are sufficiently heavy phonologically. The latter, at least for Serbo-Croat, seems to be the case for branching structures containing more than one phonological word, and multi-word person or place names.

Halpern (1995) introduces some additional considerations with respect to prosodic phrases. One is that some topicalized material is placed in a separate prosodic phrase not available as a clitic host. So we also find:

(4) Taj covek voleo =je Mariju (Halpern 1995)
that man love.ppl aux.3s Maria
"That man loved Maria"

Finally, he discusses contexts barring 2W hosting, as in

(5) *Studenti =su iz Beograda upravo stigli. (Halpern 1995)
students aux from Beograd just arrived
"Students from Beograd have just arrived"

He tentatively analyzes these contexts as ones where the material on the right of the (initial position of) the clitic forms a separate prosodic phrase. And suggests that one such "fortress" is created by a branching constituent whose first word is its head.

3. Alternative Formal Approaches

We consider three different approaches to providing for 2P clitic placement within an LFG grammar. The approaches have in common the assumption that an initial grammar is constructed without considering final clitic placement, and then rules are added or modified to account for the 2P clitic phenomena, and they share some components. But they differ significantly in the assumptions made about the relationship between prosodic considerations and syntax, and about the role of c-structure.

To briefly describe each here before examining them in more detail:

1. *PrInv by post-syntactic operation* implicitly considers 2P clitic placement to be outside the realm of syntax and c-structure. The method, related in spirit to a comment by Bresnan (correspondence), consists of building a standard LFG grammar, but including a prosodic projection associating prosodic weights with c-structure elements. A post-syntactic phase is then employed to (a) annotate the generated string by associating, with each word, a list of prosodic weights for constituents it completes, and (b) modify the string by a few rules which either leave the 2P clitics in situ, if they are adjacent to a suitable host, or move them as a group if a suitable host can be found in the appropriate (language dependent) direction.
2. *PrInv by rule transformation to yield surface rules* avoids extending the set of LFG modes of expression as required by the first method. It instead transforms an initial set of syntax rules, largely by automated means, to account for prosodic phenomena. The resulting surface rules generate structures which include the 2P clitics in both initial and surface positions if these are distinct, with the initial positions represented as traces and used as the basis for F-structure derivation for the clitics. In this approach, no effort is made

in transforming the rules to do anything but ensure that the surface-only representatives appear in the correct positions in the surface string; structurally they are embedded in other constituents.

3. Finally, *PrInv by rule-transformation to yield modified syntax rules* is a variant of the above suggested by M. Dalrymple (private conversation). The variant assumes that prosodic considerations contribute to syntax as do other types (e.g. discourse), but that it is convenient to transform the basic syntactic rules to account for 2P clitic placement. The resulting rules are simpler than those generated by the former method, omitting traces and viewing the moved clitics as interrupting discontinuous constituents.

Each of these approaches is described in more detail, below.

4. PrInv by Post-Syntactic Reordering

This approach requires two types of rules. The first are syntactic rules assigning 2P clitics to the positions they would occupy if movement were not required. Constraints associated with the rules also specify a prosodic (P) projection from c-structure capturing the prosodic aspects of the words and constituents found. Words of the generated string are marked with features based information from the prosodic projection, and then the string is adjusted based on those features. The approach is applicable both in parsing and generation. The prosodic projection is discussed immediately below, followed by consideration of reordering rules, and then bi-directional use of the approach.

4.1 The Prosodic Projection

The goal of the prosodic projection here is to attach to each c-structure node a set of features determining its adequacy as a clitic host. The features needed may vary among languages; based on data supplied by Inkelas and Zec (1990), a critical feature for Serbo-Croat might be:

WEIGHT = NULL | WORD | PHRASE | IGNORE

where WEIGHT = NULL if the node spans an extent less than a prosodic word, WORD if it spans a prosodic word but not a separate prosodic phrase, PHRASE if it spans a prosodic phrase not capable of serving as a clitic host, and IGNORE if it spans more than one of the above.

