

Reasoning with Natural Language
ESSLLI 2004

Ian Pratt-Hartmann
Department of Computer Science
University of Manchester
Manchester M13 9PL, UK

Contents

1	Introduction	5
2	Background	7
2.1	Grammars	7
2.2	Language and logic	8
2.3	Higher-order logic	10
2.4	Montague grammars	13
3	Some Fragments of English	15
3.1	The syllogistic fragment	15
3.2	Relative clauses	17
3.3	Transitive verbs	20
4	Computational Complexity	23
4.1	Resolution theorem-proving	23
4.1.1	Clauses	23
4.1.2	Resolution and factoring	24
4.1.3	Ordered resolution	26
4.2	The syllogistic fragment	27
4.3	Syllogisms with relative clauses	28
4.4	Transitive verbs	30
5	Fragments with Anaphora	35
6	Concluding Remarks	41

Chapter 1

Introduction

The primary function of language—we flatter ourselves—is to convey information; and these lectures are about the information conveyed by certain English sentences, and how to make inferences with it. For example, consider the following argument:

$$\begin{array}{l} \text{John is a boy} \\ \text{Every boy loves Mary} \\ \hline \text{John loves Mary.} \end{array} \quad (1.1)$$

Intuitively, the two sentences above the line *imply* the sentence below it: whatever the world might be like, if it is such that John is a boy and every boy loves Mary, then it is such that John loves Mary. The aim of these lectures is to show how techniques from Linguistics, Logic, and Computer Science can provide a comprehensive account of patterns of implication involving sentences of natural language.

This subject has a long pedigree. Aristotle's *Prior Analytics* contains a thorough, if somewhat dessicated, account of logical arguments for a very restricted fragment of language. This account consists of a list of inference patterns, each with two premises and a single conclusion, for example:

$$\begin{array}{ll} \begin{array}{l} \text{Every } A \text{ is a } B \\ \text{Every } B \text{ is a } C \\ \hline \text{Every } A \text{ is a } C \end{array} & \begin{array}{l} \text{Every } A \text{ is a } B \\ \text{Some } A \text{ is a } C \\ \hline \text{Some } C \text{ is a } B. \end{array} \end{array} \quad (1.2)$$

Such inference patterns are known as *sylogisms*. (The particular syllogisms in (1.2) are known according to a mediæval system of mnemonics as *Barbara* and *Datisi*, are respectively.) On Aristotle's account, devising logical arguments involving sentences in this fragment is a matter of chaining together syllogisms.

Of course, the syllogistic involves only very simple forms of sentences. In particular, the second sentence in (1.1) is not—or at least is not obviously—of

any of these forms. Famously, the nineteenth century mathematician Augustus de Morgan complained that the syllogistic cannot account for the evident validity of the argument:

$$\frac{\text{Every horse is an animal}}{\text{Every horse's head is an animal's head.}}$$

Indeed, even the *Prior Analytics* itself contains an example of an argument in geometry which cannot—or at least cannot obviously—be expressed in the usual syllogistic form! Although I shall discuss the syllogism in these lectures, I certainly do not want to limit my syntactic horizons in this way. I am interested in identifying inference patterns involving more complex fragments of natural language. The approach taken in these lectures is presented, rather tersely, in a recent paper (Pratt-Hartmann [17]). In these notes, I have tried to provide a self-contained exposition, accessible to students with a relatively limited knowledge of the requisite theoretical background.

Chapter 2

Background

2.1 Grammars

My first task is to establish some means of specifying the fragments of language to be analysed. For the syllogistic fragment, this was easy: the fragment simply consists of all instances of a finite collection of sentence forms. But there is no need for us to set our sights so low.

The formalism I shall employ for this purpose is that of a *context-free grammar*. Here is an example of a context-free grammar:

Syntax	Formal lexicon	Content lexicon	
$S \rightarrow NP, VP$	$Det \rightarrow \text{some}$	$N \rightarrow \text{boy}$	
$VP \rightarrow V, NP$	$Det \rightarrow \text{every}$	...	
$NP \rightarrow PropN$	$Det \rightarrow \text{no}$	$PropN \rightarrow \text{Mary}$	(2.1)
$NP \rightarrow Det, N$		...	
		$V \rightarrow \text{loves}$	
			

Context-free grammars generate sets of sentences by successive expansion of their non-terminal nodes. A formal definition can be found in textbooks; however, as most readers will be familiar with this material, I merely illustrate the procedure using the diagram of Fig. 2.1.

We take the set of sentences generated by a context-free grammar G to be the *language defined by G* . Any context-free grammar defines a language in this way. In the sequel, I shall consider various context-free grammars, defining various (approximations to) fragments of English. Context-free grammars are not the only formalism for specifying the grammatical strings in a language. However, they are the only such formalism that I shall be concerned with in these lectures.

The division of the productions in a context-free grammar into syntax, formal lexicon and content lexicon reflects my special concern in these lectures with

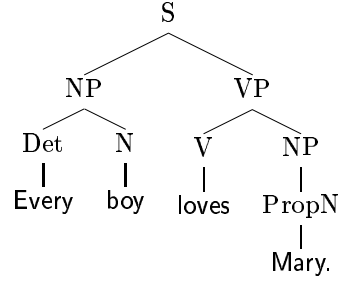


Figure 2.1: A sentence generated by the grammar (2.1).

questions of logical form. The syntax contains the rules for phrasal categories; the formal lexicon contains the rules for the closed-class lexical items; and the content lexicon contains the rules for the open-class lexical items. Thus, we can think of the syntax and formal lexicon as contributing to ‘form’, and the content-lexicon as contributing to ‘content’. From the point of view of the set of sentences which the grammar generates, however, this tripartite division of the productions has no significance.

