Stitching phases together: Domains and edges as modules and interfaces

Universität Leipzig, December 19th, 2014

Thomas McFadden
Zentrum für allgemeine Sprachwissenschaft (ZAS), Berlin
`mcfadden@zas.gwz-berlin.de`

In this talk I'm going to explore an alternative conception of phase theory, according to which phases are first constructed independent of each other, and are then stitched together late in the derivation.

☞ This is roughly the reverse of how things are done in standard phase theory, where a single structure is built up, and phase domains are periodically removed from it and sent to the interfaces.

☞ I submit that this alternative can achieve essentially the same coverage of locality and successive-cyclicity as the standard theory, but does so in a way that is more straightforwardly modular.

The ideas to be presented here are quite immature, and I've only worked out a fraction of the consequences. But they offer us a new way to think about some of the things that are problematic for phase theory, and promise at least some improvements:

- At first blush it seems to offer some advantages in using phase theory to model performance.
- It may also give us a way to improve on places where the 'overgenerate and filter' seems to be required, e.g. in driving intermediate movement steps.
- It shows some promise in dealing with CED effects.

1 Locality, successive-cyclicity and SpeDO

Syntactic relationships appear to be subject to locality.

- E.g. a verb has to agree with the subject of its own clause:

(1) a. *I am* stinky.
    b. *She is* stinky.

(2) a. She thinks [that *I am* stinky].
    b. * *She* thinks [that I *is* stinky].

c. * She *think* [that *I* am stinky].

- And a DP can A-move out of a non-finite clause:

(3) Walt cooks.
(4) *Walt* seems to <*Walt*> cook.

- But not across an intervening DP or out of a finite clause:

(5) * *Walt* is believed **Jesse** to seem to <*Walt*> cook.
(6) * *Walt* is believed [that seems to <*Walt*> cook].

But movement e.g. isn't local in absolute terms:

☞ It's possible to move something over an unbounded distance, as long as there is a series of structures of the right kind in between the starting point and the ultimate landing site of the movement:

(7) *Who* did Ophelia insinuate [<*who*> that Randolph said [<*who*> that Mortimer thought [<*who*> that Louis claimed [<*who*> that Billy Ray saw <*who*>]]]]?
(8) *Walt* appears [<*Walt*> to be alleged [<*Walt*> to be believed [<*Walt*> to seem [<*Walt*> to <*Walt*> cook]]]].

Such apparently long-distance movement seems to proceed in cycles.

☞ Note e.g. that unlike 8, 9 is bad:

(9) * *Walt* appears [<*Walt*> to be alleged [that Jesse believes [<*Walt*> to seem [<*Walt*> to <*Walt*> cook]]]].

☞ We can make sense of this if *Walt* can only cross long distances by moving through every intermediate clause. The finite clause built around *believe* would then present a problem.

☞ If movement of *Walt* could be truly long-distance, it should be able to just skip over that intermediate clause, and the sentence would be grammatical.

Similarly, in at least some languages, agreement can be triggered over long distances as long as intervening clauses are of the right type (Bhatt 2005):

(10) Vivek-ne [*kitaab paṛh-nii*] *chaah-ii*
Vivek-ERG book.F read-INF.F want-PFV.F
'Vivek wanted to read the book.'

In order to deal with facts like these, we assume some version of the following:

(11) **Locality**
Syntactic operations and relationships are fundamentally required to apply within a local domain.

(12) **Successive-cyclicity**
However, local relationships can be chained together under certain restricted circumstances, deriving what on the surface appear to be non-local relationships.

The standard implementation of these ideas in current Minimalism is in terms of **phases** (Chomsky 2000, 2001, etc.). Phases achieve locality with successive-cyclicity by being **opaque domains** with **transparent edges**:

- What's inside one phase is essentially invisible to what's inside another. They are **encapsulated** from one another and cannot interact.
- Except that the upper part — the edge — of a given phase is transparent, i.e. visible to the phase above.
- Edges thus function as **escape hatches**, deriving **successive-cyclicity**, as exemplified in 7 and 8 above.

The basic idea of opaque domains with escape hatches has been around for a while and comes in many versions. The innovation of phase theory was in how these properties fall out of the workings of the derivation:

☞ The big idea is that the operation that sends completed bits of structure to the interfaces can apply more than once in the derivation of a single sentence (Uriagereka 1999).

☞ We then assume that this removes the relevant chunks of structure from the syntactic workspace, so that it creates opacity effects.

☞ Then crucially, we **don't** assume that what is Spelled-out in this way is the complete structure up to that point — that would yield absolute locality, i.e. no interactions across phases at all, which is clearly too strong.

☞ Instead, we let the top bit of the current structure stick around to participate in the next phase of the derivation, deriving successive-cyclicity through this escape hatch.

A bit more concretely:

- Assume that a head H defines a phase P. Upon completion of P, the complement of H, which we call the **Domain** of P, is sent to Spell-out.
- The **Edge** of P, consisting of the head H and any specifiers and adjuncts, remains, and will only be sent to Spell-out as part of the domain of the next phase up.

This straightforwardly gives us an intermediate level of locality restriction along the lines we're looking for:

- We can have relationships between phases if e.g. a head in an upstairs domain Agrees with the head in the downstairs edge.
- And an element can move out of a phase as long as it passes through the edge along the way.
- All other cross-phasal relationships are ruled out.

From here on out I will refer to this standard version of phase theory in terms of its defining feature as **Spelling Domains Out** — **SpeDO** for short.

2 On phases, modularity and StiPT

Note then that the intuition behind phases — that they should be opaque domains with transparent edges — is actually reminiscent of the basic idea of a modular system:

☞ We want to have a series of objects that operate for the most part independently, each doing its own job.

