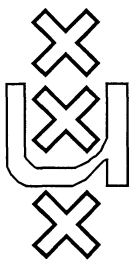
Institute for Language, Logic and Information

LOGICAL SYNTAX

Johan van Benthem

ITLI Prepublication Series 86-06



University of Amsterdam

1. Introduction

The syntax of formal languages in logic is usually considered a rather boring subject, to be restricted to a few hurried paragraphs on 'matters of notation'. Undue attention to points of logical syntax calls up images from the early days of logic in this century, with scholars engaged in remorseless formalization and orgies of Sheffer strokes. And the current neglect may even be reinforced by the rising interest in natural languages, whose more lively logical structure is being brought to light these days.

In this paper, we try to modify this verdict, arguing by means of various examples, that logical syntax can be as interesting as (say) logical semantics. Our first illustration is one of grammatical complexity. Formal languages are usually context-free, but complexity will be shown to increase, once apparently innocuous formal conditions are added. This phenomenon may be contrasted with recent discussions of the complexity of natural languages, where estimates tend rather to decrease (from context-free to regular).

The second illustration concerns natural fragments of formal languages. A formalism like predicate logic can be viewed as the result of letting certain linguistic constructions (negation, quantification, etcetera) go on ad infinitum. Looking back, one can cut it up into ascending hierarchies of fragments of growing complexity (measured, e.g., by depth of nesting for quantifiers). But, what may be counted as a natural fragment is a highly 'intensional' notion, depending on intended applications and viewpoints. We shall present some linguistically motivated examples of cutting the cake of predicate logic, which generate new open questions about this reputedly 'totally known' language.

Next, logical semantics itself is not unaffected by changes in syntactic perspective. We consider interpretation strategies for formal languages, using notions from Automata Theory to implement recent ideas about 'radical' or 'progressive' interpretation. In particular, it turns out that such interpretation is already possible on regular languages where (scope) disambiguation has not yet fully taken place. In fact, the general thesis defended is that non-ambiguity is a result, rather than a precondition of succesful interpretation.

Finally, some repercussions of all this are discussed for the notion of inference. Some results are given on the complexity of logical theorems, as well

as proofs. (The latter can be viewed as pieces of text, rather than single assertions.) But also, the issue of the proper syntactic format for doing inference is considered: the outcome being that there is no single level where inference must take place.

Thus, a more subversive purpose of this paper becomes visible. The syntax, semantics and logic of formal languages are often presented in a 'unitary' manner, suggesting that, e.g., natural language semantics must do likewise: creating one all-purpose level of logical form where interpretation and inference take place (in the proper compositional spirit). We hope to 'loosen up' this picture, because it is already too strict for logic itself.

2. Grammatical Complexity

One of the most famous and persuasive parts of the linguistic classic Chomsky 1957 is its investigation of the complexity of natural language. A hierarchy of possible grammatical formats is presented, ascending in strength:

regular < context-free < context-sensitive < type 0 .

It is then argued that the first two levels are inadequate, by reference to specific linguistic constructions beyond their reach: natural languages are at least context-sensitive. This conclusion has become accepted, by and large, throughout the linguistic community since.

In recent years, however, the case has been re-opened. Many of the early arguments for non-context-freeness have been shown defective, and estimates for natural language complexity are being lowered accordingly. A good survey of this trend is Gazdar and Pullum 1985. Indeed, a strong case can be made for a purely regular character of the major syntactic constructions. These turn out to involve only so-called 'tail recursion', that can be mimicked already by Kleene iteration (see Ejerhed and Church 1982). So, in the end, natural languages might be at most context-free, with a regular backbone.

In this light, a comparison becomes of interest with the complexity of formal languages, usually regarded as paradigms of simplicity. The latter are easily seen to be at least context-free: and sometimes even more complex than that.

Formal language complexity

First, consider a propositional language, with alphabet $\{p,\neg,\wedge,),(\}$ and the usual formation rules for well-formed fomulas. It has a context-free grammar:

$$S \Rightarrow p \ , \quad S \Rightarrow \neg S \ , \quad S \Rightarrow (S \wedge S) \ .$$

(To create infinitely many proposition letters with a finite alphabet, one might

use instead

$$S \Rightarrow A\ , \quad A \Rightarrow p\ , \quad A \Rightarrow A'\ .)$$

No simpler grammar suffices:

PROPOSITION. Propositional formulas are not a regular set of expressions.

PROOF. By an application of the Pumping Lemma for regular languages (see Hopfroft and Ullman 1979). Let n be the iteration value in that lemma. Consider the formula

$$(p \wedge (p \wedge \ldots \wedge (p \wedge p) \quad [n \text{ brackets}])\ .$$

There must be some non-empty part of the final string of brackets which can be duplicated without disturbing recognition. But, the resulting expression is not a well-formed formula, as it lacks the proper balance of brackets. □

So, propositional logic requires one coordination at a distance: something which needs precisely context-free rewriting rules.

With predicate logic, however, one seems to require an additional coordination, viz. that between predicate letters (with certain arities) and corresponding numbers of arguments. With a finite set of predicate letters, this can be handled still by enumeration of all possible atomic types, but we are interested in the general case of P^i_j ('the j-th predicate of arity i') . In principle, two coordinations seem to go beyond context-free grammars - but in this case, one is still available.

