# Some ways to think about linguistic structure (for philosophers)

Fintan Mallory
fintan.mallory@kcl.ac.uk

## Introduction

This short note introduces a few different ways of thinking about structure for philosophers. They are grammar-theoretic, automata-theoretic, model-theoretic and algebraic. While these techniques were developed in the context of formal language theory, there is no principled reason why the conceptions of structure they provide should be confined to linguistic entities. This is an important point. While the examples of use will all concern formal languages, i.e. sets of strings, the formal tools can be applied to characterise other domains. When we understand the tools in this broader sense, the differences between these frameworks can be seen to capture different philosophical views on the relationship between descriptions and their objects (or minds and reality, if you wish).

*I'm a philosopher, why should I care about this?*

One of the great tools of philosophy is the ability to think about the same problem from different perspectives. The reason mathematical equivalence theorems are neat is that they demonstrate the equivalence of different ways of thinking about the same thing. These theorems demonstrate that certain, very different ways of thinking about structure are actually equivalent.

*Fine. But can you give me some more grandiose rhetoric? I'm not interested enough yet and I'll need to be interested to put in the work.*

Very well. These different approaches are associated respectively with the concepts of rules, recognition, representations and emergence. The grammar-theoretic approaches characterise structure by appealing to the rules one would have to follow to construct that structure. The automata-theoretic approach characterises structure by appealing to the properties of the devices required to recognise that structure. Model-theoretic approaches characterise structure by appeal to descriptions of that structure. Algebraic approaches characterise structure, broadly, as something which emerges from a set of objects. Each of these approaches have associated metaphysical commitments. The grammar-theoretic view treats structure as constructed by the application of rules - the structure does not exist independently of the rules. The automata-theoretic view can be interpreted as providing a response-dependent conception of structure, i.e. to have the structure is to be recognised as having the structure. Model-theoretic approaches treat structure as *sui generis* and existing independent of our means of describing it. Algebraic approaches treat structure as extensional and holistic. When you take a holistic view of linguistic structure it does not make sense to ask what the structure of a sentence is considered independently of the language as a whole. These are profound philosophical differences but the equivalence theorems show us how they connect.

*That was a bit much. Does this have any empirical significance?*

Yes. Even outside of linguistics and computer science, ideas from formal language theory are increasingly being applied in cognitive science and ethology to characterise the kinds of structures and patterns which different species can identify. For example, the syntactic structures of human languages are generally agreed to be *mildly-context-sensitive* whereas the Bengalese Finch can only manage regular structures (Berwick et. al., 2011). Algebraic approaches to structure are increasily being used to develop machine learning techniques in the field of language acquisition (Clark, 2013).

# 1 Formalisms, a shoppers guide

## 1.1 Grammar-theoretic

Phrase structure rules consist of symbols separated by an arrow. The symbols on the right tell you what you can replace the symbols on the left with. The symbols are either terminals or non-terminals. Terminals terminate an application of a rule whereas non-terminals are like variables and allow you to apply a rule again. A derivation is complete when you have no more non-terminals left. Non-terminals are traditionally written as capital letters while terminals are written in lower-case.

For example, the rule,

$$\mathrm{A} \rightarrow \mathrm{Ab}$$

tells you that you can replace an A with an A followed by an b. An application of the rule produces the string Ab. If we add a different rule which says A → a, then these rules together could generate the set of strings ab, abb, abbb... (we call this the language ab*). These strings have very simple structures but there are infinitely many of them.

The more complex the rules, the more complex the structures they can generate. The rule A → Ab has a single non-terminal on the left and a single non-terminal on the right. This means that it is *regular*. If we allow there to be multiple non-terminals on the right

$$\mathrm{A} \rightarrow \mathrm{aAb}$$

then our rule is *context-free*. It is context free because the context of of the non-terminal on the left doesn't matter. If we add a rule A → ab then these rules could generate the set of strings ab, aabb, aaabbb $\mathrm{a}^{\mathrm{n}}\mathrm{b}^{\mathrm{n}}$. This language cannot be described by a regular grammar. If, when applying a rule, we take the context on the left into account, as you would with a rule like

$$\mathrm{aBc} \rightarrow \mathrm{abc}$$

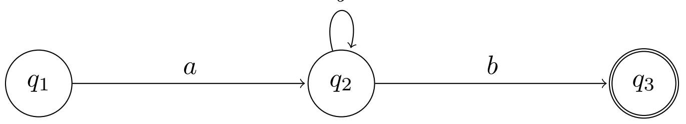
then the rule is called *context-sensitive*. This can generate languages which are beyond the scope of context-free grammars, e.g. $\mathrm{a}^{\mathrm{i}}\mathrm{b}^{\mathrm{i}}\mathrm{c}^{\mathrm{i}}$.

