Connectedness: Papers by and for Sarah VanWagenen
UCLA Working Papers in Linguistics 18: 161–174
Carson T. Schütze & Linnaea Stockall (eds.), 2014

Beyond the Apparent*

Cognitive Parallels Between Syntax and Phonology

Thomas Graf

Introduction

One of the central changes in 20th century linguistics was the reconceptualization of language as a cognitive ability rather than merely an abstract relational system of signs — in the terminology of Chomsky (1986), the move from E-language to I-language. This shift entails that linguistic theory is no longer measured exclusively by empirical coverage, succinctness of description, and conceptual appeal, but must also address the question of how language is integrated into our mental architecture. A lot of prominent work along these lines has focused on the supposed modularity of language, i.e. whether language is autonomous or shares resources with other mental capacities. This debate has been conducted at a very high level of abstraction, and as a result it sidesteps a more tangible issue: what kind of cognitive resources are actually needed to support the data structures and operations employed by I-language?

Formal language theory makes it possible to tie the complexity of linguistic patterns to specific claims about memory organization and thus provides an indirect way of measuring the cognitive demands of language. This is not a straight-forward exercise, though, and care must be taken to conceptualize these patterns in the right way. In this paper, I argue based on a battery of previous results (Graf 2010, 2013; Heinz 2010; Kobele et al. 2007; Michaelis 1998, 2001; Mönnich 2006, 2007, 2012) that even though syntax and phonology involve patterns of vastly different complexity, a linguistically informed perspective of the underlying dependencies reveals surprising similarities regarding their memory infrastructure.

*I'm a grinch, a grouch, a grump. Yet from every discussion, every hallway exchange I had with Sarah I walked away with my mood lightened. She is the only person I would call infectiously positive in a rational way. But I wish there were more people like her.

1 String Dependencies in Syntax and Phonology

1.1 String Languages, Automata, and Memory

The Chomsky hierarchy is the standard way in formal language theory to measure the complexity of string languages, where a string language is a possibly infinite set of strings and strings are finite sequences of labeled nodes. For example, *John likes Mary* can be viewed as a string with three nodes such that the first one is labeled *John*, the second *likes*, and the third *Mary*. English, in turn, would be the string set of all sentences deemed grammatical by an idealized speaker of English. The Chomsky hierarchy has been extended and revised over the years (McNaughton and Pappert 1971; Joshi 1985; Joshi et al. 1991; Seki et al. 1991; Rogers et al. 2010), a simplified sketch is given in Fig. 1. The higher a string language L is in the Chomsky hierarchy, the more powerful a grammar formalism needs to be in order to generate all strings in L but none that do not belong to L. This increase in power also translates into higher demands on memory size and organization.

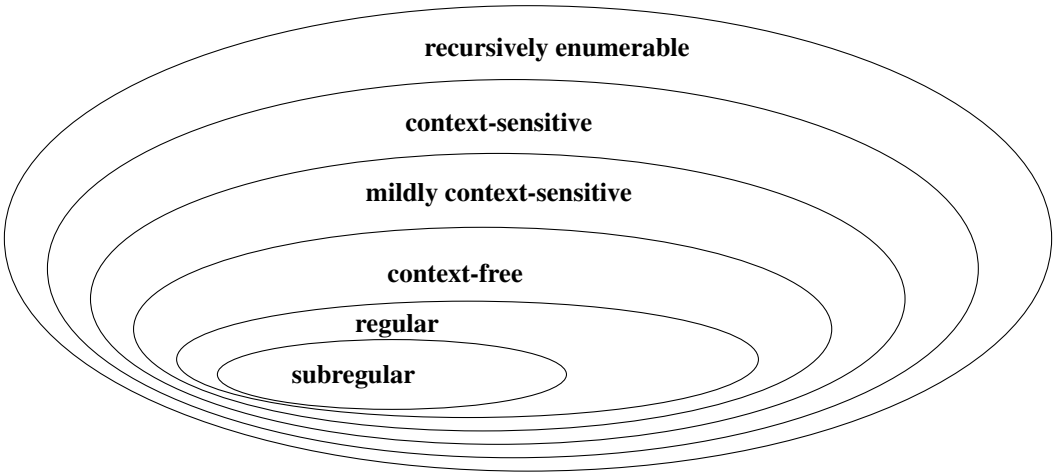


Figure 1: A simplified version of the modern Chomsky hierarchy; the subregular and mildly context-sensitive subhierarchies have been truncated

1.2 Regular Languages and Finite-State Automata

Consider regular languages, which are close to the bottom of the hierarchy. A language is regular iff it is accepted by a specific type of machine that is called a *finite-state automaton*. Such a finite-state automaton assigns every node in a string one of finitely many *states* depending solely on I) the state of the preceding node (if it exists), and II) the label of the current node. The automaton accepts a string iff the state assigned to the last node is a designated *final state*.