2.2 Language and logic

Any discussion of logical relations between sentences in natural language must depend on some account of what the sentences of that language *mean*. And the best way of representing sentence meanings for logical purposes is to use a formal language such as first-order logic. For example, here is a translation of the sentences featured in (1.1) into first-order logic:

John is a boy	$\text{boy}(\text{john})$	
Every boy loves Mary	$\forall x(\text{boy}(x) \rightarrow \text{loves}(x, \text{mary}))$	(2.2)
John loves Mary	$\text{loves}(\text{john}, \text{mary}).$	

What is important about this translation is that the target language comes equipped with a truth-theoretic semantics in the style popularized by Tarski. Let us briefly review the key ideas here.

Formulas of first-order logic are evaluated with respect to *structures*. A structure is, in essence, a mathematical model of the way the world might be. A structure $\mathfrak{A}$ is always based on some non-empty set A , called the *domain of interpretation*. We think of the domain of interpretation as a specification of which objects or individuals exist. (It is customary to use the *fraktur* letters $\mathfrak{A}$, $\mathfrak{B}$, $\mathfrak{C}$, ... to denote structures, and the corresponding Roman letters A , B , C , ... to denote their respective domains of interpretation.) The job of $\mathfrak{A}$ is then to interpret all the requisite non-logical symbols—constants, predicates, function-symbols—over A .

Thus, for example, suppose that A consists of five distinct elements: $A = \{a_1, a_2, a_3, a_4\}$. If our language includes the constants *john* and *mary*, then $\mathfrak{A}$ specifies which individuals in A these constants denote, as it might be:

$$\text{john}^{\mathfrak{A}} = a_3 \qquad \text{mary}^{\mathfrak{A}} = a_1.$$

If, in addition, our language includes the unary predicates *boy* and *girl*, then $\mathfrak{A}$ specifies which properties (sets of individuals) over A these predicates denote, as it might be:

$$\text{boy}^{\mathfrak{A}} = \{a_2, a_3, a_4\} \qquad \text{girl}^{\mathfrak{A}} = \{a_1, a_5\}.$$

And if our language includes the binary predicate *loves*, then $\mathfrak{A}$ specifies which relation (set of ordered pairs of individuals) over A this predicate denotes, as it might be:

$$\text{loves}^{\mathfrak{A}} = \{\langle a_1, a_1 \rangle, \langle a_2, a_1 \rangle, \langle a_3, a_1 \rangle, \langle a_4, a_1 \rangle, \langle a_4, a_2 \rangle\}.$$

The semantics of first-order logic determines, for any formula (with no free variables) and any structure, whether that formula is true in that structure. Thus, for example, in the structure $\mathfrak{A}$ just defined, all of the formulas in (2.2) are true. (These rules may be found in almost any logic textbook, and I shall not rehearse them here.) If formula ϕ is true in structure $\mathfrak{A}$, we write $\mathfrak{A} \models \phi$. If Φ is a set of formulas and $\mathfrak{A} \models \phi$ for all $\phi \in \Phi$, we write $\mathfrak{A} \models \Phi$. A formula ϕ is said to be *satisfiable* just in case there exists a structure $\mathfrak{A}$ such that $\mathfrak{A} \models \phi$; similarly for a set of formulas Φ . If, for all structures $\mathfrak{A}$, either $\mathfrak{A} \not\models \Phi$ or $\mathfrak{A} \models \phi$, we say that the *sequent with premises Φ and conclusion ϕ is valid*, and we write $\Phi \models \phi$.

Exercise 1. Explain why $\Phi \models \phi$ if and only if $\Phi \cup \{\neg\phi\}$ is not satisfiable.

In this way, the logical translations of English sentences provide a formal reconstruction of the intuitive concept of a valid argument. Consider, for example, the sentences in (2.2). It is easy to check that

$$\{\text{boy}(\text{john}), \forall x(\text{boy}(x) \rightarrow \text{loves}(x, \text{mary}))\} \models \text{loves}(\text{john}, \text{mary}),$$

reflecting our intuition that the argument (1.1) is valid. Henceforth, I shall always take validity in first-order logic to be the touchstone of validity in natural language: an argument couched in natural language is valid just in case the formulas to which its premises and conclusion translate form a valid sequent.

I claimed above that the formulas in the right-hand column in (2.2) are *translations of* the corresponding English sentences in the left-hand column. Exactly what this claim amounts to is harder to specify than one might at first suppose. Structures are abstract objects, and the semantics of first-order logic assigns truth values to formulas only with respect to structures. English sentences, by contrast, (typically) speak of real objects, and not—or at least not obviously—of structures at all. True, we are encouraged to think of structures

as (abstractions of) ways the world might be, and perhaps ways the world might be are something English sentences can speak about. But again, what it is we are really being encouraged to think here is quite mysterious.

To circumvent this problem, I complicate our view of languages and, consequently, of the grammars that define them. Instead of taking a language to be a set of sentences, and a grammar a machine for generating that set, I instead take a language to be a set of sentences *paired with logical translations specifying the meanings of those sentences*, and a grammar to be a machine for generating such a pairing. This will, hopefully, become clearer as we proceed. In order to explain how this new concept of grammar works, I need to review some technical material concerning the logic of function-application; it is to this topic that I turn next.

2.3 Higher-order logic

Recall that, in mathematics, a *function* is a set of ordered tuples. Thus, for example, the squaring function on $\mathbb{N}$ is the infinite set of pairs

$$\{\langle 0, 0 \rangle, \langle 1, 1 \rangle, \langle 2, 4 \rangle, \langle 3, 9 \rangle, \dots\} \quad (2.3)$$

whose second element is the square of the first. Likewise, the addition function on $\mathbb{N}$ is the infinite set of triples

$$\{\langle 0, 0, 0 \rangle, \langle 1, 0, 1 \rangle, \langle 0, 1, 1 \rangle, \langle 1, 1, 2 \rangle, \dots\} \quad (2.4)$$

whose third element is the sum of the first two.