☞ If we encapsulate each object from the others, we can simplify the design and use of the system, because each object can be defined and operate without needing to know anything about the internal affairs of the others.

☞ But since the objects ultimately need to cooperate to build a complex system, they need a way to communicate with each other, at least a little bit.

☞ The way to do this while maintaining the greatest degree of modularity is if the system provides well-defined and restricted interfaces between the objects, and all communication has to proceed via these interfaces.

SpeDO achieves this, but in a roundabout way that isn't as modular as we might like:

- Broadly speaking, there is a single derivational line for an entire sentence, i.e. it is built up as one, not-so-modular, unified structure.
- But this structure is broken up periodically, bringing modularity in sort of through the back door.
- The timing of this breaking up is then carefully calibrated to allow the desired interface between the modules.

Here I would like to explore the idea that we can get the same broad results — and in some cases achieve an improvement — by doing things the other way around, and building in modularity from the beginning:

- Assume that phases are created independently, each built up and manipulated in its own separate **workspace**.
- Only when they're completed can they be **stitched** together to yield the final complete structure.
- All locality-sensitive operations have to take place in the individual phases before they are stitched together.
- Transparency at the edges is achieved by having the edges be literally shared by two adjacent phases, which allows them to communicate, and ensures above all that only the right kinds of phases can be stitched together.

I'm going to call this approach **Stitching Phases Together — StiPT** for short. In the rest of this talk I'll work it out in some detail and see how it fares as an alternative to SpeDO.

3 An alternative conception of phases

3.1 The basic idea

In SpeDO, an essentially unified structure is built up in the active workspace, and pieces (phase domains) are periodically removed from it:

	Workspace			Interfaces
a.			C D	
b.		B	C D	
c.		B	$\boxed{\text{C D}}$	
d.		B	$\rightarrow$	$\boxed{\text{C D}}$
e.	A	B		$\boxed{\text{C D}}$

☞ So in the figure above, A can't have any relationship with D because D is removed from the workspace and shipped to the interfaces before A enters the Workspace.

In StiPT, phases are built up independently, each in its own workspace, and then stitched together into a single structure, in what we can call the **Staging Area**, at the end:

	WS1	**Staging**	**WS2**
a.			C D
b.			$\boxed{\text{C D}}$
c.	A B		$\boxed{\text{C D}}$
d.	$\boxed{\text{A B}}$		$\boxed{\text{C D}}$
e.	$\boxed{\text{A B}}$	$\rightarrow$ $\leftarrow$	$\boxed{\text{C D}}$
f.		$\boxed{\text{A B}}\boxed{\text{C D}}$	

- Again, A can't have any relationship with D, but under StiPT it is because they are in two separate workspaces when all of the syntactic action happens.
- By the time their phases are stitched together, it's too late for any interaction of their internals.

3.2 Opacity through independence

As in SpeDO, opacity is relatively straightforward under StiPT:

- Each phase is built up independently of all others in its own workspace.
- All familiar syntactic operations and relationships — in particular Merge and Agree — take place or hold in the workspace stage.
- There is a single special operation, Stitch, which assembles all the phases together at the end to create the complete structure in the Staging Area.

By hypothesis, Stitch cannot have any effect on the phases it stitches together.

☞ It can only make sure that they fit together properly (we'll come back to how this is done below), and then create a larger structure out of them.

☞ This straightforwardly derives opacity, because elements in two different phases will never be in a single workspace, where the normal syntactic operations are defined to apply.

☞ Stitch can operate over larger structures, but all it can do is combine them together. It can't change anything around internal to a phase or create any new dependencies between phases.

3.3 Transparency through edge matching

Again, as in SpeDO, transparency requires a little something extra. As things stand, we're left again with absolute opacity and hence extreme locality.

- Merge (including Move as internal Merge) and Agree are restricted to happen within individual phases, so there's no way for something to move from one phase to another, or to Agree with something in another phase.
- I.e. there is no successive-cyclicity, hence no way to derive unbounded dependencies, and also no selection or interaction at all between phases.

Notice, however, that we do have an operation that can serve as the appropriate place to handle dependencies between phases:

☞ The job of Stitch is to take completed phases and put them together to build up the full structure of a sentence.

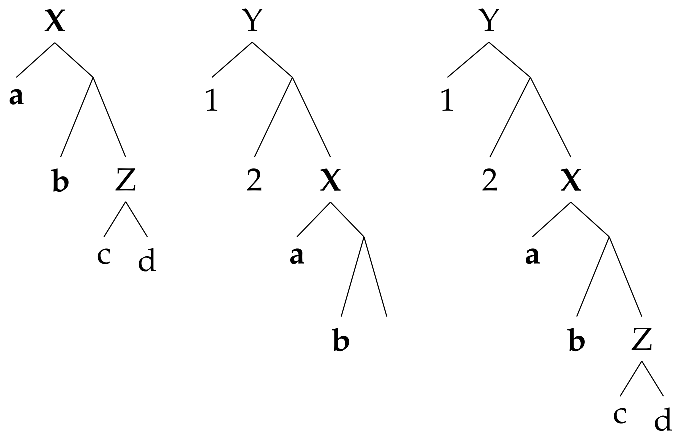
☞ But of course it needs to be able to make sure that the two phases it combines actually fit together in that order, otherwise it would be able to construct all sorts of nonsense.

I think that we can actually use a single restriction on Stitch, formulated in 13 below, to simultaneously get the dependencies between phases and create transparent edges which can derive successive-cyclicity:

(13) **Edge-matching restriction on Stitch**
Stitch can combine together two phases X and Y, as long as the edge at the root of X Matches an edge interior to Y. As a result of Stitch, those edges become a single shared structure.