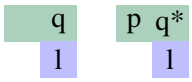
Example. In order to show that the language of all strings over a, b, and c with an even number of as and a c at the end is regular, it suffices to specify a finite-state automaton that accepts all the strings in this language, and only those. To this end, we have to define

- the set of states used by the automaton, and

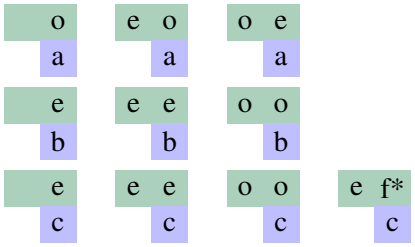
- the state assignment rules, and
- which states are final states.

All of this can be accomplished in one fell swoop by using a specific format for the state assignment rules. The instruction to assign a node with label l the state q if the previous node has state p is depicted as

If there is no preceding node, the field containing p is left empty instead, and if q is a final state, this is indicated by an asterisk.



With this format the required automaton is defined by ten rules:



The automaton has the three states o, e and f, out of which only f is a final state. Intuitively, the automaton uses o and e to keep track of whether is has encountered an odd or an even number of as so far. At the end of the word, it switches into the final state f if it has seen an even number of as and the last node is labeled c. Upon reflection it should be clear that this automaton accepts a string iff it belongs to the string language described at the beginning of this example.

For illustration, consider the strings *bcabbac* and *acabbac*. The former belongs to our language and is also accepted by the automaton. The latter is not accepted by the automaton, as no final state can be assigned to the last node. This is the correct result, as this string does not contain an even number of as and thus is not part of the desired string language. The state assignments for these strings are depicted in Fig. 2 and 3, respectively. □

For our purposes, the most interesting aspect of finite-state automata — and by extension regular languages — is that states can be regarded as highly abstracted metaphors for working memory configurations. A finite state automaton, then, represents a memory architecture that satisfies two properties that are very appealing from a psychological perspective.

- **Finite working memory**
  The limitation to finitely many states means that there is an upper bound on the working memory required to compute all regular string dependencies.
- **Simple control**
  Updating the working memory is a simple process that depends only on the current working memory configuration and the input being read.

