Minimalist Parsing of Subjects Displaced from Embedded Clauses in Free Word Order Languages

Asad B. Sayeed
Department of Computer Science
University of Maryland at College Park
A. V. Williams Building
MD 20742 USA
asayeed@mbl.ca

Abstract

In Sayeed and Szpakowicz (2004), we proposed a parser inspired by some aspects of the Minimalist Program. This incremental parser was designed specifically to handle discontinuous constituency phenomena for NPs in Latin. We take a look at the application of this parser to a specific kind of apparent island violation in Latin involving the extraction of constituents, including subjects, from tensed embedded clauses. We make use of ideas about the left periphery from Rizzi (1997) to modify our parser in order to handle apparently violated subject islands and similar phenomena.

1 Introduction

In Sayeed and Szpakowicz (2004), we started by describing the difficulty of parsing sentences in languages with discontinuous constituency in a syntactically robust and cognitively realistic manner. We made the assumption that semantic links between the words of a sentence are made as soon as they arrive; we noted that this constrains the kinds of formalisms and algorithms that could be used to parse human sentences. In the spirit of the Minimalist Programme, we would like to produce the most economical parsing process, where, potentially controversially, we characterize economy as computational complexity. Discontinuity of phrases (usually noun phrases) in e.g. Latin provides a specific set of challenges in the development of a robust syntactic analysis; for instance, in the process of building parse trees, nouns must often be committed to positions in particular structures prior to the arrival of adjectives in an incremental parsing environment.

Inspired by work such as Stabler (2001), we proposed a formalism and algorithm[1] that used feature set unification rather than feature cancellation, which Stabler uses to implement basic Minimalist operations such as MOVE and MERGE. We demonstrated the workings of the algorithm given simple declarative sentences—in other words, within a single, simple clause. What we wish to do now is demonstrate that our algorithm parses Latin sentences with embedded clauses, and in particular those with constituents displaced beyond the boundaries of embedded clauses where this displacement does not appear to be legitimate wh-movements; these are, in a sense, another form of discontinuity. In doing this, we hope to show that our formalism works for a wider subset of the Latin language, and that we have reduced the problem of developing a grammar to one of choosing the correct features.

2 Background

Noun phrases in Latin can become discontinuous within clauses. For instance, it is possible to place a noun before a verb and an adjective that agrees with the noun after the verb. However, for the most part, the noun phrase components stay within CP. Nevertheless, Kessler (1995) noted several instances where, possibly for intonational effect, Latin prose writers extracted items into matrix clauses from embedded clauses and clauses embedded within those embedded clauses. For example,

(1) Tametsi *tu*
Although you-NOM-SG
scio *quam*
know-IND-PRES-1SG how

[1] For the purpose of clarification, our algorithm can be found at http://www.umiacs.umd.edu/∼asayeed/discont.pdf

Proceedings of the ACL Student Research Workshop, pages 97–102,
Ann Arbor, Michigan, June 2005.

sis *curiosus*
are-SUBJ-PRES-2SG interested-NOM-SG
'Although I know how interested you are' (Caelius at Cicero, Fam 8.1.1)

In this and other cases provided by Kessler, a word is extracted from an embedded clause and moved to the beginning of the matrix clause. (The italicized words consist of the extracted element and the clause from which it was extracted.) Note in particular that 1 involves the dislocation of the subject from a tensed embedded clause, something that would ordinarily be a well-known island violation (Haegeman, 1994).

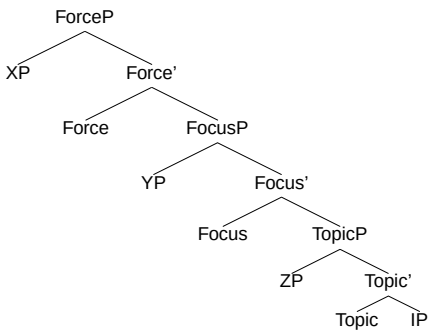
According to Kessler, this situation is rare enough that many contemporary accounts of Latin syntax neglect discussion of this kind of device. It is likely that Cicero occasionally wrote this way for prosodic reasons; however, there is no reason why prosody should not have syntactic consequences, and we attempt to account for the parsing of such sentences in this document.