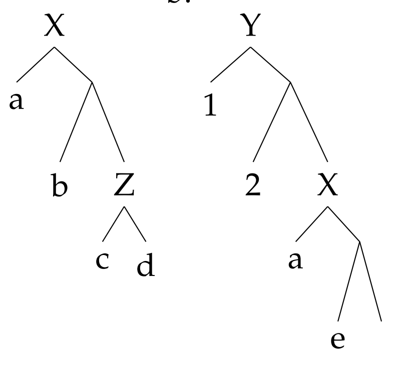
- So we can Stitch 14a with 14b to yield 14c, because the outer edge of 14a (everything above Z) Matches with an inner edge of 14b (everything from X down):

(14) a. b. c.



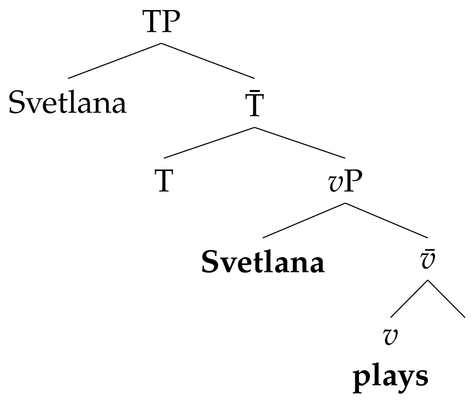
However, there's no way to Stitch together the structures in 15a and 15b, because they don't have any matching edges:

(15) a. b.

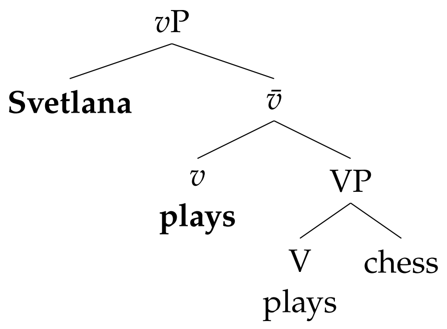


More concretely, we can take something like 16a, and Stitch it together with something like 16b to yield the structure in 17:

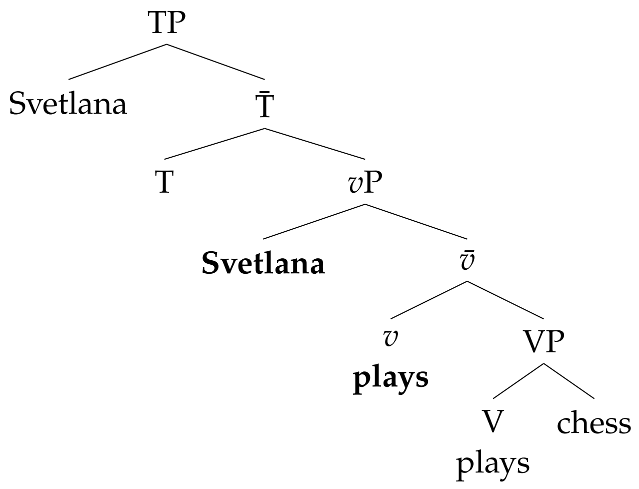
(16) a.



b.



(17)



To sum up, consider what this operation Stitch with the restriction in 13 does for us:

1. It gives us a simple way to describe dependencies between phases.
   - So we can set things up with the specification of the edges so that a C phase will combine with a *v* phase.
   - And we can do more interesting things to make sure that a *v*P phase built on *try* will only Stitch together with a phase corresponding to a control infinitive.
2. It gives us an escape hatch between phases.
   - In order for Stitch to work, the two phases getting together will **both** have to contain a copy of the edge that they share.
   - This means that something in the edge will be able to participate in local relationships in both phases, which are however unified and restricted in the escape hatch of the edge.

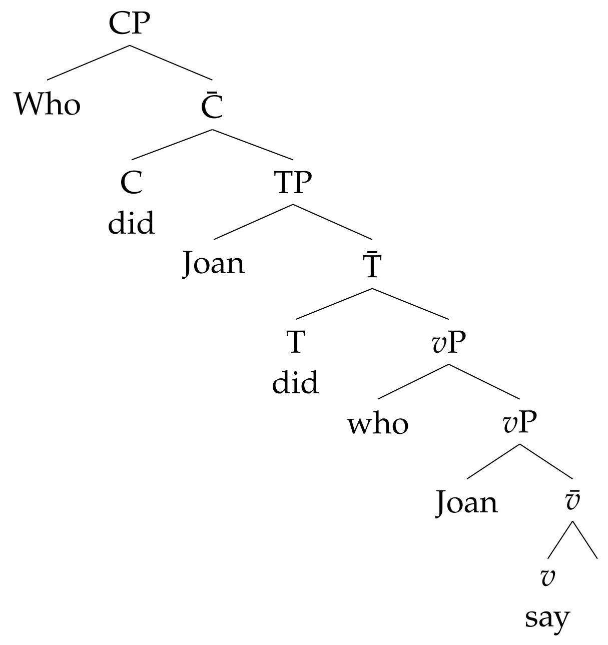
If nothing else, I think this represents a more direct implementation of the idea that derivation by phase is modular.

☞ Phases are independently created, modular objects, with encapsulated domains, which communicate through the well-defined interface of their shared edges.

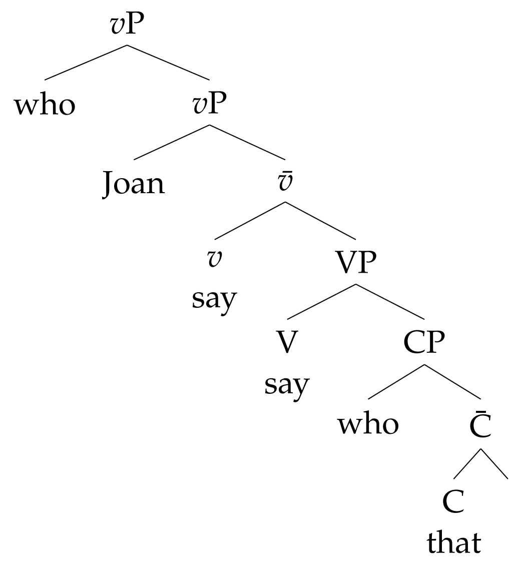
As a quick demonstration, here are the structures that we'd need for handling a basic example of successive-cyclic *wh*-movement:

(18) Who$_i$ did Joan t$_i$ say t$_i$ that Ray t$_i$ ignored t$_i$.