EXAMPLE. Let the language have an alphabet $\{P,x,',\neg,\wedge,\forall,),(\}$. Informally, variables will be $x,x',x'',\ldots,$ and predicate letters all combinations $'\ldots'P'\ldots'$, with the arity indicated on the left. Here is a recognizing grammar:

$$V \Rightarrow x\ , \qquad V \Rightarrow V' \qquad \text{(individual variables)}$$

$$\Pi \Rightarrow p\ , \qquad \Pi \Rightarrow \Pi' \qquad \text{(proposition letters)}$$

$$A \Rightarrow \Pi\ , \qquad A \Rightarrow 'AV \qquad \text{(atomic formulas)}$$

$$S \Rightarrow A\ , \quad S \Rightarrow \neg S\ , \quad S \Rightarrow (S \wedge S)\ , \quad S \Rightarrow \forall\ V\ S \quad \text{(formulas)}.$$

Note the trick with the arity count. E.g., $''Pxx'$ is constructed by the stages P , $'Px$, $''Pxx'$.

Non-vacuous binding

A more natural class of predicate-logical formulas, however, (and one intended by many users in the first place) has an additional restriction: every quantifier occurrence is to bind at least one variable. Such simple variations may already increase complexity:

PROPOSITION. The well-formed predicate-logical formulas without vacuous quantification do not form a context-free set.

PROOF. By an application of Ogden's Lemma (Hopcroft and Ullman 1979, lemma 6.2), which reads as follows

"Let L be a context-free language. Then there is a constant n such that, if z is any word in L, and we mark any n or more positions of z 'distinguished', then we can write $z = u\,v\,w\,s\,y$, such that
1) v and s together have at least one distinguished position,
2) for all $i \geq 0$, $u\,v^i\,w\,s^i\,y$ is in L".

Consider the following (non-vacuously quantified) formula:

$$\forall x^1 \forall x^2 \ldots \forall x^n\; {}^n P\, x^1\, x^2 \ldots x^n \;;$$

where $x^1 = x'$, $x^2 = x''$, etcetera.

Ordinary pumping may be admissible here, as in

$$\forall x' \forall x'' \; {}''P\, x'\, x'' \Rightarrow \forall x' \forall x'' \; {}'''P\, x\, x'\, x'' .$$

But, by marking all universal quantifier symbols $\forall$, one can make sure that at least one quantifier must be involved in the Ogden iteration: and this always produces at least one vacuous quantifier occurrence.

Examples:

i) $\underline{\forall x^1} \forall x^2 \ldots \Rightarrow \forall x^1 \forall x^1 \forall x^2 \ldots$

ii) $\forall \underline{x^1 \forall x^2} \ldots \Rightarrow \forall x^1 \forall x^1 \forall x^2 \ldots$

iii) $\forall x' \forall x'' \; \underline{\forall x'''} \ldots \Rightarrow \forall x^1 \forall x^2 \forall x^5$ (!) $\forall x^3 \ldots$,

but no 'matching' duplication is possible taking care of both the required additional argument for P and the increased arity count. □

REMARK. This result appeared without proof in Van Benthem 1977. A prompt by Hugues Leblanc led to writing up the above argument in 1985. In the meantime, Theo Janssen has pointed at Partee and Marsh 1985, which also gives a proof of the preceding result - as well as a further investigation just how far these additional constraints push us up the language hierarchy.

The Partee-Marsh argument also works for a fixed finite set of predicates without the above arity mechanism. But, with one further restriction, non-vacuously quantified predicate logic does become decidable: viz. when only finitely many individual variables can occur. Let $x_1, \ldots, x_n$ be those variables. Then create auxiliary symbols S_X for each set X of variables ('the formulas in which exactly the variables in X occur freely'). Now, binding can be encoded in the rewriting rules. E.g., with one binary predicate letter P, there will be atomic clauses such as

$$S_{\{x_1, x_2\}} \Rightarrow Px_1x_2 \;,\; Px_2x_1 \;.$$

Further administration can then be done as follows:

$$S_{X_1 \cup X_2} \Rightarrow (S_{X_1} \wedge S_{X_2}) \ ,$$

$$S_{X-\{x_1\}} \Rightarrow \forall x_1 \, S_X \qquad \text{(provided that } x_1 \in X\text{)} .$$

This last restriction may not be unreasonable for natural languages, whose supply of anaphoric pronouns is limited - but it is less reasonable for formal languages.

Discussion

It is tempting to conclude from the above observations that formal languages can be more complex than natural languages. But there is something perverse in this conclusion. After all, the language of predicate logic is learnt in one session, with or without the proviso on vacuity - whereas natural languages, regular or not, require a good deal more effort. Does this point at a defect in the conceptualization of complexity, through levels in the hierarchy of grammar ? Or could it be that the main problem in language learning is the mastery of 'width' rather than 'depth': i.e. size rather than intricacy of the grammar ?

Another reason for caution concerns the possible dependence of the above results on choices of notation. (For instance, the stroke notation used earlier for predicate logic was certainly non-standard - even if not unmotivated.) Usually, it is enough to know that different notations are effectively equivalent. But, when the fine-structure of complexity is at issue, the complexity of notational transformations itself becomes important. For instance, rewriting propositional formulas in $\neg, \wedge$ to $\sim, \&$ form takes only a finite state transducer (in the sense of Aho and Ullmann 1972), replacing symbols as they occur. But e.g., rewriting such formulas into Polish forms requires memory space, and in fact a push-down transducer. And one can easily find yet other 'equivalent' notations whose conversion requires further Turing machine power. This is evidently an issue for further investigation.

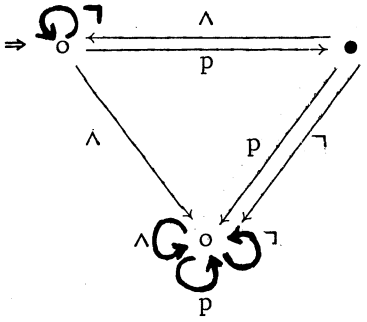
Appendix: Regular languages

The location of the boundary regular/context-free will be important again in Section 4 on interpretation. Therefore, we add some preliminary observations on this topic here.