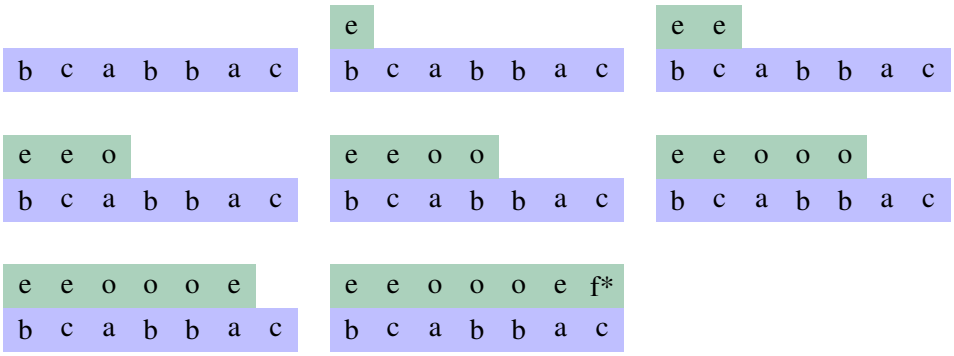


Figure 2: Step-wise state assignment for the well-formed string *bcabbac*

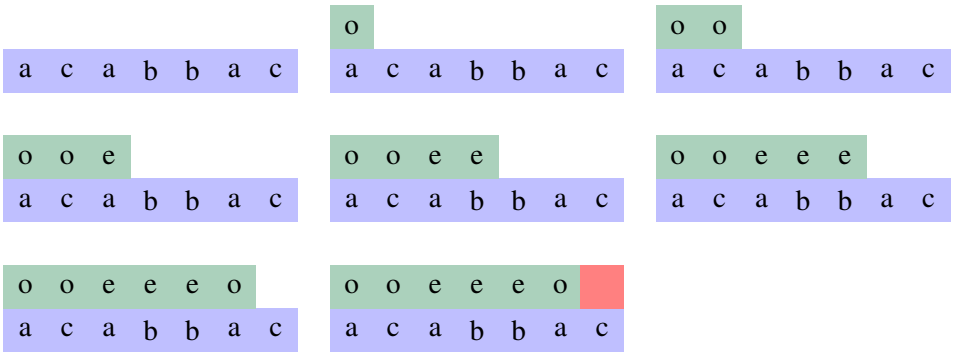


Figure 3: No valid state assignment for the illicit string *acabbac*

1.3 Context-Free Languages and Beyond

Beyond regular languages, the memory requirements of the corresponding automata quickly increase in size and complexity. The canonical automaton model for context-free string languages, for instance, is the *pushdown automaton*, which not only uses a finite set of states but also a dedicated memory stack to keep track of information. The stack uses its own set of symbols and is unbounded in size. It is also unidirectional in that it can only be manipulated at the very top: symbols can be read or removed only if they are at the top of the stack, and new symbols can only be added at the top.

A pushdown automaton assigns every node a state depending on I) the label of the node, II) the state of the preceding node (if it exists), and III) the highest symbol on the stack. In addition, which state is assigned also determines whether a symbol might be added to or removed from the top of the stack. Without going into further detail (see Hopcroft and Ullman 1979 or Sipser 2005) it is already apparent that pushdown automata are a lot more complex than finite-state automata.

- **Infinite working memory**
  There is no upper bound on the size of the pushdown stack, so an unbounded amount of memory may be consumed during the state assignment process.

- **Intertwined control**
  A pushdown automaton uses tightly integrated finite and infinite memory. Neither memory can be updated in isolation, both memory configurations need to be taken into account. Finite memory updates depend on the most recently added symbol in the infinite memory, and what is stored in the infinite memory depends on how the finite memory is updated.

For mildly context-sensitive string languages, the memory needs to be extended with auxiliary stacks that I) can always be written to, II) are readable only if the main stack is empty, and III) may themselves be stacks of stacks (Vijay-Shanker 1987; Becker 1994). Alternatively, a pushdown automaton can be supplemented with a second pushdown stack that is not subject to the read and write restrictions outlined above. Such an automaton is extremely powerful and can even handle the recursively enumerable string languages, which occupy the highest level in the Chomsky hierarchy. This means that beyond the context-free languages, it becomes difficult to make meaningful inferences about what kind of memory might be employed. The addition of a second pushdown stack is conceptually simpler than a multitude of narrowly restricted stacks yet offers enough memory to handle even the most demanding dependencies. Nonetheless there is a clear jump in memory complexity when moving from regular to context-free string languages: the latter require some kind of unbounded memory, the former do not.

1.4 The String Complexity of Syntax and Phonology

Syntax and phonology involve very different kinds of dependencies, and this is also reflected by their place in the Chomsky hierarchy.

Phonology is mostly about local dependencies, and where non-local dependencies arise, they usually take the form of "segment *x* may (not) be preceded/followed by segment *y* (if/

unless segment *z* intervenes)". As the reader may try to verify for himself, such dependencies aren't too difficult to enforce with finite-state automata. Quite generally there is a wide consensus in computational linguistics that phonology is at most regular, which is vindicated by Kaplan and Kay's (1994) demonstration that SPE as used by phonologists only generates regular string languages. Heinz (2010) considers a variety of phonological dependencies and concludes that the majority of them are subregular, which prompts the question whether a simpler memory model than finite-state automata would be sufficient for phonology. However, Graf (2010) shows that primary stress assignment in Cairene Arabic as described in Mitchell (1960) and Langendoen (1968) is regular, but not subregular. It seems, then, that phonology does indeed require the full power of finite-state automata, but not any more than that.

The demands of syntax, on the other hand, greatly outstrip what finite-state automata have to offer. The unlimited productivity of center embedding construction such as *the cheese that the mouse that the cat chased ate was poisoned* shows that syntax cannot be regular. Hence it is at least context-free, which entails that some form of infinite memory is used. Cross-serial dependencies in Dutch and Swiss German furthermore establish that syntax isn't context-free either (Huybregts 1984; Shieber 1985), increasing the memory requirements even more. The current consensus is that syntax is mildly context-sensitive, which puts it between the context-free and the context-sensitive languages, although there is some disagreement about the maximum complexity of syntactic patterns (Culy 1985; Radzinski 1991; Michaelis and Kracht 1997; Bhatt and Joshi 2004; Kobele 2006).