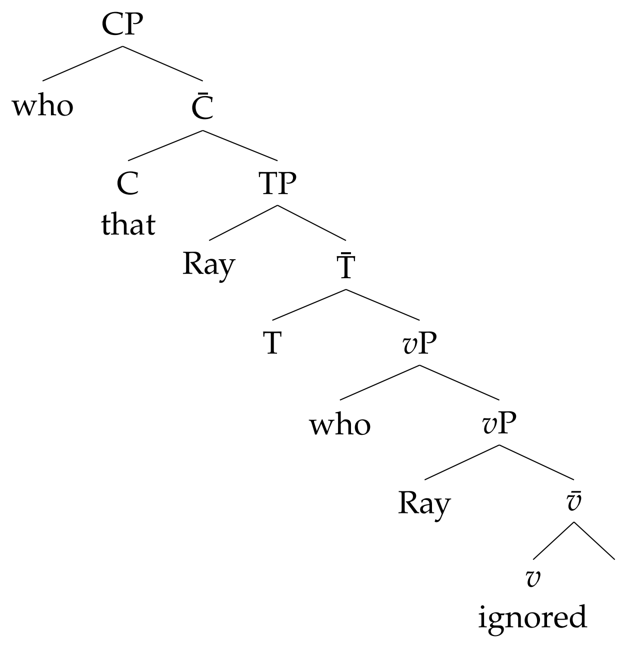
(19) a.



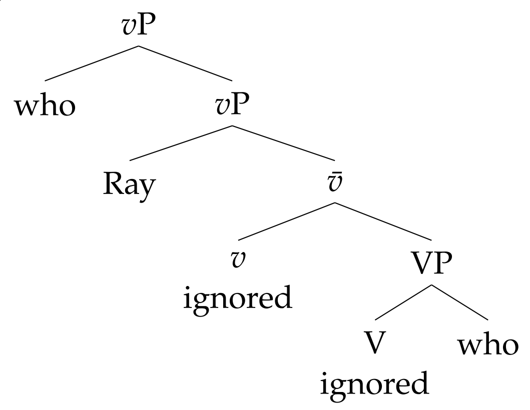
b.



c.



d.



4 Why StiPT is worth considering

4.1 It isn't worse than SpeDO

In order to justify spending further time on StiPT, we have to make sure that it isn't obviously worse than the competition, i.e. SpeDO. The first part of this is easy:

- Broadly speaking, StiPT can handle all the same data on locality and successive-cyclicity that SpeDO can, because it is designed to mimic its effects in this area.
- Both boil down to the idea of opaque domains with transparent edges, and in both we can play around with the definitions of those domains and edges to get the facts right.

The second part is a bit trickier:

☞ If both alternatives have essentially the same empirical coverage, Occam's Razor tells us that we should prefer the simpler of the two.

☞ At first blush, StiPT may appear more complicated, in that it requires all of the existing syntactic operations — Internal and External Merge and Agree — plus an additional one — Stitch.

☞ What's more, this new operation seems to play a suspiciously similar role to Merge, in that it takes two objects and combines them to create a new object.

If we think carefully, however, (most of) the force of this argument goes away, because SpeDO needs additional, ill-understood operations with properties similar to Stitch too.

- For one thing, the derivation of specifiers and adjuncts, however exactly it is done, requires the assumption of multiple parallel workspaces, the output of which is Merged to create larger structures.
- For another, we clearly need operations at the interfaces if not sooner which can reassemble the Spelled-out phase domains to re-create the complete structure of the sentence to be produced or interpreted.
- These operations don't get a whole lot of press in SpeDO, but that does not mean that they aren't necessary.

➪ So in a way, the proposal of Stitch here is really just making explicit what happens at that stage of the derivation and making use of it to handle additional effects.

➪ It is **not** stipulating some additional operations that SpeDO can do without, so we can conclude that StiPT is not obviously more complicated than SpeDO.

4.2 Performance issues

An interesting and potentially useful property of StiPT follows from the full modularity of the derivation of individual phases:

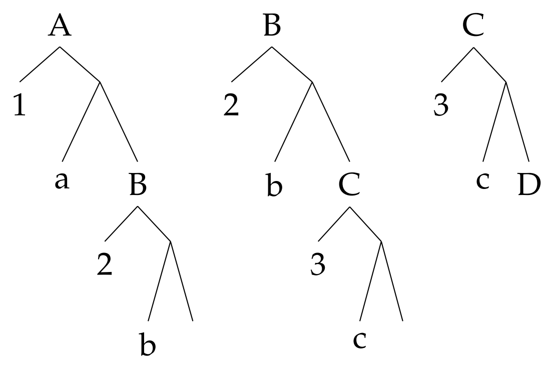
☞ In SpeDO, phases are built up sequentially in a cyclic fashion. But in StiPT, phases are built up completely independent of each other in their own workspaces.

☞ So, while in SpeDO higher phases depend on the output of lower ones, and hence lower ones must be constructed first, in StiPT the order in which phases are constructed is irrelevant.

☞ Furthermore, the order in which phases are Stitched together is irrelevant, because the way their edges interact essentially depends on static matching.