Although functions *are* just sets of tuples, these sets are usually *specified* using a succinct expression of some language whose terms have agreed interpretations over the domain in question. Thus, for example, the squaring function (2.3) might be succinctly specified using the expression $\lambda x[x^2]$, where the “ λx ” indicates that we are dealing with a function taking x as input, and the “ $[x^2]$ ” gives the output for a given input x . Similarly, the addition function (2.4) might be succinctly specified using the expression $\lambda x \lambda y[x + y]$, where the “ $\lambda x \lambda y$ ” indicates that we are dealing with a function taking x and y as input, and the “ $[x + y]$ ” gives the output for given inputs x and y . The body of such a λ -expression (the bit inside the square brackets) is just an expression of some formal language: provided we can interpret the non-logical vocabulary of this language, this notation specifies functions in the obvious way. Remember, however, that the object denoted by a λ -expression (relative to a structure interpreting the non-logical vocabulary) is not a rule or a procedure, but a set: the set of tuples consisting of all possible input(s) and the corresponding output.

Another example should help make this clear. Consider the structure $\mathfrak{A}$ defined in Section 2.2. With respect to this structure, the expression $\lambda x[\text{girl}(x)]$ denotes the function:

$$\{\langle a_1, T \rangle, \langle a_2, F \rangle, \langle a_3, F \rangle, \langle a_4, F \rangle, \langle a_5, T \rangle\}$$

mapping each individual to the indicated truth-value. Similarly, the expression $\lambda x[\text{boy}(x)]$ denotes the function:

$$\{\langle a_1, F \rangle, \langle a_2, T \rangle, \langle a_3, T \rangle, \langle a_4, T \rangle, \langle a_5, F \rangle\}.$$

Thus, on this view, a property defined on some set of objects is a function from that set of objects to the set of truth-values $\{T, F\}$.

Consider again the expression $\lambda x \lambda y[x + y]$. I suggested above that this expression denotes a function of two arguments returning a number as output. However, it is actually simpler to regard it instead as denoting a function of one argument x , which returns as value a function—namely, the function which takes one argument y and returns the number $x + y$ as value, that is:

$$\begin{aligned} \lambda x \lambda y[x + y] = \\ \{\langle 0, \{\langle 0, 0 \rangle, \langle 1, 1 \rangle, \dots \} \rangle, \langle 1, \{\langle 0, 1 \rangle, \langle 1, 2 \rangle, \dots \} \rangle, \langle 2, \{\langle 0, 2 \rangle, \langle 1, 3 \rangle, \dots \} \rangle, \dots \}. \end{aligned}$$

Using this trick, we may assume without loss of generality that all functions have one argument, though they may produce functions as values. This process of converting multi-input functions to single-input functions is sometimes called *currying*, after Haskell B. Curry, a logician.

Just as functions may produce functions as outputs, so too they may take functions as inputs. Such functions are sometimes called *higher-order* functions. Thus, for example, the property of being a property possessed by Mary is denoted by

$$\lambda p[p(\text{mary})], \tag{2.5}$$

the property of being a property possessed by every girl, by

$$\lambda p[\forall x(\text{girl}(x) \rightarrow p(x))], \tag{2.6}$$

and the property of being a property possessed by some boy, by

$$\lambda p[\exists x(\text{boy}(x) \wedge p(x))]. \tag{2.7}$$

In expressions (2.5)–(2.7), the variable p is taken to range over properties of objects (functions from objects to truth-values). That is why, in the bodies of these expressions, p functions syntactically as a predicate. Thus, in this formalism, variables are *typed* according to the kind of entities they range over: individuals, truth-values, functions from individuals to truth-values, and so on. In these lectures, I mark this typing informally by using $x, y, \text{etc.}$ to range over individuals, $p, q, \text{etc.}$ to range over properties, and (occasionally), other letters for still higher types. The formal language we are employing here is usually known as *higher-order logic*. Technically, of course, I should specify the syntax and semantics of this language. However, such a digression would take us too far from the main subject of these lectures; and most people get the hang of the notation with a little practice.

Higher-order logic incorporates two rules for manipulating expressions: α -conversion and β -reduction. The rule of α -conversion allows the renaming of bound variables, thus:

$$\lambda x[\phi(x)] \implies \lambda y[\phi(y)]$$

where y is a variable not occurring free in ϕ . The operation of β -reduction allows us to compute the result of applying functions to arguments, thus:

$$\lambda x[\phi(x)](\psi) \Longrightarrow \phi[\psi/x]$$

where $\phi[\psi/x]$ indicates the result of substituting ψ for all free occurrences of the variable x in $\phi(x)$, provided that ϕ and ψ have no variables in common. Of course, this proviso is never a problem in practice, since it can always be realized by applying α -conversion as necessary to either ϕ or ψ .

To illustrate how β -reduction gives sensible results, let us see what happens when we apply the squaring function to some argument—say the number 6. The result is:

$$\lambda x[x^2](6) = 6^2 = 36.$$

As a second example, let us see what happens when we apply the function representing the property of being a property possessed by Mary to the property of being happy:

$$\begin{aligned} \lambda p[p(\text{mary})](\lambda x[\text{happy}(x)]) &= \\ \lambda x[\text{happy}(x)](\text{mary}) &= \\ \text{happy}(\text{mary}), \end{aligned}$$

by two applications of β -reduction. Of course, under a particular structure interpreting the non-logical vocabulary, the resulting expression denotes a truth-value: T if, in the chosen structure, Mary is happy, F otherwise. This equation expresses the judgment that the property of being happy has the (higher-order) property of being a property possessed by Mary just in case Mary is happy.

Here is another example of β -reduction. We can think of the function $\lambda p\lambda q[\forall x(p(x) \rightarrow q(x))]$ as a relation between properties: the relation holds of properties p and q if and only if every individual which possesses property p also possesses property q . (Remember, however, that we are currying functions, so this expression actually denotes a function which takes one property as input and produces another function as output.) Suppose that we apply this function to the property of being a boy and β -reduce:

$$\begin{aligned} \lambda p\lambda q[\forall x(p(x) \rightarrow q(x))](\lambda x'[\text{boy}(x')]) &= \\ \lambda q[\forall x(\lambda x'[\text{boy}(x')](x) \rightarrow q(x))] &= \\ \lambda q[\forall x(\text{boy}(x) \rightarrow q(x))]. \end{aligned} \tag{2.8}$$

The result is the (higher-order) property of of being a property true of all boys. Applying this in turn to the property of being happy, we get:

$$\begin{aligned} \lambda q[\forall x(\text{boy}(x) \rightarrow q(x))](\lambda x'[\text{happy}(x')]) &= \\ \forall x(\text{boy}(x) \rightarrow \lambda x'[\text{happy}(x')](x)) &= \\ \forall x(\text{boy}(x) \rightarrow \text{happy}(x)), \end{aligned}$$

which is a truth-value: true (in a given structure) if every boy is happy, false otherwise.