First, the context-free complexity of propositional formulas is due entirely to what is essentially a semantically inspired auxiliary construction: their disambiguating bracket structure:

PROPOSITION. The propositional formulas with their brackets removed form a regular language.

PROOF. Here is a finite state recognizer for this language:



●: accept

o: reject

□

Now, bracketing structure is (almost) absent in natural language - and indeed, one can often 'regularize' prima facie context-free grammars. For instance, the conditional sentences produced by the two rules

$$S \Rightarrow p\ , \qquad S \Rightarrow S\ \underline{\text{if}}\ S$$

can also be described by the Kleene notation

$$p\ \ (\underline{\text{if}}\ \ p)^* .$$

Or, consider the following recursive rewrite rules generating noun phrases:

$$NP \Rightarrow \text{Det}\ N\ , \qquad N \Rightarrow N\ PP\ , \qquad PP \Rightarrow \text{Prep}\ NP$$

$$\text{Det} \Rightarrow \underline{a}\ , \underline{no}\ , \qquad N \Rightarrow \underline{dog}\ , \underline{boss} \qquad \text{Prep} \Rightarrow \underline{with}\ , \underline{without}\ .$$

An equivalent regular version is

$$\left\{\begin{matrix}\underline{a}\\ \underline{no}\end{matrix}\right\}\left\{\begin{matrix}\underline{dog}\\ \underline{boss}\end{matrix}\right\}\left(\left\{\begin{matrix}\underline{with}\\ \underline{without}\end{matrix}\right\}\left\{\begin{matrix}\underline{a}\\ \underline{no}\end{matrix}\right\}\left\{\begin{matrix}\underline{dog}\\ \underline{boss}\end{matrix}\right\}\right)^* .$$

Of course, these pairs of grammars are only weakly equivalent: as they assign quite different constituent structures. So, the question becomes if one can make semantic sense of the regular format. After all, once 'unitary' prejudices are given up, one need no longer assume that a simplest syntactic format for a language is necessarily also the best format for its semantic interpretation. We shall consider interpretation on regular syntax in Section 4 below.

As a final point, there might also be room for both types of description simultaneously. Say, with some context-free 'competence grammar' and some coarser regular 'performance grammar' modelling the blurring of recursive structure which we experience after a few rounds. In this connection, one would want to have some natural notion of 'regular closure' of any given context-free system.

3. Natural Fragments

The examples in Section 2 also suggest a more 'structured' view of the language of predicate logic, as a web of various fragments. There are many principles of subvision here, for instance by the arity of predicates ('monadic' predicate logic versus polyadic versions), or the depth of nestings of quantifiers (a useful measure for model-theoretic complexity). In particular, many fragments have been isolated in the search for decidable subsystems of full predicate logic (see Dreben and Goldfarb 1979).

In addition to these more purely logically motivated fragments, analogies with natural languages suggest further subdivisions. In particular, it has often been noticed that natural languages are less liberal with (bound) variables, preferring constructions circumventing overt display of pronouns. Compare, e.g.,

"every house has a price" with

$\forall x\ (Hx \rightarrow \exists y(Py \wedge \mathcal{R}\, xy))$.

Or also the use of Passive in stating the inference pattern

$\exists x\ \forall y\ K\ xy \Rightarrow \forall y\ \exists x\ \check{K}\ xy$:

"someone knows everybody" implies

"everybody is known by someone" , rather than

"for everybody there is someone who knows him" .

(The latter point is made in Dummett 1973.) Such phenomena invite a study of variable-free notations for predicate logic - as has been done in Quine 1971. In Quine's system, predicate-logical formulas express predicates, with proper 'bindings' between argument positions handled by operators of 'identification' and 'permutation'. Such a perspective suggests entirely different fragments of predicate logic, depending on which operators are allowed. Correspondingly, the issue can be raised if some of these give us 'large' decidable fragments of predicate-logical inference in some practical sense (cf. Bacon 1985).

REMARK. Incidentally, variable-free or variable-poor notations also occur inside mathematics. For instance, the common notation for composition is a way of avoiding variables - and Recursion Theory even has a complete variable-free notation for recursive functions, involving operators for composition, recursion and minimalization. It would be of interest to compare the necessary 'tricks' here, including so-called 'projection functions', with those of Quine.

Also more generally, current systems of 'flexible' categorial grammar for natural language can be viewed as variable-free mechanisms for encoding meanings. For instance, applying a rule of Functional Composition to recognize a constituent such as $[\text{knows everybody}]_{NP}$ has the following semantic effect:

types: $(e,(e,t)) + ((e,t),t) \Rightarrow (e,t)$

meaning: $\lambda x_e \cdot \text{everybody}_{(e,(e,t))} (\lambda y_e \cdot \text{knows}_{(e,(e,t))} (y_e)(x_e))$.

See van Benthem 1986, chapter 7, for a systematic account of these meaning instructions. The basic system here (correspondary to the so-called Lambek Calculus) has the following strong restriction:

'every lambda operator binds exactly one occurrence of a free variable'.

This leads to an obvious numerical hierarchy, of a rather unusual kind.

As an illustration, consider single bond predicate logic, having such formulas as

$\forall x\, Ax$, $\exists x \forall y\, Rxy$, $\exists x\, Ax \rightarrow \forall x \forall y\, Ryx$.

Non-formulas are $\forall x\, Rxx$, $\exists x\, (Ax \wedge Bx)$. With a fixed finite non-logical vocabulary , this fragment, though admitting arbitrary depth of quantifier nesting, is rather simple:

PROPOSITION. Single-bond predicate logic is decidable.

PROOF. First, a normal form result is needed.

LEMMA. Every single-bond sentence is logically equivalent to a positive Boolean compound of basic single-bond formulas of the form

< sequence of quantifiers - (negation of) atomic formula >.