It is interesting to note how in these examples, the displaced element moves somewhere near to the beginning of the outer clause. Rizzi (1997) suggests a structure for this "left periphery" based on observations from Italian:

(2) ...Force...(Focus)...(Topic)...

Within Rizzi's GB-based framework, this is suggested to be the internal structure of CP. In X-bar terms, it looks something like this:

(3)



Focus and Topic in most languages have prosodic effects, so if words displaced from embedded clauses for prosodic reasons happen to have been raised to the beginning, it suggests that the word has become part of some form of articulated CP structure.

Since our parsing algorithm is inspired by minimalism, we cannot make use of the full X-bar system. Instead, we use Rizzi's analysis to develop an analysis based on features and checking.

3 The Parser in Action

3.1 A Run-through

Our parser (2004) is incremental, meaning that it does not have access to the end of the sentence at the beginning of a derivation. It is also "semantically greedy", meaning that it attempts to satisfy the semantic requirements (through checking) as soon as possible. So each step in the derivation consists of attempting to see whether or not checking can be accomplished using the current items in the "processing buffer" and those in the "input queue," and if not, shifting a word from the input queue onto the processing buffer. The distinction is marked, in our notation, by a |: the words and trees before | are in the processing buffer, and those that are after | are in the input queue.

The algorithm also prefers move before merge. This also ensures that trees do not have multiple pending resolvable semantic dependencies, which can represent a state of ambiguity in determining which dependency to resolve and how.

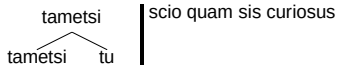
We will now present an example parse of the above sentence. But we will first present the general outline of the parse, rather than the full details using the formal representation; after that, we will demonstrate the formalism. We sketch the steps of the parse first so that we can deduce what features we would need to make it work with the system.

We first start with everything in the input queue, after the |:

(4) |tametsi tu scio quam sis curiosus

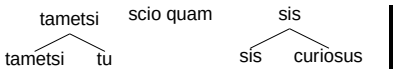
Now we need to shift (hear) two words for any parsing operations to be performed. So we shift *tametsi* and *tu*. *tametsi* ("although") consists of *tamen*, *et*, and *si*: "nevertheless", "and", and "if." These suggest that *tametsi* is part of a CP, and, most likely, Force. Since *tu* has been displaced from the embedded clause, probably for prosodic reasons, it likely has features that can be gleaned from the intonation and the context, such as Focus. Since these are part of our CP system, we merge them.

(5)

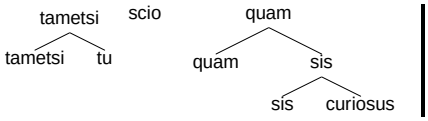


Now we have to shift *scio*. But the verb *scio* does not have a complement and cannot merge with *tametsi*

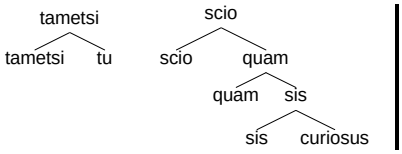
until it is a complete VP. The same is true for *quam* ("how") and *sis* since *sis* ("you are") needs a complement: *curiosus*. So the system waits to shift everything and then merges *sis* and *curiosus*.

(6) 

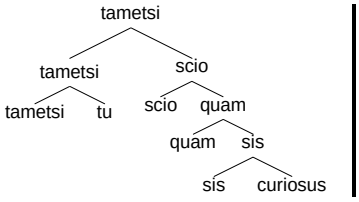
Now we can merge *sis* and *quam*, since *sis* now has a complement. Latin is a pro-drop language, so we can perform the merge without having an explicit subject, which is currently part of another tree.

(7) 

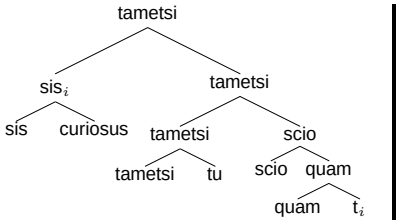
quam has been given its complement. Now as a complete CP, it is ready to be a complement of *scio*.

(8) 

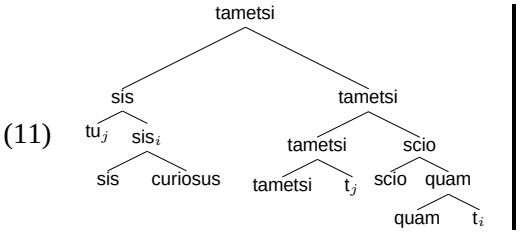
We have a CP (the *tametsi* tree) and a VP (*scio*), and we need to merge them to form one CP.

