Lecture 15

Syntax is and isn't Context-Free

We are now operating under the assumption that syntax can be adequately described in terms of regular tree languages. They are sufficiently expressive to capture advanced linguistic concepts like headedness, projection, subcategorization, agreement, adjunction, and instances of displacement like wh-movement and topicalization in English. At the same time, they can be automatically converted into CFGs, which makes them suitable for a variety of well-understood parsing techniques. Today we will look once more at what can and cannot be done with regular tree languages, and we will see that, puzzlingly, syntax is both within and outside this class, depending on what kind of object we take syntax to define.

1 Syntactic Constraints

1.1 A General Template for Constraints

The syntactic literature is full of an enormous number of constraints. Some examples include:

- the Empty Category Principle (ECP), which restricts the possible landing sites of movement,
- Binding theory, which restricts the distribution of pronouns (*him*, *her*) and reflexives (*himself*, *herself*),
- the Person Case Constraint (PCC), which blocks certain combinations of pronouns in a sentence,
- the Shortest Derivation Principle (SDP), which favors shorter derivations over longer ones.

This is obviously not a complete list, and linguists keep coming up with new constraints or modify old ones to better fit the data. How could we possibly say that all constraints in natural language syntax fall within the class of regular tree languages?

As with any other empirical science, there is of course the possibility that a new discovery completely contradicts our current theories and requires new, more powerful machinery. But with any new piece of data we discover that matches current proposals, this scenario becomes less and less likely. After 50 years of generative linguistics with many competing formalisms, an abstract template for the formulation of constraints

has crystallized. Constraints that are stated with respect to a single tree usually involves four components:

- a finite number of nodes between which the dependency holds (e.g. the target site and the source of a moving phrase),
- a tree-geometric relation that picks out these nodes (e.g. c-command)
- a locality domain within which the dependency must be satisfied,
- a logical control mechanism that triggers the dependency (e.g. "if X is in position Y, then Z must be satisfied")

Comparative constraints like the SDP, where the well-formedness of a tree cannot be decided without looking at other trees, do not obviously fit this template, and we will not discuss them here. However, Graf (2013) shows that they, too, can be broken down into templates of this form (using ideas we encountered in the proof that certain variants of OT generate regular string languages).

The existence of such a template for constraints is crucial because if all four factors stay within the realm of regular tree languages, then all constraints that obey this template are regular, too. In the early 90s it was realized that constraints obeying the template can be easily expressed as statements of a formal description language that can be automatically translated into refined strictly 2-local tree grammars, thereby establishing their regularity. This insight forms the backbone of what is now known as *model-theoretic syntax* (Blackburn et al. 1993; Backofen et al. 1995; Kracht 1997; Rogers 1998; Potts and Pullum 2002; Pullum 2007).

1.2 Constraints, Logics, and Model Theory

If one abstracts away from all matters of implementation, a constraint c over trees is simply a method for defining the largest set of trees that satisfy c, or equivalently, do not violate c. So we can equate every constraint with a (possibly infinite) set of trees.

Something remarkably similar can be found in mathematical logic. There are many different logics, e.g. propositional logic, first-order logic, or modal logic. What makes each one of them a logic is that they are formal systems with a well-defined syntax, which defines what strings are well-formed formulas of the logic, and a semantics that assigns a specific interpretation to each formula.

For first-order logic, the syntax can be defined in a way that's very close to a context-free grammar. First, we have to fix a vocabulary, also called a *signature*. It includes a finite number of relational symbols R_i^n of arity n, and a set O of logical operands:

$\wedge$	and
$\vee$	or
$\rightarrow$	implies
$\leftrightarrow$	if and only if
$\neg$	not
$\forall$	for every
$\exists$	there exists

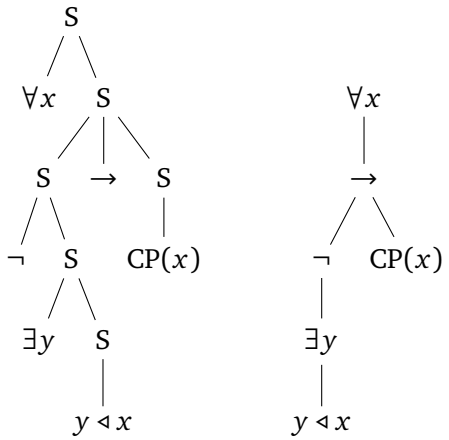
In addition, there is an infinite set of variables $x_1, \ldots, x_n, \ldots$, and the bracketing symbols "(" and ")". The set of well-formed formulas is given by the following rules:

