A Memory Architecture for Merge
`Draft`[*]

David Adger
Queen Mary University of London

February 2017

1 Introduction

Restrictiveness and simplicity are not the happiest of bedfellows. An easy means of restricting the yield of a generative system is to place extra conditions on its operation with the result that the system as a whole becomes more complex. Simplifying a system typically involves reducing or removing these extra conditions, potentially leading to a loss of restrictiveness.

Chomsky's 1995 introduction of the operation Merge, and the unification of displacement and structure building operations that it accomplishes, was a marked step forward in terms of simplifying the structure building component of generative grammar. But the simplicity of the standard inductive definition of syntactic objects that incorporates Merge has opened up a vast range of novel derivational types. Recent years have seen for example, derivations that involve rollup head movement, head-movement to specifier followed by morphological merger (Matushansky 2006), rollup phrasal movement (Svenonius 2007), undermerge (Pesetsky 2013), countercyclic tucking-in movements (Richards 2001), countercyclic late Merge (Takahashi and Hulsey 2009), and, the topic of this brief paper, sidewards movement, or, equivalently, Parallel Merge (Nunes 2001, Citko 2005).

[*]Many thanks to the audiences at my Baggett Lectures at the University of Maryland (January 2016) and at Queen Mary University of London's Linglunch group for comments.

An alternative to adding conditions to a generative system as a means of restricting its outputs is to build the architecture of the system in such a way that it allows only a restricted range of derivational types, that is, to aim for an architecture that embodies the constraints rather than representing them explicitly (cf. Pylyshyn 1993). This opens up the possibility of both restricting a system and simplifying it. In Adger (2013), for example, I argued for a system that does not project functional categories as heads, following Brody (2000). This immediately removes derivational types involving certain kinds of head movement from the computational system. Apparent head movement effects have to be, rather, a kind of direct morphologization of syntactic units in certain configurations. No heads means no rollup head movement, no head to specifier movement followed by morphological merger, no 'undermerge' (Pesetsky 2013) and no parallel merge derivations for head movement (Bobaljik 1995). That same system (Adger 2013) also rules out roll-up phrasal movements via an interaction between the structure building and labelling components of the grammar (essentially, roll-up configurations lead to structures with two complements). It follows that the kinds of roll-up remnant derivations argued for by Kayne (2004) or Cinque (2005) are ungenerable and the empirical effects they handle must be dealt with otherwise. In all of these cases the concern was to reduce the range of derivational types by constructing a system whose architecture simply does not allow them. Adger 2013 makes the argument that the system presented there is at least no more complex than standard Bare Phrase Structure architectures (Chomsky 1995).

In this paper I extend this reasoning to Parallel Merge. I develop a theory of the core computational operation of the generative system which makes Parallel Merge impossible. This involves attributing a memory architecture to Merge. The standard approach takes there to be long term memory (items stored in the lexicon) and a kind of cache-memory (the Workspace) that stores the items that are involved in the current derivation and serves as a symbol space for their manipulation. To this I add an analogue of a register-memory: a very small, in fact binary, memory space which is where the operation of Merge takes place. I show that this architecture rules out Parallel Merge derivations, and provide some empirical reasons for thinking that this is the right result.

2 A Framework

Collins and Stabler (2016) provide an explicit formulation of one version of minimalist syntax that uses workspaces and Merge. For them, a workspace is a set and a derivation successively modifies it, giving a succession of workspaces. Collins and Stabler implement conditions on Merge as explicit constraints, as follows:

(1) For some A, B

a. A $\in$ W$_i$,

b. either A contains B or W$_i$ immediately contains B

c. W$_{i+1}$ = (W$_i$ $-${A, B}) $\cup$ {Merge(A,B)}

Here, W$_i$ is the current workspace, and A and B are syntactic objects. This definition captures one conception of how the system operates. However, the restrictions imposed by the definition seem arbitrary. The first disjunct in (1) (b) allows Internal Merge and the second allows External Merge. This system rules out Parallel Merge derivations by fiat. To see why, consider a simple parasitic gap example:

(2) What did Lilly play with while eating?