PROOF. First, by well-known equivalences, push negations inward until they reach atomic subformulas. This does not affect the single-bond constraint. Then, starting with innermost quantifiers, drive these inward too, using the well-known prenex equivalences. Note that every connective $\wedge, \vee$ will be crossed, since, e.g., $\exists x(\phi \vee \psi)$ must be equivalent to either $\exists x\phi \vee \psi$ or $\phi \vee \exists x\psi$ (x occurs freely in ϕ or in ψ , but not in both). □

Next, validity can be computed as follows. Modulo logical equivalence, there are only finitely many basic formulas. Now, the Boolean compound of the lemma will be valid if and only if some finite number of disjunctions of basic formulas is valid. So, to answer the general question of validity, it suffices to have the validity of these (finitely many) possible disjunctions precomputed. □

REMARK. Choosing fragments may have unexpected repercursions. For instance, we have to check if formerly acceptable rules of inference stay within the area. E.g., an equivalence which would not have been admissible in the above arguments is Distributivity:

$Ax \wedge (By \vee Cz) \leftrightarrow (Ax \wedge By) \vee (Ax \wedge Cz)$

Of course, the single-bond restriction is too severe for natural language in general. A restriction which has been proposed, however, for empirical reasons, is that of bounded dependencies. It seems that bindings in most natural languages cannot cross arbitrary layers of operators in a sentence. Again, predicate logic is a good testing ground for such phenomena (a suggestion which is due to Frank Heny).

Let the bound-1 fragment have only formulas in which any quantifier occurrence binds only occurrences of its variable x for which it is the first quantifier in whose scope such an x occurs. For similar quantifiers, this condition holds by convention:

$$\exists x\ (Ax \wedge \exists x\ Bx) \quad .$$

But now, one also forbids such crossings as the following:

$$\exists x\ (Ax \wedge \exists y\ Rxy) \quad .$$

On the other hand, arbitrary width is allowed:

$$\exists x\ (Ax \wedge Bx \wedge Cx \wedge \ldots) \quad .$$

PROPOSITION. All bound-1 formulas are equivalent with formulas of quantifier depth 1, and vice versa.

PROOF. Formulas of quantifier depth 1 are automatically bound-1. Conversely, one can remove possible nestings of quantifiers by the prenex equivalences. Example: $\exists x\ (Ax \vee \exists y\ R\ yz)$ is equivalent with $\exists x\ Ax \vee \exists y\ Ryz$. Note that x cannot occur freely in the scope of disjuncts with quantifiers in front. □

For greater depths of dependency, however, this simple correspondence breaks down:

EXAMPLE. The following formula is bound-2:

$$\exists x\ \forall y\ (Ryx \vee \exists z\ Rzy) \quad .$$

But, its quantifier depth is 3 - and this cannot be reduced to 2, witness the following situation. The two graphs G_1 , G_2 are '2-equivalent' in the sense of Ehrenfeucht-Fraïsse games (player II has a winning strategy for the case of two rounds; see, e.g., Hodges 1983 for definitions):

It follows that G_1, G_2 validate precisely the same sentences in R of quantifier depth up to 2. On the other hand, G_1 validates the above formula (letting x be one of the right end-points), while G_2 does not. □

So, we have all kinds of questions for such a new division of predicate logic into fragments. Is there an independent semantic motivation for the levels bound-1, bound-2, etcetera (perhaps, in terms of different games)? Are the logics of these successive fragments decidable? Such rather immediate open questions suggest that, contrary to prevailing opinions, even such a familiar system as predicate logic remains largely terra incognita.

4. Strategies of Interpretation

Lowering syntactic complexity by adopting a different grammar for a language may have a cost too: perhaps, interpretation becomes more difficult (or even impossible) - and the same could be true for inference. In particular, such questions can be raised at the border-line of regular versus context-free grammars, encountered in Section 2.

As a preliminary observation, not even context-free grammars need be easily interpretable. Of course, by their known equivalence with categorial grammars, the latter's semantics can be borrowed - but this need not produce faithful Fregean rule-by-rule interpretation. For instance, there just is no semantics for a rule like $V \Rightarrow V'$ in the earlier grammar for predicate logic, and at best a rather fanciful one for the atomic formation rule $A \Rightarrow 'AV$. For further reflections on interpreting context-free syntax, see Suppes 1976, Janssen 1983.

Interpreting regular syntax

The question here is rather if we can still interpret regular syntax in some plausible fashion. For instance, the earlier grammar $p \bullet (\text{if } p)^*$ will produce flat conditional sentences with all conditional clauses treated as adjoints, on a par. Now what do we mean, intuitively, by a sentence such as

"John cries if Mary laughs if Porky grunts" ?

There seems to be a tendency toward right association here:

(J if (M if P)) .

And in any case, we can study certain systematic strategies of interpretation, generating such preferences. (A 'deeper' treatment might consist in providing a more conjunctive co-ordinating interpretation for these iterated clauses, rather than the usual subordinating one arising from obligatory bracketing. This would be more in line with the methodology of Barwise and Perry 1983.)

There are several reasons for exploring interpretation on regular syntax. First, the usual bracketing structures seem somehow imposed, rather than being really there. In a sense, they are one step on the way to regimented logical form. But then, they might even give too little: as the 'worst case' representation level for natural language sentences needs not only scope disambiguation, but also anaphoric linking (and perhaps further decisions as well). By contrast, one might want to study at which level of representation one can do already what kind of interpretation - without too rigid preconceptions about the proper methodological order of Syntax and Semantics.

As suggested above, we can formulate various interpretative strategies on regular syntax, such as

I 'Give each successive operator the widest possible scope',

or rather

II 'Give each successive operator the smallest possible scope'.

