Cyclicity in Minimalist Syntax

Gregory M. Kobele

Abstract

What is the relationship between the extension condition and the cycle in e.g. phonology? I explore the analytical landscape and conclude that we must distinguish derivational and derived structure.

1. Introduction

The cycle, and references to cyclicity, appears throughout linguistics. Many terms in linguistics do double duty both as a name for a mechanism or analysis and as a label for a set of facts that that mechanism is intended to explain. As noted by Freidin (1978, 1999), cyclicity was introduced in transformational grammars to restrict the power of the model. Certain kinds of ungrammatical constructions (superraising, relativized minimality) could be blocked by means of this mechanism. By a *façon de parler* any mechanism which blocks these constructions could be termed cyclic. One might be forgiven, however, for wondering whether this situation is the reflection of a deep truth about language, as opposed to a historical accident. If this were the case, we might expect that the phenomena associated with cyclicity could and should be given a uniform formal treatment. In this short paper I will investigate whether cyclicity as currently implemented in minimalist syntax can be unified with cyclicity as implemented in other domains, and if so, how. My answer will be that cyclicity is exclusively about the mapping from one structure to another, and thus that there is no meaningful unification of *syntactic cyclicity* with *interface cyclicity*. However, I will argue that the popular conception of syntactic cyclicity is more properly viewed as an interface condition.

Cyclicity, 73–82
Mariia Privizentseva, Felicitas Andermann & Gereon Müller (eds.)
LINGUISTISCHE ARBEITS BERICHTE 95, Universität Leipzig 2023

2. Cyclicity in Syntax and Elsewhere

Cyclicity is used in the domain of (morpho-)phonology to describe the interaction between modules: phonological rules apply to the intermediate outputs of bottom-up morphological structure building. As an equation:

$$\pi([\mathsf{m}\ \text{-}\mathsf{AFF}]) = \textsc{Phon}(\pi(\mathsf{m}), \text{-}\mathsf{AFF})$$

Here, $\pi(\cdot)$ is the (phonological) interpretation of a morphological structure. The operation $\textsc{Phon}(\cdot,\cdot)$ is responsible for attaching the right form of the affix to the right spot in $\pi(\mathsf{m})$, and applying the relevant phonological rules to the result. This allows for both phonological rules to depend on morphological structure, as well as limitations on this dependence to be stated.

Looking for an analogue of this in syntax, we would expect cyclicity to be a property of the interfaces: semantic terms (for example) should be computed incrementally during the derivation. However, Chomsky (1995: chap.3) asserts that requiring that all structure building target the root of the tree (with the implicit assumption that it target *only* the root of the tree) is the proper way to bring the concept of (strict) cyclicity into modern syntactic theory. Demanding that syntactic structure building target the root—the Extension Condition (EC)—seems a completely different kind of thing from allowing rules from one domain to depend on rules from another domain.

When cyclicity was first applied to syntactic theory, the formal model of syntax was very different. By *Aspects*, Chomsky (1965) had adopted the idea of a context-free base component. A grammar consisted of a context-free base, together with a (sometimes ordered) set of transformational rules. As a given transformational rule could apply in principle to any number of parts of a given input, some strategy was necessary to adjudicate between possibilities. Simple and natural rules seemed to require a *cyclic* (or *inside-out*) mode of scheduling: rules applied to a subtree before applying to anything containing it. This statement could perhaps be interpreted to allow a rule to apply to a subtree by actually applying to a proper subpart of this subtree. A *strict* reading of this statement would require that the rule apply to the entire subtree in question. An obvious question was which subtrees rules should apply to. A very simple proposal (see McCawley 1988: Chapter 6) is that rules should apply at *every* subtree. McCawley (1988) observes that this allows for rule ordering to be abandoned: rules apply in a (strict) cyclic manner whenever their structural descriptions are met. *This* sounds a lot more like what we began with: if you apply as soon as you can, you will (of course) be targeting the root of the current tree! A more influential proposal has it that rules are applied only at subtrees with certain syntactic properties (i.e. having a particular syntactic category). They then needn't apply to the entire subtree, but rather to any part of it that properly includes the previous subtree with that property. This in turn sounds very similar to ideas about *feature inheritance* (Chomsky 2008), which necessitates mild abrogations of the EC. It would seem that the identification of cyclicity in minimalist syntax with the EC (or relaxations of it) is faithful to the original conception of cyclicity in transformational grammar.