A Parallel Merge approach to this would involve a derivation where *what* is Merged with *eating*, and then later Merged with *with*. At some point, the workspace looks as follows:

(3) { ... {eating, what}, ..., play, with, ...}

In order to Internally Merge *what* and *with*, one must contain the other (which obviously is not the case). For External Merge, *with* is a member of the Workspace, and the second part of the disjunction in (1) (b) requires that *what* is also immediately contained in the workspace. It is therefore impossible for *with* to Merge with *what*, a case of Parallel Merge.

But why have these restrictions? Why not "A immediately contains B" or "W$_i$ contains B'? These options would give, respectively, movement of just the daughters of some syntactic object, or Parallel Merge. The Collins and Stabler system simply specifies what kinds of Merge are available, a shortcoming which they are well aware of. They note (p8) "However, it is important to highlight that relatively minor changes in our assumptions about derive-by-Merge would allow sideward Merge."

Turning to the workspace itself, we can raise the question how do elements get into a workspace? Since Chomsky (1995), the suggestion has been that a lexical array of some sort furnishes workspaces with the elements that are available to Merge via an operation that Chomsky calls Select. Collins and Stabler formalize Select as an operation that places elements from a finite array of lexical items (LA) into a workspace. Their definition looks as follows:

(4) Let S be a stage in a derivation S=<LA, W>.
If lexical token A $\in$ LA, then Select(A, S)=<LA$-\{$A$\}$, W$\cup\{$A$\}$>

So for Collins and Stabler, Select removes lexical items from a lexical array and enters them into a workspace, which is then manipulated via Merge. This is faithful to the general architecture laid out by Chomsky (1995), however, more recent formulations (e.g. Berwick and Chomsky 2016) adopt a more general approach.

> We can think of the computational process as operating like this. There is a workspace, which has access to the lexicon of atomic elements, and contains any new object that is constructed. To carry a computation forward, an element X is selected from the workspace, and then a second element Y is selected. X and Y can be two distinct elements in the workspace, ..., what is called External Merge. Or one can be part of the other, called Internal Merge, ... Berwick and Chomsky 2016

This proposal for Merge licenses computations that look as follows. External Merge is what we get when both X and Y are drawn from the workspace:

(5) a. Select X from workspace
b. Select Y (from workspace)
c. Merge(X, Y) = {X, Y}

Internal Merge is identical, except the source of Y is now from within X:

(6) a. Select X from workspace
b. Select Y (from X)
c. Merge(X, Y) = {X, Y}

Unlike in Collins and Stabler's formalization, this approach to Select has it applying to not just lexical items, but also to complex syntactic objects. Select identifies the arguments for Merge.

In that system we can construct a Parallel Merge derivation as follows:

(7) a. Select *with* in WorkSpace
b. Select *what* from {eating what} (already constructed and in Workspace)
c. Merge(with ,what)= {with,what}
d. ...

It is clear that what allows Parallel Merge here is the capacity to Select from inside an already constructed object in the Workspace, a capacity that Collins and Stabler rule out via an explicit constraint on what is accessible in a workspace. It is not obvious how one might apply such a constraint in the Berwick and Chomsky architecture. Part of the core workings of such an architecture is that it is possible to select from inside an already constructed object. This is what allows the unification of Move and Merge. To rule out Parallel Merge, we need it to be the case that an object contained in an object in the workspace is inaccessible to Merge. Ideally this should follow from the structure of the system, as opposed to an external constraint.

3 Revising the Framework

Rather than stating explicit constraints on what properties the computational system should have, I'd like to think about what the basic elements and processes of the computation might actually be, following the framework sketched in the quote from Berwick and Chomsky above.

Let's first consider the role of the workspace.

I'll assume, as is standard, that the workspace has access to the lexicon, and that it is a multiset (as opposed to a set) so that it allows repetitions. A typical workspace might then look at follows (I'll abstract away from whether vP is a phasal category and hence defines its own workspace):

(8) [the, the, dog, cat, chases, v, T, C]

I use the standard [ ] notation for multisets, and { } for sets.