$$
\begin{array}{lcl}
S & \rightarrow & (S \wedge S) \\
S & \rightarrow & (S \vee S) \\
S & \rightarrow & (S \rightarrow S) \\
S & \rightarrow & (S \leftrightarrow S) \\
S & \rightarrow & (\neg S) \\
S & \rightarrow & (\forall x_i\ S) \\
S & \rightarrow & (\exists x_i\ S) \\
S & \rightarrow & R_i^n(x_1, \ldots, x_n)
\end{array}
$$

Assuming that our only relational symbols are binary $\triangleleft$ and unary CP, the rules above tell us that the following is a well-formed formula:

$$(\forall x((\neg(\exists y(\triangleleft(y, x)))) \rightarrow \mathrm{CP}(x))$$

Assuming certain standard conventions about operator scope switching to infix notation for binary relations, we can drop some of the brackets and simplify the formula to $\forall x(\neg\exists y(y \triangleleft x) \rightarrow \mathrm{CP}(x))$, and we can also represent it in terms of a tree (the left format shows the phrase structure, the right one the dependency structure).



Defining the set of well-formed formulas gives us only one half of a logic, the much more important half is to endow each well-formed formula with a specific meaning. This is achieved by the notion of a *model*. A mathematical structure M is a model of formula ϕ (written $M \models \phi$) iff ϕ holds in M under a fixed interpretation. This interpretation is defined in a piece-wise fashion, where $\phi[x \leftarrow u]$ is the result of replacing every instance of x by u.

$$
\begin{array}{lcl}
M \models & \phi \wedge \psi & \text{iff both } M \models \phi \text{ and } M \models \psi \\
M \models & \phi \vee \psi & \text{iff } M \models \phi \text{ or } M \models \psi \\
M \models & \phi \rightarrow \psi & \text{iff } M \models \phi \text{ implies } M \models \psi \\
M \models & \phi \leftrightarrow \psi & \text{iff } M \models \phi \text{ implies } M \models \psi \text{ and the other way round} \\
M \models & \neg\phi & \text{iff } M \models \phi \text{ does not hold} \\
M \models & \forall x\phi & \text{iff } M \models \phi[x \leftarrow u] \text{ for all } u \text{ of } M \\
M \models & \exists x\phi & \text{iff } M \models \phi[x \leftarrow u] \text{ for some } u \text{ of } M
\end{array}
$$

In addition, one must also define an interpretation for all relational symbols. For example, if we assume that our class of models is the set of all trees, we may take $\triangleleft$ to denote the mother-of relation, whereas $\mathrm{CP}(x)$ means that node x has the label CP. In that case, the formula above holds of all trees whose root is labeled CP. Or the other way round, every tree whose root is labeled CP is a model for this formula.

Just like every constraint can be identified with the set of trees that satisfy it, every formula can be identified with its set of models. But this means that we can actually think of constraints in terms of logical formulas: a constraint c is a logical formula ϕ such that the set of structures obeying c is exactly the set of models of ϕ.

Example 15.1 A Formula for Trace Licensing

The ECP states that every trace must be c-commanded by the moving phrase that left behind this trace. We will formalize a simplified variant that ensures the presence of a c-commanding phrase that could be the original mover. To this end, we use first-order logic with a signature that contains $\lhd^+$ for proper dominance, $\approx$ for equivalence, and a unary predicate for every label in our intended tree alphabet. Note that we do not have to add the mother-of relation to the signature because it can be defined in terms of proper dominance:

$$x \lhd y \iff x \lhd^+ y \wedge \neg\exists z[x \lhd^+ z \wedge z \lhd^+ y]$$

The same is true of c-command:

$$\text{c-com}(x, y) \iff \neg(x \lhd^* y) \wedge \neg(y \lhd^* x) \wedge \forall z[z \lhd^+ x \rightarrow z \lhd^+ y]$$

Here $\lhd^*$ denotes reflexive dominance, which is also definable from proper dominance and equivalence.

$$x \lhd^* y \iff x \approx y \vee x \lhd^+ y$$

So now we have all the predicates we need to talk about the tree geometric configuration between traces and movers, but we still need a way to identify potential movers.

