Do model-theoretic grammars have fundamental advantages over proof-theoretic grammars?

Richard Moot

Do model-theoretic grammars have fundamental advantages over proof-theoretic grammars?

Richard Moot

1 Introduction

In a series of articles, Pullum & Scholz (2001, 2005), and later Pullum (2013*a*, 2019, 2020) have advocated model-theoretic syntax as a 'radical alternative' to syntactic theory as it has essentially ever been practiced before.

They contrast model-theoretic syntax (MTS) with what they call generative enumerative syntax (GES). Where generative enumerative syntax uses tools from proof theory, model-theoretic syntax uses the tools of model theory, even though in either case the concrete links with model theory and proof theory are generally rather tenuous.

While I can only applaud the introduction of new mathematical tools into formal linguistics, especially when they lead to deep mathematical results and novel perspectives (Rogers 1998), Pullum's advocacy of MTS is based on claims that there are a number of important facts any linguistic theory should explain, and that the MTS explanations for these facts are inherently superior to any explanation GES could provide, since the flaws of GES are "unavoidable side effects of the deepest aspects of the generative type of formalism itself" (Pullum 2020).

These claims about the superiority of MTS are frequently repeated without argument or discussion in the MTS literature. Postal (2010, p. 4) even claims "there are unrefuted arguments in the literature that adequate [natural language] grammars cannot be proof-theoretic". What surprised me about model-theoretic syntax is that although its advantages have been touted for more than 20 years now, few people appear to have bothered trying to demonstrate these advantages in practice. This makes it worth investigating precisely what these claimed advantages are, and how much evidence has been presented in their favour. Is there any substance to these supposed advantages or are they, to cite Pullum (2013*a*, p. 532) in a different context, "like cosmologists' references to dark energy, known to be out there somewhere but as yet neither empirically located nor theoretically accounted for in detail"?

2 Background

In this section we will briefly define the basic terms, and make explicit a number of distinctions which are often glossed over.

2.1 Logic and algorithms

Unfortunately, there is some confusion in the literature between declarative definitions and procedures, with procedural terminology being used in the proof-theoretic context and declarative terminology in the model-theoretic context. This can give the misleading impression that the main distinction between model-theoretic and proof-theoretic grammars is that the first are declarative and the second procedural.

It is therefore important to remember the following well-known dictum (Kowalski 1979).

$$\text{algorithm} = \text{logic} + \text{control} \tag{1}$$

In a declarative programming paradigm, we give a logical description of our problem, then add a control strategy to obtain an algorithm. As an example, suppose we write a simple definite clause grammar (without function symbols). When we use the Prolog execution mechanism, we obtain a top-down parser, but when we use the Datalog execution mechanism we obtain a bottom-up parser. The logic stays exactly the same, but the control mechanism produces very different algorithms.

Pullum & Scholz (2001, p. 19) are right to warn about 'procedural metaphors', and these procedural metaphors appear to go back at least to the early work of Chomsky. As a consequence, it is strictly wrong to say that generative grammars provide "a method or program for random construction" (Pullum 2013*a*, p. 495), or that they are "nondeterministic random construction procedures" (Pullum 2020, p. 4). Generative grammars are "nondeterministic random constrution procedures" in the same sense that linguistic structures in model-theoretic syntax are non-empty strings over the alphabet $\{0, 1\}$, and grammars use logic as a type of random access machine to verify these strings satisfy certain properties: strictly speaking both of these statements are valid interpretations, but they leave out so much context as to be exceptionally misleading, a rhetorical strategem that would merit a number of Pinocchios at many fact checking sites. Generative grammars, whatever the formalism used, are declarative statements, and they become algorithms only with the addition of procedural control. Therefore, the distinction Pullum (2020, ibid) makes between 'expansion oriented' and 'composition oriented' is purely a choice of control mechanism and has no relation to the logic used.

It is a *strength* of proof-theoretic syntax that it is generally fairly easy to given grammars a procedural interpretation. But even though procedural metaphors are a good tool to describe and explain grammar rules, we should always remember this is just a procedural interpretation of the rule: the literature on parsing and automated theorem proving demonstrates there are often many ways to turn the same declarative rules into different procedures.

2.2 Proofs and models

Given that proofs and models will be important throughout this paper, I will review some of the basic notions. Most readers with basic familiarity in logic will be able to skip this section.

The basic objects in model theory are structures, sets with associated relations and/or functions, defined as follows.

Definition 1 *A* structure *is a tuple of the form.*

$$\mathcal{M} = \langle M, \sigma, I \rangle$$

In this definition M is a non-empty domain, σ is the signature, specifying the non-logical symbols (predicates, functions, constants) and their arity, and I is an interpretation function. When f is a function of arity $\sigma(f)$, $I(f)$ is a function from $M^{\sigma(f)}$ to M. When p is a predicate of arity $\sigma(p)$, $I(p)$ is a subset of $M^{\sigma(p)}$. Constants c are treated as functions of arity 0; $I(c)$ is therefore a member of M.

This is the basic definition of a structure as you might find it in mathematical logic. It defines the non-logical symbols in the formal language of the logic we use, and ensures each symbol receives an interpretation of the appropriate type: an n-ary predicate is interpreted as the set of n-tuples for which it the predicate it true, an n-ary function is a function from n entities in our domain to an entity in our domain. This ensures that structures defined this way are possible models for logics with the same signature σ.

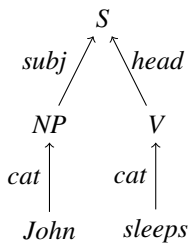
Example 1 *As a concrete example, a very crude representation of "John sleeps" in the model-theoretic framework could look as follows. We take*

$$\mathcal{E} = \langle A, \sigma, I \rangle$$

A, σ and I as follows.

$$\begin{aligned}
A &= \{a, b, c, d, e\} \\
\sigma(John) &= 1 & \sigma(NP) &= 1 \\
\sigma(sleeps) &= 1 & \sigma(V) &= 1 \\
\sigma(head) &= 2 & \sigma(S) &= 1 \\
\sigma(subj) &= 2 & \sigma(cat) &= 2 \\
I(John) &= \{a\} & I(sleeps) &= \{b\} \\
I(NP) &= \{c\} & I(VP) &= \{d\} \\
I(S) &= \{e\} \\
I(cat) &= \{(a, c)\} & I(cat) &= \{(b, d)\} \\
I(head) &= \{(d, e)\} & I(subj) &= \{(c, e)\}
\end{aligned}$$

When we portray nodes as their labels (the unary predicate which is true at this node), and binary predicates as arrows from their first to their second argument, we can draw this structure as follows.



$\Diamond$

In structure $\mathcal{E}$ of Example 1, a number of formulas are satisfied, for example $\exists x.NP(x)$ (choose $x = c$) and $\forall xNP(x) \Rightarrow \exists y.S(y) \wedge \text{subj}(x, y)$ (we verify the formula for all x such that $NP(x)$ holds, only c in the structure, and we can choose e for y). These formulas are not particularly interesting from a linguistic point of view, but $\mathcal{E}$ is still a model for these formulas, so we can write, for example $\mathcal{E} \vDash \exists x.\text{NP}(x)$. For our grammatical principles, we generally want to state things which are true in all models, not just specific ones.

The basic objects of proof theory are proofs. These are defined inductively. We have a set of axioms, the basic building blocks of proofs, and a set of rules which take one or more proofs to produce a new proof. For example, from a proof of A and a proof of $A \Rightarrow B$ we produce a proof of B using the familiar modus ponens rule.

$$\frac{\begin{array}{cc}\vdots & \vdots \\ A & A \Rightarrow B\end{array}}{B}$$

There are generally many different proof systems for a logic. For example, we can give a rule to introduce a conjunction '$A \wedge B$' as follows.

$$\frac{\begin{array}{cc}\vdots & \vdots \\ A & B\end{array}}{A \wedge B}$$

This rule simply states we can combine a proof of A with a proof of B to produce a proof of $A \wedge B$.

We can also formulate a system with many axioms and just a few rules, generally just modus ponens plus some rules for the logical quantifiers. In proof systems like this, the previous rule becomes the axiom $A \Rightarrow B \Rightarrow A \wedge B$. We can combine this axiom with a proof of A and a proof of B to produce a proof of $A \wedge B$ using modus ponens as the only rule.

$$\frac{\begin{array}{cc}\vdots & \\ B & \end{array} \qquad \frac{\begin{array}{cc}\vdots & \\ A & \end{array} \qquad \overline{A \Rightarrow B \Rightarrow A \wedge B}}{B \Rightarrow A \wedge B}}{A \wedge B}$$

What this shows is that we can often turn rules into axioms and vice versa. For some purposes it is more convenient to have a logic with few rules and many axioms, because this means there are few inductive cases to verify. For other purposes it is more convenient to have a logic with many rules and few axioms (often only $A \vdash A$) because this makes proof search easier, and also because some logically desirable properties can be more easily proved (such as consistency and the subformula property).

Ideally, a logic has both a proof theory and a model theory related by soundness and completeness proofs. These proofs relate model theory to proof theory by proving that A is derivable if and only if it is true in all models. We can therefore switch between proof theory and model theory in different ways. Instead of showing A is not provable, we can construct a model where A is false (a countermodel).