The very formulation here presupposes some order of processing, and in fact, the literature contains various suggestions to this effect. For instance, Hintikka has pointed at a 'Progression Principle' in interpreting sentences (see Hintikka and Carlson 1979), and Barwise and Perry 1983 stress 'radical interpretation', from left to right. (Of course, the 'resultant force' of understanding a text is necessarily from left to right - but the question is how strictly this order must be respected within sentences.) The model suggested by these views is that of standard automata, traversing a string of symbols while (perhaps) using a working memory, to build up an interpretation. How versatile we are as human readers may be judged from our position, when viewed as being such automata, in the usual Machine Hierarchy. (For a general development of 'semantic automata' in a congenial, though not identical spirit, see van Benthem 1986, chapter 8.) E.g., are we just finite state evaluators - or can we hope to be Turing machines?

This perspective enables us to use results from Automata Theory. For instance, the well-known equivalence proof between context-free grammars and push-down store automata may now be viewed as a 'linearization' result. It tells us which assumptions need to be made about a linear processor recognizing, or evaluating, a given context-free language. (In this light, the 'regularization' mentioned in Section 2 might result from restricting the size of the push-down stack to some empirical limit.)

Propositional formulas

A simple, but instructive test case for ideas of this kind is ordinary propositional logic. Which kinds of machine are needed for its evaluation?

First, a representation is needed of the interpretative information. We have a string of symbols, but also a valuation assigning truth values to proposition letters - and the problem is to compute the proper truth value for the string. Calling the valuation may be viewed as an atomic subroutine, producing values 0,1 in the place of proposition letters. Thus we need machines operating on strings in which the latter replacement has been made throughout, which produce truth values themselves. I.e., in principle, they are to be (rather simple) transducers.

The most elementary case is that of finite state evaluators, being finite state machines with a 0 or 1 value attached to certain states, and 3 ('non-interpreted') to others. Not surprisingly, such evaluators are inadequate for ordinary bracketed syntax. For instance, as in the argument of Section 2, they will make 'errors of the first kind': mistakenly evaluating non-formulas. More seriously, however, they will also make 'errors of the second kind': mis-evaluating well-formed formulas. (The second kind of error does not arise in the usual case of syntactic recognition.)

PROPOSITION. No finite state evaluator evaluates all propositional forms correctly.

PROOF. (For greater perspicuity, Polish notation is used here.) Let n be the number of states in the putative evaluator. Consider the following formula:

$$\wedge\ [n \text{ times}]\ \wedge\ \vee\ [n \text{ times}]\ \vee\ 0[n+1 \text{ times}]\ 0\ 1[n \text{ times}]\ 1\ ,$$

whose truth value equals 0. In evaluating this string, the machine will loop in the $\vee$-part (say, with cycle $k \leqq n$), and also in the 1-part (say, with $1 \leqq n$). Now, consider the above formula with $k \cdot 1$ copies of $\vee$ interpolated, and as many copies of 1. The result is again a formula, with truth value 1 this time. (The total $\vee$-subformula gets value 1 now, as its initial $\vee$ will capture an argument 1.) But the evaluator will produce the same outcome as before: being 0. □

From now on, we will use ordinary notation again (leaving the possible effects of notational variance for some future occasion - as in Section 2).

A suitable machine for correctly interpreting propositional formulas is a simple variant of their push-down store recognizer. The latter will be constructed first.

EXAMPLE. The machine has three states:

1 ('going to read a formula'),

2 ('have read a formula'),

3 ('failure').

Its stack alphabet is $\{\not<,\not\in\}$, where $\not\in$ symbols indicate that the main connective for this bracket is still to be located.

Here are the crucial transition rules (stated for binary $\wedge$ only):

state	stack	symbol read	⇒ new state	action on stack
1	empty	$\wedge$	3	none
		(	1	store $\not\in$
		)	3	none
		p	2	none
1	$\not\in$	$\wedge$	3	none
		(	1	store $\not\in$
		)	3	none
		p	2	none
1	$\not<$	$\wedge$	3	none
		(	1	store $\not\in$
		)	3	none
		p	2	none
2	empty	any symbol	3	none
2	$\not<$	$\wedge$	3	none
		(	3	none
		)	2	erase top $\not<$
		p	3	
2	$\not\in$	$\wedge$	1	replace $\not\in$ by $\not<$
		(	3	none
		)	3	none
		p	3	none

To understand how this works, it is useful to write out a full trace of recognition, e.g., for the string

$$(((p \wedge q) \wedge r) \wedge (p \wedge r)) :$$

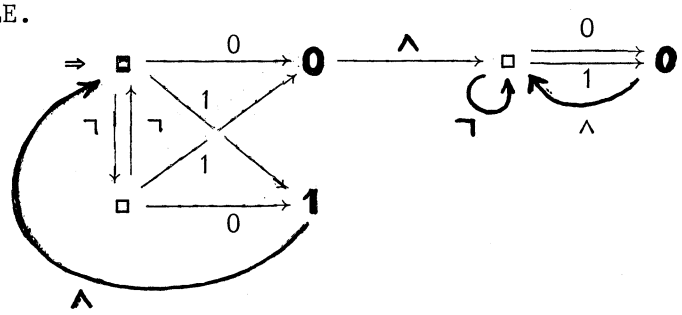
	read	state	stack									
1	-	: 1 ,	-	7	q	: 2 ,	$\not\in\not\in\not<$	13	(	: 1 ,	$\not<\not\in$
2	(	: 1 ,	$\not\in$	8	)	: 2 ,	$\not\in\not\in$	14	p	: 2 ,	$\not<\not\in$
3	(	: 1 ,	$\not\in\not\in$	9	$\wedge$	: 1 ,	$\not\in\not\in$	15	$\wedge$	: 1 ,	$\not<\not<$
4	(	: 1 ,	$\not\in\not\in\not\in$	10	r	: 2 ,	$\not\in\not<$	16	r	: 2 ,	$\not<\not<$
5	p	: 2 ,	$\not\in\not\in\not\in$	11	)	: 2 ,	$\not\in$	17	)	: 2 ,	$\not<$
6	$\wedge$	: 1 ,	$\not\in\not\in\not<$	12	$\wedge$	: 1 ,	$\not<$	18	)	: 2 ,	-

Now, to obtain an evaluator from this recognizer, one has to multiply states, encoding whether a true or a false formula has been read. Concurrently, one can extend the stack alphabet with symbols $\dagger_0$, $\dagger_1$, for those cases where the value of a first component already determines that of the whole formula (i.e., $0 \wedge \phi$, $1 \vee \phi$). Details are omitted here.