The workspace stores the resources for the continuing derivation. So once we have applied Merge, (8) becomes (9):

(9) [the, {the, dog}, cat, chases, v, T]

For Collins and Stabler, the operation Merge applies to elements in the workspace. However, I'd like to make a minor modification to the architecture here, and split the workspace in two. Let's call the multiset that contains the resources for the derivation going forward the Resource Space. In addition to the Resource Space, I propose a second memory buffer. I'll call this the Operating Space. The Operating Space has access only to the Resource Space, and is a set (as opposed to a multiset) with maximal cardinality 2.

The Operating Space is where Merge takes place. What Merge does is to combine the two 'cells' in the Operating space into a single object. The general architecture is then as follows:

(10) Lexicon $\rightarrow$ Resource Space=[...] $\leftrightarrow$ Operating Space = { , }

The lexicon is a static structure. When a lexical item is copied from the Lexicon to the RS, it leaves the lexicon unchanged. RS and OS are dynamic, and a derivation is essentially a list of RS, OS pairs, with information being transfered between the two memory buffers, and Merge creating single objects only in OS. The computation minimizes the size of RS and OS at each step in a way that will become clear.

We can now define Select as an operation that reads and writes information in these memory buffers. I take it to work as follows:

(11) Select(i, D$_{\mathit{1}}$, D$_{\mathit{2}}$) succeeds if i is a member* of D$_{\mathit{1}}$ and is written to D$_{\mathit{2}}$.

The predicate member* here is transitive membership relativised to the domain. If the domain is a set, then member* recursively picks out all members and member*s of those members that are sets (i.e. what are standardly called *terms*). If the domain is a multiset, then member* recursively picks out all members and member*s of those members that are multisets. Member* is a kind of search that picks a data structure, and sticks to it until the search is complete.

These architectural choices have a number of consequences. RS is a multiset but since it only draws from the lexicon or from OS (after Merge), its members will be lexical items or sets, and not multisets. It follows that member* applied to RS only picks out the members of RS, not the members of the sets in RS. Since member* is relativized to the domain, if it applies to a multiset, it will only identify members of that multiset and members of any members that are multisets (recursively).

The Operating Space is, however, a set, and its members will be sets or lexical items. Member* applied to OS picks out members of the OS and member*'s of those members that are sets.

From this it follows that Select will be a kind of search that is restricted to being horizontal in the Resource Space (that is, it cannot see inside elements in the Resource Space), but can be vertical in the Operating Space, by virtue of the nature of the datastructures involved.

This leads to four logical possibilities, which I enumerate here.

1. The first is that Select(i, RS, OS) takes a member* of RS and writes it to OS. There is a question of whether this writing operation leaves i in RS. If the computation minimizes the memory buffers where possible, then i will delete from RS. No information is lost across the system, since i is in OS. Let's adopt this assumption: when Select applies to RS, the element is removed from RS (cf. the definition of Select applied to lexical array given by Collins and Stabler 2016 above). It also has the consequence that RS will not end up filled with copies of everything that has undergone Select.

   A plausible motivation for this, mentioned by Chomsky (2008), is that RS is the input to Transfer, so that when a single object of the relevant size (e.g. a phase) has been constructed, the contents of RS are Transfered (equivalently, the singularity of RS marks a point where the object constructed is evaluated for convergence). Extra copies of syntactic objects that have undergone Select operations would make this simple statement of convergence impossible.

2. The second possibility is Select(i, OS, OS). This takes a member* of OS and writes it to OS. In this case there is no need to delete i, as OS is never checked for convergence or any other kind of interpretive procedure. If i is just the sole member of OS, then we have a case of Self-Merge. If i is a member* of the single member of OS, then i is written to the other cell in OS, and we have a case of Internal Merge. Just as in the standard system, there is no distinction in the operations that lead to structure building and displacement. The difference is simply where Merge's arguments come from.