Whatever the precise location of syntax might be in the Chomsky hierarchy, it is far removed from phonology. Phonology only requires a very simple memory architecture whose size is finitely bounded. Syntax, on the other hand, needs to supplement and intertwine this architecture with at least two stacks of unbounded size. From a psychological perspective, this is rather surprising. If syntax has access to such an elaborate memory system, why is phonology not granted the same privilege? Or if one assumes that both use the same memory system, why does phonology fail to fully exploit its power?

Cognitive Conundrum Why do we find such a marked difference between phonology and syntax with respect to their minimum memory requirements?

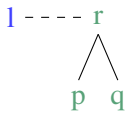
2 Syntactic Dependencies are Finite-State Too

The Cognitive Conundrum arises from the assumption that syntax and phonology should have access to the same memory architecture and thus display string dependencies of comparable complexity. Crucially, though, plenty of linguistic evidence suggests that syntactic dependencies are never string dependencies. Instead, syntactic dependencies seem to be computed over trees. One might speculate, then, that the chasm between syntax and phonology with respect to string complexity isn't due to different memory architectures but rather the use of different data structures: strings for phonology, trees for syntax.

2.1 *Finite-State Tree Automata*

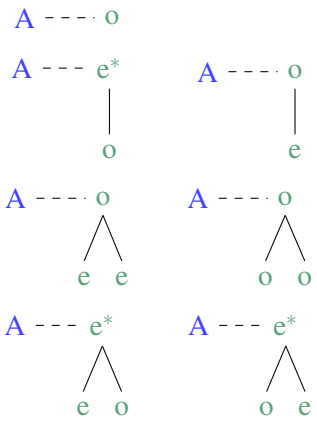
In order to make this idea more precise, we need a tree analogue of finite-state automata, and we have to show that these automata can handle all syntactic dependencies. A finite state automaton over trees is called a *finite-state tree automaton*. Like a finite state (string) automaton, it assigns every node in a tree one of finitely many states depending on I) the label of the current node, and II) the states of the daughter nodes. It accepts the tree iff the root of the tree is assigned a final state. The major difference between the string automaton and the tree automaton, then, is that the former only has to take one state into account (the state of the preceding node), whereas the latter has to look at the states of all daughter nodes, which in general will be more than one.

Example. Defining a finite-state tree automaton involves the same three components as the definition of a finite-state string automaton: the set of states, the state assignment rules, and which states are final states. This can be done using a minor generalization of the rule format we used for string automata.



The rule above states that a node labeled *l* is assigned the state *r* iff its left daughter has state *p* and its right daughter has state *q*.

Now consider the set of all trees whose nodes are labeled *A* and that contain an even number of nodes. The automaton below will accept all trees belonging to this tree language, and only those.



Once again states are used to keep track of whether an odd or even number of *A*s has been encountered, except that this time the underlying arithmetic is a little bit more complicated for nodes with more than one daughter. Since the sum of two even numbers or two odd numbers is always even, a node whose daughters have the same states is assigned the state *o* — an even number of *A*s distributed over the two daughters plus the one *A* of the current node means an odd number of *A*s has been seen so far. Conversely, a node is assigned state *e* if it has two daughters with distinct states. An example is given in Fig. 4. □

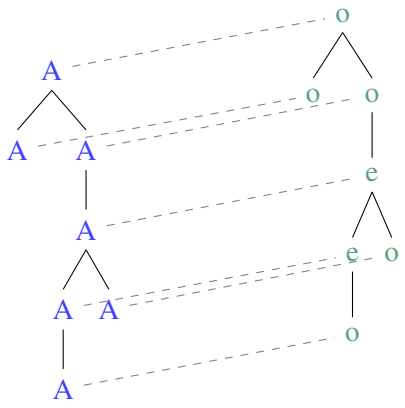
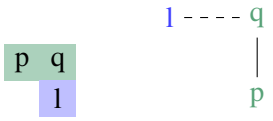


Figure 4: State assignment for a tree with an odd number of nodes

That finite-state tree automata are a natural generalization of finite-state string automata is witnessed by the fact that the latter can be reinterpreted as tree automata over strictly unary branching trees. Rotating a string counterclockwise by 90 degrees yields a unary branching tree, and every string automaton rule can be converted into a tree automaton rule by transposing it (i.e. mirroring it along the diagonal from the bottom left corner to the top right corner):



Now if syntax can indeed be handled by finite-state tree automata, then it is closely related to phonology on a cognitive level after all.