Finite state evaluators

To explore possible interpretation of regular syntax, one can look at some concrete examples.

EXAMPLE.



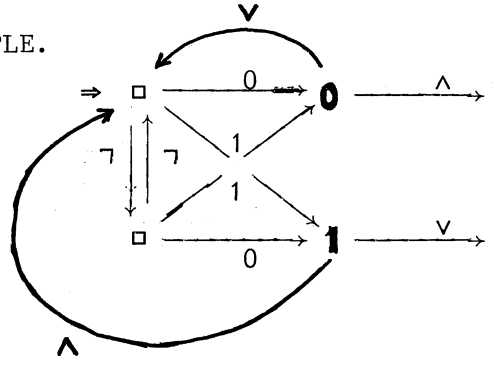
This finite state machine has 'neutral' states □, and evaluating states **0** and **1**. Moreover, there is a failure state (not indicated here) where all omitted transition arrows end. It will produce truth values for all propositional formulas with brackets removed. Its action may be described as follows:

'small scope for ¬, wide scope for ∧'.

E.g., $\neg p \wedge q$ is evaluated as $(\neg p) \wedge q$, and $p \wedge \neg q \wedge r$ as $(p \wedge ((\neg q) \wedge r))$. (Note how initial truth values determine the evaluation of the first ∧ encountered. If it is 0, then the value is 0, provided that the remaining string is a formula. (This is the function of the right-most subroutine.) If the initial value is 1, then the procedure starts again.)

Actually, 'wide scope' is not an appropriate term here, as <u>any</u> way of handling the conjunction will produce the same value (because of Associativity). Differences will only show up when two binary connectives interact, as in the following illustration.

EXAMPLE.



"check if formula: then **0**"

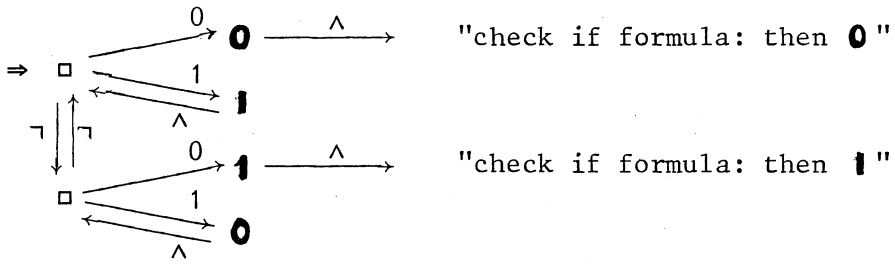
"check if formula: then **1**"

This machine will evaluate, e.g., $p \wedge q \vee r$ to $p \wedge (q \vee r)$, rather than $(p \wedge q) \vee r$.

Evidently, many variations are possible for these machines. A principled choice might be to select evaluators with a minimal number of states, seeing what strategies these incorporate.

For instance, evaluating with wide scope for $\neg$ will be more costly:

EXAMPLE.



This machine will interpret, e.g., $\neg p \wedge q$ as $\neg(p \wedge q)$. (Note the two dominating 'positive'/'negative' states in front.)

There are many further possibilities here for experimentation. We conclude with two comments. One is that there might be just one basic strategy of interpretation, with deviations triggered by syntactic indicators ("both ... and ...", "if ... then ...", "if ... , ...") or phonological ones (emphasis, pauses). These can be processed as indicators that some special subroutine is to be called - still at the level of finite state automata.

The other point is that we may still be ensnared by traditional notions, which are now obsolete -- and 'scope' itself is one example. This point is made in Barwise and Perry 1983; but we can now make it precise. The above evaluators start interpreting before full syntactic structure is known: evaluation and syntactic construction proceed in parallel. (This too is central in current 'Stanford philosophy', witness also Fenstad et al. 1985.) Therefore, it makes less sense to talk about scope. For instance, when evaluating a conjunction $p \wedge \phi$, the procedure depends crucially on the value of p. If it is 0, the internal structure of ϕ is largely irrelevant for the outcome (being 0 anyway). (A similar point, concerning 'sequential interpretation', has been made in van der Sandt 1986.)

REMARK. Here is a speculation on another possible virtue of the finite state/regular syntax framework. Writing regular languages in Kleene notation suggests an analogy with regular programs, as used e.g. in Dynamic Logic (see Harel 1984). The grammar is a program which we can activate to produce sentences. As the

program is non-deterministic (because of the available choices), every actual sentence is the trace of an 'intensional' decision process as to what to say. But of course, in some settings (such as Gricean conversation), this intensional view of what is actually said versus what might have been said is of the essence. Can one also push the analogy in other directions, e.g., toward developing a 'correctness theory' for regular languages ?

Finally, up till now, interpretation has been taken in a very narrow sense, as computing some truth value. But, there are much richer senses which could be explored. For instance, in Discourse Representation Theory (Kamp 1984), processing raw syntax results in the incremental construction of discourse representations. Then, evaluators might be rather complex transducers, turning simple syntax into more elaborate structures. Perhaps, the present hierarchical viewpoint could help in distinguishing more elementary from more fanciful mechanisms of representation - and thereby introduce some constraints on the enterprise.