2.4 Montague grammars

We are now in a position to augment the context-free grammar given in (2.1) with semantic information so as to allow automatic translation into first-order logic. Every entry in the grammar's lexicon is associated with an expression of higher-order logic representing the meaning of the word in question; in addition, every entry in the grammar's syntax is associated with an expression of higher-order logic specifying how the meaning of the phrase it licenses is assembled from the meanings of its immediate constituents. In this case, the augmented grammar is:

Syntax	Formal lexicon
$S/y_1(y_2) \rightarrow \text{NP, VP}$	$\text{Det}/\lambda p\lambda q[\exists x(p(x) \wedge q(x))] \rightarrow \text{some}$
$\text{VP}/y_1(y_2) \rightarrow \text{V, NP}$	$\text{Det}/\lambda p\lambda q[\forall x(p(x) \rightarrow q(x))] \rightarrow \text{every}$
$\text{NP}/y_1 \rightarrow \text{PropN}$	$\text{Det}/\lambda p\lambda q[\forall x(p(x) \rightarrow \neg q(x))] \rightarrow \text{no}$
$\text{NP}/y_1(y_2) \rightarrow \text{Det, N}$	
Content lexicon (2.9)	
$\text{N}/\lambda x[\text{boy}(x)] \rightarrow \text{boy}$	
...	
$\text{PropN}/\lambda p[p(\text{mary})] \rightarrow \text{Mary}$	
...	
$\text{V}/\lambda s\lambda x[s(\lambda y[\text{loves}(x, y)])] \rightarrow \text{loves}$	
...	

Here, each syntax rule is annotated by an expression involving the variables y_1 and y_2 : y_1 represents the meaning of the first constituent on the right-hand side of the rule, and y_2 the meaning of the second constituent on the right-hand side of the rule (if present). The expression in question—for example, $y_1(y_2)$ —then specifies how the meaning of the phrase licensed by the rule depends on the meanings of its immediate constituents.

Consider, by way of illustration, the production $\text{NP}/y_1(y_2) \rightarrow \text{Det, N}$. Given that **every** is a Det with meaning $\lambda p\lambda q[\forall x(p(x) \rightarrow q(x))]$ and **boy** an N with meaning $\lambda x[\text{boy}(x)]$, this rule licenses the categorization of **every boy** as an NP, with meaning computed by applying the meaning of **every** to the meaning of **boy**. In detail:

$$\begin{aligned}
 [[\text{NP every boy}]] &= [[\text{Det every}]]([[\text{Det boy}]]]) \\
 &= \lambda p\lambda q[\forall x(p(x) \rightarrow q(x))](\lambda x'[\text{boy}(x')]) \\
 &= \lambda q[\forall x(\text{boy}(x) \rightarrow q(x))].
 \end{aligned}$$

The last line here is a step of β -reduction: the details of the calculation are spelled out in (2.8). In the sequel, I always (silently) β -reduce as much as possible, to minimize notational clutter. Applying this procedure for other productions in (2.9), we obtain the phrase-structure of Fig. 2.2. Notice how



this procedure, in addition to generating the sentence **Every boy loves Mary**, simultaneously translates it into first-order logic.

Exercise 2. Verify the computation of Fig. 2.2.

Chapter 3

Some Fragments of English

With the tools introduced above at my disposal, I am now in a position to define some simple fragments of English, and provide those fragments with a formal semantics mapping their sentences into formulas of first-order logic. In the next lecture, I analyse the difficulty of making inferences in these fragments.

3.1 The syllogistic fragment

Consider the following Montague grammar:

Syntax	Formal lexicon
$IP/y_1(y_2) \rightarrow NP, I'$	$Det/\lambda p\lambda q[\exists x(p(x) \wedge q(x))] \rightarrow \text{some}$
$I'/y_1 \rightarrow \text{is a } N'$	$Det/\lambda p\lambda q[\forall x(p(x) \rightarrow q(x))] \rightarrow \text{every}$
$I'/\lambda x[\neg y_1(x)] \rightarrow \text{is not a } N'$	$Det/\lambda p\lambda q[\forall x(p(x) \rightarrow \neg q(x))] \rightarrow \text{no}$
$NP/y_1 \rightarrow \text{PropN}$	
$NP/y_1(y_2) \rightarrow Det, N'$	Content lexicon
$N'/y_1 \rightarrow N.$	$N/\lambda x[\text{man}(x)] \rightarrow \text{man}$
	$N\lambda x[\text{mortal}(x)] \rightarrow \text{mortal}$
	...
	$\text{PropN}/\lambda p[p(\text{socrates})] \rightarrow \text{Socrates}$
	

Let us call the fragment of English defined by this grammar Cop, since the most salient syntactic construct is the *copula*. Fig. 3.1 shows a sample derivation in Cop. It is straightforward to verify that these grammar rules recognize sentences of the following six forms, and translate them to first-order logic as follows:

Every A is a B	$\forall x(a(x) \rightarrow b(x))$	No A is a B	$\forall x(a(x) \rightarrow \neg b(x))$
Some A is a B	$\exists x(a(x) \wedge b(x))$	Some A is not a B	$\exists x(a(x) \wedge \neg b(x))$
S is a B	$b(s)$	S is not a B	$\neg b(s).$