(9) 

So this leaves us in the position of having a *tu* and *sis* in one tree. However, we cannot bring them together. In Sayeed and Szpakowicz (2004), we required (in order to limit tree searches) that movement during parsing be to positions that command the trace of movement. Clearly, *tu* does not command *sis*. We only permitted raising, so what should we raise? If we raised the entire CP, we would get a tree in which neither *tu* nor *sis* commands the other. We would have to make another move to get *sis* to command *tu*. So we take a simpler route and just move *sis*.

(10) 

Now *sis* commands *tu*. We can now move *tu*.

(11) 

Note that *sis* still projects after the merge, seeing that *sis* holds the requirement for a subject—*tu* is now in what would be known as a specifier position. It does not matter that *tu* does not presently command its trace; this is something in our account of parsing that differs from GB and minimalist accounts of movement in generation. Instead, the position with which it must be merged after movement can be the one that commands the original position. This allows the target position to be the one that projects, as *sis* has.

3.2 Now with Features

Now all dependencies are satisfied, and we have a complete tree. What we need to accomplish next is an account of the features required for this parse under the system in Sayeed and Szpakowicz (2004). We add one extra characteristic to Sayeed and Szpakowicz (2004) which we will explain in greater detail in forthcoming work: optionally-checked features; this is required primarily to avoid having to imagine empty categories when parsing such phenomena as dropped subjects, which exists in Latin.

First of all, let us account for the lexical entries of the initial two words, *tametsi* and *tu*. We need features that represent the discursive effect represented by the displacement of *tu*. We shall assume that this is Focus. Also, however, we need a feature that will prepare *tametsi* to merge with *scio*. So we represent these two as

(12) tametsi: {UNCH?(Disc:Focus), UNCH(Type:V)}
tu: {unch(Disc:Focus) → unch(Case:Nom, Pers:2, Num:Sg)}

Features are grouped together into feature bundles, which allow simultaneous checking of features. Note that the ? in one of the feature bundles of *tametsi* means that it is optional; it does not have to be checked with a focus feature on an adjacent constituent if such a feature does not exist, but it must if there is one.

For *tu* we are using feature paths as we defined in Sayeed and Szpakowicz (2004); what is to the right of a feature path cannot be checked before what is to

the left. In this case, we must check the focus feature before we can check *tu* as a constituent of its proper VP (headed by *sis*).

We express the trees using the same horizontal indented representation as in Sayeed and Szpakowicz (2004). We use this notation because the nodes of this tree are too large for the "normal" tree representation used above. So we start with

(13) | tametsi tu scio quam sis curiosus

We need to shift two words before we can do anything. We thus create nodes with the above features.

(14) [tametsi {UNCH(Disc:Focus), UNCH(Type:V)}]
[tu {unch(Disc:Focus) → unch(Case:Nom, Pers:2, Num:Sg)}]
| scio quam sis curiosus

The Focus features can be checked. Using our system, unch and UNCH feature bundles are compatible for checking, and the node with the UNCH feature projects. This form of merge among the items already shifted can only be performed with the roots of adjacent trees. We specified this to prevent long-distance searches of the processing buffer.

(15) [tametsi {CH(Disc:Focus, Case:Nom, Pers:2, Num:Sg),
UNCH(Type:V)}]
tametsi
[tu {ch(Disc:Focus) → unch(Case:Nom, Pers:2, Num:Sg)}]
| scio quam sis curiosus

When UNCH and unch features bundles are checked, their features are unified (and replaced with the result of unification). UNCH and unch become CH and ch. Meanwhile, *tametsi* has acquired the features of *tu* in the CH bundle. The purpose of this mechanism is to transfer information up the tree in order to support incremental parsing of discontinuous NP constituents, but we find an additional use for this below.