But what of the *other* notion of cyclicity, that which seemed to point towards the interfaces? How do these notions of cyclicity relate to one another, aside from onomastically? We will see that the original notion of cyclicity in transformational grammar, properly understood, was in fact the same as the interface notion. However, this raises issues with the glib identification of cyclicity in minimalist syntax with the EC.

3. Cyclicity in Transformational Grammar

The popular conception of transformational grammar is that a derivation proceeded by first obtaining a base structure (a tree), and then applying transformations to it in a cyclic manner. Transformations could in principle interact with (i.e. feed, bleed, etc) one another, but these could not retroactively affect the base structure—if, working bottom up, transformations change what was once a VP into an NP, this doesn't change the fact that this NP née VP was combined with an NP on its left to make an S in the base component. In other words, *syntactic selection* was purely a matter of the context-free base component, whereas transformations manipulated the God-given base structure.

Cyclic rule application over a tree is naturally stated in terms of interleaving the tree construction process with the rule application process. For this purpose, it is more useful to view context-free grammar productions in a bottom-up way; thus the production $S \rightarrow NP\ VP$ is viewed *not* as rewriting an S into NP and VP, but rather, as combining a pre-existing NP and VP together to obtain an S. Because the bottom-up interpretation of a production rule is less familiar, I will use a different notation so as to remind us that productions are to be so interpreted: $S \vdash NP\ VP$. Thinking of rules in a bottom-up way, we now actually *have* a particular NP (say u) and a particular VP (say v) that we are combining to produce an S. This S is constructed out of the NP u and the VP v by introducing a new node labeled S as the parent of these two (in that order): $[_S\ u\ v]$. This information is implicit in the production rule notation, but we can make it explicit in the bottom-up notation by writing the derived object in parentheses after the category name: $X(w)$, which can be read as 'w is an object of category X'. Our rule now looks as follows: $S([_S\ u\ v]) \vdash NP(u)\ VP(v)$.

As we want to treat categorial selection differently from the objects we construct (selection cannot be changed by transformational rules, the object we build can), it is useful to have a notation of this sort that distinguishes them. Curry (1961) calls the aspect of grammar dealing with categorial selection *tectogrammar*, and the aspect dealing with the objects constructed *phenogrammar*. This same distinction is made in Abstract Categorial Grammars (de Groote 2001: (ACGs)), where it is easier to see how this notion applies in general to *interfaces*: the tectogrammatical structure is the input to an interface, and the *phenogrammatical* structure is its output. The ACG perspective views the rule $S([_S\ u\ v]) \vdash NP(u)\ VP(v)$ as an operator ρ of type $NP \to VP \to S$, which is interpreted as a function $[\![\rho]\!] = \lambda u, v.\, [_S\ u\ v]$ of type $\mathsf{tree} \to \mathsf{tree} \to \mathsf{tree}$. A well-typed term M of atomic type can be thought of as representing a derivation of the tree $[\![M]\!]$, where $[\![M(N)]\!] = [\![M]\!]([\![N]\!])$.

Let us now write tr for the process of applying an ordered sequence of transformational rules to a structure. Then, as we surely want to apply transformations at S nodes, we can add this information to our base rule as follows: $S(\mathsf{tr}([_S\ u\ v])) \vdash NP(u)\ VP(v)$. In other words, if we have an NP u and a VP v, we can construct an S from them by applying a round of transformational rules to the object $[_S\ u\ v]$. Note that the property of being an S (say) is different from being an object whose root is labeled with the symbol S. Being an S means that you are something that can be used by a rule that has S on its right hand side. What is important here is that we see that the transformational component (the symbol tr) lies squarely in the phenogrammatical component. This shows that the transformational notion of cyclicity is in fact the same as the interface notion of cyclicity familiar in the morphophonological world: it's just that the 'interface' here is the map between the base structure and the surface structure.