We assume that lexical items are assigned weights of either NULL (for, e.g., Serbo-Croat prepositions), or WORD, and that PHRASE weights are found by a method not described here.

4.2 Post-syntactic reordering rules

Given the above feature assignment, we assume that each word of a generated string *S* is annotated as either NOTHOST if the weight is NULL or the word is contained in a prosodic phrase, and HOST otherwise.

The string *S* is input to a post-syntactic phase governed by a few rules, such as the following which might be appropriate for 2D/2W languages such as Serbo-Croat:

1. If a 2P clitic group is immediately preceded by a HOST word (and the clitic group does not mark the beginning of a 2P scope), the surface string *S'* is equal to *S*.
2. Otherwise if the next word *w* following the clitic group is marked with weight-feature WORD, the surface string *S'* is equal to that which would be obtained by moving the clitic group immediately following *w*.
3. Otherwise fail.

4.3 Parsing using post-syntactic reordering rules

This approach is suitable both descriptively, i.e., in a generative direction, and also in utterance analysis. In

parsing, the syntactic and post-syntactic rules may be applied as follows:

1. Attempt to parse the surface string using the syntactic rules.
2. If this does not succeed, assume that PrInv has taken place, and iteratively
 - move the 2P clitics in the string one position to the left, and
 - attempt to parse the string.
3. If no parse succeeds by the above, fail.
4. Finally, apply the 2P clitic movement rules specified in the previous section to the successfully parsed string. If this produces the original surface string, succeed. Otherwise fail.

The last step is needed to ensure that the successful analysis obtained is indeed an analysis of the surface string. Since movement to the left in step 2 above is unconstrained, the last step guards against obtaining a successful parse by moving the clitic group outside its original domain.

4.4 Discussion

As indicated earlier, this approach is consistent with the assumption that 2P clitic placement takes place outside the syntax. Also, it has the advantages of simplicity and expressiveness, because the rules are not embedded within syntactic rules. However, it does add another phase to the framework, and the need for additional formalism (not addressed in this paper).

5. PrInv by rule transformation to surface rules

This next approach avoids introducing additional processing phases. Instead, the basic syntactic rules are transformed so as to properly position the 2P clitics. Like the previous approach, it might also be considered to assume that 2P clitic placement is extra-syntactic, in that the transformed rules only ensure that the surface order is correct; "moved" clitics can be embedded within other constituents.

The method starts with C-structure rules placing the clitics in an assumed position, for example adjoined to C0 as suggested by King (private conversation) for Serbo-Croat, giving something like:

1. CP --> DP C'
2. C' --> C0 IP
3. C0 --> C0 CLcat1 CLcat2 ... CLcatn
C0 --> complementizer
4. IP --> I0 VP

where CLcati is "clitic category i", and the categories might be AUX, DAT, ACC, etc, as listed by King (1995) for a variety of Slavic languages.

The rules are augmented to develop prosodic projection values different from those of the previous method, and then transformed to include surface clitic placement, retaining traces in the original positions of any moved clitics.

The prosodic (P) projection here employs additional attributes:

- LWT = weight of constituent if NULL | WORD | PHRASE
weight of leftmost component if WEIGHT=IGNORE (recursive)
RWT = weight of constituent if NULL | WORD | PHRASE
weight of rightmost component if WEIGHT=IGNORE(recursive)

The transformation employs an additional COMM projection to communicate the need for/occurrence of prosodic inversion, and to communicate clitic-related lexical information between surface positions and their

syntactic traces, where the information is used to build F-structures identical to those which would have been obtained if no movement occurred.

The COMM projection is described in somewhat more detail below, followed by the development of the transformed rules, and then of a mechanism for achieving the more laborious transformations automatically.

5.1 The COMM projection

The COMM projection is responsible for the "bookkeeping" of prosodic inversion. It contains two types of features, hosting features and lexical features. Grammar-dependent hosting features, here LEFTHOST1 and LEFTHOST2, are used to communicate the need for prosodic inversion. They can be thought of as being assigned values based on prosodic characteristics of the initial clitic environment, with the values tested at initial clitic positions and potential prosodic inversion sites (or vice versa).