We make one change here to the unification of feature bundles as described by Sayeed and Szpakowicz (2004): when we replace feature bundles with the result of unification, we replace them with the features of the entire path with which we are checking. This ensures that in the process of checking, we do not "hide" features that are further on in the path. So *tametsi* also gains the gender, person, and case features. This is actually quite a logical extension of the idea we expressed in Sayeed and Szpakowicz (2004) that a feature being checked with a feature further down a path should be compatible with all the previous features on the path. In both cases, the system should reflect the idea that features further down a path are dependent on the checking status of previous features. As with unification in general, compatibility means lack of a conflict in $\tau : \phi$ pairs (i.e., no case conflicts, and so on).

Now, as per 6, we need to shift all the remaining words into the buffer before we get a compatible set. So we need to determine lexical entries for all of the remaining words. First, *scio*:

(16) scio: {UNCH?(Case:Nom, Pers:1, Num:Sg),
UNCH(Wh:0) → unch(Type:V)}

We once again use a feature path. In this case, it means that *scio* ("know") must have a wh-phrase complement[2] before it is ready to be checked by something that takes a VP complement (such as a complementizer). So this leads us to an entry for *quam*:

(17) quam: {UNCH?(Disc:Focus), UNCH(Type:V) → unch(Wh:0)}

For *quam*, we also have an optional Focus feature, because it is the head of a CP as *tametsi* is above. (We might have other optional discourse features there, but they would be superfluous for this discussion.) And, like *tametsi*, it has a feature that allows it to take a VP complement. Checking this feature releases the wh-feature that allows it to become the complement of *scio*.

Now we only need entries for *sis* and *curiosus*

(18) sis: {UNCH?(Case:Nom, Pers:2, Num:Sg),
UNCH(Case:Acc) → unch(Type:V)}
curiosus: unch(Case:Acc, Gen:Masc, Num:Sg)

We use an optional feature for the requirement of a nominative subject on *sis*, subjects being optional in Latin. However, we do require it to take an accusative object. We are able to shift everything as we did prior to 6.

(19) [tametsi {CH(Disc:Focus, Case:Nom, Pers:2, Num:Sg),
UNCH(Type:V)}]
tametsi
[tu {ch(Disc:Focus) → unch(Case:Nom, Pers:2, Num:Sg)}]
[scio {UNCH?(Case:Nom, Pers:1, Num:Sg),
UNCH(Wh:0) → unch(Type:V)}]
[quam {UNCH?(Disc:Focus), UNCH(Type:V) → unch(Wh:0)}]
[sis {UNCH?(Case:Nom, Pers:2, Num:Sg),
UNCH(Case:Acc) → unch(Type:V)}]
[curiosus unch(Case:Acc, Gen:Masc, Num:Sg)] |

Now *sis* and *curiosus* can merge. The resulting merger between compatible unch and UNCH features, by Sayeed and Szpakowicz (2004), also causes the contents of those feature bundles to be unified.

(20) [tametsi {CH(Disc:Focus, Case:Nom, Pers:2, Num:Sg),
UNCH(Type:V)}]
tametsi
[tu {ch(Disc:Focus) → unch(Case:Nom, Pers:2, Num:Sg)}]

[2] The 0 is just a placeholder meaning that the Wh is a singleton, not a pair like many of the other features.

[scio {UNCH?(Case:Nom, Pers:1, Num:Sg),
UNCH(Wh:0) → unch(Type:V)]
[quam {UNCH?(Disc:Focus), UNCH(Type:V) → unch(Wh:0)}]
[sis {UNCH?(Case:Nom, Pers:2, Num:Sg),
CH(Case:Acc, Gen:Masc, NumSg) → unch(Type:V)}]
sis
[curiosus ch(Case:Acc, Gen:Masc, Num:Sg)] |

Now that the left feature on the feature path on *sis* is checked, the verb type feature is free. It can check with the corresponding feature on *quam*.

(21)
[tametsi {CH(Disc:Focus, Case:Nom, Pers:2, Num:Sg),
UNCH(Type:V)}]
tametsi
[tu {ch(Disc:Focus) → unch(Case:Nom, Pers:2, Num:Sg)}]
[scio {UNCH?(Case:Nom, Pers:1, Num:Sg),
UNCH(Wh:0) → unch(Type:V)]
[quam {UNCH?(Disc:Focus), CH(Type:V) → unch(Wh:0)}]
quam
[sis {UNCH?(Case:Nom, Pers:2, Num:Sg),
CH(Case:Acc, Gen:Masc, NumSg) → ch(Type:V)}]
sis
[curiosus ch(Case:Acc, Gen:Masc, Num:Sg)] |

Feature paths allow *quam* to merge with *scio* as in 8.