2.2 Syntactic Dependencies and their Encoding

It has been known for a long time that if a string language L is the yield of a tree language that is accepted by a finite-state tree automaton, then L is context-free (an easy corollary of a theorem in Thatcher 1967). That is to say, finite-state tree dependencies can capture context-free string dependencies, but not more than that. But syntax is not context-free, wherefore finite-state tree automata cannot handle all syntactic dependencies. So we have once again reached an impasse; computing syntactic dependencies over trees instead of strings does lower the memory requirements, but it is not enough to capture all syntactic dependencies with only a finite amount of working memory.

But is our *modus tollens* argument actually valid? Our primary interest lies in capturing syntactic dependencies, but who is to say that the dependencies we find in the strings are purely syntactic in nature? Maybe there is some non-syntactic factor that increases the complexity of the string dependencies beyond what is handled by syntax. For instance, linear order is only of peripheral importance according to most syntacticians, and all context-free languages are in fact regular if linear order is ignored (Parikh 1966). So maybe the task of

putting the nodes in a tree into the correct linear order should be factored out of the problem of capturing syntactic dependencies.

This is precisely the line pursued in the *two-step approach* (see Morawietz 2003, Mönnich 2006, 2007, 2012, and references therein). The idea is that syntax can be factored into two components: a language of trees encoding the underlying syntactic dependencies, and a mapping from these abstract trees to linearly ordered surface phrase structure trees. Crucially, if both components are required to be finite-state (in a specific technical sense), one obtains exactly the class of mildly context-sensitive string languages. So not only does syntax — construed as the theory of syntactic dependencies — appear to be finite-state, the mapping that mediates the relation between syntactic dependencies and surface realizations is too.

Abstract as this may sound, it can be given a very natural interpretation in two prominent grammar formalisms, Minimalist grammars (Stabler 1997, 2011) and Tree Adjoining Grammars (TAG; Joshi 1985; Joshi and Schabes 1997). In both formalisms, derivation trees serve the role of encoding syntactic dependencies independent of matters of linear order. The mapping, in turn, relates every derivation tree to the structure that is built by the grammar if one actually executes the steps in the derivation. Figure 5 shows a Minimalist derivation and the corresponding output tree with all elements in the correct surface order. Since the mapping from derivations to surface trees is deterministic in both formalisms, the latter are strictly speaking redundant for specifying the set of well-formed structures. And in both formalisms the derivation tree languages are finite-state (Kallmeyer 2009; Kobele et al. 2007; Michaelis 1998, 2001). This shows that syntax can indeed be thought of as a finite-state theory over trees.

Cognitive Parallelism Hypothesis Phonological dependencies are finite-state dependencies over strings. Syntactic dependencies are finite-state dependencies over trees. The string patterns instantiated by syntax are the image of syntactic structures under a finite-state mapping.

3 Some Open Issues

We have arrived at an appealing picture where phonology and syntax use the same kind of memory architecture but operate over different data structures, which in turn gives rise to the apparent complexity differences we see on the level of strings. But as is often the case, things might not be quite as elegant in reality.

Under the strictest possible reading, the Cognitive Parallelism Hypothesis is actually violated by syntax because the mapping from abstract structures to surface structures is not finite-state in the same sense as our tree automata are finite-state. In the case of Minimalist grammars, for example, the mapping is computed by a device called a deterministic multi bottom-up tree transducer (Kobele et al. 2007). This machine uses only a finite number of states, but each state can be used to store subtrees of the derivation. So these states can contain a fixed number of objects of unbounded size, and consequently the analogy between states and finite working memory breaks down to a certain extent; the number of objects that can be stored in memory is still bounded, but their size is not.