5. Varieties of Inference

The previous discussion of non-orthodox syntax and semantics has various repercussions for the study of inference, the traditional testing ground for semantic theories. Some observations to this effect are gathered here, under two headings.

Inferential complexity

Like sets of well-formed formulas, sets of logical validities can be viewed as languages of a certain complexity. This is what happens already when logicians prove decidability, and when computer scientists ask 'how' decidable various logics are (in polynomial time ? in exponential time ?). In line with Section 2, we look at grammatical complexity of logics:

PROPOSITION. On a finite set of proposition letters, the set of propositional tautologies is context-free.

REMARK. With infinitely many proposition letters, complexity goes up. (The latter situation is the setting for the usual complexity arguments about 'Boolean satisfiability' being NP-complete.)

PROOF. Up to logical equivalence, there are only finitely many equivalent formulas, say with representatives $F_1,\ldots,F_n$. Now, the relevant grammar has auxiliary symbols $S_1,\ldots,S_n$ for each of these, with rewrite rules

$S_i \Rightarrow \neg S_j$ if F_i is equivalent to $\neg F_j$

$S_i \Rightarrow (S_j \wedge S_k)$ if F_i is equivalent to $(F_j \wedge F_k)$

$S_i \Rightarrow p$ if p is equivalent to F_i.

A simple induction shows that, for any propositional formula ϕ, ϕ is equivalent with F_i if and only if F_i rewrites to ϕ in this grammar. In particular, the S_i corresponding to the tautological F_i rewrites to all tautologies. □

All that is used in this proof is the 'logical finiteness' of classical propositional logic. With logics lacking this property, one can expect higher complexity:

PROPOSITION. The set of validities in the minimal modal logic on two proposition letters is not context-free.

PROOF. First, we need a semantic observation.

LEMMA. A modal formula of the form $(\Box^i p \wedge \Box^j q) \rightarrow \Box^k(p \wedge q)$ is valid if and only if $i = j = k$.

PROOF. 'If': this is a well-known type of validity.

'Only if': Consider the frame $\langle IN,S \rangle$ at the point 0. Let $V(p) = \{i\}$, $V(q) = \{j\}$. Then $\Box^i p, \Box^j q$ hold at 0, and hence so does $\Box^k(p \wedge q)$. This can only happen if $k = i = j$. □

Then, the intersection of the set of all validities with the <u>regular</u> set $(\Box^* p \wedge \Box^* q) \rightarrow \Box^*(p \wedge q)$ is the non-context-free language $\{(\Box^n p \wedge \Box^n q) \rightarrow \Box^n(p \wedge q)\}_n$. And so, the set of validities itself cannot be context-free. □

REMARK. This observation also extends to other modal logics. For instance, in S4 the above lemma breaks down; but it can be imitated as follows.
Define 'alternating' modal operators $[A]_p^n \phi$:

$$[A]_p^0 \phi := \Box(p \rightarrow \phi)$$

$$[A]_p^{n+1} \phi := \Box(p \rightarrow \Box(\neg p \rightarrow [A]_p^n \phi)).$$

Note that $[A]_p^n \phi$ implies all $[A]_p^{n+i} \phi$ in S4.

CLAIM. S4 proves $([A]_p^i q \wedge [A]_p^j r) \rightarrow [A]_p^k (q \wedge r)$ if and only if $k \geq$ maximum (i,j).
And then, an argument similar to the above establishes non-context-freeness.

But, one can also look at grammatical complexity of sets of proofs, instead of merely theorems. For instance, axiom systems for predicate logic often carry such restrictions as the following:

'$\forall x\phi \rightarrow \phi^{t}_{x}$, provided that t is free for x in ϕ'.

The latter condition is reminiscent of those studied in Section 2, destroying context-freeness. But also, purely propositional axioms can have this effect. For instance, consider an axiomatization of propositional conditional logic in the Hilbert style, with the usual axioms

$(\phi \rightarrow (\psi \rightarrow \phi))$

$((\phi \rightarrow (\psi \rightarrow \chi)) \rightarrow ((\phi \rightarrow \psi) \rightarrow (\phi \rightarrow \chi)))$.

Let proofs be finite sequences of formulas, separated by commas, created with instances of these axioms and the rule of Modus Ponens.

PROPOSITION. Hilbert proofs for conditional logic do not form a context-free set.

PROOF. Again, the method of proof is the above reduction; using the following observation.

LEMMA. A formula of the form $(\neg\alpha \rightarrow (\neg\beta \rightarrow \neg\gamma))$ is a (one-element) proof if and only if $\alpha = \gamma$. And again, the set $\{(\neg\alpha \rightarrow (\neg\beta \rightarrow \neg\alpha))\}_{\alpha}$ is not context-free. (Compare $\{ww \mid w \in \Sigma^*\}$, as in Hopcroft and Ullman 1979.)

Note: Intersection is to be taken with respect to the regular set of all comma-free strings of the form

(¬ "string" → (¬ "string" → ¬ "string")) . □

A possible use of these considerations may be in substantiating our intuitive feeling that some proof formats are simpler than others. As with 'equivalent notations' (cf. Sections 2,3), 'equivalent systems of proof' by standard logical methods can be quite different. Compare, e.g., a Hilbert type axiomatic system with a cut-free Gentzen calculus. But also, in an obvious sense, various rules within one calculus have quite different complexities.
To see this, compare the elimination rule for → in Natural Deduction (being Modus Ponens) with the introduction rule of Conditionalization, which requires global book-keeping of assumptions higher up in the tree.