**Info:** These grammars were initially developed by the logician Emil Post. In his paper *Recursive Unsolvability of a Problem of Thue*, Post considered whether it was possible to mechanically determine for any arbitrary pair of strings whether it is possible to get from one to the other by substituting substrings in a step-by-step manner. In showing the problem to be recursively unsolvable, he contributed significantly to the modern understanding of the Turing machine.

## 1.2 Automata-theoretic

Automata-theoretic approaches characterise a structure in terms of an automaton capable of recognising it. More accurately, the automaton must be capable of recognising a string possessing that structure. An automaton recognises a string if it halts after receiving the string as an input on a piece of paper.

An automaton is a set of states which are characterised by input-output relations (it might help to think of these as dispositional properties). The complexity of the automaton is determined by the complexity of these relations. For example, a finite state automaton, can change state in response to the input on a tape. If the automaton is in state $\mathrm{q}_1$ and it encounters an 'a' on the tape, it changes to state $\mathrm{q}_2$. The only input is what is on the tape, the only output or action the automaton can perform is to change its internal state. The state the machine is in is determined solely by what state it was in previously and what is on the tape. We can represent these automata with diagrams. For example, an automaton that recognises only the language ab* above would have 3 states and look like this.


$$\begin{tikzcd} q_1 \arrow[r, "a"] & q_2 \arrow[loop above, "b"] \arrow[r, "b"] & q_3 \end{tikzcd}$$

To recognise more complex languages (i.e. structures), we need more complex input-output relations. For example, to recognise context-free languages, the automaton needs to be able to track more than just what state it was in previously. This means allowing it to record things to memory. A *Push-down stack automaton* can not only change state in response to inputs but also push a symbol onto a memory stack. With this in place, we can build what is at the top of the stack into the input relation. This means that what the machine does will depend upon what is on the tape *and* what is at the top of the stack. This kind of memory is needed to recognise languages like $a^n b^n$ (this should be intuitive as the machine needs to be able to count the number of 'a's and 'b's that it has seen.

## 1.3 Model-theoretic

model-theoretic approaches simply state what structures there are in a formal metalanguage. For example, if we want to characterise the language ab*, we simply have to state in our metalanguage that each string starts with an 'a' and ends with any number of 'b's. If we want to characterise $a^n b^n$, we simply have to say that every string starts with an a, ends with a 'b', and that there are the same number of 'a's as 'b's. To state these properties, we can use predicates to tell us what letter something is and a relation R corresponding to 'preceeds'.

$$\exists x(Ax \wedge \forall y\ Rxy \rightarrow By)$$

Things become interesting when we play with the expressive power of our metalanguages. First order languages can describe the kind of structures produced by regular grammars. However, if you want to talk about languages like $a^n b^n$, you'll need to be able to quantify over sets and so you'll need to use at least (weak) Monadic Second Order Logic. Within a formal metalanguage you can state *theories* about certain tree sets and in this case, the kinds of ordering relations your theory has access to will determine the kinds of linguistic structures you can describe. Confining our focus to Second-order theories, with one ordering relation, you can speak about regular languages. If you have two ordering relations (e.g. precedence and dominance), you can describe the hierarchical structures of context-free languages. Interestingly, if we want to describe some of the structures that occur in natural languages, we need a third ordering relation. This suggests that while spoken sentences are one dimensional, all a speaker can do is put one word in front of another, syntactic structures in the brain are at least *three dimensional*.

## 1.4 Algebraic

Algebraic approaches consider the relations between parts of our languages and the algebraic structures they give rise to. For our purposes, an algebra is a set and an operation $\langle M, \circ \rangle$. Recall that we can think of a language (or whatever structured entities we are interested in) as a set of strings.