In addition, the memory requirements of the mapping can increase depending on what kinds of movement are allowed in the grammar. For Minimalist grammars without copy

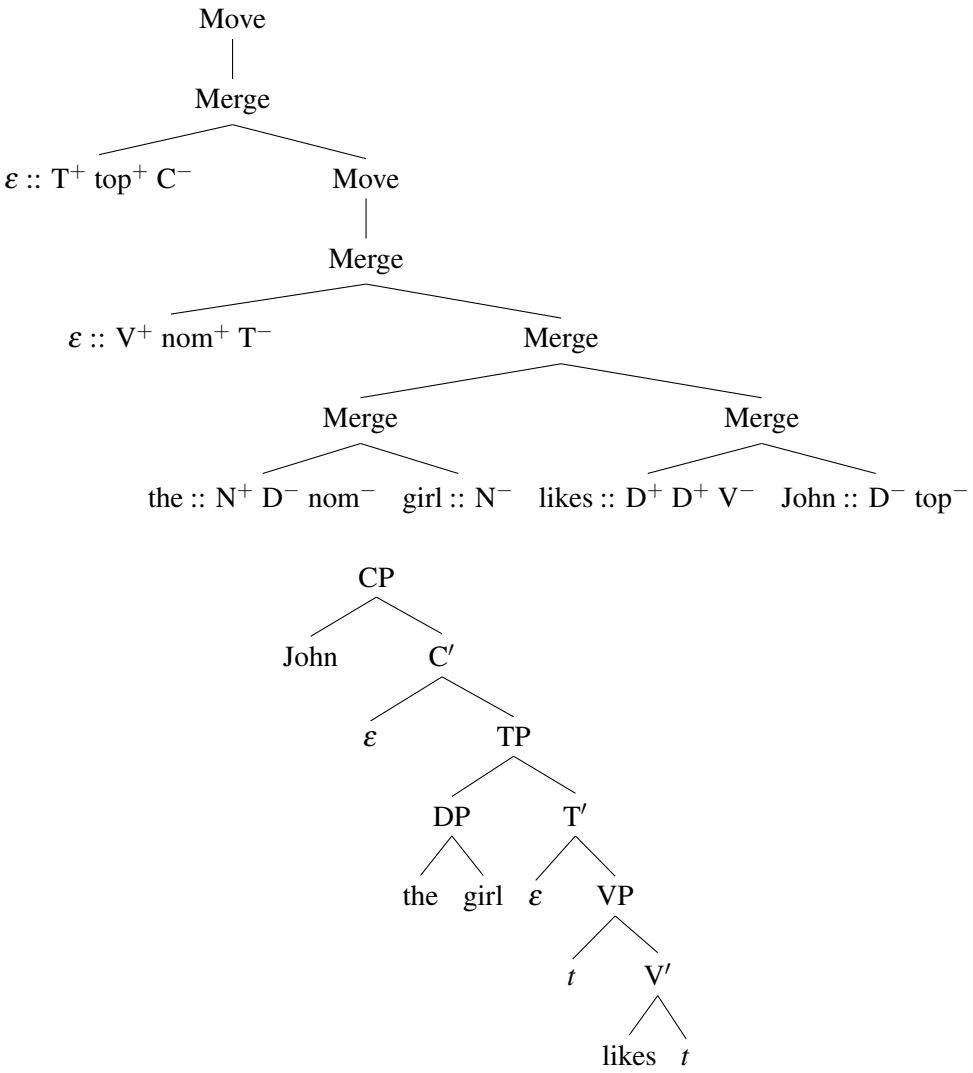


Figure 5: Minimalist derivation (top) and corresponding surface structure (bottom)

movement, the upper bound is given by macro tree transductions of linear size increase (Engelfriet and Maneth 2003; Graf 2012b). In this case, states may also be nested, so a state can contain both subtrees and states, which in turn may contain subtrees and states, and so on. So even though all these machines are finite-state in a literal sense, the kind of states they employ can store a lot more information than those of string and tree automata.

Phonology and syntax do not seem to be perfectly matched even if one ignores the memory requirements of the syntactic mapping and focuses purely on the abstract, underlying trees instead. As shown in Graf (2012a, 2013), Minimalist derivation trees can be characterized in terms of first-order definable constraints. The power of finite-state tree automata eclipses that of first-order logic, so Minimalist derivation tree languages can actually be computed with a slightly more restricted memory architecture. Phonology, on the other hand, displays at least one dependency that is not first-order definable, namely primary stress assignment in Cairene Arabic as described in Mitchell (1960) and Langendoen (1968).