In what follows, we will only consider logics which are both sound and complete. Soundness and completeness allow us to switch between model theory and proof theory as shown below. Remark that 4 and 5 are simply the negations of 2 and 3. Equivalence 5 shows that we can show satisfiability of A by showing that $\neg A$ is not provable, exploiting the standard logical definition of '$\exists$' as '$\neg\forall\neg$'.

$$\vdash A \iff \forall\mathcal{M}.\mathcal{M} \vDash A \tag{2}$$

$$\nvdash A \iff \exists\mathcal{M}.\mathcal{M} \vDash \neg A \tag{3}$$

$$\vdash \neg A \iff \forall\mathcal{M}.\mathcal{M} \vDash \neg A \tag{4}$$

$$\nvdash \neg A \iff \exists\mathcal{M}.\mathcal{M} \vDash A \tag{5}$$

But what about the models themselves? These are structures and quite different objects from proofs. There are elementary model-theoretic results which allow us to translate formulas to models and models to formulas. A model corresponds to a set of atomic formulas and vice versa (Hodges 1997, Chapter 1). The model $\mathcal{E}$ of example 1 corresponds to the following set of atomic formulas (the set of all atomic formulas which are true in $\mathcal{E}$).

$$\begin{aligned}\{\ &\text{John}(a), \text{sleeps}(b),\\ &\text{NP}(c), \text{VP}(d), \text{S}(e),\\ &\text{cat}(a, c), \text{cat}(b, c), \text{head}(d, e), \text{subj}(c, e)\ \}\end{aligned}$$

The only complication here is that the set of atomic formulas corresponding to a model might not be finite (this is obvious when we look at models of Peano arithmetic for the natural numbers). We can turn this set into a single formula using conjunction. $A = A_1 \wedge \ldots \wedge A_n$, and we then have $A \vdash \exists x.\text{NP}(x)$ as the proof theoretic counterpart of $\mathcal{E} \vDash \exists x.\text{NP}(x)$.

The things to remember are that model theory and proof theory are different ways to study the same objects, and while some things are easier to prove using model theory and others using proof theory, we can move between model theory and proof theory rather easily: instead of showing there is a model of A, we can instead show that $\neg A$ has no proofs. In many cases, we can also switch back and forth between models and formulas, and between rules and axioms.

2.3 Proof-theoretic grammars, model-theoretic grammars, and their differences

We should be precise about the distinction between model-theoretic grammars and generative-enumerative grammars — I prefer term proof-theoretic grammars to generate-enumerative grammars and will treat these terms as synonymous throughout this article. Although the term model-theoretic syntax had been used before, the contrast between was most clearly developed by Pullum & Scholz (2001).

Proof-theoretic grammar A 'generative' grammar according to Pullum (2020, Section 1, p. 4) is any grammar specifying syntactic structures in the following way.

1. a set of one or more relevant objects [which is] given initially,

2. a set of operations for constructing new objects from those already available.

Rephrased more succinctly, proof-theoretic theoretic grammars define syntactic structures inductively.

Model-theoretic grammar A model-theoretic grammar, again according to Pullum (2020, Section 3), is defined as follows.

1. a grammar is a set of logical postulates or axioms (Pullum often uses the word 'constraints'),

2. syntactic structures are exactly those structures which satisfy these postulates.

In other words, model-theoretic syntax defines grammars as a logical theory and is interested in the structures which satisfy the postulates of this theory. It is important to emphasise that these definitions do not provide a dichotomy: a grammar can be both model-theoretic and proof-theoretic, or neither. For example, a formalism specifying transderivational constraints is neither proof-theoretic nor model-theoretic[1], and a model-theoretic formalism whose structures can be inductively defined is both model-theoretic and proof-theoretic.

Key differences Since nearly all proof-theoretic grammars can be seen as logical theories as well, the main difference is that model-theoretic grammars are satisfaction based rather than validity based. For model-theoretic grammars, the basic question is "what are the models which satisfy our axioms?" and these models are our basic linguistic structures, for proof-theoretic grammars it is "what can we derive using our axioms?", and these derivations are our basic linguistic structures. Stating things in the same terms, linguistic structures in model-theoretic syntax are true in *some* model, whereas linguistic structures in proof-theoretic syntax are true in *all* models.

More precisely, in an MTS grammar specified as a set of axioms $\mathcal{T} = A_1, \ldots, A_n$, the grammatical syntactic structures of this grammar are the $\mathcal{M}$ such that $\mathcal{M}$ satisfies all of the A_i, i.e. $\mathcal{M} \vDash A_1 \wedge \ldots \wedge A_n$. In a proof-theoretic grammar, given a logical theory $\mathcal{T}$, the syntactic structures are exactly the derivable statements in the logic, that is, the set of C such that $\mathcal{T} \vdash C$.

In a satisfaction-based theory, like MTS, we start by allowing everything permitted by the signature of our logic, then prohibit more and more by adding axioms. In a validity-based theory, we start by deriving only $A \vdash A$, and adding rules and axioms allows us to derive more and more.

Another key difference is the internal versus the external perspective. In proof-theoretic syntax, our derivations *are* our syntactic structures, whereas in model-theoretic syntax, our axioms describe these structures (Pullum 2013*a*, p. 498).

[1] In this context, a transderivational constraint would be a proof rule with $\Gamma \nvdash C$ as a premiss, or a model constraint of the type *if* $\sigma_1 \vDash \mathcal{T}$ *then* $\sigma_2 \nvDash \mathcal{T}$, two different ways of making the grammaticality of a structure/proof depend on the ungrammaticality of another.

	PTS	MTS
basic logical principle	validity	satisfaction
grammatical objects	derivations	structures
generated language	derivable statements	structures satisfying the theory
grammatical principles	theorems about derivations	logical consequences of the theory
grammaticality checking	proof checking	satisfaction checking
parsing	theorem proving	model building, proof theory
grammar cardinality	proof-theoretic analysis	spectrum theory

Table 1: A summary of the differences between the proof-theoretic and model-theoretic view of syntax

Table 1 gives a summary overview of the differences between the proof-theoretic and model-theoretic perspective. Most of these are simple consequences of the initial difference between validity and satisfaction. For example, in proof-theoretic syntax finding proofs of the form $\mathcal{T} \vdash C$ is used for parsing, whereas in model-theoretic syntax it is used to derive grammatical principles from the axioms, as done on many occasions by Johnson & Postal (1980). In a proof-theoretic formalism, grammatical principles correspond to theorems about derivations, that is, we prove properties which hold of all derivations, generally by induction on their structure.

Another difference is that while for proof-theoretic grammars, parsing is simply a form of proof search (parsing-as-deduction), model-theoretic grammars do not have such a direct way to decide whether a sentence is grammatical or not. Their main focus is deciding whether a fully specified *structure* satisfies all axioms (and this can be very inefficient: proof checking is easy, but model checking is at least PSPACE-complete for most of the standard logics used for MTS). There are a number of strategies we could follow for deciding whether a string is grammatical in a model-theoretic framework.

- *Model building*. Generate all possible structures with the given words as its yield, for example as done by Palm (2007).

- *Theorem proving*. Keep the grammar as it is, and code the input string as a formula. For a grammar $\mathcal{T}$ and an input string coded as A, we try to prove $\neg(A \wedge \mathcal{T})$. If we find a proof, the sentence is ungrammatical: the sentence contradicts the grammatical theory. If this fails, we can construct a model from the structure of the failed proof. This essentially the approach used by Palm (2004) using temporal logic. Appendix A presents a version of this theorem proving approach using first-order logic. Theorem proving for first-order logic or MSO is of course undecidable. While MSO has some important decidable fragments, these are deeply intractable: even when we restrict ourselves to MSO over regular string languages, theorem proving has non-elementary complexity[2].

- *Grammar translation*. Translate the grammar into a proof-theoretic grammar and use theorem proving. This is the approach of Morawietz (2003), who translates a

[2]More precisely, the complexity is a tower of exponentials, with the height k of the tower depending on the number of quantifier alternations, and the topmost exponent n depending on the size of the quantifier-free part of the formula (Meyer 1975, Libkin 2004).

model-theoretic grammar written in MSO to a tree automaton. This compilation step is still non-elementary, but parsing is then linear.

There is also a difference in the way we determine how many structures are generated by our grammatical theory. For a proof-theoretic grammar, we simply use the inductive definition, and in many cases there will be a countably infinite number of structures in our grammar. However, these are theorems obtained by analysing our logic, not principles imposed on it from outside. For model theory, the analysis of the number of (non-isomorphic) models of a theory is called 'spectrum theory' (Shelah 1978).

As we will see in what follows, the main problems with the comparison between proof-theoretic and model-theoretic syntax of Pullum & Scholz (2001) is that their criticism are either at the wrong level or do not distinguish in any meaningful way between model-theoretic and proof-theoretic grammars.

2.4 Model theory, proof theory, and natural language

There are two distinct ways to view model-theoretic syntax, a weak and a strong version.

- Weak MTS. The grammar has a proof-theoretic, inductive backbone generating candidate structures, with a set of axioms filtering out ill-formed ones. Or, alternatively, model-theoretic syntax and proof-theoretic syntax are merely different perspectives on the same structures.
- Strong MTS. There is no proof-theoretic backbone generating candidate structures, only axioms specifying properties of the allowed structures.

Weak MTS is essentially proof-theoretic syntax with constraints, or an alternative way to see the structures generated by proof-theoretic syntax. Pullum (2013*a*) makes a similar distinction, claiming that Government and Binding theory and Generalised Phrase Structure Grammar (GPSG) are 'hybrid' (weak MTS in my terms) whereas Head-driven Phrase Structure Grammar (HPSG) and Arc Pair Grammar, again according to Pullum, are 'clear examples of the MTS perspective' (strong MTS in my terms). Pullum argues in favour of strong MTS, but even his 'clear examples' are quite obviously instances of weak MTS.

Strong MTS grammars, because of the absence of inductively defined structures, exhibit 'structural holism'. The structures generated by model-theoretic grammars are not built up from basic objects and operations combining these, since that would make the grammar proof-theoretic[3]. This means that model-theoretic syntax, at least when it is not at the same time proof-theoretic syntax, has no inductive structure, and grammatical correctness is a holistic property of structures. The only clear example of strong MTS in Pullum's writings is the idea of expressing grammars by forbidden configurations of trees or strings (Pullum 2019). This is an interesting idea which would deserve to be further developed, opening up connections with graph minor theory from theoretical computer science (Robertson & Seymour 2004). Even then, I consider it quite

[3]It is possible to claim that grammatical structures are generated co-inductively instead, but this makes a difference only when we consider the basic linguistic objects to be infinite structures.

likely that grammars so developed would also be proof-theoretic (e.g. by taking the complement).