Take the word 'cat' in our language. This letter only occurs in certain contexts, e.g. the **cat** sleeps, a **cat** purrs. Consider the set of contexts in which 'cat' appears {the,sleeps}, {a,purrs}, call this set of contexts C(cat). Expressions are congruent if they share their contexts. The syntactic monoid of a language L is the set of congruence classes of L induced by the congruence relation: $u \equiv_L v$ iff $C_L(u) = C_L(v)$ where $C_L(w) = \{(l, r) \in \Sigma^* \times \Sigma^* \mid lwr \in L\}$ For a regular language, the syntactic monoid $\langle M, \circ, 1 \rangle$ is finite (this is the *Myhill-Nerode theorem* which we'll discuss below). More complex languages give rise to more complex algebraic structures. The canonical algebraic structure corresponding to a context-free language is a complete idempotent semiring.

## 1.5 Correspondences between language classes

| Complexity Classes | | | |
|---|---|---|---|
| Language | Grammar | Model | Algebra |
| MCFG | MCFL | | Paired complete idempotent semiring |
| TAG | Tree-Adjoining Grammar | wS$\omega$S T3 | |
| CFG | Context-Free | wS$\omega$S T2 | Complete Idempotent Semiring |
| Regular | Regular | wS1S | Finite Monoid |
| Star-Free | | First-Order (transitively closed successor) | Finite aperiodic monoid |
| Piecewise-Testable | Piecewise Testable | | Finite *J*-Trival |
| Locally-Testable | | First-Order (successor) | |

## 1.6 What's the difference?

There are other ways we can view structures than those listed above and further equivalences exist. For example, if you understand a lack of structure to be randomness (where everything is as likely as everything else), then you might understand structure to be *deviation from equiprobability*. This is how Zellig Harris understood linguistic structure in his later work.

The kinds of *phrase-structure grammars* shown above fell out of use in linguistics long ago and research has tended to focus on categorial grammar or minimalist grammars. While the terminals in phrase-structure grammars are simple atomic symbols, the basic units of contemporary grammars are structurally much more complex and might possess a range of properties called 'features' which are responsible for the kinds of combinatorial relation the expression can end up in. This shift in the burden of complexity means that grammarians no longer state rules of grammar but describe the complex atoms which can be joined to each other.

Perhaps the most interesting difference between these approaches is where they locate *recursion*. The 'recursiveness' of a phrase-structure grammar is captured in the rules which generate sets of structure. When a rule is stated with the same non-terminal on both the left and right of the arrow, e.g. $X \rightarrow yXy$, then that rule can 'apply to itself' to produce unboundedly many strings. In the model-theoretic approach, the recursive element is shifted into the interpretation function for our formal metalanguage. If our symbols of monadic second-order logic are to mean anything, they must be interpreted, i.e. mapped to elements in a domain. The account of grammar is still dependent upon recursion but what the model-theoretic approach does is shift the recursive element of the grammar to the metalanguage of the metalanguage (the metametalanguage in which the interpretation function is given). In the automata-theoretic approach, the recursion is implemented within the system (at least, this is what the Church-Turing thesis would suggest).

At the beginning of this paper we hinted that these different ways of thinking about structure might correspond to different ways we can consider the relation between the mind and reality. We have done *absolutely nothing* here to justify this claim. However, if it is true, then there is presumably something profound in the fact that each of these methods place the locus of recursion in different places. Unpacking what this means may be a valuable philosophical task.

# 2 Appendix

None of these proofs are particularly complete but they should indicate the methods used and relations stated between different technical devices. They all involve Kleene's Theorem in some form or another.

We'll start with a couple of definitions:

**Definition 1.** A non-deterministic finite state automaton is a tuple A = {Q, $\Sigma$, $\delta$, $q_0 \in$ Q, F $\in$ Q } in which Q is a set of states, $\Sigma$ is an alphabet, $\delta$ is a relation ( or 'transition function') Q $\times$ $\Sigma \rightarrow$ Q, $q_0$ is an initial state of Q, and F is a set of final states.
**Definition 2.** An automaton congruence $\sim_A$ is a relation between states in A. Two strings x, y are congruent with respect to the automaton A iff $\delta^*(q_m, x) = \delta^*(q_m, y)$.[1]
**Definition 3.** The right congruence of a string w $\in$ L is a relation $\sim_R$ s.t. x $\sim$ y $\rightarrow$ xz $\sim$ yz for x, y, z $\in \Sigma^*$. A left congruence is defined as you'd expect.
**Definition 4.** The distribution of of a string w $\in$ L is $C_L = \{\langle l , r\rangle \in \Sigma^* \times \Sigma^* \mid lwr \in L\}$
**Definition 5.** Two words x, y $\in$ L are syntactically congruent w.r.t L (x $\sim_L$ y) iff $C_L(x) = C_L(y)$. This relation partitions the strings of L into a set of equivalence classes. The class of words in L congruent with w is $[w]_L$
**Definition 6.** A congruence *saturates* a language iff x $\sim$ y $\rightarrow$ x $\in$ L iff y $\in$ L.
**Definition 7** A path through an automaton is a (finite) sequence, $(q_n, a, q_{n+1}),(q_m, b, q_{m+1})$... from Q $\times$ $\Sigma$ $\times$ Q. We denote paths below $\delta^*$.