3. The third possibility is that Select(i, OS, RS) takes a member* of OS and writes it to RS. In this case, i deletes from OS as a consequence of

the system minimizing memory, which has the effect of emptying OS for next operation. This is the operation that returns the constructed syntactic object to the RS for further manipulation

4. Finally, a logical possibility is Select(*i*, RS, RS). This takes a member* of RS and writes it to RS. However, if the previous decision we made about RS is correct, *i* deletes from RS. Such an operation is, of course, vacuous.

These are the four logical possibilities given this architecture. There is a single read-write operation, and a single combinatorial operation.

One noteworthy point about this architecture is that whether an element is deleted from a memory space is a somewhat 'global' property. When Select applies to write the content of OS to RS, OS is emptied, or the system would simply halt. Similarly, unless elements are deleted after Select applies to the RS, the system will never halt. The system conspires so that the only situation where Select applies and there is no subsequent deletion is for the Select(*i*, OS, OS) case.

3.1 Derivational Types

This architecture is a particular development of the proposal in Berwick and Chomsky (2016), with specific choices made. As we shall see directly, it reduces the range of derivational types. Specifically, Parallel Merge and Late Merge derivations are impossible in this system.

External Merge works as follows, giving the usual result. Let us start with a state of the system where the RS looks as follows:

(12) State n: RS = [the, cat, jump, ...]

Now a computational procedure applies.

(13)

Operation	RS	OS
	[the, cat, jump, ...]	{ }
Select(cat, RS, OS)	[the, jump, ...]	{cat}
Select(the, RS, OS)	[jump, ...]	{cat, the}
Merge({cat, the})	[jump, ...]	{{cat, the}}
Select({the, cat}, OS, RS)	[{cat, the}, jump, ...]	{ }

(14) State n+1: RS = [{cat, the}, jump, ...]

Merge in this system just applies to the two elements in OS creating a single object.

(15) Merge(A, B) = {A, B}

Internal Merge also follows in a standard way. Take a state of the system where T has been Merged to vP:

(16) State n: RS = [{T, {{the, cat}, { v, jump}}}, ...]

Abbreviating the structure in (16), where the vP internal subject is in situ as $\bar{\text{T}}$ for reasons of space, we have:

(17)	Operation	RS	OS
		[$\bar{\text{T}}$, ...]	{ }
	Select($\bar{\text{T}}$, RS, OS)	[...]	{$\bar{\text{T}}$}
	Select({the, cat}, OS, OS)	[...]	{{the, cat}, {T, {{the, cat}, { v, jump}}}}
	Merge({{the, cat}, $\bar{\text{T}}$)	[...]	{{{the, cat}, {T, {{the, cat}, { v, jump}}}}}
	Select(TP, OS, RS)	[TP ...]	{ }

Here TP abbreviates {{the, cat}, {T, {{the, cat}, { v, jump}}}}. The subsequent state of the system is:

(18) State n+1: RS = [{{the, cat}, {T, {{the, cat}, { v, jump}}}}, ...]

Self-Merge derivations, of the type used by Adger 2013, are cases of Internal Merge, with a single object in OS targeted by Select, so that there are two instances of it:

(19)	Operation	RS	OS
		[X, ...]	{ }
	Select(X, RS, OS)	[...]	{X}
	Select(X, OS, OS)	[...]	{X, X}
	Merge(X, X)	[...]	{{X, X}}
	Select({X, X}, OS, RS)	[{X, X} ...]	{ }

The object {X, X} is interpreted by the system as simply {X}, since the two members are token identical.

3.2 No Parallel Merge

A straightforward consequence of this system is that Parallel Merge/Sidewards Movement derivations are not possible. Take a classical example where Parallel Merge/Sidewards Movement is used, such as a parasitic gap construction under the analysis of Nunes (2001):

(20) Guess what Anson recommended before Anders read?