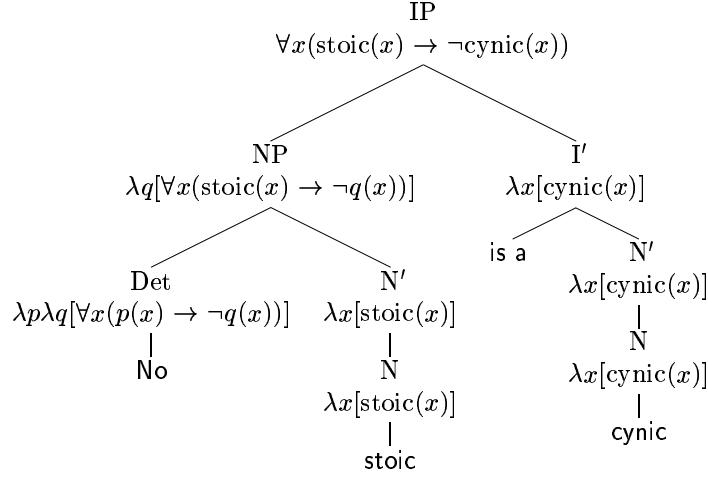


Figure 3.1: A derivation in Cop

Here, a , b and s are of course the non-logical primitives specified in the content lexicon entries for A , B and S . Thus, the logic generated by Cop is at least as expressive as the syllogistic.

Does the grammar of Cop recognize any sentences of any other forms? Yes, in addition to the forms (3.1), we have the following two forms (and no others):

$$\text{Every } A \text{ is not a } B \quad \forall x(a(x) \rightarrow \neg b(x)) \quad \text{No } A \text{ is not a } B \quad \forall x(a(x) \rightarrow \neg \neg b(x)).$$

These sentence forms do not traditionally feature in syllogisms (indeed, they are arguably grammatically marginal). However, the corresponding formulas $\forall x(a(x) \rightarrow \neg b(x))$ and $\forall x(a(x) \rightarrow \neg \neg b(x))$ are logically equivalent to formulas in (3.1), so that these sentences provide no additional expressive power. Thus, our fragment Cop has exactly the expressive power of the syllogistic. Note, however that, in contrast to the methods of traditional logic, I have adopted the now standard non-presuppositional account of universal quantifiers, according to which a sentence **Every** A is a B is true if no A exist. (Standard treatments of the syllogism assign the reverse truth-value in this case.) But I pass over this minor detail.

Since Cop is a Montague grammar, and so translates the sentences it recognizes to first-order logic, the notions of satisfiability and validity for the relevant fragment can be reconstructed in the standard way. Thus, if E is a set of sentences in Cop which translate to a set of formulas Φ , I take E to be satisfiable if Φ is satisfiable; if, in addition, e is a sentence of Cop which translates to a formula ϕ , I take the argument from E to e to be valid just in case $\Phi \models \phi$. Modulo the issues of presuppositionality mentioned above, the traditional syllogistic can be seen as a *test* for the validity of arguments in Cop. Given a set of premises E and a putative conclusion e , the validity of the argument from

E to e can be tested by trying to chain together syllogisms in such a way as to derive e from some subset of E . Whether this is an efficient method, or even a correct method, are matters of historical interest which need not detain us here. But the important point is that determining the satisfiability of a set of sentences in Cop, or determining whether one sentence in Cop follows from a set of such sentences are well-defined problems. It is therefore natural to ask what the intrinsic computational complexity of these problems is. Since Cop is closed under negation, satisfiability and validity are really just complementary problems (see Exercise 1). In the sequel, therefore, I follow established custom and focus on the satisfiability problem.

If X is any expression or set of expressions (either in a natural or a formal language), I take the *size* of X , denoted $|X|$, to be the number of atomic symbol (tokens) occurring in X . For natural languages, the atomic symbols are the lexemes (and perhaps morphemes); for formal languages, the atomic symbols are the logical connectives, the variables and the members of the signature. Using this notion of size, we can formulate questions about computational complexity in the usual way. Thus, the complexity of the satisfiability problem for some English fragment $\mathcal{E}$ is the number of steps of computation required to determine algorithmically whether a given finite set of sentences $E \subset \mathcal{E}$ is satisfiable, expressed as a function of $|E|$.

Our first result states that satisfiability in the fragment Cop is computationally tractable.

Theorem 1. *The problem of determining the satisfiability of a set of sentences in Cop is in PTIME.*

That is: the number of computational steps required to determine whether a set E of sentences in Cop is satisfiable is bounded by some fixed polynomial of the size of E .

I prove this result in the next lecture. For the present, I note that it suggests a programme of work: take a fragment of English delineated in terms which respect the syntax of the language; then determine the computational complexity of deciding satisfiability in that fragment, if, indeed, the fragment is decidable. From this standpoint, the syllogistic can be regarded as just one such fragment, with very restricted syntax, and a correspondingly efficient decision procedure. In the sequel, I shall investigate what happens as we expand our syntactic horizons.

3.2 Relative clauses

One way to generalize the fragment Cop is to add relative clauses, thus accommodating arguments such as:

$$\begin{array}{l}
 \text{Every philosopher who is not a stoic is a cynic} \\
 \text{Every stoic is a man} \\
 \text{Every cynic is a man} \\
 \hline
 \text{Every philosopher is a man.}
 \end{array}
 \tag{3.2}$$

Our approach to the semantics of relative clauses loosely follows that of Heim and Kratzer [10]. Let Cop+Rel be the fragment defined by the grammar rules for Cop together with the following additional syntax rules and formal lexicon rules.