Since the definition of proof-theoretic grammar is any grammar formalism where linguistic structure can be defined inductively (possibly augmented with some constraints in the weak MTS perspective), this makes it quite hard to show that a grammar is not proof-theoretic, because this amounts to showing that there is no inductive structure to grammatical objects. Remember that it is quite irrelevant whether the inductive structure is specified using rules or using axioms. But there are easy ways of showing a grammar is proof-theoretic: a formalism which has any of the standard notions of constituency, dependency, or dominance is proof-theoretic, since these are all ways of defining structures inductively. An easy diagnostic is the following: if the formalism has an Earley-type or CKY-type parsing algorithm, then it is proof-theoretic.

When we look at HPSG, it has a clear generative backbone: there are basic objects defined in the lexicon and operations for creating new objects from already defined ones, such as the head-subject, head-specifier, and the head-complement principle. These principles are all tree construction rules, and clearly presented as such in the authoritative textbooks (Pollard & Sag 1994, Section 1.5), (Sag, Wasow & Bender 1999, Section 4.5). Even the versions of HPSG Pullum designates as clearly model-theoretic have such a proof-theoretic backbone.

Arc pair grammars have a clear inductive structure as well, and their grammatical objects can be seen as a type of (multiple overlapping) constituent trees with additional edges (and relations between those edges). For example, we find notions of constituent structures and principles stating that clauses are trees headed by verbs (Johnson & Postal 1980, PN Law 35, p. 212).

Similarly, Property grammars can be seen as inductively constructing constituent trees[4] with some additional edges for representing precedence (Blache & Prost 2014).

The problem with this is that most of the arguments against proof-theoretic syntax are actually arguments against inductively defined structures, that is, arguments in favour of structural holism. This means that whatever arguments are presented against inductively defined structures, they apply with equal force to HPSG, Arc Pair grammars and Property grammars, but perhaps not to a strong MTS formalism which might be proposed in the future.

3 Proof-theoretic syntax

Before discussing Pullum's criticisms in detail, I will present a version of proof-theoretic syntax. This will give a concrete formalism to help with later discussion.

The logic I will present is the Lambek calculus (Lambek 1958). Even though it has its known limitations, is it simple enough to allow a discussion with concrete examples and I will refer the reader elsewhere for a discussion of the different variants and extensions of the Lambek calculus and their linguistic applications (Moortgat 2010, Morrill, Valentín & Fadda 2011, Barker & Shan 2014, Kubota & Levine 2020, Moot 2021). The generic name for these formalisms is type-logical grammars.

[4]Technically, both Arc Pair grammars and Property grammars can be seen as inductively constructing constituent trees which are free trees (that is, acyclic, connected graphs).

$$\frac{}{A \vdash A}\ Ax$$

$$\frac{\Gamma \vdash A/B \quad \Delta \vdash B}{\Gamma, \Delta \vdash A}\ /E \qquad\qquad \frac{\Gamma, B \vdash A}{\Gamma \vdash A/B}\ /I$$

$$\frac{\Gamma \vdash B \quad \Delta \vdash B\backslash A}{\Gamma, \Delta \vdash A}\ \backslash E \qquad\qquad \frac{B, \Gamma \vdash A}{\Gamma \vdash B\backslash A}\ \backslash I$$

Figure 1: Lambek natural deduction rules

In a sense, type-logical grammars are the purest examples of proof-theoretic syntax, because the syntactic theories of type-logical grammars are *logics* rather than logical theories.

Figure 1 presents the logical rules for the Lambek calculus in natural deduction format. There is a single axiom $A \vdash A$ and only four rules[5], two for each connective.

The elimination rules are a form of modus ponens: from B together with B entails A we derive A. However, our logic is resource conscious (that is, we care how many occurrences of each formula there are) and sensitive to direction. We therefore have two implications A/B, which combines with a B to its right to form an A, and $B\backslash A$ which combines with a B to its left to form an A.

We can then paraphrase the $/E$ rule as follows: if we have derived some structure Γ to be of type A/B and some other structure Δ to be of type B, then the concatenation of these two structures Γ, Δ is of type A. The $\backslash E$ rule is symmetric.

For the $\backslash I$ rule, if B, Γ is a list of formulas from which we can derive A, then when we remove B from the list, we can derive $B\backslash A$. This is the deduction theorem in Hilbert-style formulations of logics: if from Γ and B we can derive A, then from Γ alone, we can derive B entails A. In this case, if B, Γ derives A, then Γ alone derives $B\backslash A$, that is, something missing a B to its left to form an A.

Model theory Even though this is a purely proof-theoretic interpretation of the logic, the Lambek calculus has a very natural semigroup interpretation for which it is sound and complete (Pentus 1995). If M is a set and '$\circ$' is the binary semigroup operation over elements of M, then formulas are interpreted as follows.

$$v(p) \subseteq M \tag{6}$$

$$v(B\backslash A) = \{y \mid \forall x \in v(B)\ x \circ y \in v(A)\} \tag{7}$$

$$v(A/B) = \{x \mid \forall y \in v(B)\ x \circ y \in v(A)\} \tag{8}$$

Atomic formulas p are interpreted as subsets of M. The interpretation of $B\backslash A$ is the set of those y such that for every x in the interpretation of B, $x \circ y$ is in the interpretation of

[5]Lambek (1958) presented a sequent calculus and additional rules for a product connective.

A. In other words, the objects in $B\backslash A$ are exactly the things which can be combined with any object in B (to their left) to produce an A. The interpretation of A/B is symmetric.

This result is much more profound than it appears at first sight: the Lambek calculus is a logic over strings, and it is sound and complete when interpreted over semigroups. From the point of view of mathematical logic, it is very desirable to have simple, well-understood mathematical objects as models for our logic[6].

Logic and grammar For the Lambek calculus, and type-logical grammars in general, the lexicon is simply a set-valued function from words to formulas. While we will return to the lexicon and lexical dependance in Section 4.3, for now we will simply note that the lexicon is not a part of the logic.

A tiny lexicon is presented below.

$$\begin{aligned}
\text{Lex}(\textit{he}) &= np & \text{Lex}(\textit{John}) &= np \\
\text{Lex}(\textit{him}) &= np & \text{Lex}(\textit{chair}) &= n \\
\text{Lex}(\textit{likes}) &= (np\backslash s)/np & \text{Lex}(\textit{of}) &= (n\backslash n)/np \\
\text{Lex}(\textit{is}) &= (np\backslash s)/np & \text{Lex}(\textit{his}) &= np/n \\
\text{Lex}(\textit{are}) &= (np\backslash s)/np & \text{Lex}(\textit{'s}) &= (np\backslash np)/n \\
\text{Lex}(\textit{the}) &= np/n & \text{Lex}(\textit{department}) &= n
\end{aligned}$$

Using this lexicon, we can find the following derivation for "John is the chair of his department".

$$\dfrac{np \vdash np \qquad \dfrac{(np\backslash s)/np \vdash (np\backslash s)/np \qquad \dfrac{np/n \vdash np/n \qquad \dfrac{n \vdash n \qquad \dfrac{(n\backslash n)/np \vdash (n\backslash n)/np \qquad \dfrac{np/n \vdash np/n \quad n \vdash n}{np/n, n \vdash np}\,/E}{(n\backslash n)/np, np/n, n \vdash n\backslash n}\,\backslash E}{n, (n\backslash n)/np, np/n, n \vdash n}\,/E}{np/n, n, (n\backslash n)/np, np/n, n \vdash np}\,/E}{(np\backslash s)/np, np/n, n, (n\backslash n)/np, np/n, n \vdash np\backslash s}\,\backslash E}{np, (np\backslash s)/np, np/n, n, (n\backslash n)/np, np/n, n \vdash s}\,\backslash E$$

While proofs like this are the objects of study for proof theorists, they are not that readable to ordinary humans. It is therefore fairly common in type-logical grammars to use some 'syntactic sugar' and replace formulas on the left hand side of the turnstile by the corresponding word from the lexicon. This gives the following, more readable

[6]The Lambek calculus also has Kripke models, which are a staple in model theory, and which extend more easily to variants of type-logical grammars (Kurtonina & Moortgat 1997).

proof.

$$\dfrac{\text{John} \vdash np \qquad \dfrac{\text{is} \vdash (np\backslash s)/np \qquad \dfrac{\text{the} \vdash np/n \qquad \dfrac{\text{chair} \vdash n \qquad \dfrac{\text{of} \vdash (n\backslash n)/np \qquad \dfrac{\text{his} \vdash np/n \quad \text{dept} \vdash n}{\text{his dept} \vdash np}\,/E}{\text{of his dept} \vdash n\backslash n}\,\backslash E}{\text{chair of his dept} \vdash n}\,/E}{\text{the chair of his dept} \vdash np}\,/E}{\text{is the chair of his dept} \vdash np\backslash s}\,\backslash E}{\text{John is the chair of his dept} \vdash s}$$

Even though this proof is more readable, it is isomorphic to the original proof, at least when the lexical assignments have been given.

Terms and features

We can extend the Lambek calculus with some restricted form of linguistic features by using the first-order quantifiers. To keep the logic simple, we only use implicit quantification here.