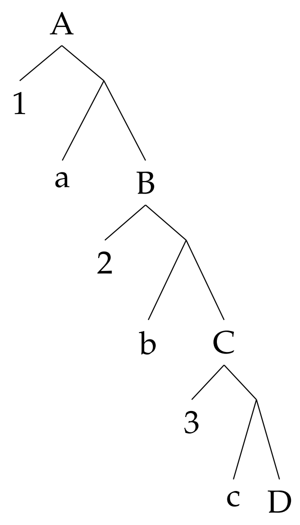
I should clarify that I do not mean **linear** or **hierarchical** order here, but **derivational** order. Take e.g. 3 phases with the structures in 20:

(20)



- Their shapes along with 13 will ensure that B is stitched into A, and C is stitched into B yielding 21:

(21)



- No other combination — e.g. B stitching into C or C stitching into A — will be possible, because the relevant edges don't Match. So the resulting hierarchical order, and the linear order derived from it, are relevant and pre-determined.
- However, it is irrelevant whether we first stitch C into B, and then B into A, or the other way around. Both derivational orderings will yield the same structure in 21.

This is important because it allows us an interesting approach to a well-known tension about directionality of derivations:

- ☞ On the one hand, we have evidence from things like basic compositionality and cyclicity effects that derivations proceed in a bottom up fashion.
- ☞ But we also have evidence from production and parsing that performance must operate to a large extent in a left-to-right fashion.
- ☞ Bottom up and left-to-right aren't exactly opposites, but they are incompatible with each other, so we have an apparent inconsistency.

Given what we've just said, however, StiPT has the potential to resolve this tension:

- Derivational steps within a single workspace, i.e. inside a phase, are bottom up.
- Stitching phases together, however, can proceed left-to-right.
- We can then construct our theories of competence and performance to take advantage of this distinction, modeling particular phenomena according to which kind of directionality effects they show.

Note that one potential interpretation of all of this is that Stitch isn't strictly speaking a derivational operation, but rather a constraint on well-formed representations.

- ☞ It is not accidental that we see an overlap here with arguments for "constraint-based" approaches (Jackendoff 2011, Müller 2013) on the basis of their ability to model performance.

One example of how this might be useful comes from the kinds of apparently changing constituency facts discussed by Phillips (2003).

- Phillips starts from the well-known fact that coordination seems to be able to target lots of linearly adjacent strings which we don't expect on other grounds to be constituents:

  (22) [Wallace designed] and [Gromit built] an enormous tin moon-rocket.

- He argues that these strings are indeed constituents, but only at an intermediate stage of the derivation. The can thus be targeted by certain operations (like coordination) but not by others (e.g. movement).

- In order to get this to work, he proposes that the derivation really works from left to right, rather than bottom up, so that constituency actually changes over time.

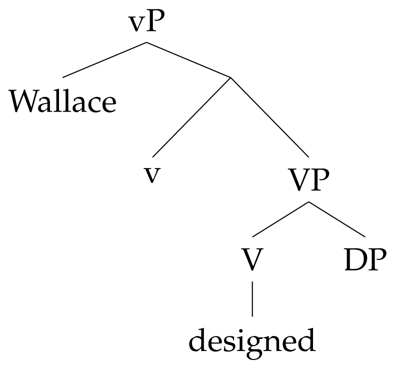
Now, Phillips's system gets these coordination facts to come out right and also makes sense of certain other data about processing, but at a price that most theoreticians are not willing to pay:

☞ Merge has to be redefined to apply to right branches rather than the root, which is formally more complex and also makes the relationship between the derived structure and the derivation more opaque.

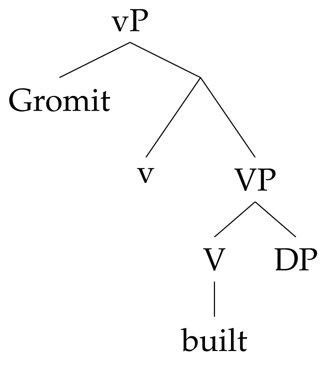
Under StiPT, we have an alternative, that I think lets us have our cake and eat it too. Here's the basic idea:

- First we build the vP phases involving Wallace designing and Gromit building:

(23) a.



b.



- Note crucially that they include the object DP, but assuming that DPs are phases, it's only a representation of the edge of the DP phase. So we're still going to need to Stitch in the object.

- If we conjoin the two vP phases at this point (however exactly that works), under a reasonable interpretation of the coordinate structure constraint, any subsequent operation that applies to one of them will have to apply to both.

- So a single DP phase will have to Stitch in and simultaneously be interpreted as the object of both verbs. The fact that it's only pronounced to the right of *built* makes sense if linearization statements involve precedence but not adjacency.

4.3 Triggering intermediate movement

StiPT also allows a better story for a common concern of phase-based work: the treatment of intermediate steps of successive-cyclic movement. Here's a quick description of the issue, based on 24, simplified to ignore *v*P as a phase:

(24) *Who* did Stringer think [$_{CP}$ <*who*> that Omar shot <*who*>]?

- We assume that the matrix C has a feature on it that drives a *wh*-element to its specifier, because this is clearly a syntactic requirement of English *wh*-interrogative clauses.
- It can't get there directly from its base position, which will be shipped off to Spell-out before matrix C enters the derivation.
- This problem can be overcome if it moves successive-cyclically via the lower Spec-CP which, being in the edge, will be visible to the next phase up.

But what actually makes *who* move to the Spec of the lower CP on its way up?

☞ The embedded CP does not have interrogative force, so in and of itself it has no requirement to have a *wh*-element in its specifier. In fact, it can't really have one end up there:

(25) * Stringer thinks [$_{CP}$ *who* (that) Omar shot <*who*>]

☞ It's clear that the higher Spec-CP wants to force movement, and that that movement will be blocked if *who* doesn't move to the intermediate Spec-CP on the way.