Syntax	Formal lexicon
$N'/y_2(y_1) \rightarrow N, CP$	$C \rightarrow$
$CP/y_1(y_2) \rightarrow CSpec_t, C'_t$	$RelPro/\lambda q \lambda p \lambda x[p(x) \wedge q(x)] \rightarrow \text{who,}$
$C'_t/\lambda t[y_2] \rightarrow C, IP$	which
$NP/y_1 \rightarrow RelPro$	
$CSpec_t \rightarrow$	

In addition, I assume that, following generation of an IP by these rules, relative pronouns are subject to wh-movement. For our purposes, we may take the wh-movement rule to require: (i) the empty position $CSpec_t$ must be filled by movement of a RelPro from within the IP which forms its right-sister (in more technical jargon: which it *C-commands*); (ii) every RelPro must move to some such $CSpec_t$ position; (iii) every RelPro moving to $CSpec_t$ leaves behind a trace t , which contributes the semantic value $\lambda p[p(t)]$. The semantic information with which these rules are augmented can then be understood as for Cop, with meanings computed after wh-movement. Figure 3.2 illustrates the structure of the first sentence in argument (3.2), with the arrow representing wh-movement in the obvious way.

Exercise 3. Verify the computation of Fig. 3.2.

The grammar for Cop+Rel thus translates the argument (3.2) to:

$$\frac{\begin{array}{l} \forall x(\text{philosopher}(x) \wedge \neg \text{stoic}(x) \rightarrow \text{cynic}(x)) \\ \forall x(\text{stoic}(x) \rightarrow \text{man}(x)) \\ \forall x(\text{cynic}(x) \rightarrow \text{man}(x)) \end{array}}{\forall x(\text{philosopher}(x) \rightarrow \text{man}(x))}.$$

I mention in passing that the wh-movement rule (and the associated use of free variables in meanings) can actually be avoided: it is a simple matter to simulate these effects using the apparatus of Montague grammars introduced above. I have helped myself to a formulation based on wh-movement because it will be more familiar to many readers.

Our second result states that the problem of determining satisfiability in the fragment Cop+Rel is likely to be harder than the corresponding problem for Cop.

Theorem 2. *The problem of determining the satisfiability of a set of sentences in Cop+Rel is NP-complete.*

Again, I will explain in the next lecture how to prove this result.

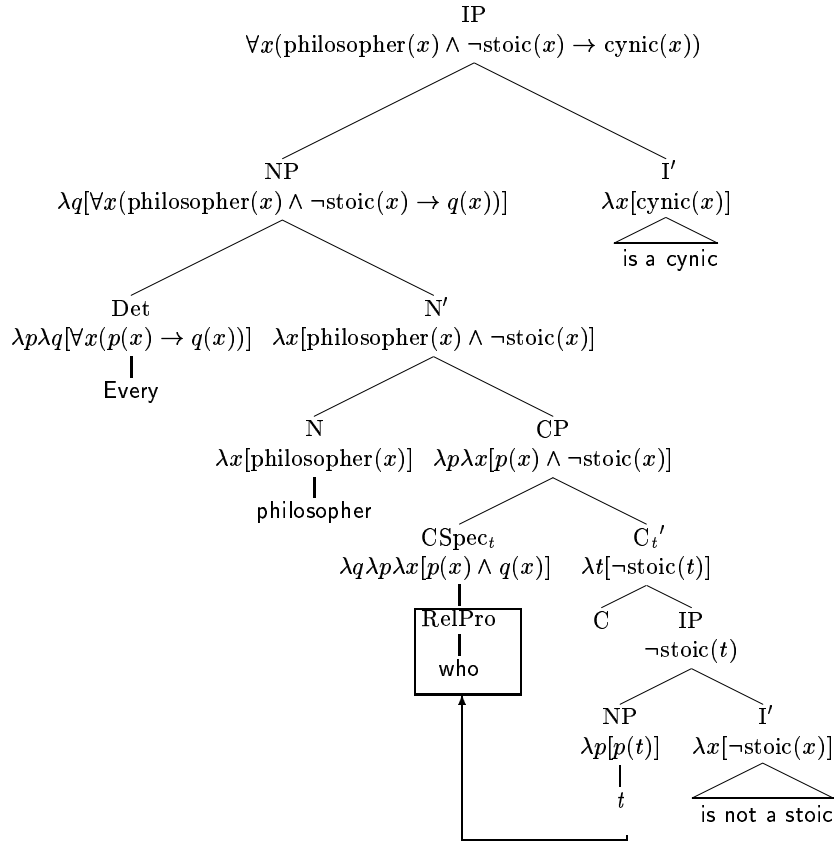


Figure 3.2: Typical phrase-structure in the fragment Cop+Rel

3.3 Transitive verbs

I have already mentioned de Morgan's complaint that the syllogistic cannot account for the evident validity of the argument:

$$\frac{\text{Every horse is an animal}}{\text{Every horse's head is an animal's head.}}$$

The problem is of course that this argument involves *relational* facts, whereas the syllogistic mentions only (non-relational) *properties*. Let us see what happens when we add relations to our fragment of English.

The possessive constructions featured in de Morgan's argument involve some delicate syntactic and semantic handling; however, transitive verbs, which also carry relations as their semantic payloads, are more straightforward; so it is these that I turn to now. Let Cop+Rel+TV be the fragment defined by the grammar rules for Cop+Rel together with the following additional grammar rules:

Syntax	Content Lexicon
$I'/y_1 \rightarrow \text{VP}$	$V/\lambda s \lambda x[s(\lambda y[\text{admire}(x, y)])] \rightarrow \text{admires}$
$I'/y_1 \rightarrow \text{NegP}$	$\dots$
$\text{NegP}/\lambda x[\neg(y_1(x)) \rightarrow \text{does not, VP}]$	$\dots$
$\text{VP}/y_1(y_2) \rightarrow \text{V, NP.}$	

The wh-movement rule is carried over from Cop+Rel. Note that this rule allows any NP in a relative clause—in either subject or object position—to fill the appropriate CSpec. Fig. 3.3 shows a typical derivation in this fragment.

Exercise 4. Verify the computation of Fig. 3.3.