In the logic, this only changes the elimination rules. When combining A/B and B' for the $/E$ rule, we compute the most general unifier of B and B' and do a global substitution for the computed unifier. This is standard methodology for theorem proving in first-order logic (Fitting 1996).

$$\dfrac{\Gamma \vdash A/B \quad \Delta \vdash B'}{\Gamma, \Delta \vdash A}\,/E \qquad\qquad \dfrac{\Gamma \vdash B' \quad \Delta \vdash B\backslash A}{\Gamma, \Delta \vdash A}\,\backslash E$$

This allows us to give a simple analysis of "John likes coffee" as follows.

$$\dfrac{\text{John} \vdash np(3, \text{sing}, X) \qquad \dfrac{\text{likes} \vdash (np(3, \text{sing}, \text{nom})\backslash s(\text{fin}))/np(Y, Z, \text{acc}) \quad \text{coffee} \vdash np(3, \text{sing}, V)}{\text{likes coffee} \vdash np(3, \text{sing}, \text{nom})\backslash s(\text{fin})}\,/E}{\text{John likes coffee} \vdash s(\text{fin})}\,\backslash E$$

We have kept the variables from the lexical entries in place. Unification of $np(Y, Z, \text{acc})$ with $np(3, \text{sing}, V)$ for the $/E$ rule produces $Y = 3$, $Z = \text{sing}$ and $V = \text{acc}$, whereas unification of $np(3, \text{sing}, X)$ with $np(3, \text{sing}, \text{nom})$ produces $X = \text{nom}$.

Structural rules, linear logic, intuitionistic logic

We can also extend the Lambek calculus with a number of structural rules, which operate on the list of formulas in the antecedent. Figure 2 lists the standard structural rules which connect the Lambek calculus to other logics. The permutation rule, $[P]$, allows us to swap the order of two adjacent formulas on the list. This turns the antecedent from a list into a multiset, and the logic into intuitionistic linear logic, or the Lambek-van Benthem calculus LP.

The contraction C and weakening W rules, read from premiss to conclusion, allow us to remove duplicates (C) and add arbitrary formulas (W). Adding all three rules turns the logic into intuitionistic logic[7]. Given that we are interested in the lambda

[7] Adding P and W produces affine logic, adding P and C a type of relevant logic.

$$\frac{\Gamma, B, A, \Delta \vdash C}{\Gamma, A, B, \Delta \vdash C}\ P$$

$$\frac{\Gamma, A, A, \Delta \vdash C}{\Gamma, A, \Delta \vdash C}\ C \qquad\qquad \frac{\Gamma, \Delta \vdash C}{\Gamma, A, \Delta \vdash C}\ W$$

Figure 2: Structural rules, linear and intuitionistic

terms generated by our proofs for computing meaning in the formal semantics tradition of Montague (1974), intuitionistic logic is the most permissive logic we could use for this purpose, at least when we restrict ourselves to the implicational fragment.

Complexity

The complexity of theorem proving for most type-logical grammars is NP-complete, with some of the more expressive formalisms going up to PSPACE-complete. For comparison, the simpler task of model-checking for first-order logic is already PSPACE complete.

4 Criticisms of proof-theoretic syntax

The next sections discuss the main criticisms of Pullum & Scholz (2001) and Pullum (2020) and show how they fail to provide clear advantages for model-theoretic syntax.

4.1 Generative holism

As we have seen in Section 2.4, under Pullum's conception of model-theoretic syntax (what I call strong MTS), grammatical structures exhibit 'structural holism': grammatical correctness is a holistic property of structures. Pullum argues that proof-theoretic syntax exhibits what he calls 'generative holism', properties of a grammar are holistic properties of all the combined rules and axioms, and he even argues that "no individual operation says anything about any expression" (Pullum 2020, p. 6).

Let's investigate this more closely. What, to take Pullum's example, does the context-free rule 9 below entail?

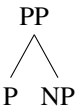
$$\text{PP} \rightarrow \text{P NP} \tag{9}$$

Pullum's criticism of what this rule does or doesn't say seem focused on some of the powerful operations allowed is some, mostly older, versions of generative grammar, which I have no particular interest in defending. For example, some versions of mainstream generative grammars have operations which necessarily delete or replace some symbols. However, this criticism is of highly specific operations in one type of formalism. It is therefore odd to see this criticism as part of a list of "unavoidable side

effects of the deepest aspects of the generative type of formalism itself" (Pullum 2020, ibid).

But when we look at context-free grammars, this rule does make a number of claims: there are prepositional phrases, and some of these consist of a prepositional phrase followed by a noun phrase[8].

Another way to interpret this rule is as stating that the following local tree is admissible in our grammar (McCawley 1968).



A final way to see 9 is as the following logical axiom.

$$\forall xy.[\mathrm{P}(x) \wedge \mathrm{NP}(y) \Rightarrow \mathrm{PP}(x + y)] \tag{10}$$

This is another standard interpretation of our example context-free rule, where the '+' infix function symbol denotes string concatenation. It states that if string x is a preposition and string y a noun phrase, then their concatenation is a prepositional phrase.

Pullum claims the following example, which expresses part of rule 9 model-theoretically, is superior.

$$\forall x.[\mathrm{PP}(x) \Rightarrow \exists y.[\mathrm{Head}(x, y) \wedge \mathrm{P}(y)]] \tag{11}$$

First of all, while there are differences in the type of things that 10 and 11 express, these differences are difference in grammar design and how we encode certain grammatical properties in our logic. Nothing specific with respect to model theory or proof theory can be said when we see one of these formulas in isolation.

We can also criticise 11 in ways rather similar to Pullum's critique of 9: this axiom is compatible with structures where prepositions and prepositional phrases are identical (i.e. the axiom allows $x = y$), it is compatible with structures where prepositions have multiple heads (i.e. are not trees), it is compatible with structures were $\mathrm{Head}(y, x)$ holds, and it is compatible with structures containing a path of countably infinite length between x and y. Excluding these structures requires knowledge of the other axioms. In other words, just like for the proof-theoretic versions, many properties of the grammar can only be determined globally, exhibiting a similar type of 'holism' if this is the term you want to use for it. In axiomatic systems, especially systems with many axioms, the individual axioms often make sense only when we know the intended interpretation of the predicate and function symbols, and when we have studied how this axiom interacts with the other axioms. One of the main reasons why proof theorists look for ways to replace axioms with rules as much as possible is to avoid such unintended interactions

[8] We assume here and elsewhere that neither P nor NP is a 'useless' variable, that is, there are strings in the grammar which derive P and strings which derive NP. This is easy to verify, and grammars containing useless variables can, by elementary transformations, be transformed into simpler grammars without such variables.

Note that the same type of criticism can be levelled at model-theoretic versions of this rule, but there it need not be easy to show that there are expressions assigned P and NP.

(Gentzen 1934, Dowek, Hardin & Kirchner 2003, Burel, Bury, Cauderlier, Delahaye, Halmagrand & Hermant 2020).

In the proof-theoretic context of Section 3, assigning a formula $(np\backslash s)/np$ to a word like "is" means this formula will necessarily occur in a subproof of the following form (for some structures x and y).

$$\dfrac{\begin{array}{c}\vdots\\ x \vdash np\end{array} \qquad \dfrac{\text{is} \vdash (np\backslash s)/np \quad \begin{array}{c}\vdots\\ y \vdash np\end{array}}{\text{is } y \vdash np\backslash s}\,/E}{x \text{ is } y \vdash s}\,\backslash E$$

We can make very precise claims about the shape of the proofs in which formulas in type-logical grammars occur. There is a useful interpretation of proofs in type-logical grammars where each formula corresponds to a 'constraint' (of the type shown above) on the proofs it can be a part of, and proofs correspond to the simultaneous satisfaction of all these constraints. In spite of what this terminology might suggest, this approach is fully proof-theoretic (Moot 2021).

The general point appears to be fairly trivial: there are some properties about proof-theoretic grammars which are harder to prove than for an equivalent model-theoretic grammar, requiring reference to the entire grammar. But this point is not very deep, as there are other properties which are easier to prove about proof-theoretic grammars. When a grammar writer adds subject-verb agreement as an axiom, it seems odd to then congratulate him about how easy it is to show the structures defined by his grammar satisfy subject-verb agreement.

4.2 Gradience

One of the main arguments against proof-theoretic syntax is the often repeated claim that these formalisms *necessarily* impose sharp boundaries, whereas model-theoretic syntax provides a simple account of gradience since it can make some of its axioms optional.

In this way, model-theoretic syntax can then define multiple boundaries: the fully grammatical structures, satisfying all axioms, the structures not satisfying one axiom but satisfying all the others, etc. In proof-theoretic syntax, when seen as a set of axioms, removing axioms means less sentences are derivable, so we would have to add axioms. The argument then appears to be that it is easier to remove axioms than to add them, which is something I can agree with. However, if adding axioms is hard, then grammar learning is hard (we will briefly return to grammar learning in Section 5.1).

From a more formal perspective the problem looks as follows. Take a theory $\mathcal{T}$ and a structure S with corresponding formula A. In the model-theoretic, satisfaction-based context we have $S \not\models \mathcal{T}$ (that is, S is not grammatical for our theory) and we look for the largest $\mathcal{T}' \subset \mathcal{T}$ such that $S \models \mathcal{T}'$. The advantage of the model-theoretic case is that these $\mathcal{T}'$ can be enumerated. In the proof-theoretic, validity-based context we have $\mathcal{T} \nvdash A$ and we look for the smallest $\mathcal{T}' \supset \mathcal{T}$ such that $\mathcal{T}' \vdash A$ (we should also verify that $\mathcal{T}'$ is consistent, i.e. $\mathcal{T}' \nvdash \bot$). We can reformulate this as looking for the smallest formula B such that $\mathcal{T} \wedge B \vdash A$. Logically, the proof-theoretic case is a type

of abduction, and there are solutions for finding the formula B from the failed proof of $\mathcal{T} \nvdash A$, essentially by computing the minimal formulas needed to close the open branches in a tableau proof (Mayer & Pirri 1993).

However, there is a much simpler solution. First, it is important to remember that model-theoretic syntax defines gradience over structures. Given a structure, a model-theoretic grammar can then indicate, for each axiom, whether it is satisfied or violated.