Assuming that there is no special derivational distinction between movement and External Merge in terms of copies etc, Sidewards Movement and Parallel Merge are identical (Citko 2005). The derivation of (20) will involve a workspace (that is a Resource Space) that looks as follows:

(21) RS = [Anson, {recommended what}, read, ...]

The derivation will proceed by attempting to Merge *what* inside *recommend what* with *read*, as follows. First we need to write both *recommend what* and *read* into OS

(22) a. Select({recommended what}, RS, OS)
b. Select(read, RS, OS)

(23) OS = {{recommended what}, read}

But now, within OS, we need to Select *what* from the OS (which is possible, since *what* is a member* of the OS) and write it to the OS. This is not possible, however, as the OS has a maximal cardinality of 2. There is simply not space in memory to carry out the operation.

(24) a. Select(what, OS, OS) *crash*

This result is, I think, a welcome one, at least for parasitic gaps. If Parallel Merge derivations are available to the child, it follows that both gaps in a parasitic gap construction should behave identically. It is well known that there are reconstruction asymmetries in parasitic gaps constructions (Munn 2001, Nissenbaum 2000, a.o):

(25) a. Which pictures of each other did the boys hide before my sister could sell?
b. *Which pictures of each other did my sister sell before the boys could hide?

However, there is a possibility that these effects may be dealt with by an independent condition, perhaps in processing, which impacts the leftmost gap (that is the one first encountered in the parse after the extractee) differentially. Subject parasitic gaps seem to allow reconstruction, perhaps backing up this approach.

(26) a. *Which picture of herself did every boy who saw __ say Mary liked __
b. Which picture of himself did every boy who saw __ say Mary liked __

However, reconstruction is not the only difference that one finds in parasitic gap constructions. Munn (2001) notes that amount readings are unavailable for a parasitic gap:

(27) a. It was amazing the wine we drank __ that night (amount or individual)
b. It was amazing the wine Bill drank __ after Fred spilled __ on the floor ($\neq$ amount reading)

Munn doesn't give subject PGC examples, but these turn out to be equally bad, ruling out some kind of 'leftness' effect as an explanation:

(28) a. Sorry, I bought that wine that everyone who drank __ (last night) hated __
b. It'd fill a bathtub, the wine that everyone who was at the party drank __ (last night) (=amount)
c. *It would fill a bathtub, the wine that everyone who drank __ (last night) hated __

A parallel merge derivation will need some extra stipulation which changes the interpretation of both gaps to something incompatible with an amount meaning, and it is not clear what would motivate that stipulation.

Yet more challenging for a Parallel Merge account is the behaviour of what Munn calls sloppy functional readings. Munn points out that, in contrast to ATB extraction, parasitic gap constructions don't allow paired answers in a conjoined question:

(29) a. Which restaurant did Bill review __ on Monday and Fred review __ on Tuesday

b. Paired answer: Bill reviewed the first on his list and Fred the last

(30) a. Which restaurant did Bill review __ on Monday before Fred reviewed __ on Tuesday

b. *Paired answer: Bill reviewed the first on his list and Fred the last

In these cases, there is only a reading where Bill reviewed the first (say) on his list, and Fred reviewed the same one. This means that we need a true copy for there to be a functional reading available for the trace in the main clause. On the other hand, we need something quite distinct for the parasitic gap in the adjunct in order to rule out the functional reading.

The important point is that the evidence for the asymmetry here is negligible as far as the language acquirer is concerned. If Parallel Merge is possible, it should be available to the child for such constructions, and no asymmetry should be detectable. That there is an asymmetry strongly points to the unavailability of a Parallel Merge strategy. The lack of c-command between the gaps must be sufficient for the child to treat the parasitic gap as an independent A-bar extraction configuration, creating a predicate which is conjoined with the main predicate (as in Nissenbaum's updating of the proposals in Chomsky (1982)).