Curiously, though, Graf's (2010) proof that Cairene Arabic phonology is regular hinges on the alleged absence of secondary stress, which has been doubted in the literature. If it turned out that secondary stress is overtly realized in Cairene Arabic, its primary stress rule would be a lot less complex. As a result, phonology would be at most star-free, which is exactly the class of first-order definable string languages. That is to say, if Cairene Arabic could be shown to have overt secondary word stress, then syntax and phonology would once again satisfy the Cognitive Parallelism Hypothesis because they would both be restricted to first-order definable dependencies. Until this has been established conclusively, though, phonological dependencies appear to be more demanding than syntactic ones in certain respects.

Phonology poses a second puzzle in the form of reduplication. Reduplication involves copying a substring of potentially unbounded length and is a mildly context-sensitive phenomenon. If reduplication is part of phonology, then why are there no phonological dependencies that fall between the regular and the mildly context-sensitive levels in the Chomsky hierarchy? Perhaps phonology should also be factored into two components like syntax, such that one keeps track of phonological dependencies while the other is involved in the assembly of the output structures. At this point it is unclear how exactly such a system could be set up in a way that allows for regular patterns and reduplication, but nothing else. Maybe the answer lies once again in adopting a more elaborate data structure than just strings (cf. Heinz et al. 2011).

Conclusion

A mentalist view of language naturally gives rise to the question what kind of cognitive resources are involved in the computation of linguistic dependencies. If one thinks of phonology and syntax as systems of string dependencies, then the two diverge significantly regarding the memory infrastructure they require. However, a linguistically informed perspective that recognizes that syntactic dependencies operate over trees rather than strings paints a picture of phonology and syntax as theories that do not differ with respect to memory architecture but rather the type of data structure they operate over.

References

Becker, Tilman. 1994. A new automaton model for TAGs: 2-SA. *Computational Intelligence* 10:422–430.

Bhatt, Rajesh, and Aravind Joshi. 2004. Semilinearity is a syntactic invariant. A reply to Kracht and Michaelis 1997. *Linguistic Inquiry* 35:683–692.

Chomsky, Noam. 1986. *Knowledge of language: Its nature, origin, and use*. New York: Praeger.

Culy, Christopher. 1985. The complexity of the vocabulary of Bambara. *Linguistics and Philosophy* 8:345–351.

Engelfriet, Joost, and Sebastian Maneth. 2003. Macro tree translations of linear size increase are MSO definable. *SIAM Journal on Computing* 32:950–1006.

Graf, Thomas. 2010. Logics of phonological reasoning. Master's thesis, UCLA.

Graf, Thomas. 2012a. Locality and the complexity of minimalist derivation tree languages. In *Formal Grammar 2010/2011*, ed. Philippe de Groot and Mark-Jan Nederhof, volume 7395 of *Lecture Notes in Computer Science*, 208–227. Heidelberg: Springer.

Graf, Thomas. 2012b. Movement-generalized minimalist grammars. In *LACL 2012*, ed. Denis Béchet and Alexander J. Dikovsky, volume 7351 of *Lecture Notes in Computer Science*, 58–73.

Graf, Thomas. 2013. Local and transderivational constraints in syntax and semantics. Doctoral Dissertation, UCLA.

Heinz, Jeffrey. 2010. Learning long-distance phonotactics. *Linguistic Inquiry* 41:623–661.

Heinz, Jeffrey, Chetan Rawal, and Herbert G. Tanner. 2011. Tier-based strictly local constraints in phonology. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, 58–64.

Hopcroft, John E., and Jeffrey D. Ullman. 1979. *Introduction to automata theory, languages, and computation*. Reading, Mass.: Addison Wesley.

Huybregts, M. A. C. 1984. The weak adequacy of context-free phrase structure grammar. In *Van periferie naar kern*, ed. Ger J. de Haan, Mieke Trommelen, and Wim Zonneveld, 81–99. Dordrecht: Foris.

Joshi, Aravind. 1985. Tree-adjoining grammars: How much context sensitivity is required to provide reasonable structural descriptions? In *Natural language parsing*, ed. David Dowty, Lauri Karttunen, and Arnold Zwicky, 206–250. Cambridge: Cambridge University Press.

Joshi, Aravind, and Y. Schabes. 1997. Tree-adjoining grammars. In *Handbook of formal languages*, ed. Grzegorz Rosenberg and Arto Salomaa, 69–123. Berlin: Springer.