This notion is easily adapted to the proof-theoretic context: like the model-theoretic case, we look at a structure then mark any local subtree which does not correspond to any of our rules as incorrect. This gives a rather similar notion of gradience as the one advocated for the model-theoretic case. We have the structures where all rule applications are correct, we have those where exactly one application is incorrect, etc. In this scenario, gradience is a form of proof checking, in the same way that it is a form of model checking for MTS.

Pullum's remark that such a structure is not generated is completely besides the point: it confuses parsing/theorem proving with proof checking. He himself simply assumes fully specified structures are given as input to his grammars, and we similarly assume a fully specified proof candidate is given for evaluation to our proof checker. We should either compare model checking with proof checking, as we do here, or model building with theorem proving. We will see below that it is fairly easy to set up a proof-theoretic syntax in such a way that we can generate ungrammatical sentences, thereby giving an account of gradience which operates on strings.

I will illustrate this in the context to categorial grammars by taking Pullum's examples of successively worse versions of "He is the chair of his department", repeated below, complete with his error counts.

(1)	a.	He is the chair of his department.	
	b.	* He are the chair of his department.	(1 error)
	c.	** Him are chair of his department.	(2 errors)
	d.	*** Him are chair of he's department.	(3 errors)
	e.	**** Him chair are of he's department.	(4 errors)
	f.	***** Him chair are he's department of.	(5 errors)

Starting with example (1-b), there is a unification failure between "he" and "are" either of number $sing \neq plur$ or of person $2 \neq 3$. This can be handled by the basic treatment of features in the Lambek calculus introduced before, giving the following schematic proof for "he are" with the $\backslash E$ rule marked as incorrect because of the unification failure.

$$\frac{\text{he} \vdash np(sing, \ldots) \quad \begin{array}{c}\text{are} \ldots \vdash np(plur, \ldots)\backslash s \\ \vdots\end{array}}{\text{he are} \ldots \vdash s}\ \backslash E_*$$

For example (1-c), we count the missing determiner counts as an additional error, because even when we add a rule like $n \vdash np$ it would be restricted to plurals.

$$\dfrac{\text{him} \vdash np \qquad \dfrac{\text{are} \vdash (np\backslash s)/np \qquad \dfrac{\text{chair} \vdash n \qquad \dfrac{\text{of} \vdash (n\backslash n)/np \qquad \dfrac{\text{his} \vdash np/n \quad \text{dept} \vdash n}{\text{he's dept} \vdash np}\,/E}{\text{of he's dept} \vdash n\backslash n}\,\backslash E}{\text{chair of his dept} \vdash n}\,*}{\text{chair of his dept} \vdash np}\,/E}{\text{are chair of his dept} \vdash np\backslash s}\,\backslash E_*}{\text{him are chair of his dept} \vdash s}$$

For example (1-d), it is possible to analyse "he's" as a similar unification failure, by modelling the "'s" genitive as requiring an uncased noun phrase to its left[9]. This gives the following proof.

$$\dfrac{\text{him} \vdash np \qquad \dfrac{\text{are} \vdash (np\backslash s)/np \qquad \dfrac{\text{chair} \vdash n \qquad \dfrac{\text{of} \vdash (n\backslash n)/np \qquad \dfrac{\dfrac{\text{he} \vdash np \quad \text{'s} \vdash np\backslash(np/n)}{\text{he's} \vdash np/n}\,\backslash E_* \quad \text{dept} \vdash n}{\text{he's dept} \vdash np}\,/E}{\text{of he's dept} \vdash n\backslash n}\,\backslash E}{\text{chair of he's dept} \vdash n}\,*}{\text{chair of he's dept} \vdash np}\,/E}{\text{are chair of he's dept} \vdash np\backslash s}\,\backslash E_*}{\text{him are chair of he's dept} \vdash s}$$

From example (1-d) to (1-e), only the $/E$ rule marked with $*$ changes. However, the permutations which produce reasonable sentences are quite limited: the only other reasonable permutation would be "chair of he's dept are".

$$\dfrac{\text{him} \vdash np \qquad \dfrac{\text{are} \vdash (np\backslash s)/np \qquad \dfrac{\text{chair} \vdash n \qquad \dfrac{\text{of} \vdash (n\backslash n)/np \qquad \dfrac{\dfrac{\text{he} \vdash np \quad \text{'s} \vdash np\backslash(np/n)}{\text{he's} \vdash np/n}\,\backslash E_* \quad \text{dept} \vdash n}{\text{he's dept} \vdash np}\,/E}{\text{of he's dept} \vdash n\backslash n}\,\backslash E}{\text{chair of he's dept} \vdash n}\,*}{\text{chair of he's dept} \vdash np}\,/E_*}{\text{chair are of he's dept} \vdash np\backslash s}\,\backslash E_*}{\text{him chair are of he's dept} \vdash s}$$

Finally, from example (1-e) to (1-f) the topmost starred $/E$ essentially becomes a

[9]We can also attribute the error to the morphology component of our grammar, and then count " he's" as incorrect for reasons similar to why we would count "betted" and "curiouser" as wrong.

$\backslash E$ and is therefore counted as an additional error.

$$\dfrac{\text{him} \vdash np \qquad \dfrac{\text{are} \vdash (np\backslash s)/np \qquad \dfrac{\dfrac{\text{chair} \vdash n \qquad \dfrac{\text{of} \vdash (n\backslash n)/np \qquad \dfrac{\dfrac{\text{he} \vdash np \quad \text{'s} \vdash np\backslash(np/n)}{\text{he's} \vdash np/n}\backslash E_* \quad \text{dept} \vdash n}{\text{he's dept} \vdash np}/E}{\text{he's dept of} \vdash n\backslash n}/E_*}{\text{chair he's dept of} \vdash n}\backslash E}{\text{chair he's dept of} \vdash np}\,{}^{*}/E_*}{\text{chair are he's dept of} \vdash np\backslash s}/E_*}{\text{him chair are he's dept of} \vdash s}\backslash E_*$$

The proposal I sketched here adequately accounts for the desired degrees of granularity, it allows us point to where specific rules are incorrectly applied, and the approach is fully constructive and local.

Gradience and proof search

We have only compared proof checking with model checking in our discussion so far, which is a comparison at the same level. We will now investigate how to handle gradience in proof search. Given that removing axioms in model-theoretic syntax amounts to adding rules in proof-theoretic syntax, or at least relaxing their conditions, this points to a fairly obvious solution.

A simple candidate rule for relaxing derivability would be the permutation rule $[P]$ of Figure 2. We can even add the contraction and weakening rule, and some non-logical axioms between atomic formulas (e.g. $n \vdash np$) without affecting decidability. Similarly, instead of incorporating unification into the $/E$ and $\backslash E$ rules, we treat unification of the arguments as a side condition to the rule to be verified when the proof is completed.

This gives us a logic for generating proof candidates, with Lambek calculus proofs as a proper subset. We can then use a standard proof search procedure to obtain proofs, combined with a proof checking mechanism to verify the generated proofs are Lambek calculus proofs like we did in the previous section.

The complexity of proof search remains NP-complete when adding commutativity, which compares very favourably to parsing in model-theoretic frameworks.

Obviously, some work is needed to turn this into a fully-fledged theory of gradience: permutations degrade the acceptability of a sentence very rapidly, and weakening appears acceptable mostly in cases of self-correction. However, all that I needed to show here is that gradience can be treated in a proof-theoretic formalism, and that this does not require us to abandon any of the foundational assumptions of proof-theoretic syntax.

Gradience in model-theoretic syntax

The basic idea of handling gradience in model-theoretic syntax is making axioms optional. While this sounds simple enough, it is not enough on its own to make accurate predictions. Pullum apparently believes this is trivial, since he doesn't give the slightest indication of how structures (grammatical or not) are generated from strings or how he intends to obtain the error scores for his example sentences. If we model subject-verb

agreement by an axiom of the schematic form shown in 12 below, then this axiom is satisfied by example (1-b), simply because the antecedent of the material implication is false. The result will be that we obtain a forest consisting of two trees, one with root NP and one with root VP, but we have no idea where a constraint was violated (in general, when an axiom with an existential quantifier is not satisfied, we cannot say where).

$$\forall x.[\mathrm{NP}(x, \ldots) \wedge \mathrm{VP}(x, \ldots) \Rightarrow \mathrm{S}(\ldots)] \tag{12}$$

We can change 12 to the slightly more complicated 13 below, but then we will have the same problem when we use backward chaining proof search (or a top-down parser).

$$\forall xy.[\mathrm{NP}(x, \ldots) \wedge \mathrm{VP}(y, \ldots) \Rightarrow \mathrm{S}(\ldots) \wedge x = y] \tag{13}$$

So even getting subject-verb agreement correct requires careful engineering of the axioms, and knowledge of the algorithm used for generating partially correct structures.

For a vivid example showing why model-theoretic syntax does not have a fundamental advantage over proof-theoretic syntax, take the string "of the and a but through or", which Pullum & Scholz (2001) rightfully consider to be 'complete gibberish'. It has the following model-theoretic analysis.

P	D	Crd	D	Crd	P	Crd
of	the	and	a	but	through	or

When we analyse the string like this, it violates only a single tree axiom, namely $A1$ from Rogers (1998) which stipulates the existence of a node which dominates all others. Other than that, every word is assigned a syntactic category. It satisfies several other axioms as well: all prepositional phrases have a preposition as their head, all coordinate structures have a coordinator. All these axioms are vacuously satisfied, of course, but this is the way it should be. It is not clear what type of principle should be added to make this structure violate more axioms: when we add an axiom requiring a vertex to have a parent, we always make an exception for the root node. We could restrict the categories allowed for the root node, we could also require sister or parent nodes for certain categories, but in either case the treatment of fragments would disappear.