4. Cyclicity in Minimalism

One of the changes that occured on the road from transformational grammar to minimalism was the syntactification of transformations. In other words, transformations were subsumed by the operations of the base—in particular by the single binary operation MERGE.[1] The effects of MERGE are commonly thought of in terms of adding a new node which is an immediate parent of both of its arguments: $[\![\textsc{Merge}(\alpha,\beta)]\!] = [[\![\alpha]\!]\ [\![\beta]\!]]$.[2]

In terms of the derivation, MERGE as defined above applies directly to its two arguments. This builds in the extension condition, at least derivationally (i.e. syntactically). As we wish to see how extension relates to cyclicity, we would like to avoid building extension into the system. In linguistic terms, we incorporate *timing* into the definition of MERGE. This requires us to permit MERGE to apply to subparts of expressions. In order to make this precise, MERGE must take *four* arguments—$\textsc{Merge}(A,a,B,b)$—where two arguments (lower case a and b) are the two terms which will be merged together, and the other two arguments (upper case A and B) indicate at what point in the derivation this is to happen. We intend a to be a subterm of A, and b to be a subterm of B.[3] This is easiest to visualize via a picture, as shown in figure 1. In the figure, the left and right subtrees are the A

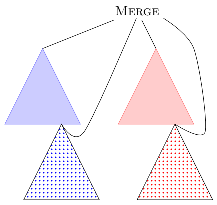


Figure 1: $\textsc{Merge}(A,a,B,b)$

and B arguments respectively. The dotted subtrees of each are the a and b

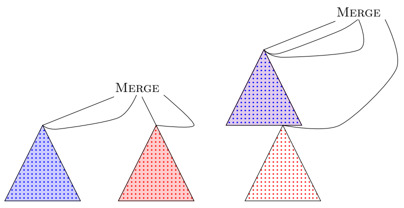
[1]This is not entirely accurate. The effects of transformations have been distributed across components of the grammar. Some effects of transformations have been moved to the interfaces, for example copy deletion/trace conversion.

[2]This is canonically written in set notation: $[\![\textsc{Merge}(\alpha,\beta)]\!] = \{[\![\alpha]\!],[\![\beta]\!]\}$. This is of course just another notation for unordered trees where all daughters of a node are distinct.

[3]We can enforce this by changing the types of a and b from terms to pointers to nodes: $\textsc{Merge} : \mathsf{forall}\ (A : \mathsf{Term}),\ \mathsf{Node}(A) \to \mathsf{forall}\ (B : \mathsf{Term}), \mathsf{Node}(B) \to \mathsf{Term}$.

arguments respectively. Intuitively, we want to understand $\textsc{Merge}(A,a,B,b)$ as saying "merge objects a and b together, but retroactively, after embedding them in A and B respectively".

We will say that MERGE is countercyclic in a if $A \neq a$, and countercyclic in b if $B \neq b$.[4] Being countercyclic in this sense means the same thing as violating the extension condition (in a particular argument). If $A = a$ and $B = b$ then we have normal (cyclic) external merge. Internal merge obtains if $A = a$ and $A = B$ but $B \neq b$. That is, internal merge is countercyclic in b. Note that in both cases there are exactly two distinct arguments: in (cyclic) external merge these are the $A = a$ and $B = b$ arguments, and in internal merge these are the $A = a = B$ and the b arguments. Thus in both cases, merge can be treated as a binary operation. These cases of merge are depicted in figure 2.



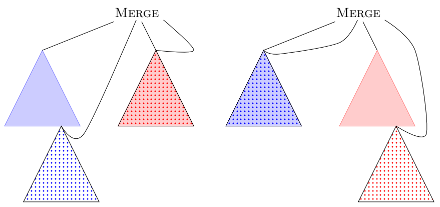
(a) external (b) internal

Figure 2: Canonical merge

Now let $A \neq B$. If $A \neq a$ but $B = b$ then we have countercyclic merge of b inside of A, what Citko (2005) calls *parallel merge* and van Riemsdijk (2006) calls *grafting*. In the reverse situation ($A = a$ but $B \neq b$), we have what Nunes (2001) calls *sideward movement*. This is depicted in figure 3.