**Myhill-Nerode Theorem:** *A language is regular iff it has a finite number of congruence classes (i.e. $\Sigma/{\sim_L}$ is finite)*

**Proposition 1**: If L is regular, then $\sim_L$ has a finite number of congruence classes.

**Quick proof:** This direction is simple. If L is regular, then Kleene's theorem tells us that there exists an accepting DFA A, s.t. L = L(A), we can use this automaton to pick out a finite set of congruence classes saturating L. Assume L = L(A), it follows that for w $\in$ L, $\delta^*(q_{n-1}, w) \in F$. In other words, there is a path which upon recognizing w, transititions into a final state. Assume x,y are right congruent, x $\sim_R$ y, then $\delta^*(q_{n-1}, x) = \delta^*(q_{n-1}, y)$. Does this saturate? Well, if xz $\in$ L then $\delta^*(q_m, xz)$ and so $\delta^*(q_m, yz)$ which means yz $\in$ L. This means that if x $\sim_R$ y, then x $\sim_A$ y. This entails that $\sim_L$ has at most as many conguence classes as $\sim_A$ which is, by definition, finite.[2] The Myhill-Nerode theorem doesn't only tell us that every regular language has a finite number of congruence classes but that if a language has a finite number of congruence classes, then it is regular. This requires:

**Proposition 2**: If $\sim_L$ has a finite number of congruence classes, then L is regular.

**Quick proof** : What we need is to show that when a language has a finite partition into congruence classes under $\sim_L$, then we can build a recognising automaton for the language. Naturally, we use the classes of $\Sigma/{\sim_L}$ to generate the states of our automaton. Let $S_1$, $S_2$... $S_n$ be the classes of L under $\sim_L$ The union of these classes is our alphabet $\Sigma^*$. The transitions are defined $\delta^*([x]_L, a) = [xa]_L$. The set of final states are defined $F = \{[x]_L \mid x \in L\}$. The start state $q_0$ is simply identified with the class $[\epsilon]_L$.

The Myhill-Nerode theorem show us the connection between regular languages and finite semi-groups $\Sigma/{\sim_L}$ via finite state automata. The Büchi-Elgot-Tracktenbrot Theorem will show us the connection between regular languages and monadic second-order logic again with reference to automata. MSOL contains the usual logical connectives ($\wedge$, $\vee$, $\forall$...) along with a countable number of variables, $x_1$,$x_2$...$x_n$ predicates, $P_1$, $P_2$... $P_n$ and second-order predicates of arity 1. Below we'll also make use of the successor function 's' which is second-order definable.

**Büchi-Elgot-Tracktenbrot Theorem** : *A language is regular iff it is definable by a MSO (with successor) formula.*

**Proposition 3**: (BET left to right) If a language is regular, then it is definable in MSO (with successor) (henceforth S1S) .

From Kleene's theorem (again) we know that if a language is regular, then it is recognised by some fi-

[1]Definition 2 says that two strings are congruent with regard to an automaton iff the state the automaton reaches when one is the input is the same as that reached when the other is the input .
[2]In fact each equivalence class of $\sim_L$ is a union of equivalence classes of $\sim_A$

nite state automaton {Q, $\Sigma$, $\delta$, $q_0 \in Q$, $F \in Q$ }. All we need to do then, is to characterise this automaton in MSO by describing the accepting paths of the automaton. first-order variables will range over positions in a path while second-order variables range over states of the automaton. For each $q \in Q$ we have an MSO variable $X_q$ which encodes the set of positions a path visits en route to q.[3] We need the sentence to state that these states are pariwise disjoint (at no position are we in two different states) and that if a path passes from one state q to another state q' through the edge labelled $a$ then (q, $a$, q') $\in \delta^*$ (the accepted transitions of the automaton).
Such a sentence would look like:

$$\phi = \bigwedge_{q \neq q'} \neg\exists x(X_q(x) \wedge X_{q'}(x)) \wedge (\forall x \forall y\ S(x, y) \rightarrow \bigvee_{q,a,q' \in \delta^*} (X_q(x) \wedge a(x) \wedge X_{q'}(y))$$