Since this structure violates only a single axiom, it should therefore be comparable in grammaticalness to other structures violating a single axiom, such as "He are the chair of his department" or "his and my book"

The problem with the model-theoretic approach to gradience is that when axioms become mere suggestions, then the structure satisfying the greatest amount of these suggestions might actually be a quite impoverished one. It is therefore no coincidence that researchers who have experimented with gradience in model-theoretic syntax use, first of all, a proof-theoretic base grammar to generate candidate structures inductively, but also much more sophisticated machinery to compute an acceptability score, essentially a type of percentage of constraints satisfied among those which could potentially apply (Blache & Prost 2005), and weighted axioms Duchier, Prost & Dao (2009). So

while some excellent work has been done on gradience in model-theoretic syntax, it needs some fairly complex calculations, weights, and a proof-theoretic backbone.

So in spite of giving the impression of being a strong argument in favour of model-theoretic syntax seen from an abstract, high level, this is only because the comparison is at the wrong level and details of empirical evaluation are conveniently ignored. Model-theoretic syntax has no advantages over proof-theoretic syntax when we compare them at the same level, or when we look in detail at concrete examples.

4.3 Lexical dependence

The final main claim of Pullum, which he calls "devastating", is that proof-theoretic syntax is necessarily dependant on the lexicon whereas model-theoretic syntax is not.

The claim that proof-theoretic syntax is necessarily dependant on the lexicon is certainly true in a boring way: the definitions of many grammar formalisms generally start by specifying an explicit signature Σ. These basic units of a grammar formalism need not be words: they can be smaller units (morphemes, phonemes, characters) or abstract categories like part-of-speech tags. But in the same way, whatever logical formalism is used for a model-theoretic approach, an alphabet and a signature are generally required. So if we want to analyse "Pirots karulize elatically" in any way, our logical signature needs all these symbols in its alphabet to assign them to nodes in the structure: an expression like $karulize(x)$ is a well-formed formula only when *karulize* is a unary predicate symbol in our logic. In other words, a model-theoretic grammar cannot say anything about "karulize" because it is not even in the language.

This may seem pedantic, but just like most logic texts only specify an explicit signature for some of their more basic examples, so do most formal grammars (and proof-theoretic grammars) only specify an explicit terminal alphabet for some toy grammars.

Pullum proposes to analyse phrases containing invented words by simply assuming there is some morphological process assigning "Pirots karulize elatically" the correct NP V Adv part-of-speech tags. However, once we have the part-of-speech tags, we can use a proof-theoretic grammar to analyse it, and whatever mechanism is used for assigning these part-of-speech tags, for Pullum's argument to be valid, he must show that such an assignment is *necessarily* model-theoretic.

For example, we could add rules like the following to our grammar.

$$\forall x, w.\ string(x, w + s) \Rightarrow postag(x, \mathrm{NP}(pl)) \wedge meaning(x, f(\mathrm{NP}(pl), w)) \tag{14}$$

$$\forall x, w.\ string(x, w + ize) \Rightarrow postag(x, \mathrm{V}(3s)) \wedge meaning(x, f(\mathrm{V}, w)) \tag{15}$$

$$\forall x, w.\ string(x, w + ally) \Rightarrow postag(x, \mathrm{ADV}) \wedge meaning(x, f(\mathrm{ADV}, w)) \tag{16}$$

In the rules above, x is a vertex, w a string, and $+$ the string concatenation operation, *postag* assigns a part-of-speech tag to a vertex, and *meaning* assigns a meaning to a vertex, while f is a function computing a meaning given part-of-speech tags and the root w (the details of this function are up to the grammar designer).

Rule 14 can informally be summarised as stating that words ending with the suffix "s" can be plural noun phrases. Similarly, rule 15 states that words with root w and suffix "ize" can be third-person singular verbs, and rule 16 states that words with suffix

"ally" can be adverbs[10].

Even when adding the above rules to a model-theoretic grammar, "pirot", "karul" or "elatic" are possible as values for x only when we have made these terms part of the signature of our language in advance.

For words not appearing in the lexicon, the following strategies (and many combinations) are possible.

1. *Part-of-speech tagging.* In computational linguistics, part-of-speech tagging is a task which has been considered solved for a long time. This allows us to use the following strategy for the lexicon: if the word is not in the lexicon, we simply use its part-of-speech tag in the grammar. This is again completely orthogonal to model-theoretic or proof-theoretic syntax, and Pullum (2020) uses this strategy himself without comment or reflection.

2. *Change the lexical units.* Our lexical entries can contain units other than fully inflected words. Our lexical entries can instead be morphemes, phonemes or characters (part-of-speech tags, as suggested by solution 1, can be another possibility).

3. *Machine learning.* While Pullum & Scholz (2001) are mostly right about rejecting statistics/machine learning for *gradience*, machine learning has achieved stunning successes for resolving lexical ambiguity (so-called supertagging) and a side-effect of this is that we can assign complex lexical assignments to unknown words without problem (Bangalore & Joshi 2011), this explicitly includes type-logical grammars (Kogkalidis & Moortgat 2022).

4. *Inference.* In type-logical grammars, we can in many cases simply solve for the unknowns analytically (Moortgat 1988, Emms 1993, Mirzapour 2017).

5. *Default rules.* These are rules like 14 to 16 above where, when no more specific rule is found, we use a generic rule as backup. Some logics — non-monotonic logic, default logic — have been developed for cases much like this.

6. *Parameters.* In model theory, we can extend our structure with parameters, essentially a type of additional constants. It is possible to argue that "pirots" is a parameter. However, this amounts to admitting our *grammar* doesn't have anything to say about this word. Using parameters is morally equivalent to the proof-theoretic solution of Neeleman (2013), who proposed to temporarily add words to the lexicon. Given his comments on this paper, it would be hard for Pullum (2013*b*) to consistently advocate for the parameter approach in the model-theoretic context.

So again, if lexical dependence is a problem, then it is a problem for model-theoretic syntax as well, and the different available solutions are not tied to any specific framework.

[10] Such rules would of course need to be refined with some sort of default mechanism to avoid the generation of incorrect forms for irregular plurals e.g. "foots" instead of "feet" and "mans" instead of "men".

5 Minor arguments

5.1 Learnability

There have been a number of results about the learnability of certain classes of language. In a very abstract way, the language learner is exposed to a number of strings in the language, then progressively forms and refines his hypothesis about the structure of the language.

With respect to learnability, the difference between the model-theoretic and proof-theoretic perspective is the following. From the proof-theoretic perspective, the 'blank slate' is where nothing is grammatical, and a grammar is constructed by adding more and more rules or axioms to account for the expressions encountered by the learner. From the model-theoretic perspective, the blank slate is allowing everything, and the learner hypothesises postulates to account for the regularities in the expressions encountered, reducing the allowed structures. Even though Pullum emphasises the negative results about learnability from the proof-theoretic perspective, there have been quite a few positive results as well. So while Gold (1967) has shown that there are classes of regular languages which are not learnable according to his theoretical model, Shinohara (1990) has shown that even context-sensitive languages are learnable in this paradigm, provided that we impose a bound on the number of rules. There are also quite expressive type-logical grammars which are learnable (Costa-Florêncio 2003).

This means it is rather up to the proponents of model-theoretic syntax to show learnability of their specific set of axioms, and it is not at all clear how language learners would acquire their logical postulates, or what would cause them to revise a postulate. One possibility is that all these postulates are innate and that the learner simply has to pick and choose postulates from this fixed, innate set, e.g. choose between head first or head final. This is rather similar to the old Principles and Parameters type of learning[11] and would therefore not be a point in favour of model-theoretic syntax with respect to its proof-theoretic counterparts.

But there doesn't seem to be any reasonable alternative for model-theoretic syntax. If the learner is allowed to hypothesise any well-formed formula, then it appears exceedingly unlikely that there is a process which converges to something resembling the correct grammar given grammatical strings as input and feedback, positive and/or negative, about the learner's hypothesised grammars and structures. While it is certainly possible that something could be worked out, doing so is a research program, the success of which remains to be determined.

In sum, there are a number learnability results for proof-theoretic syntax, some negative but also some important positive ones. There are no such results, positive or negative, for model-theoretic syntax. I fail to see how anyone can reasonably claim advantages with respect to learnability for model-theoretic syntax.

[11] I find it quite implausible that something functionally equivalent to the formal apparatus of mainstream generative grammar is innate, and would be similarly skeptical about innate logical postulates, making a possible exception for the tree axioms.

5.2 Fragments

Pullum & Scholz (2001, Section 3.2) claim that proof-theoretic grammars cannot say anything about expression fragments or incomplete expressions, such as "and of the". The argument is rather similar to the gradience argument discussed before: given a structure representing an incomplete expression, a model theoretic grammar can say which axioms it satisfies and which ones it doesn't, Pullum & Scholz are silent about how to produce a structure for a fragment, but proof-theoretic syntax is criticised for failing to produce the structure which is simply assumed as given in the model-theoretic case.

This criticism can therefore be addressed the same way as the gradience criticism: given a proof candidate, we can use a proof checker to verify correct applications of the rules and count the incorrect applications in the same way Pullum & Scholz would count the axioms not satisfied.

However, in the case of the Lambek calculus, there is an even simpler solution. We can simply compute the formula assigned to "and of the" as follows.

$$\dfrac{\text{and} \vdash (pp\backslash pp)/pp \qquad \dfrac{\text{of} \vdash pp/np \qquad \dfrac{\text{the} \vdash np/n \quad n \vdash n}{\text{the } n \vdash np}\,/E}{\text{of the } n \vdash pp}\,/E}{\dfrac{\text{and of the } n \vdash pp\backslash pp}{\text{and of the} \vdash (pp\backslash pp)/n}\,/I}\,/E$$

This is a perfect Lambek calculus proof. We might argue that the proof should indicate it depends crucially on associativity of the basic logic, but there are simple technical solutions to that (Kurtonina & Moortgat 1997, Moortgat 2010). So this is another non-issue.