Clearly a phrase is a mover if it occupies a position where it cannot have been selected as an argument. According to the standard theory of movement, movers can only occur in specifiers, which are usually reserved for the second argument of a head. So it suffices to look at the head of the phrase and check whether it selects a second argument (for the sake of simplicity, we will assume that no head has an optional second argument, that phrases obey a strict X′-template where every phrase has at most one specifier, and we also ignore adjuncts, which would be identified as potential movers with this approach).

First we need to define a predicate that identifies whether a node is a specifier. A phrase is a specifier iff it is the sibling of a node labeled X′. Rather than assuming that the trees include this information in their interior node labels, we define a predicate for identifying the X′-node projected by a lexical item, which is then referenced in the definition of the predicate for specifiers.

$$\begin{aligned}
\text{bar}(x,y) &\iff \exists z[x \lhd z \wedge z \lhd y \wedge \bigvee_{l \text{ a lexical item}} l(y)] \\
\text{sibling}(x,y) &\iff \exists z[z \lhd x \wedge z \lhd y] \\
\text{spec}(x,y) &\iff \exists z[\text{sibling}(x,z) \wedge \text{bar}(z,y)]
\end{aligned}$$

Now we can finally define a predicate to identify moving phrases as specifiers of a head that takes no second argument.

$$\text{mover}(x) \iff \exists y[\text{spec}(x,y) \wedge \bigvee_{l \text{ takes at most one argument}} l(y)]$$

In the last step we require every trace to be c-commanded by a phrase.

$$\forall x[t(x) \rightarrow \exists y[\text{mover}(y) \wedge \text{c-com}(y,x)]]$$

Note that this is an actual formula, whereas all the previous formulas are just definitions that define predicates as a shorthand for specific subformulas that we use in definition of the ECP constraint.

Our implementation of the ECP leaves a lot to be desired, of course (a more adequate implementation is given in Rogers 1998, a logical formalization of GB that takes up over 150 pages). One glaring omission is that a mover cannot have left a specific trace behind if its category feature does not fit the position in which the trace occurs, for in that case the mover cannot have originated there. This can be added to the formula above by defining a predicate that checks what type of category is required for the position occupied by a trace and checks that the mover has this category — a rather simple exercise. A more interesting aspect is that movement is usually assumed to be locally bounded, which means that the licensor of a trace must occur within a specific domain, e.g. the smallest CP containing the trace. Such domain restrictions are very common across linguistic constraints, but fortunately they pose no challenge to first-order logic.

$$\begin{aligned}
\text{smallest}(x,y,ZP) &\iff ZP(y) \wedge y \lhd^{+} x \wedge \neg\exists z[ZP(z) \wedge y \lhd^{+} z \wedge z \lhd^{+} x] \\
\text{local}(x,y,ZP) &\iff \exists z[\text{smallest}(x,z,ZP) \wedge \text{smallest}(y,z,ZP)]
\end{aligned}$$

$$\forall x[t(x) \rightarrow \exists y[\text{mover}(y) \wedge \text{c-com}(y,x) \wedge \text{local}(x,y,\text{CP})]]$$

While the example above implements a specific constraint, it is easy to see that the tricks and techniques work for virtually all linguistic constraints. Lexicalization and projection means that all local information can be inferred from the head of a phrase, and we can keep track of this information via newly defined predicates. More complex tree geometric notions like c-command or m-command are easily defined

in terms of dominance, and the same holds for the notions of closeness and locality domain that are ubiquitous in linguistics. With a little bit of ingenuity, pretty much all syntactic constraints can be expressed in first-order logic.

First-order logic is a fragment of monadic second-order logic (MSO), which enriches first-order logic with the option to quantify over sets of nodes. It turns out that MSO is exactly as powerful as refined strictly 2-local tree grammars, so one can freely translate between grammars and MSO formulas (Büchi 1960). The construction is fairly involved (see Morawietz 2003 for details), and the computational complexity of the translation is unbounded — it doesn't get any less tractable than that. Nonetheless there have been several successful implementations, and the fact that linguistic constraints can be stated without set quantification reduces the complexity a lot. Overall, then, it is a viable strategy to formalize linguistic theories in terms of first-order formulas, which are then automatically translated into refined tree grammars or possibly even CFGs.

2 Pushing the Limits

2.1 Polarity Items

2.2 Binding Theory

2.3 Copy Movement

2.4 Crossing Dependencies

3 Towards a Context-Free Solution