The lexical features of COMM, which we will name LXcati (e.g., LXAUX, LXDAT,.) are used to transmit clitic-associated lexical information between the two potential positions within the tree. Because these clitics belong to closed classes, we here assume that the values of LXcat features are lexical strings, e.g., "mi" "ti". At the syntactic nodes (traces) representing moved clitics, appropriate F-structure values can be set by constraints such as:

```
{ [(COMM * LXDAT)=mi (! CASE)=DAT (! PERS)=1 (! NUM)=SG]
  | [(COMM * LXDAT)=ti (! CASE)=DAT (! PERS)=2 (! NUM)=SG]
  etc.
```

5.2 Rule Transformation

This section describes the kinds of rule transformations needed to implement the approach; the next one discusses how the more laborious transformations might be automated.

The transformations developed here extend the initial rule set to:

- Determine whether the clitics can remain in their standard position based on P projection information.
- Place the 2P clitics, using the P projection information to constrain the placement of the surface elements.
- Use COMM projection features to pass information between surface and syntactic clitic positions.

For concreteness we will assume the skeleton grammar rules for Serbo-Croat listed above. However, the method is applicable to alternative grammar formulations.

We first want to modify rules 1-3 to include the determination of whether the clitics should remain in their original positions. This can be done by rules referencing the P projection as outlined in an earlier section, and setting COMM hosting attributes accordingly.

```
1'. CP --> DP: (P * RWT)=WORD => (COMM M* LEFTHOST1)=+;
    any original constraints
    C': (COMM M* )=(COMM *)
    any original constraints
2'. C' --> C0: (COMM M* )=(COMM *)
    any original constraints.
    IP: (COMM M* )=(COMM *)
    any original constraints.
3'. C0 --> C0: (P * RWT)=WORD => (COMM M* LEFTHOST2)=+;
    any original constraints
    CLcat1M: (COMM M*)=(COMM *)
    ....
    CLcatnM: (COMM M*)=(COMM *)
```

where we will use the notation (X * Y) to mean attribute Y of projection X of the current node, and (X M* Y) to mean attribute Y of projection X of the parent node. We then add rules expanding the new CLcat1M categories, and associating lexical information for the clitics with F-structures:

```
3-1. CLcat1M --> { CLcat1: { (COMM M* LEFTHOST1) | (COMM M* LEFTHOST2)
  | e:      ~{ (COMM M* LEFTHOST1) | (COMM M* LEFTHOST2) }
              (COMM M* LXcat1) "percolated value from PrInv"
              != (COMM M* LXcat1)
  }
  MACRO_LXcat1
```

```
3-n. CLcatnM --> { CLcat1: { (COMM M* LEFTHOST1) | (COMM M* LEFTHOST2)
  | e:      ~{ (COMM M* LEFTHOST1) | (COMM M* LEFTHOST2) }
              (COMM M* LXcatn) "percolated value from PrInv"
              != (COMM M* LXcatn)
  }
  MACRO_LXcatn
```

Rules 3-i above test whether the clitics belong in their initial position or, if not, whether they can be found in another position (by the existence constraint (COMM M* LXcatn)). If the latter, the percolated information is obtained for use in building the F-structure. MACRO_LXcatn is a macro abbreviation of the F-structure setting constraints shown in the last section, for example, MACRO_LXDAT might expand to:

```
{ [(COMM * LXDAT)=mi (! CASE)=DAT (! PERS)=1 (! NUM)=SG]
 | [(COMM * LXDAT)=ti (! CASE)=DAT (! PERS)=2 (! NUM)=SG]
 etc.
```

While the above rules look complicated, they are limited in size and can be obtained by transforming a small number of rules associated with a limited number of categories. This is not the case for rules implementing the inversion by finding 2W positions; for relatively free-word-order languages, transformations may be needed for rules associated with almost all categories. But the transformations involved are quite regular and susceptible to automation.