The fragment Cop+Rel+TV includes the sentences in the following (lightly transcribed) version of the de Morgan argument:

$$\frac{\text{Every horse is an animal}}{\text{Every head which some horse has is a head which some animal has.}}$$

Exercise 5. Verify that the above rules translate this argument as follows.

$$\frac{\forall x(\text{horse}(x) \rightarrow \text{animal}(x))}{\forall x(\text{head}(x) \wedge \exists y(\text{horse}(y) \wedge \text{has}(y, x)) \rightarrow \text{head}(x) \wedge \exists y(\text{animal}(y) \wedge \text{has}(y, x)))}.$$

In the next lecture, I prove the following result:

Theorem 3. *The problem of determining the satisfiability of a set of sentences in Cop+Rel+TV is EXPTIME-complete.*

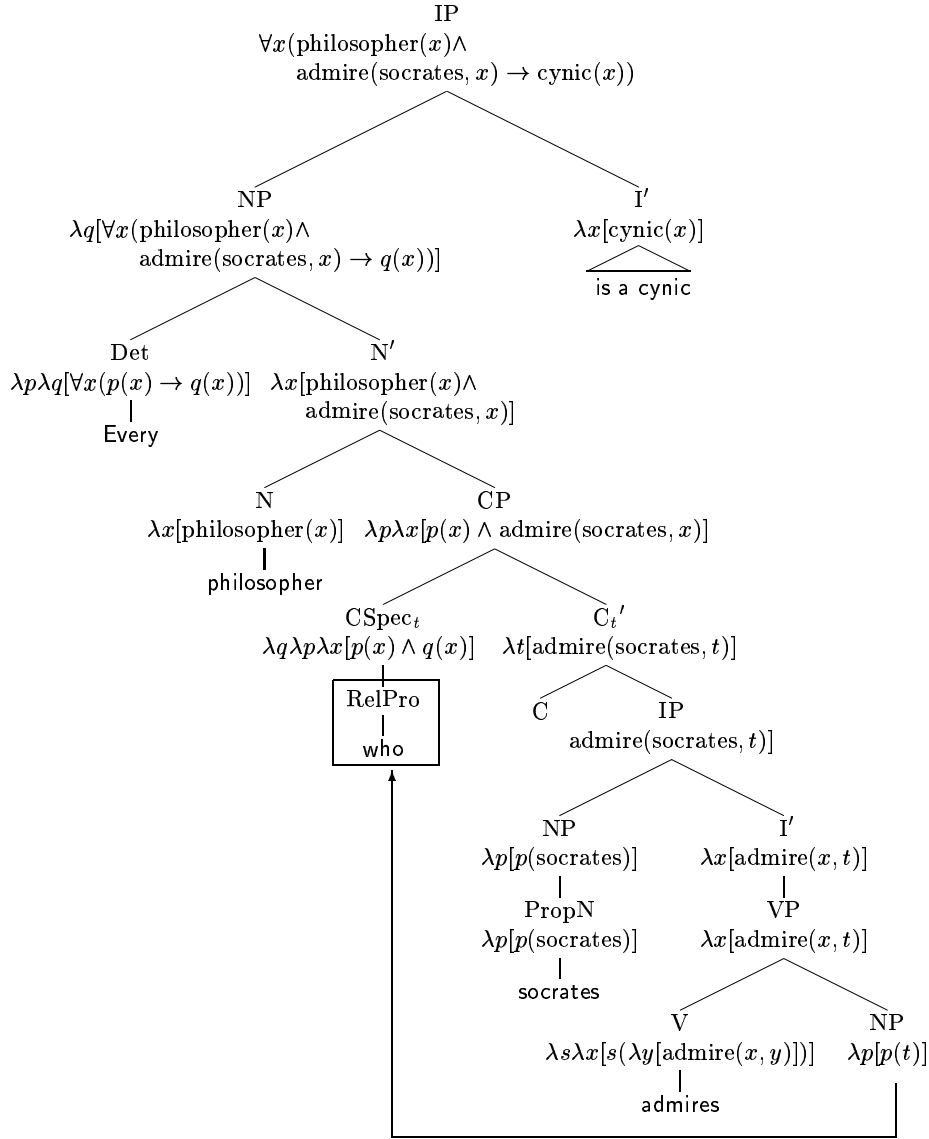


Figure 3.3: Typical phrase-structure in the fragment Cop+Rel+TV

It follows that there is no algorithm for determining the satisfiability of sets of Cop+Rel+TV-sentences which runs in polynomial time. Adding transitive verbs and relative clauses to the syllogistic thus increases the computational complexity of determining logical relations between English sentences.

To be sure, the fragment Cop+Rel+TV is a rather rough approximation to English as she is spoken. I have ignored this issue of verb-inflections. I have ignored possibility of quantifier rescoping. I have ignored the issue of negative polarity determiners, thus allowing dubious sentences such as

No cynic does not admire no stoic. (3.3)

Finally, I have failed to ban the notorious phenomenon of centre-embedding, thus allowing

Every cynic is a cynic who every sceptic who some stoic admires
despises. (3.4)

Exercise 6. Show that the sentences (3.3) and (3.4) are generated by Cop+Rel+TV, and compute their translations into first-order logic.

Evidently, a more sensitive grammatical treatment would require some additional effort. In fact, however, plugging these leaks in our grammar cannot alter the complexity of reasoning within it. This surprising result follows from the analysis of the next lecture.

Chapter 4

Computational Complexity

4.1 Resolution theorem-proving

4.1.1 Clauses

The following notation is standard in the literature on Automated Theorem-Proving. A *literal* is an atomic formula or the negation of an atomic formula; a *clause* is a disjunction of literals. Thus, $p \vee q$, $\neg p'(x) \vee q'(x, y)$, $\neg p''(x, y, z) \vee \neg q''(x, f(x)) \vee \neg r(g(x, f(x)))$ are clauses, but $p \rightarrow q$, $p'(x) \vee \neg \neg q'(x, y)$ and $p''(x, y, z) \wedge q''(x, f(x))$ are not. The formula $\perp$ is taken to be the disjunction of zero literals, and thus counts as a clause. In these lectures, I assume that clauses do not contain the equality predicate.