☞ But given the logic of phase theory, the higher C won't yet have entered the structure at the point when this movement would have to be triggered!

➪ So we have to do something clever to trigger intermediate movement in case it is needed, but make sure that we don't get it in cases where it is not needed.

There are various ideas out there about how to do it, but most boil down to some version of 'overgenerate and filter':[1]

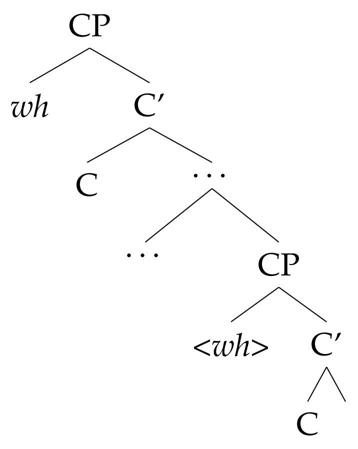
- We assume some mechanism that can trigger movement in contexts where it is not locally required, and make this somehow be optional.

[1] Heck and Müller (2000), Müller (2011) have an approach that doesn't really fit into the overgenerate and filter category, based on the idea of phase balance. If I understand correctly, though, phase balance requires access to a numeration for the entire sentence, not just information about the current phase, so that, in a sense, while you can't look ahead to the structure that's going to be built, you can look ahead at what elements are going to be involved.

- We make sure that there are constraints to rule out structures with the wrong combination of upper and lower portions — e.g. an interrogative main clause and a lower clause without intermediate movement, or a declarative main clause and a lower clause with intermediate movement.
- We then generate all conceivable combinations, and the ill-formed ones get filtered out at the interfaces for having unchecked uninterpretable features or the like.
- This does the basic job of course, but seems to suggest a waste of computational resources, as a large subset of derivations (the (vast) majority?) will lead to a crash.

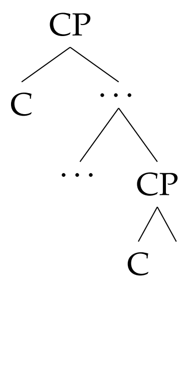
I think StiPT gives us a fairly satisfactory way to implement the idea behind overgenerate-and-filter without actually overgenerating:

☞ Higher phases with *wh*-movement landing sites (where the base position of the *wh*-element isn't local) will always have a copy of the *wh*-element in their inner edge, i.e. they'll look like this:

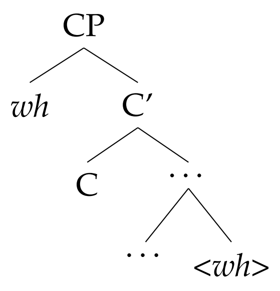


☞ So they will only be able to Stitch with lower phases that also have such an element in their outer edge. I.e. Stitch will simply not happen with a structure without intermediate movement.

☞ Higher phases without a *wh*-movement landing site will **not** have a copy of a *wh*-element in their inner edge, i.e. they'll look like this:



☞ So they will only be able to Stitch with lower phases that also have an outer edge without a *wh*-element in the specifier. I.e. Stitch will simply not happen with a structure like the one below with intermediate movement:



Here's how it has to work:

- We set things up so that Merge and Agree can create all of the phase types that will actually be needed in convergent sentences of the language.
- This (in part) amounts to saying that, for every phase generated with an internal edge, at least one phase will also be generated with a matching external edge.
- Then we don't have to actually overgenerate and filter. We generate a number of individually convergent phase pieces, and then choose the appropriate ones to Stitch together.
- Non-convergent structures (at least of the kind relevant here) can never be generated, so there's no need to filter them out.

4.4 The edge of the highest phase

Another minor but clear advantage comes in dealing with the edge of the highest phase in a sentence.

☞ In SpeDO, a given bit of structure is only Spelled-out (sometime) subsequent to the merging of the phase head above.

☞ This prompts the question of what happens with the material in the edge of the highest phase — how does it get Spelled-out, since there is no higher phase head?

☞ There are clearly sentences, like those with overt *wh*-movement, where the standardly assumed structure has material in the edge of the highest phase that clearly has to be Spell-out, since it is overtly pronounced and is interpreted at LF.

In order to get that material Spell-out, some additional assumptions are needed:

- We could posit an additional silent and uninterpreted phase head above all of the familiar material, which is never interpreted, but solely has the function of triggering Spell-out of all remaining material.

- Or we could posit a second way to trigger Spell-out, which applies at the end of the derivation of a sentence, flushing the workspace when there are no more phase heads to come.

☞ These are not at all unreasonable ideas, and there are others we could imagine as well. The point is that none of them come for free from the main architecture of SpeDO.

StiPT, on the other hand, does provide a simple solution to this problem, or rather it doesn't create the problem in the first place:

☞ Since phases are constructed independently and then stitched together, there is nothing particularly special about the highest phase, and the transparency of the edge doesn't depend on delayed Spell-out.

☞ So the highest phase will be constructed, stitched in and interpreted just like any of the phases below it.

4.5 CED effects

One last reason why StiPT is worth pursuing is the insight it may allow into tradtional CED effects (see Huang 1982, Müller 2011, and lots of other stuff in between).

(26) **Condition on Extraction Domains**
You can only move things out of complements, i.e. not out of specifiers or adjuncts.

Here are some quick examples demonstrating the basics:

(27) a. Rhonda likes [drinking water].
b. What does Rhonda like [drinking <*what*>]? **(✓From complement)**
(28) a. [Drinking water] makes Rhonda thirsty.
b. * What does [drinking <*what*>] make Rhonda thirsty. **(*From specifier)**
(29) a. Gary shouted [that Mary brought beer].
b. What did Gary shout [that Mary brought <*what*>]? **(✓From complement)**
(30) a. Gary shouted [because Mary brought beer].
b. * What did Gary shout [because Mary brought <*what*>]? **(*From adjunct)**

Now, once you start looking at different kinds of subjects, complements and adjuncts, and different languages, things get way more complicated than 26 would suggest.