5.3 Finite model theory

Pullum (2020, p. 10) invokes finite model theory as an important logical foundation for research in model-theoretic syntax. I agree that finite model theory is a research area which has revealed many profound connections between logic, complexity theory and formal language theory. I also agree that there are undoubtedly still many connections to explore between finite model theory and research in formal linguistics. However, when it comes to expressing linguistically relevant properties in various logics, many results in finite model theory are negative. For example, if we choose to use first-order logic we cannot express that our graphs are connected, acyclic or trees. Other linguistically useful properties not expressible in first-order logic are queries like "does this phrase contain a gap?". Given that these are some of the most basic types of constraints to impose upon linguistic structures and that all linguistic theories impose at least one of these upon their structures, finite model theory therefore shows first-order logic to be inadequate for expressing natural language grammars. In addition, finite model theory is concerned with model checking and has relatively little to say about model building (parsing).

What are some possible solutions to these negative results?

1. *Use a more expressive logic.* The most obvious candidate here is monadic second-order logic (MSO), and several prominent authors in formal linguistics and model-theoretic syntax have used MSO for modelling natural language (Rogers 1998, Morawietz 2003). An alternative would be to add a form of fixed-point to our logic, essentially using Datalog to write our grammars. However, this means that even the most basic questions such as "is this fully specified structure grammatical according to our logical axioms?" become PSPACE-complete or worse[12].

2. *Use a more restrictive logic.* Several authors have proposed more restrictive logics, generally some type of modal logic, to deal explicitly with tree structures (Blackburn & Meyer-Viol 1994, Palm 2004). While this ensures better complexity-theoretic properties than monadic second-order logic, it is unclear whether these logics have the required expressivity for natural language syntax. Some of the most important points of Rogers (1998) and Morawietz (2003) are the connections they make to standard linguistic theories, thereby showing that MSO has the required expressivity. It remains to be shown that any of these more restricted logics has the expressivity required to implement Pullum's program.

5.4 Cardinalities, fixed or not

Pullum (2020) is right in claiming that, given a proof-theoretic grammar we can generally decide more or less easily what the size of the given grammar is in terms of the generated objects. In most cases, the number of expressions generated will be countably infinite: all it takes is that the grammar contains a recursive definition which is not primitive recursive. Whether or not there are languages which do not generate an infinite set of strings is ultimately an empirical question, open for reasoned debate.

However, where Pullum is wrong is when he claims that model-theoretic syntax is silent about the cardinalities of languages. There has been a fruitful line of work exploring the number of models (up to isomorphism) of theories, and there are many definite answers to these questions (Shelah 1978). So in many cases, model-theoretic grammars *do* make claims about the cardinality of the structures described by a grammar, but they can make claims which are quite different from those expressible in proof-theoretic formalisms.

One important difference is that model-theoretic grammars can define theories which have an uncountable number of models (and several standard theories have many such models), whereas the number of proofs in a proof theory is generally countable. While it is true some authors have argued in favour of natural languages being uncountable (Langendoen & Postal 1984), these claims depend on a number of assumptions, such as the existence of infinite-length sentences (i.e. length ω, not simply unbounded), which we can combine using infinitary operations, to successively construct larger and larger

[12] Under the reasonable assumption that the graphs representing linguistic structures are sufficiently tree-like, we can compute these queries in linear time, but this requires a compilation of MSO formulas to tree automata. This compilation has non-elementary complexity even when the MSO theory describes a regular string language; more precisely, there is a stack of exponentials whose height is bound by the number n of quantifier alternations (Libkin 2004, Chapter 7). This compilation is therefore deeply intractable. Morawietz (2003, Chapter 6) provides a good description of parsing MSO grammars and presents experimental results for compilation of some crucial parts of his grammars.

uncountable classes. Hart (2021) presents a detailed argument about the mathematical problems with the arguments of Langendoen & Postal (1984). There are a number of problems with sentences of uncountable lengths[13]. For example, it entails that sentences are not subsets of Σ^* (that is, sequences of letters from an alphabet consisting of a finite number of letters or morphemes), and that it is impossible to give a lexicographic ordering on sentences.

When we make the linguistically reasonable restriction to finite models[14], there is no clear distinction between proof theory and model theory for this question.

6 Conclusions

Might I just be "too conservative to like the idea that MTS offers better theories of linguistic phenomena" (Pullum 2013*b*)? It's just an idea, and what's up for debate is whether or not it is true, and the weight of the evidence presented in its favour. While the advantages of model-theoretic syntax have been loudly proclaimed since the seminal article of Pullum & Scholz (2001), when we look at the much-repeated arguments in favour of model-theoretic syntax, we find that they have little substance on closer examination. This should not surprise anyone skeptical about theoretical advantages unless they are backed up by concrete empirical advantages.

There is no evidence that model-theoretic syntax has the fundamental advantages over proof-theoretic syntax that Pullum & Scholz (2001) have been arguing. Indeed, I am skeptical about the most fundamental point that it is possible (or even useful) to define grammatical structures for natural language without using any form of structural induction.

There has been a lot of excellent research done under the banner of model-theoretic syntax, and I hope I will see a lot more of it in the future. But I believe that model-theoretic proselytism would do better to showcase the actual successes of model-theoretic syntax (Rogers 1998, Rogers 2003, Morawietz 2003) rather than claiming abstract, high-level advantages which somehow fail to materialise empirically.

References

Bangalore, S. & Joshi, A. (2011), *Supertagging: Using Complex Lexical Descriptions in Natural Language Processing*, MIT Press.

Barker, C. & Shan, C. (2014), *Continuations and Natural Language*, Oxford Studies in Theoretical Linguistics, Oxford University Press.

Blache, P. & Prost, J.-P. (2005), Gradience, constructions and constraint systems, *in* H. Christiansen, P. R. Skadhauge & J. Villadsen, eds, 'Constraint Solving and

[13] Uncountable structures are very strange, so we should be careful. A tree with an uncountable number of vertices need not have any uncountable branches nor have an uncountable number of descendants at any specific depth.

[14] When we abandon finite models in first-order logic, the upward Löwenheim-Skolem theorem will immediately give us uncountable models, i.e. fully grammatical sentences with uncountable structures.

Language Processing', Vol. 3438 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 74–89.
URL: *http://dx.doi.org/10.1007/11424574*$_5$

Blache, P. & Prost, J.-P. (2014), Model-theoretic syntax: property grammars, status and directions, *in* 'Constraints and Language', Cambridge Scholars Publishing, chapter 3, pp. 37–59.

Blackburn, P. & Meyer-Viol, W. (1994), 'Linguistics, logic and finite trees', *Logic Journal of the IGPL* **2**(1), 3–29.

Burel, G., Bury, G., Cauderlier, R., Delahaye, D., Halmagrand, P. & Hermant, O. (2020), 'First-order automated reasoning with theories: when deduction modulo theory meets practice', *Journal of Automated Reasoning* **64**(6), 1001–1050.

Costa-Florêncio, C. (2003), Learning Categorial Grammars, PhD thesis, Utrecht Institute of Linguistics OTS, Utrecht University.

Dowek, G., Hardin, T. & Kirchner, C. (2003), 'Theorem proving modulo', *Journal of Automated Reasoning* **31**(1), 33–72.

Duchier, D., Prost, J.-P. & Dao, T.-B.-H. (2009), A model-theoretic framework for grammaticality judgements, *in* 'International Conference on Formal Grammar', Springer, pp. 17–30.

Emms, M. (1993), Parsing with polymorphism, *in* 'Proceedings of the Sixth Conference of the European Association of Computational Linguistics', pp. 120–129.

Fitting, M. (1996), *First-order logic and automated theorem proving*, Graduate texts in computer science, 2 edn, Springer.

Gentzen, G. (1934), 'Untersuchungen über das logische Schließen', *Mathematische Zeitschrift* **39**, 176–210, 405–431.

Gold, E. M. (1967), 'Language identification in the limit', *Information and Control* **10**(5), 447–474.
URL: *https://www.sciencedirect.com/science/article/pii/S0019995867911655*

Hart, K. P. (2021), A critique of 'the vastness of natural languages' by Langendoen and Postal, Technical report, Technical University Delft.

Hodges, W. (1997), *A shorter model theory*, Cambridge university press.

Johnson, D. E. & Postal, P. M. (1980), *Arc pair grammar*, Princeton University Press.

Kogkalidis, K. & Moortgat, M. (2022), 'Geometry-aware supertagging with heterogeneous dynamic convolutions', *arXiv preprint arXiv:2203.12235* .

Kowalski, R. (1979), 'Algorithm = logic + control', *Communications of the ACM* **22**(7), 424–436.

Kubota, Y. & Levine, R. (2020), *Type-Logical Syntax*, MIT Press.

Kurtonina, N. & Moortgat, M. (1997), Structural control, *in* P. Blackburn & M. de Rijke, eds, 'Specifying Syntactic Structures', CSLI, Stanford, pp. 75–113.

Lambek, J. (1958), 'The mathematics of sentence structure', *American Mathematical Monthly* **65**, 154–170.

Langendoen, D. T. & Postal, P. M. (1984), *The vastness of natural language*, Basic Blackwell.

Libkin, L. (2004), *Elements of finite model theory*, Springer.

Mayer, M. C. & Pirri, F. (1993), 'First order abduction via tableau and sequent calculi', *Logic Journal of the IGPL* **1**(1), 99–117.

McCawley, J. D. (1968), 'Concerning the base component of a transformational grammar', *Foundations of language* pp. 243–269.

Meyer, A. R. (1975), Weak monadic second order theory of succesor is not elementary-recursive, *in* 'Logic colloquium', Springer, pp. 132–154.

Mirzapour, M. (2017), Finding missing categories in incomplete utterances, *in* 'TALN: Traitement Automatique des Langues Naturelles'.