This leaves open the question of how to treat ATB extraction. ATB is typologically widespread , and tied to coordination (as opposed to parasitic gaps, which are rarer, incidentally supporting the notion that they are to be distinguished). One suggestion is to implement coordinative syntax via a second resource space, whose accessibility is tied to conjunctions, as in multidimensional approaches to coordination (Goodall 1987, Moltmann 1992). I leave development of such an idea to another time.

4 Conclusion

This brief note suggests a way of further educing the range of derivational types available to the computational system of human language by providing a memory architecture of the computation. The fundamental innovation is the splitting of the workspace into two: a Resource Space and an Operating Space, mimicking the cache-register structure of computers. Restricting the Merge operation to the Operating Space means that parallel Merge derivations are impossible.

References

Adger, David. 2013. *A Syntax of Substance*. Cambridge, MA: MIT Press.

Berwick, Robert C. and Chomsky, Noam. 2016. *Why Only Us?*. Cambdridge, MA: MIT Press.

Bobaljik, Jonathan. 1995. In terms of Merge: Copy and head movement. In Rob Pensalfini and Hiroyuki Ura, eds., *Papers on Minimalist Syntax*, volume 27, 41–64, MIT Working Papers in Linguistics.

Brody, Michael. 2000. Mirror theory: syntactic representation in perfect syntax. *Linguistic Inquiry* 31:29–56.

Chomsky, Noam. 1982. *Some Concepts and Consequences of the Theory of Government and Binding*. MIT Press.

Chomsky, Noam. 1995. *The Minimalist Program*. Cambridge, MA: MIT Press.

Chomsky, Noam. 2008. On phases. In Robert Freidin, Carlos P. Otero, and Maria Luisa Zubizarreta, eds., *Foundational Issues in Linguistic Theory*, 133–166, Cambridge, MA: MIT Press.

Cinque, Guglielmo. 2005. Deriving Greenberg's Universal 20 and its exceptions. *Linguistic Inquiry* 36:315–332.

Citko, Barbara. 2005. On the nature of merge: External merge, internal merge, and parallel merge. *Linguistic Inquiry* 36:475–496.

Collins, Chris and Stabler, Edward P. 2016. A formalization of minimalist syntax. *Syntax* 19:43–78.

Goodall, Grant. 1987. *Parallel structures in syntax*. Cambridge: Cambridge University Press.

Kayne, Richard S. 2004. Prepositions as probes. In Adriana Belletti, ed., *Structures and Beyond*, 192–212, Oxford: Oxford University Press.

Matushansky, Ora. 2006. Head movement in linguistic theory. *Linguistic Inquiry* 37:69–109.

Moltmann, Frederike. 1992. On the interpretation of three-dimensional syntactic trees. In Chris Barker and David Dowty, eds., *Proceedings of SALT*, volume 2, 261–281, Ohio State Working Papers in Linguistics.

Munn, Alan. 2001. Explaining parasitic gap restrictions. In Peter Culicover and Paul Postal, eds., *Parasitic gaps*, 369–392, Cambridge, Massachusetts: MIT Press.

Nissenbaum, Jon. 2000. Covert movement and parasitic gaps. In Masako Hirotani, Andries Coetzee, Nancy Hall, and Ji-yung Kim, eds., *Proceedings of the North East Linguistic Society*, 541–556, Rutgers University: Graduate Linguistic Student Association.

Nunes, Jairo. 2001. Sideward movement. *Linguistic Inquiry* 32:303–344.

Pesetsky, David. 2013. *Russian case morphology and the syntactic categories*. Cambdridge, MA: MIT Press.

Pylyshyn, Zenon. 1993. Rules and representations: Chomsky and representational realism. In Asa Kasher, ed., *The Chomskyan Turn*, 231–251, Cambridge, Massachusetts: Blackwell Publishers.

Richards, Norvin. 2001. *Movement in Language: Interactions and Architectures*. Oxford: Oxford University Press.

Svenonius, Peter. 2007. Projections of p, lingBuzz/000484.

Takahashi, Shoichi and Hulsey, Sarah. 2009. Wholesale late merger: Beyond the a/ā distinction. *Linguistic Inquiry* 40:387–426.