$\phi$ characterises the states and transitions of our automaton A. The first 'clause' says that states are disjoint while the second gives us a way to assign for successive states in Q and labels in $\Sigma$, disjunctive conditions that represent the possibilty of the transitions from q to q'. To indicate that paths are successful we must also be able to say that the path's minimal position is in the initial state $X_{q_0}$ and the maximal position is in the set of final states. Minimal and maximal positions are easily defined by first-order formulas: x = min iff $\exists x \forall y \neg S(y, \min)$ (max is defined conversely.)

$$\phi_+ = \phi \wedge (\bigvee_{q \in I} X_q(\min)) \wedge (\bigvee_{q \in F}(\max))$$

We can now encode an automaton entirely with a formula: $\psi = \exists X_1\ \exists X_2\ ...\exists X_n\ \phi_+$.

**Proposition 4** : If a language is definable by a S1S sentence, then it is regular. (BET left-right)

To prove this we show by induction that for any MSO formula $\phi(X_1...X_n)$ there is a corresponding automaton which accepts exactly those words satisfying $\phi$. We need to show that for every language defined by a formula of S1S, there exists a corresponding finite state automaton which recognizes the language. To do this we show by induction that for every formation rule for formulas, there is a corresponding automata-theoretic operation.

We've already seen that you can use S1S formulas to define languages by using them to describe automata. Since we used that method for defining languages in the first half of this proof, we'll need an independent means of defining languages directly with S1S for this part. We're going to show first how a S1S can define a language

Unlike sentences, formulas might have free variables and we need to have the tools to satisfy them in our model. Say we have $n$ free first-order variables x and$m$ free second-order variables X, then we'll need an alphabet of length$m+n$ to represent all these (sets of) positions. If n = m = 0, then we have a sentence and the model is just the word w.
We don't know what any given $\phi$ will be so these word models have to satisfy a range of possible formulas with free variables. What we need then is a means of giving those formulas something to correspond to in the model. The trick is to allow them correspond to an extended word from an alphabet $\Sigma^*$ {0,1}. The $w \in \Sigma^*$ does its job as usual while the sets of positions of w denoted by X are interpreted by binary representations.

Now our new domain is more than a single word. It also includes strings whose job it is to interpret the free variables X, X of $\phi$. These free variables can be interpreted by sets of positions in a word $w$ . We code, for every free position, whether or not it is in one of the sets of positions, i.e. $x \in X$, $x \in X$? These richer models can interpret of S1S formulas.

$$\begin{array}{c} w_1, w_2 ... w_n \\ (c_1)_1, (c_1)_2 ... (c_1)_n \\ (c_2)_1, (c_2)_2 ... (c_2)_n \\ \vdots \end{array}$$

[3]States of the automaton are sets of positions the automaton can be in.

$$(c_m)_1,(c_m)_2...(c_m)_n$$

To make the transition from MSO sentences to automata clearer, we'll restrict our language to talking about set variables and set quantifiers. Invididual positions like $x_1$ will be represented as a singleton set {x}. The formulae of our restricted language will say things like: $X_j \subseteq X_k$, Sing($X_j$), Suc($X_jX_k$), and $X_j \subseteq Q_a$. A formula like $P_a(x)$ will be written sing(X) $\wedge$ X $\subseteq P_a$
Next we convert the logical expressions $\neg$, $\vee$ and $\exists$ into Boolean operations upon the regular languages.[4] This follows from the definitions.

$$L(\bigvee_{i\in I} \phi_i) = \cup_{i\in I} L(\phi_i)$$
$$L(\bigwedge_{i\in I} \phi_i) = \cap_{i\in I} L(\phi_i)$$
$$L(\neg\phi) = (\Sigma \times \{0, 1\}^n)^* - L(\phi)$$

## 3 References

* Bolhuis, B., Berwick, R.C., Okanoya,K., Beckers, G.J.L. 2011 Songs to syntax: the linguistics of birdsong Trends in Cognitive Science Mar;15(3):113-21
* Clark, A. 2013: The syntactic concept lattice: Another algebraic theory of the context-free languages?. Journal of Logic and Computation

[4]The other logical expressions can be defined by means of $\neg$, $\vee$ and $\exists$.