The general idea of the transformations is to specify alternative rules embedding the "moved" clitics after the prosodic word following the clitic, if such a word exists. To the extent that the alternatives for 2P placement may differ among languages, the transformations might differ. Here we assume that in the motion a lexical item that is not a prosodic word may be skipped over.

The to-be-automated transformation procedure starts by finding the highest level categories that may immediately follow the clitics in their initial position. In this case the only such category is IP, because of "2. C' --> C0 IP". It then finds the rules expanding those categories, here only "4. IP --> I0 VP".

For each such rule, with general form

```
XP --> a b c ...
```

the transformation procedure adds alternative forms of the rule:

```
| a_1: (COMM M*)=(COMM *)
      ~(COMM M* LEFTHOST1) ~(COMM M* LEFTHOST2)
      original constraints for a
b: as in original
rest as in original
```

```

| a: (P * WEIGHT) = NULL
    original constraints for a
b_1: (COMM M*)=(COMM *)
    ~(COMM M* LEFTHOST1) ~(COMM M* LEFTHOST2)
    original constraints for b
rest as in original...

```

and, if a (respectively b) is a preterminal category, also

```

a_1 --> a: (P * WEIGHT) = WORD
        CLCat1: (COMM M* LXCat1)=!
        CLCat2: ...

```

The effect of the added alternatives is to determine if the clitics can remain in-situ, and, if not, to look for an appropriate location in the first or second component constituent. The second constituent is a candidate if the first consists of a single lexical item which does not make up a prosodic word.

If a (respectively b) is not a preterminal category, the process is continued for its left descendant, creating rules a_1 --> etc. revising those for a (respectively b_1 -->... revising those for b).

5.3 Automating Rule Transformation

Generating the last set of rules manually is laborious, since it potentially affects all categories found under IP. But since the transformation is regular, it can be automated by a parameterized, recursive rule-transformation function. (Note: it would be preferable at some point to specify the transformations via LFG-oriented meta-rules.)

The function parameters specify the variable elements of the transformation:

1. The initial category whose rules are to be transformed (here IP),
2. the constraint used to indicate that a 2W word should be bypassed, (NULL weight),
3. the constraint used to indicate that a 2W word can serve as host
4. the constraints to be added to PrInv target candidates
5. the constraints to be used at the point of clitic insertion.

or, in pseudo-C code:

```

P2RULES(initial_category, /* e.g., IP */
        bypass_condition, /* e.g., (P * WEIGHT)=NULL */
        accept_condition, /* e.g., (P * WEIGHT=WORD */
        candidate_condition, /* e.g., (COMM M*)=(COMM *)
                               ~(COMM M* LEFTHOST1) ~(COMM M* LEFTHOST2)
        clitic_group        /* e.g., CLCat1: (COMM M* LXCat1)=!
                               CLCat2: ...
        generated_label     /* used in an internal recursive call */
)

```

For details of function operation please refer to Appendix A.

6. PrInv by rule transformation to syntactic rules

This last approach is suggested by Mary Dalrymple as a possible alternative to the previous one. In this approach, moved elements can be inserted between elements of, and at the same level as, discontinuous constituents under VP. This allows us to dispense with the empty traces and percolation of f-structure information, as the moved clitic nodes can be treated in the same way as other arguments.

While the previous method seems consistent with a view that the rules generating the moved clitics are more like "surface rules" than "syntactic rules", because they allow the clitics to be embedded in other argument constituents, this one seems to support a more unified account.

The approach is interesting because of its economy, and because there is some evidence that the constituents really are discontinuous, as in the following example given by King (1995):

```
(6)Taj =joj =ga =je      poklono covek.
    that her it aux-3SG presented man
    'That man presented her with it'
```

However, the version of the approach developed so far is limited with respect to the structures that are candidate hosts for PrInv. It can move 2W clitics only after one-word argument-level constituents, or after the first word of such constituents having single branches to the left. The viability of the approach rests on either removing this limitation, or demonstrating that it covers the relevant cases.