(22)
[tametsi {CH(Disc:Focus, Case:Nom, Pers:2, Num:Sg),
UNCH(Type:V)}]
tametsi
[tu {ch(Disc:Focus) → unch(Case:Nom, Pers:2, Num:Sg)}]
[scio {UNCH?(Case:Nom, Pers:1, Num:Sg),
CH(Wh:0) → unch(Type:V)]
scio
[quam {UNCH?(Disc:Focus), CH(Type:V) → ch(Wh:0)}]
quam
[sis {UNCH?(Case:Nom, Pers:2, Num:Sg),
CH(Case:Acc, Gen:Masc, NumSg) → ch(Type:V)}]
sis
[curiosus ch(Case:Acc, Gen:Masc, Num:Sg)] |

And, lastly, *scio* merges with the CP headed by *tametsi*.

(23)
[tametsi {CH(Disc:Focus, Case:Nom, Pers:2, Num:Sg),
CH(Type:V)}]
tametsi
tametsi
[tu {ch(Disc:Focus)
→ unch(Case:Nom, Pers:2, Num:Sg)}]
[scio {UNCH?(Case:Nom, Pers:1, Num:Sg),
CH(Wh:0) → ch(Type:V)]
scio
[quam {UNCH?(Disc:Focus), CH(Type:V) → ch(Wh:0)}]
quam
[sis {UNCH?(Case:Nom, Pers:2, Num:Sg),
CH(Case:Acc, Gen:Masc, NumSg)
→ ch(Type:V)}]
sis
[curiosus ch(Case:Acc, Gen:Masc, Num:Sg)] |

We now have a single tree, but we are in the predicament of 9. We need to be able to move *sis* to a position where it commands *tu*. And that means moving it to join with *tametsi*.

In Sayeed and Szpakowicz (2004), we proposed a mechanism by which adjuncts displaced from discontinuous NPs could reunite with their NPs even if the NP had already been merged as a constituent of a verb. This was by allowing adjuncts to merge with the verb if the verb had a compatible CH feature (without actually checking the adjunct feature bundle). A CH feature advertises that the verb had previously merged with a compatible noun, since unification would have given the noun's features to the CH feature bundle.

In this case, *tametsi* does have a CH feature bundle that appears compatible with *sis*, but UNCH features are not features that cause adjunctions in our system. We propose a minimal stipulation that will solve this problem:

(24) UNCH features (i.e., features that indicate a requirement for a constituent) can be moved or merged to meet compatible CH features.

The main problem with 24 is the possibility that unnecessary movements caused by UNCH features may occur in such a way that the UNCH feature would be moved out of the way of compatible unch features.

But this is likely not a problem. Our system prefers to exhaust all possible movements before mergers in parsing. So, if an UNCH feature had been in the tree, and an unch feature is introduced later at the root (as specified in Sayeed and Szpakowicz (2004)), the constituent containing the UNCH feature would immediately have moved to claim it. Then if a compatible CH feature arrived, it would not matter, since the UNCH feature would itself have been checked. But if a compatible CH feature had been in the tree *before* the compatible unch feature had joined, what then? The constituent containing the UNCH feature would move to join it. Then the unch feature would join the tree. It would still command the UNCH feature, which would move to claim it.

There is only one unsafe case: if the CH feature arrives before the unch feature, and it is part of a head whose constituents contain a compatible unch feature on the *wrong* constituent, then the UNCH feature would be checked with the wrong constituent according to the mechanism above. After all, the UNCH feature would command the incorrect unch feature. This possibility, however, can only exist if there is another displaced item in the tree containing the original CH that is compatible with the UNCH feature but displaced from some *other* phrase. This requires further investigation into Latin grammar, as it seems unlikely that such constructions exist, given the rarity of displacement in the first place.