Crucially, allowing any countercyclicity in MERGE (including the countercyclicity of internal merge) means that the derivations are no longer proper trees, but rather graphs (multiple dominance structures). If there is a finite upper bound on the number of possible targets of reentrant arcs in any given structure, as is enforced by Ed Stabler's so-called **SMC** constraint (Stabler

[4]This terminology is unfortunate, as it sounds like it has something to do with the notion of cyclicity under discussion. Whether it does is the subject of this paper. This terminology reflects current linguistic practice, which presupposes the connection.



(a) parallel (b) sidewards

Figure 3: Non-canonical merge

1997) in the case of internal merge, then these graphs can be encoded as trees.

4.1. The Extension Condition

As in transformational grammar, we have in minimalism two levels of structure: 1. the derivation, and 2. the structure so derived.We wish to ask whether the extension condition holds at which levels.

Here we are confronted with the fact that the EC is stated for trees, rather than for multi-dominance structures. By their very nature, all derivational operations satisfy the no tampering condition (NTC)—requiring that the inputs to an operation be preserved in the output—and this is often thought of as being stricter than the EC. On the other hand, we might wish to require that structure building *exclusively* target the root. This would then rule out as violating the EC any sort of reentrancy—in our terms, any merge step countercyclic in any of its arguments. As alluded to at the end of the previous section, reentrancy needn't be explicitly represented (and thus, can be formally eliminated) if the targets of reentrancy are uniquely determinable. As an example, many constraints on movement proposed in the minimalist program have a 'superlative' flavor—*Shortest Move*, *Attract Closest*, *Minimal Link* (though consider in this context the notion of *equidistance*)—which suggest that the identity of the mover might be uniquely recoverable just from the information that at a particular point in the derivation a movement took place. More generally, sidewards movement and parallel merge can be made compatible with this restricted derivational EC, so long as the sidewards mover and the target of parallel merger respectively can be reconstructed from the derivational stage at which merge applies.

The main question of interest is thus whether the EC holds of derived structure, which, as we have seen in the case of transformational grammar, can be thought of as the phenogrammar (i.e. in terms of interfaces). Viewing MERGE, as described above, as adding a new node to a graph, which immediately dominates its arguments, the objects it derives satisfy the EC just in case two of the following statements are true: $A = a$, $B = b$ and $A = B$.[5] In other words, this allows for exactly the structure building effects of cyclic merge and move, as the EC was designed to do. In contrast to our initial situation, now that derivation and derived structures have been distinguished, we see that this involves the interleaving of structure building and interpretation.

The difficulty with derived structure is that it is of necessity somewhat ephemeral—its entire *raison d'être* is to serve as the input to some other process, such as linearization. As we change our respresentation of derived structure, so too changes what might count as conforming to the EC. As a concrete example, consider the PF-interface, which we suppose is responsible solely for linearizing the terminals in our derived structure. As shown by Michaelis (2001) and Harkema (2001) (and intuited by Brosziewski (2003)), the mapping from derivation to string in Stabler's Minimalist Grammar formalism can be achieved by maintaining a tuple of strings without the need for derived structure. The EC no longer straightforwardly applicable to such a representation. However, if we understand the intent of the EC to be that of limiting changes to already constructed structures (along the lines of the NTC), then the generalized EC holds of a tuple of strings if only string concatenation is used to combine the components of tuples with one another (as opposed to substitution, or infixation). Splitting a single derived object up into parts (i.e. a tuple) allows for operations conforming to the generalized EC to apply which would have fallen afoul of the EC on the original derived object. For example, tucking-in movement (Richards 1999) satisfies the generalized EC so long as the sequence of specifiers of a head is split off from

[5]There are four possibilities, of which the following two have not yet been discussed: 1. $A = B = b \neq a$, and 2. $A = a = B = b$.The first case is the mirror image of internal merge, where the second to-be-merged argument of the MERGE operation properly contains the first. One could potentially think of this in linguistic terms as reprojection. The second case could be thought of as self-merge.

this head and its complement—this makes the innermost specifier position directly accessible without modifying an already built structure.