Joshi, Aravind, K. Vijay-Shanker, and David Weir. 1991. The convergence of mildly context-sensitive grammar formalisms. In *Foundational issues in natural language processing*, ed. P. Sells, S. M. Shieber, and T. Wasow, 31–81. Cambridge, MA, USA: MIT Press.

Kallmeyer, Laura. 2009. A declarative characterization of different types of multicomponent tree adjoining grammars. *Research on Language and Computation* 7:55–99.

Kaplan, Ronald M., and Martin Kay. 1994. Regular models of phonological rule systems. *Computational Linguistics* 20:331–378.

Kobele, Gregory M. 2006. Generating copies: An investigation into structural identity in language and grammar. Doctoral Dissertation, UCLA.

Kobele, Gregory M., Christian Retoré, and Sylvain Salvati. 2007. An automata-theoretic approach to minimalism. In *Model Theoretic Syntax at 10*, ed. James Rogers and Stephan Kepser, 71–80.

Langendoen, D. Terence. 1968. *The London school of linguistics: A study of the theories of B. Malinowski and J. R. Firth*. Cambridge, Mass.: MIT Press.

McNaughton, Robert, and Seymour Pappert. 1971. *Counter-free automata*. Cambridge, Mass.: MIT Press.

Michaelis, Jens. 1998. Derivational minimalism is mildly context-sensitive. *Lecture Notes in Artificial Intelligence* 2014:179–198.

Michaelis, Jens. 2001. Transforming linear context-free rewriting systems into minimalist grammars. *Lecture Notes in Artificial Intelligence* 2099:228–244.

Michaelis, Jens, and Marcus Kracht. 1997. Semilinearity as a syntactic invariant. In *Logical Aspects of Computational Linguistics*, ed. Christian Retoré, volume 1328 of *Lecture Notes in Artifical Intelligence*, 329–345. Springer.

Mitchell, Terence F. 1960. Prominence and syllabification in Arabic. *Bulletin of the School of Oriental and African Studies* 23:369–389.

Mönnich, Uwe. 2006. Grammar morphisms. Ms. University of Tübingen.

Mönnich, Uwe. 2007. Minimalist syntax, multiple regular tree grammars and direction preserving tree transductions. In *Model Theoretic Syntax at 10*, ed. James Rogers and Stephan Kepser, 83–87.

Mönnich, Uwe. 2012. A logical characterization of extended TAGs. In *Proceedings of the 11th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+11)*, 37–45. Paris, France.

Morawietz, Frank. 2003. *Two-step approaches to natural language formalisms*. Berlin: Walter de Gruyter.

Parikh, Rohit. 1966. On context-free languages. *Journal of the Association for Computing Machinery* 13:570–581.

Radzinski, Daniel. 1991. Chinese number names, tree adjoining languages, and mild context sensitivity. *Computational Linguistics* 17:277–300.

Rogers, James, Jeffrey Heinz, Gil Bailey, Matt Edlefsen, Molly Vischer, David Wellcome, and Sean Wibel. 2010. On languages piecewise testable in the strict sense. In *The mathematics of language*, ed. Christan Ebert, Gerhard Jäger, and Jens Michaelis, volume 6149 of *Lecture Notes in Artificial Intelligence*, 255–265. Heidelberg: Springer.

Seki, Hiroyuki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. 1991. On multiple context-free grammars. *Theoretical Computer Science* 88:191–229.

Shieber, Stuart M. 1985. Evidence against the context-freeness of natural language. *Linguistics and Philosophy* 8:333–345.

Sipser, Michael. 2005. *Introduction to the theory of computation*. Course Technology, second edition.

Stabler, Edward P. 1997. Derivational minimalism. In *Logical aspects of computational linguistics*, ed. Christian Retoré, volume 1328 of *Lecture Notes in Computer Science*, 68–95. Berlin: Springer.

Stabler, Edward P. 2011. Computational perspectives on minimalism. In *Oxford handbook of linguistic minimalism*, ed. Cedric Boeckx, 617–643. Oxford: Oxford University Press.

Thatcher, James W. 1967. Characterizing derivation trees for context-free grammars through a generalization of finite automata theory. *Journal of Computer and System Sciences* 1:317–322.

Vijay-Shanker, K. 1987. A study of tree adjoining grammars. Doctoral Dissertation, University of Pennsylvania.

Affiliation

Thomas Graf
Department of Linguistics
Stony Brook University
mail@thomasgraf.net