So let us implement our solution:

(25)
```
[tametsi {CH(Disc:Focus, Case:Nom, Pers:2, Num:Sg),
          CH(Type:V)}]
  [sis {UNCH?(Case:Nom, Pers:2, Num:Sg),
       CH(Case:Acc, Gen:Masc, NumSg) → ch(Type:V)}]
    sis
    [curiosus ch(Case:Acc, Gen:Masc, Num:Sg)] |
  tametsi
    tametsi
      tametsi
      [tu {ch(Disc:Focus)
           → unch(Case:Nom, Pers:2, Num:Sg)}]
    [scio {UNCH?(Case:Nom, Pers:1, Num:Sg),
           CH(Wh:0) → ch(Type:V)]
      scio
      [quam {UNCH?(Disc:Focus), CH(Type:V) → ch(Wh:0)}]
        quam
        <sis>
```

Note that the maximal projections move, not the heads of constituent trees. The maximal projections are the highest node containing the features, and we always take the highest node according to Sayeed and Szpakowicz (2004). Now *sis* commands *tu*. We can move *tu*.

(26)
```
[tametsi {CH(Disc:Focus, Case:Nom, Pers:2, Num:Sg),
          CH(Type:V)}]
  [sis {CH(Case:Nom, Pers:2, Num:Sg),
       CH(Case:Acc, Gen:Masc, NumSg) → ch(Type:V)}]
    [tu {ch(Disc:Focus) → ch(Case:Nom, Pers:2, Num:Sg)}]
    sis
      sis
      [curiosus ch(Case:Acc, Gen:Masc, Num:Sg)] |
  tametsi
    tametsi
      tametsi
      <tu>
    [scio {UNCH?(Case:Nom, Pers:1, Num:Sg),
           CH(Wh:0) → ch(Type:V)]
      scio
      [quam {UNCH?(Disc:Focus), CH(Type:V)
             → ch(Wh:0)}]
        quam
        <sis>
```

All optional unchecked features have been eliminated, and the derivation is complete.

4 Conclusions and Future Work

Using the system of Sayeed and Szpakowicz (2004), we have demonstrated a means to parse sentences with constituents extracted from embedded clauses for prosodic reasons in Latin—constituents that appear to be able to escape even subject islands. We were able to maintain the adjacency requirement of our system by making use of discourse features inspired by Rizzi's analysis of the left periphery in Italian in a GB framework. Thus, this highly constrained incremental system was able to parse a sentence with a long-distance displacement.

In order to do it, though, we had to add a stipulation to the system to allow the constituent that required the displaced one to move to a commanding position. We also took no heed to cyclicity in this system, which given the apparent island violation permitted by these constructions, may not seem so bad, especially since the displaced constituent only moves over one CP in the examples we gave. But Kessler finds that there are rare examples where it moves over two CPs. Of course, these cases are even more rare than displacement over a single CP. It could be that the difficulty in violating subjacency is what makes these cases rare, but the checking of the discourse feature that causes the displacement is more important.

One characteristic of our solution and, indeed, Sayeed and Szpakowicz (2004) in general is that in order to maintain incrementality, we do not attempt to return items displaced during generation to their original positions. We still perform only raising, just as in most GB and minimalist accounts of movement. This means that if the constituent of a phrase is higher than its rightful parent in the tree, the lower subtree raises to claim it. In this case, we had to stipulate that constituent subtrees searching for their own constituents could move to intermediate locations as adjuncts, something that Sayeed and Szpakowicz (2004) did not specify. However, we still maintain an essential property of our system: movement happens as soon as possible. This means that the first available compatible intermediate location is sought. It becomes an empirical question, then, whether an intermediate position could ever be a wrong position.

References

Liliane Haegeman. 1994. *Introduction to Government and Binding Theory*. Blackwell, Oxford, 2nd edition.

Brett Kessler. 1995. Discontinuous constituents in latin. http://www.artsci.wustl.edu/∼bkessler/latin-discontinuity/discontinuity.ps.

Luigi Rizzi. 1997. The fine structure of the left periphery. In L. Haegeman, editor, *Elements of Grammar*, pages 281–337. Kluwer, Dordrecht.

Asad Sayeed and Stan Szpakowicz. 2004. Developing a minimalist parser for free word order languages with discontinuous constituency. In José Luis Vicedo, Patricio Martínez-Barco, Rafael Muñoz, and Maximiliano Saiz, editors, *EsTAL—España for Natural Language Processing*. Springer-Verlag.

Edward P. Stabler. 2001. Minimalist grammars and recognition. In Christian Rohrer, Antje Roßdeutscher, and Hans Kamp, editors, *Linguistic Form and its Computation*. CSLI Publications, Stanford.