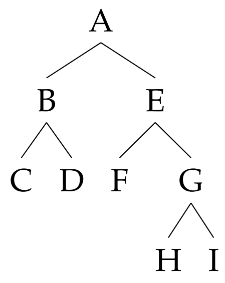
☞ I'm not even going to try to wave my hands at everything that's out there.

☞ I'll just suggest that 26 is an accurate descriptive generalization if we relativize it: it's generally easier to extract out of complements than out of specifiers or adjuncts.

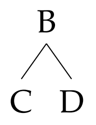
☞ StiPT gives us a straightforward way to think about why that would be so (which is clearly related to the approach of Uriagereka 1999).

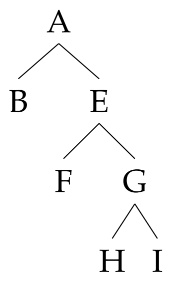
First of all, StiPT implies that all specifiers and adjuncts must be phases:

- A 'phase' is just what we call the amount of structure built up by merge within a single workspace. Any time that the output of two workspaces is to be combined, this is done by Stitch, not by Merge.
- Adjuncts and specifiers, at least branching ones, must be constructed separately before they are put into their specifier or adjunct positions — you can't combine two branching structures without using two workspaces.
- I.e. the only way to get this:



is by first building this: and this:

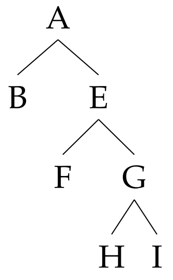




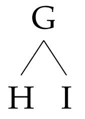
and then stitching B into A.

Now, complements **can** be phases too, but they don't have to be:

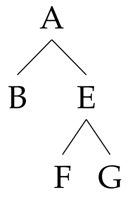
☞ So the structure



could be from building



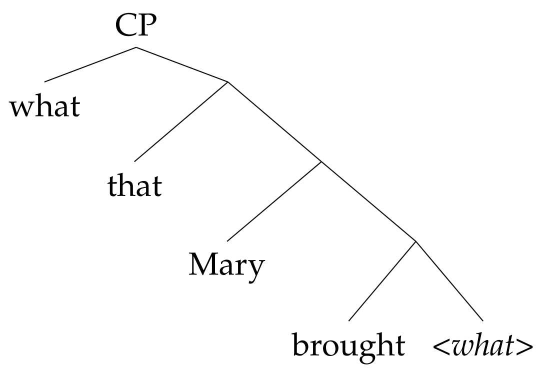
and



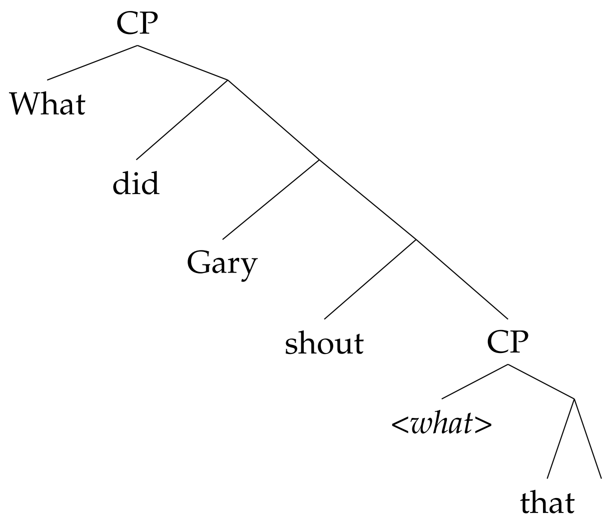
and stitching G into A (two phases) or from simply merging H and I, then F, then B (one phase).

But even when complements are phases, we can still extract out of them. Here's how it works (again, ignoring *v*Ps, which don't affect the logic of the argument):

- Build up the complement phase normally, moving the relevant element to the edge:

(31)


- Build up the matrix phase, starting with the edge of the complement, including a *wh*-element:

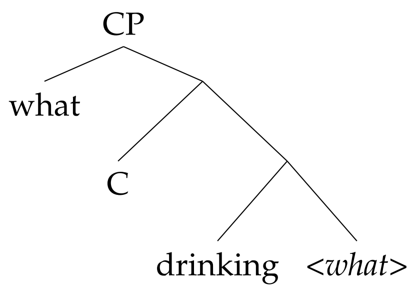
(32)


- We can then stitch 31 into 32 to get the sentence we're after.
- Note crucially that 32 could be built, including the edge of the lower phase containing the *wh*-element, purely by merge within a single line, i.e. in a single workspace.
- The fact that we can have an element in the lower edge that is associated by internal merge with a higher position within this phase is what makes extraction out of the complement possible.