The approach uses essentially the same mechanisms as the previous one:

- the P projection to express prosodic constraints on clitic attachment,
- a COMM projection, here much diminished, serving only to communicate whether an in-situ host has been found (via LEFTHOST features).
- rule transformations, both manual and automated, but with different content.

The revised rules are developed below. Starting, as before, with rules

1. CP --> DP C'
2. C' --> C0 IP
3. C0 --> C0 CLcat1 CLcat2 ... CLcatn
C0 --> complementizer
4. IP --> I0 VP

The transformed rules 1 and 2 are the same as the earlier formulation:

- 1'. CP --> DP: (P * RWT)=WORD => (COMM M* LEFTHOST1)=+;
any original constraints
C': (COMM M*)=(COMM *)
any original constraints
- 2'. C' --> C0: (COMM M*)=(COMM *)
any original constraints.
IP: (COMM M*)=(COMM *)
any original constraints.

The transformed rule 3 is simplified to:

- 3'. C0 --> C0: {(P * RWT)=WORD => (COMM M* LEFTHOST2)=+;
any original constraints
CLcat1: { COMM M* LEFTHOST1) | (COMM M* LEFTHOST2)
MACRO_LXcat1
....
CLcatn: { COMM M* LEFTHOST1) | (COMM M* LEFTHOST2)
MACRO_LXcatn

The transformations of rule 4 and descendants are revised to produce the discontinuous constituents. For rule 4, of general form

z --> a b c

the transformation is to:


```

clitic_group          /* e.g., CLCat1: (COMM M* LXCat1)=!
                      CLCat2: ...
generated_label       /* used in an internal recursive call-ignore */
)

```

To do so succinctly, a few additional pieces of notation are necessary. First, assume the elements of a rule can be accessed as a structure

```
r.0 -> r.1.n r.1.e r.2.n r.2.e .. r.n.rest
```

where r.i.n is a category name, and r.i.e is the constraint expression associated with the category in the rule, and r.n.rest represents all elements of the rule beginning with r.n.

Next, assume two additional subordinate functions:

- RULES(cat) returns all rules having "cat" on the left-hand side. Note: the multiple alternatives of a rule in LFG are here considered separate rules.
- ADDRULE(string) generates a rule consisting of the indicated string. The latter is constructed by concatenating variables with literal strings ("...").

The recursive function P2RULES can then be expressed by the following C-like pseudo-code:

```

P2RULES(initial_category, /* e.g., IP */
        bypass_condition, /* e.g., (P * WEIGHT) = NULL
        accept_condition, /* e.g., (P * WEIGHT=WORD */
        candidate_condition, /* e.g., (COMM M*)=(COMM *)
                               ~(COMM M* LEFTHOST1) ~(COMM M* LEFTHOST2)
        clitic_group       /* e.g., CLCat1: (COMM M* LXCat1)=!
                               LXCat2: ...
        gen_label          /* initially blank */
) {
    if(RULES(initial_cat) != NULL) { /* entry_cat is not preterminal */
        for (r in RULES(initial_category)) { /* for rules initial_category -> ... */
            GEN = gensym() /* new generated label */
            /* generates XP -> a_1... b: */
            ADDRULE(r.0 || gen_label || "-->" r.1.n || GEN || ":" ||
                    r.1.e || candidate_condition || r.2.rest)
            /* generates rule for a_1 */
            P2RULES(r.1.n, bypass_condition, accept_condition,
                    candidate_condition, clitic_group, GEN)
            /* generates XP -> a: ... b_1:
            ADDRULE(r.0 || gen_label || "-->" || r.1.n, r.1.e ||
                    bypass_condition ||
                    r.2.n || GEN, r.2.e || candidate_condition || r.3.rest)
            /* generates rule for b_1 */
            P2RULES(r.2.n, bypass_condition, accept_condition,
                    candidate_condition, clitic_group, GEN)
        }
    }
    else { /* entry cat is preterminal */
        ADDRULE (initial_cat || GEN || "-->" ||
                accept_condition || clitic_group)
    }
}

```