The general moral here is that it might be profitable to extend considerations of complexity to the level of texts, of which proofs are one eminent example. For instance, the level of textual complexity of natural language may be quite different from that of its sentential complexity. (One example is the explicit recursion in keeping track of nested levels of hypothetical reasoning, which cannot plausibly be made regular.)

Inference and syntax

When the locus of interpretation becomes more diffuse, because various syntactic levels can contribute 'some meaning', the same question arises with respect to inference. In fact, one could envisage different preferred levels for syntactic parsing, semantic interpretation and logical inference - with efficient parsing of, say, regular syntax incurring the price of inefficient inference at that level. (The latter possible trade-off was pointed out by Hans Kamp.)

There are some precedents for a somewhat looser order of priorities between syntax and inference. For instance, in early logics of this century, rules of inference sometimes produced theorems as well-formed formulas without defining the latter level independently. (This is still possible, e.g., with Gentzen calculi of sequents.) But also in the opposite direction, one might allow rules of inference producing ill-formed output. For instance, the rule of 'Conservativity' as used in current semantic theories for determiners

$$\text{Det N VP} \Leftrightarrow \text{Det N (VP and N)} ,$$

often produces ungrammatical output, which needs syntactic sugaring for its conclusion.

A survey of varieties of natural language inference at various levels of syntactic representation may be found in van Benthem 1986 (chapter 6), 1987. Examples presented there range from a simplest case of monotonicity inference on context-free syntax to anaphoric inference at the level of discourse representation. In the light of the preceding Sections, can we still go one step down in complexity, having inference on regular syntax? At least, this fits in with claims made, e.g., in Fenstad et al. 1985 about the possibility of inference before scopal disambiguation has taken place.

EXAMPLE. Inference from regular propositional syntax.
Here are some valid conclusions, regardless of eventual bracketing:

$$\neg p \wedge q \Rightarrow \neg(p \wedge q)$$

$$p \wedge q \vee r \Rightarrow p \vee r \ , \ q \vee r$$

In fact, this type of inference is still decidable, as it amounts to consequence from the disjunction of all fully bracketed variants of the premise. But, the question here would be if there is some natural intrinsic description.

With this final speculation, we conclude our survey of the secret charms of logical syntax.

References

1. N. Chomsky, 1957, Syntactic Structures, Mouton, Den Haag.
2. G. Gazdar and G. Pullum, 1985, Computationally Relevant Properties of Natural Languages and their Grammars, report 85-24, Center for the Study of Language and Information, Stanford.
3. E. Ejerhed and K. Church, 1982, 'Recursion-Free Context-Free Grammars', paper presented at Workshop on Scandinavian Syntax and Theory of Grammar, University of Trondheim.
4. J. Hopcroft and J. Ullman, 1979, Introduction to Automata Theory, Languages and Computation, Addison-Wesley, Reading (Mass.).
5. J. van Benthem, 1977, Logica in Vogelvlucht, lecture notes, Filosofisch Instituut, Rijksuniversiteit,Groningen.
6. W. Marsh and B. Partee, 1985,'How Non-Context-Free is Variable Binding?', manuscript, department of linguistics, University of Massachusetts, Amherst.
7. A. Aho and J. Ullman, 1972, The Theory of Parsing, Translating and Compiling, Prentice Hall, Englewood Cliffs (N.J.).
8. B. Dreben and W. Goldfarb, 1979, The Decision Problem: Solvable Classes of Quantificational Formulas, Addison-Wesley, Reading (Mass.).
9. W. Quine, 1971, 'Predicate-Functor Logic', in J.E. Fenstad (ed.), Proceedings of the Second Scandinavian Logic Symposium, North-Holland, Amsterdam, 309-315.
10. J. Bacon, 1985, 'The Completeness of a Predicate-Functor Logic', Journal of Symbolic Logic 50, 903-926.
11. J. van Benthem, 1986, Essays in Logical Semantics, Reidel, Dordrecht.
12. Th. Janssen, 1983, Foundations and Applications of Montague Grammar, dissertation, Mathematisch Centrum, Amsterdam.
13. P. Suppes, 1976, 'Elimination of Quantifiers in the Semantics of Natural Language by use of Extended Relational Algebras', Revue Internationale de Philosophie 117/8, 243-259.
14. J. Barwise and J. Perry, 1983, Situations and Attitudes, Bradford Books/MIT Press, Cambridge (Mass.).
15. J. Hintikka and L. Carlson, 1979, 'Conditionals, Generic Quantifiers, and other applications of subgames', in E. Saarinen (ed.), Game-Theoretical Semantics, Reidel, Dordrecht, 179-214.
16. H. Kamp, 1984, 'A Theory of Truth and Semantic Representation', in J. Groenendijk et al. (eds), Truth, Interpretation and Information, Foris, Dordrecht, 1-41.
17. M. Dummett, 1973, Frege. The Philosophy of Language, Duckworth, London.
18. W. Hodges, 1983, 'Elementary Predicate Logic', in D. Gabbay and F. Guenthner, eds, Handbook of Philosophical Logic, vol. I, Reidel, Dordrecht, 1-131.
19. J-E. Fenstad, P-K. Halvorsen, T. Langholm and J. van Benthem, 1985, Equations, Schemata and Situations, report 85-29, Center for the Study of Language and Information, Stanford University.
20. R. van der Sandt, 1986, 'Composition or Cancellation: a misguided dilemma', manuscript, Filosofisch Instituut, Katholieke Universiteit, Nijmegen.
21. D. Harel, 1984, 'Dynamic Logic', in D. Gabbay and F. Guenthner, eds, Handbook of Philosophical Logic, vol. II, Reidel, Dordrecht, 497-604.
22. J. van Benthem, 1987, 'Meaning: Interpretation and Inference', to appear in Synthese, (M-L. dalla Chiara and G. Toraldo di Francia, eds, 'Florence Workshop on Theories of Meaning').