Now consider what happens if we try to do the same thing with extraction from a specifier:

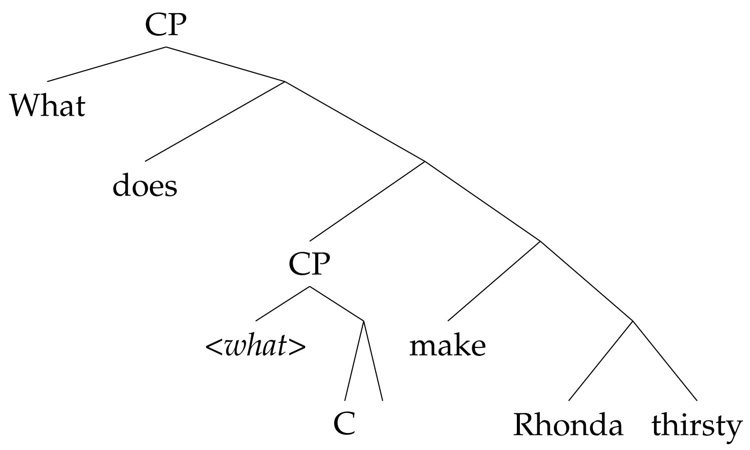
- We can build up the subject phase and move the *wh*-element to its edge without any trouble (I'm assuming for concreteness that this clause is a CP and ignoring the interior details beyond that, but its category actually makes no difference here):

(33)



- Then we would like to build the matrix phase as follows, including the specifier clause with movement out of its edge:

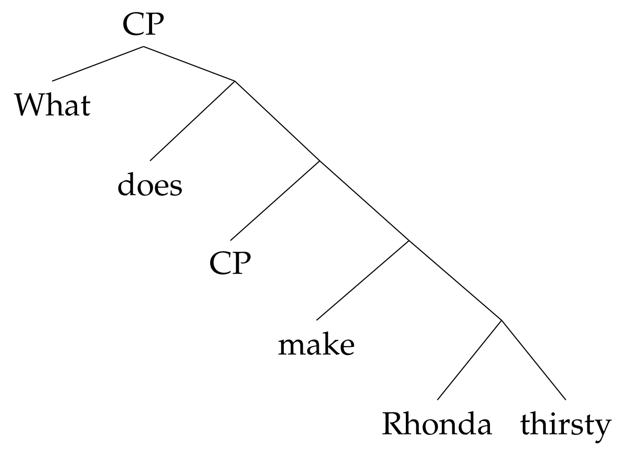
(34)



- But it's not actually possible to build the structure in 34 in a single workspace, i.e. within one phase, because the embedded CP, even just its edge, has a branching structure, so it would have to be constructed on its own.

- The best we can actually do is 35, where the embedded CP is reduced to a single non-branching node, and thus can include only very limited information – presumably things like its category, maybe even features triggering *wh*-movement and the like, but crucially not its internal structure.

(35)



- Without any access to its internal structure, we can't extract from it — note that there's no way to understand the *what* in matrix SpecCP here as a copy of anything inside the embedded CP.

Adjuncts will have the same problem, although with them it's perhaps even worse:

☞ Being unselected, it's plausible to think that they won't even be represented as minimal non-branching edges in their matrix clauses before Stitching takes place.

Again, there's far more to be said about CED effects, and about the generally worrisome status of adjuncts, but I'll leave it at that for now.

5 Issues, concerns and open questions

Again, this is very preliminary work at the present, so there are lots of loose ends and potential problems. Here I'll just mention a few that I've thought of:

- What is the definition of Match? We probably need to allow some degree of underspecification. Does it amount to something like unification?
- When do vocabulary insertion and linearization happen? We'd like to do as much as possible within the workspace, but it's not obvious how this will work. E.g. how do we pronounce a DP before it's been inserted in the large structure where its case will be determined?)
- Relatedly, when can we decide about pronunciation of copies?
- Do we ever need to merge elements with some or all of their features checked, and if so how do we make sure that it's ok?

- When building the inner edge of a phase, what do we start with? Do we just start with the lower phase head (this might lead to weird structures) or do we merge the lower phase head first with a dummy category to stand in for its opaque domain? I've been drawing the trees using a version of the latter idea to make Match more visually obvious.
- Merge and Stitch are fairly similar, in that they both take two objects and combine them to form a new larger object. Is there any way that we could unify them? Or can we further justify having both options by showing that they really do different things, both of which are necessary?
- Does StiPT offer us any insight into what kinds of complements count as phases? Can we e.g. use it to deal with variable phase size as argued for by Bobaljik and Wurmbrand (2013), Bošković (2014)?
- How does head movement work?

References

Bhatt, Rajesh. 2005. Long-distance agreement in Hindi-Urdu. *Natural Language and Linguistic Theory* 23:757–807.

Bobaljik, Jonathan, and Susi Wurmbrand. 2013. Suspension across domains. In *Distributed morphology today: Morphemes for morris halle*, ed. Ora Matushansky and Alex Marantz. Cambridge, Mass.: MIT Press.

Bošković, Željko. 2014. Now i'm a phase, now i'm not a phase: On the variability of phases with extraction and ellipsis. *Linguistic Inquiry* 45:27–89.

Chomsky, Noam. 2000. Minimalist inquiries: the framework. In *Step by step: Essays on minimalism in honor of Howard Lasnik*, ed. Roger Martin, David Michaels, and Juan Uriagereka. Cambridge, Mass.: MIT Press.

Chomsky, Noam. 2001. Derivation by phase. In *Ken Hale: A life in language*, ed. Michael Kenstowicz. Cambridge, Mass.: MIT Press.

Heck, Fabian, and Gereon Müller. 2000. Successive cyclicity, long-distance superiority, and local optimization. In *Proceedings of WCCFL 19*, 218–231.

Huang, C.T. James. 1982. Logical relations in Chinese and the theory of grammar. Doctoral Dissertation, MIT.

Jackendoff, Ray. 2011. What is the human language faculty? two views. *Language* 87:586–624.

Müller, Gereon. 2011. *Constraints on displacement. a phase-based approach*, volume 7 of *Language Faculty and Beyond*. Amsterdam: Benjamins.

Müller, Stefan. 2013. Unifying everything. *Language* 89:920–950.

Phillips, Colin. 2003. Linear order and constituency. *Linguistic Inquiry* 34:37–90.

Uriagereka, Juan. 1999. Multiple spell-out. In *Working minimalism*, ed. Samuel Epstein and Norbert Hornstein. Cambridge, Mass.: MIT Press.