5. Conclusion

Defining cyclicity uniformly as interleaving structure building with interpretation accounts for what was called the transformational cycle in transformational grammar. The extension condition in minimalism can also be viewed in this manner, with the EC regulating the mapping from derivation to derived structure. However, it is extremely sensitive to representational choices, and thus appears *ad hoc*.

The EC can be applied to syntax proper (the derivation), where it requires that the targets of operations be uniquely determinable from their point of application. However, this does not have anything to do with cyclicity as a formal mechanism.

An important aspect of cyclicity as a mechanism is that the objects being constructed are only semipermeable to subsequent manipulation. The sensitivity of the EC to representational choices can be understood in this light—instead of condemning the EC as representation dependent, we use the kind of manipulation which is possible for already constructed derived object to infer a representation which allows (just) those to take place under the EC.

References

Brosziewski, Ulf (2003): *Syntactic Derivations: A Nontransformational View*. Linguistische Arbeiten, Max Niemeyer Verlag, Tübingen.

Chomsky, Noam (1965): *Aspects of the Theory of Syntax*. MIT Press, Cambridge, Massachusetts.

Chomsky, Noam (1995): *The Minimalist Program*. MIT Press, Cambridge, Massachusetts.

Chomsky, Noam (2008): On Phases. *In:* R. Freidin, C. P. Otero and M. L. Zubizarreta, eds, *Foundational Issues in Linguistic Theory*. MIT Press, Cambridge, Massachusetts, pp. 133–166.

Citko, Barbara (2005): 'On the Nature of Merge: External Merge, Internal Merge, and Parallel Merge', *Linguistic Inquiry* **36**(4), 475–496.

Curry, Haskell B. (1961): Some Logical Aspects of Grammatical Structure. *In:* R. O. Jakobson, ed., *Structure of Language and its Mathematical Aspects*. Vol. 12 of

Symposia on Applied Mathematics, American Mathematical Society, Providence, pp. 56–68.

de Groote, Philippe (2001): Towards Abstract Categorial Grammars. In: *Association for Computational Linguistics, 39th Annual Meeting and 10th Conference of the European Chapter, Proceedings of the Conference*. pp. 148–155.

Epstein, Samuel David and Norbert Hornstein, eds (1999): *Working Minimalism*. Number 32 *in* ‘Current Studies in Linguistics’, MIT Press, Cambridge, Massachusetts.

Freidin, Robert (1978): ‘Cyclicity and the Theory of Grammar’, *Linguistic Inquiry* **9**(4), 519–549.

Freidin, Robert (1999): Cyclicity and Minimalism. *In* Epstein and Hornstein (1999), chapter 5, pp. 95–126.

Harkema, Henk (2001): Parsing Minimalist Languages. PhD thesis, University of California, Los Angeles.

McCawley, James D. (1988): *The Syntactic Phenomena of English*. Vol. 1, The University of Chicago Press.

Michaelis, Jens (2001): On Formal Properties of Minimalist Grammars. PhD thesis, Universität Potsdam.

Nunes, Jairo (2001): ‘Sideward Movement’, *Linguistic Inquiry* **32**(2), 303–344.

Richards, Norvin (1999): Featural Cyclicity and the Ordering of Multiple Specifiers. *In* Epstein and Hornstein (1999), chapter 6, pp. 127–158.

Stabler, Edward P. (1997): Derivational minimalism. *In:* C. Retoré, ed., *Logical Aspects of Computational Linguistics*. Vol. 1328 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 68–95.

van Riemsdijk, Henk (2006): Grafts follow from Merge. *In:* M. Frascarelli, ed., *Phases of Interpretation*. Vol. 91 of *Studies in Generative Grammar*, Mouton de Gruyter, pp. 17–44.