Montague, R. (1974), The proper treatment of quantification in ordinary English, *in* R. Thomason, ed., 'Formal Philosophy. Selected Papers of Richard Montague', Yale University Press, New Haven.

Moortgat, M. (1988), *Categorial Investigations: Logical and Linguistic Aspects of The Lambek Calculus*, Foris, Dordrecht.

Moortgat, M. (2010), 'Typelogical grammar', Stanford Encyclopedia of Philosophy Website. `http://plato.stanford.edu/entries/typelogical-grammar/`.

Moot, R. (2021), 'Type-logical investigations: proof-theoretic, computational and linguistic aspects of modern type-logical grammars', Habilitation à Diriger des Recherches, Université de Montpellier.

Morawietz, F. (2003), *Two-step approaches to natural language formalisms*, Vol. 64 of *Studies in Generative Grammar*, Mouton de Gruyter.

Morrill, G., Valentín, O. & Fadda, M. (2011), 'The Displacement calculus', *Journal of Logic, Language and Information* **20**(1), 1–48.

Neeleman, A. (2013), 'Comments on Pullum', *Mind & Language* **28**(4), 522–531.

Palm, A. (2004), 'Model theoretic syntax and parsing: An application to temporal logic', *Electronic Notes in Theoretical Computer Science* **53**, 261–273.

Palm, A. (2007), Parsing complexity and model-theoretic syntax, *in* J. Rogers & S. Kepser, eds, 'Model-Theoretic Syntax at 10', pp. 29–38.

Pentus, M. (1995), 'Models for the lambek calculus', *Annals of Pure and Applied Logic* **75**(1-2), 179–213.

Pollard, C. & Sag, I. (1994), *Head-Driven Phrase Structure Grammar*, CSLI, Chicago.

Postal, P. M. (2010), *Edge-based clausal syntax: A study of (mostly) English object structure*, MIT Press.

Pullum, G. K. (2013*a*), 'The central question in comparative syntactic metatheory', *Mind & Language* **28**(4), 492–521.

Pullum, G. K. (2013*b*), 'Consigning phenomena to performance: A response to Neeleman', *Mind & language* **28**(4), 532–537.

Pullum, G. K. (2019), What grammars are, or ought to be, *in* S. Müller & P. Osenova, eds, 'Proceedings of the 26th International Conference on Head-Driven Phrase Structure Grammar, University of Bucharest', CSLI Publications, Stanford, CA, pp. 58–78.

Pullum, G. K. (2020), 'Theorizing about the syntax of human language: a radical alternative to generative formalisms', *Cadernos de Linguística* **1**(1), 1–33.

Pullum, G. K. & Scholz, B. C. (2001), On the distinction between model-theoretic and generative-enumerative syntactic frameworks, *in* 'International Conference on Logical Aspects of Computational Linguistics', Springer, pp. 17–43.

Pullum, G. K. & Scholz, B. C. (2005), Contrasting applications of logic in natural language syntactic description, *in* 'Logic, methodology and philosophy of science: Proceedings of the twelfth international congress', King's College Publications London, pp. 481–503.

Robertson, N. & Seymour, P. D. (2004), 'Graph minors XX. Wagner's conjecture', *Journal of Combinatorial Theory* **92**(2), 325–357.

Rogers, J. (1998), *A Descriptive Approach to Language-Theoretic Complexity*, CSLI.

Rogers, J. (2003), 'wMSO theories as grammar formalisms', *Theoretical Computer Science* **293**(2), 291–320.

Sag, I., Wasow, T. & Bender, E. (1999), *Syntactic theory: A formal introduction*, Vol. 92, Center for the Study of Language and Information Stanford, CA.

Shelah, S. (1978), *Classification theory and the number of nonisomorphic models*, Vol. 92 of *Studies in Logic and the Foundations of Mathematics*, North-Holland Publishing.

Shinohara, T. (1990), Inductive inference from positive data is powerful, *in* 'Proceedings of the third annual workshop on Computational learning theory', pp. 97–110.

A Parsing model-theoretic grammars

Parsing is one of the most important ways to verify the predictions of our grammar: we input a sentence as a sequence of words, and our parser outputs the different structures our grammar predicts for this sentence.

When using model-theoretic grammars, parsing can still be done proof-theoretically. As a simple example, suppose we want to parse "John sleeps". The first step is to code the input string as a logical formula. For example, we can use the formula F below.

$$F = \exists xy.John(x) \wedge sleeps(y) \wedge x < y \wedge \neg\exists z.z < x \wedge \neg\exists zy < z$$

The relation '$<$' denotes the linear precedence relation between words, so F expresses that there are two nodes x (labelled 'John') and y (labelled 'sleeps') with x preceding y, x the first word, and y the last word. The formula F then codes, in a simple way, the linear order on the words of the input sentence.

We can then combine G with the simple theory $\mathcal{T}$ consisting of axioms 17-19 below.

$$\forall y.\text{John}(y) \Rightarrow \exists x.\text{parent}(x, y) \wedge \text{NP}(x) \tag{17}$$

$$\forall y.\text{sleeps}(y) \Rightarrow \exists x.\text{parent}(x, y) \wedge \text{VP}(x) \tag{18}$$

$$\forall yz.\text{NP}(y) \wedge \text{VP}(z) \wedge y < z \Rightarrow \exists x.\text{parent}(z, x) \wedge \text{parent}(z, y) \wedge \text{S}(x) \tag{19}$$

The relation parent(x, y) is true when node x is the (immediate) parent of node y. Axiom 17 then specifies nodes labelled 'John' have a parent node which is a noun phrase. Some additional axioms are necessary to ensure that nodes are distinct from their parents, and that parents are unique. Axiom 18 specifies that 'sleeps' has a verb phrase parent. Finally, axiom 19 specifies that when x is a noun phrase and y a verb phrase, with x preceding y, then they both have a sentence node as parent.

To parse this sentence, we can then either take the model-building road, first constructing all possible 1-vertex models satisfying $\mathcal{T} \wedge F$, then all 2-vertex models, etc., until we find a structure with yield 'John sleeps'. To show the sentence is ungrammatical, we do the same for $\mathcal{T} \wedge \neg F$. If our theory has the finite model property, we will either find a model of $\mathcal{T} \wedge F$, indicating the sentence is grammatical, or a model of $\mathcal{T} \wedge \neg F$, indicating the sentence is ungrammatical.

We can also parse proof-theoretically. Given this theory $\mathcal{T}$ and the formula F encoding our input sentence, a tableau prover (Fitting 1996) would produce the following schematic tableau for $\mathcal{T} \wedge F$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\neg\text{NP}(c)} \quad \dfrac{}{\neg\text{VP}(d)} \quad \dfrac{}{\neg(c < d)}}{\neg(\text{NP}(c) \wedge \text{VP}(d) \wedge c < d)} \quad \dfrac{\text{parent}(e, c), \text{parent}(e, d), \text{S}(e), \ldots}{\exists x.\text{parent}(x, y) \wedge \text{parent}(x, z) \wedge \text{S}(x)}}{\text{parent}(d, b), \text{VP}(d), \ldots} \quad \dfrac{}{\neg\text{sleeps}(b)}}{\exists x.\text{parent}(x, y) \wedge \text{VP}(x), \ldots} \quad \dfrac{\text{parent}(c, a), \text{NP}(c), \ldots}{\exists x.\text{parent}(x, a) \wedge \text{NP}(x), \ldots}}{\neg\text{John}(a) \quad \text{John}(a), \text{sleeps}(b), a < b, \ldots}}{\exists xy.John(x) \wedge sleeps(y) \wedge x < y \wedge \ldots}$$

Reading the proof from the root upwards, we first remove the existential quantifiers, replacing variables x and y by constants a and b, then unfold the conjunctions into their subformulas. We can then expand axiom 17, instantiating y to a. In a tableau proof, $A \Rightarrow B$ is treated as $\neg A \vee B$, so the implications produces two branches, the left one with $\neg$John(a), which we can close because John(a) appears in the branch). We unfold the right branch, replacing the existentially quantified x by a new variables c and removing the conjunctions. We continue with axioms 18 and 19 until no further rules can be applied[15].

The tableau cannot be closed, which means there is no proof of $\neg(\mathcal{T} \wedge G)$, and therefore that $\mathcal{T} \wedge G$ has a model. One of the attractive properties of tableau proof search is that when we fail to find a proof, we can use the constructed tableau to recover a countermodel. In our case, a countermodel of $\neg(\mathcal{T} \wedge G)$ is a model of $\mathcal{T} \wedge G$ and therefore a successful syntactic analysis of G.

To recover this model, we can simply collect the positive atomic atomic formulas along any open branch, with different open branches corresponding to different models. These positive atomic formulas are enough to construct a model (Hodges 1997). In the current case these formulas are:

$$\begin{aligned}
&\text{John}(a), \text{sleeps}(b),\\
&\text{NP}(c), \text{VP}(d), \text{S}(e)\\
&a < b, c < d\\
&\text{parent}(c, a), \text{parent}(d, b),\\
&\text{parent}(e, c), \text{parent}(e, d),
\end{aligned}$$

This gives us a model with domain $\{a, b, c, d, e\}$, the same signature σ as the logic, and the following interpretation function.

$$\begin{aligned}
I(\text{John}) &= \{a\}\\
I(\text{sleeps}) &= \{b\}\\
I(\text{NP}) &= \{c\}\\
I(\text{VP}) &= \{d\}\\
I(\text{S}) &= \{e\}\\
I(<) &= \{\langle a, b\rangle, \langle c, d\rangle\}\\
I(\text{parent}) &= \{\langle c, a\rangle, \langle d, b\rangle, \langle e, c\rangle, \langle e, d\rangle\}
\end{aligned}$$

[15] We need the tree axioms relating the 'parent' and '<' relations to close the $\neg(c < d)$ branch.