Since clauses contain no quantifiers, they will in general have free variables. If C is a clause, denote its universal closure by $\forall C$ (i.e. the formula $\forall x_1 \dots \forall x_n C$, where $x_1, \dots, x_n$ are all the variables in C). If $\mathfrak{A}$ is a structure and C a clause, say that C is *true in $\mathfrak{A}$* if $\mathfrak{A} \models \forall C$. A clause C is said to be *satisfiable* if there exists a structure in which it is true; a set of clauses $\mathcal{C}$ is said to be *satisfiable* if there exists a structure in which all of its elements are true. In this way, for example, the clause $\neg \text{man}(x) \vee \text{mortal}(x)$ expresses the universally quantified formula $\forall x(\text{man}(x) \rightarrow \text{mortal}(x))$; likewise, the clause $\neg \text{stoic}(x) \vee \neg \text{cynic}(y) \vee \text{admire}(x, y)$ expresses the universally quantified formula $\forall x(\text{stoic}(x) \rightarrow \forall y(\text{cynic}(y) \rightarrow \text{admire}(x, y)))$.

There is no direct way to express existential quantification in clauses; however, the effect of existential quantifiers can be ‘simulated’ using constants or function-symbols. Thus, for example, the trio of clauses

$$\text{stoic}(c), \quad \text{cynic}(d), \quad \text{admire}(c, d) \quad (4.1)$$

guarantees the truth of the formula $\exists x(\text{stoic}(x) \wedge \exists y(\text{cynic}(y) \wedge \text{admire}(x, y)))$. Likewise, the pair of clauses

$$\neg \text{stoic}(x) \vee \text{cynic}(fx), \quad \neg \text{stoic}(x) \vee \text{admire}(x, fx) \quad (4.2)$$

guarantees (remembering to read the free variable x as universally quantified) the truth of $\forall x(\text{stoic}(x) \rightarrow \exists y(\text{cynic}(y) \wedge \text{admire}(x, y)))$.

In fact, it transpires that *any* formula in first-order logic can be converted into clause form in this way:

Theorem 4. *If Φ is a set of first-order formulas, we can find a set of clauses $\mathcal{C}$ such that $\mathcal{C}$ is satisfiable if and only if Φ is satisfiable. Moreover, $\mathcal{C}$ can be computed in time bounded by a polynomial function of the size of Φ .*

A proof of this theorem may be found in many elementary textbooks in logic. Notice that the relationship claimed to hold between Φ and $\mathcal{C}$ is very weak: it merely guarantees that either both Φ and $\mathcal{C}$ are satisfiable, or neither Φ nor $\mathcal{C}$ is satisfiable.

4.1.2 Resolution and factoring

Suppose we know that the propositional formulas p and $p \rightarrow q$ are both true. Evidently, we may infer that q is also true. This inference rule, known as *modus ponens*, may be displayed, with the obvious interpretation, as follows:

$$\frac{p \rightarrow q \quad p}{q}.$$

Re-writing $p \rightarrow q$ in the logically equivalent clausal form $\neg p \vee q$, this inference rule becomes:

$$\frac{\neg p \vee q \quad p}{q}. \quad (4.3)$$

The (propositional) *resolution rule* is the following generalization of (4.3):

$$\frac{\neg p \vee C \quad p \vee D}{C \vee D}, \quad (4.4)$$

where C and D are arbitrary propositional clauses. (To obtain (4.3) as a special case of (4.4), let C be the clause q and D the empty clause.) This inference rule is undoubtedly valid: any structure which makes the clauses above the line true makes the clause below the line true too.

The very same rule works in the general case, provided that appropriate substitutions are chosen for variables in the resolved literals. Again, the easiest way to understand what is going on is to begin with a special case:

$$\frac{\neg p(x) \vee q(x) \quad p(a)}{q(a)}. \quad (4.5)$$

Remembering that free variables in clauses are understood as universally quantified, this inference schema is simply a familiar syllogism in disguise: if all p 's are q 's and a is a p , then a is a q .

More generally, if A and A' are atoms, we say that A and A' are *unifiable* if there is a substitution σ of terms for variables such that $A\sigma$ and $A'\sigma$ become identical. It turns out that, if A and A' are unifiable, then there is substitution θ

such that $A\theta = A'\theta$ and, for any substitution σ such that $A\sigma = A'\sigma$, we have $\sigma = \theta\rho$ for some substitution ρ . In other words, if A and A' are unifiable, then they have a *most general unifier* (m.g.u.) θ , from which any other unifying substitution can be obtained by further substitution of terms for variables. Moreover, θ is unique up to renaming of variables, and can be efficiently computed from A and A' .

Generalizing (4.5) then, the (first-order) *resolution rule* is the inference schema

$$\frac{\neg A \vee C \quad A' \vee D}{(C \vee D)\theta}, \quad (4.6)$$

where C and D are clauses and A and A' are unifiable atoms with m.g.u. θ . Again, this schema is undoubtedly valid: in any structure, if the clauses above the line are true, then so is the clause below the line.

The inference rule of (propositional) *factoring* is still simpler. Suppose we have a clause with repeated literals, say $p \vee p \vee q \vee p$. Obviously, this clause entails $p \vee q$: that is, we may eliminate the repeated literals. Like resolution, factoring generalizes to the first-order case using the notion of an m.g.u. If C is any (first-order) clause, let C^0 denote the result of eliminating repeated literals in C . Then the (first-order) *factoring rule* is the inference schema

$$\frac{C}{(C\theta)^0}, \quad (4.7)$$

where C is a clause and θ the m.g.u. of some nonempty subset of literals in C .

Equipped with the inference rules of resolution and factoring, we can define a sensible notion of proof. A *proof* from a set of clauses $\mathcal{C}$ is simply a finite sequence of clauses such that every clause in this sequence is either an element of $\mathcal{C}$ or follows by either resolution or factoring from earlier clauses in the sequence. In practice, it is usually easier to display a proof as a tree indicating the various inference steps in the obvious way. Note that we can use proofs to determine implications between sets of clauses: if, for some structure $\mathfrak{A}$, every clause in $\mathcal{C}$ is true in $\mathfrak{A}$, and if there is a proof from $\mathcal{C}$ of another clause C , then C is also true in $\mathfrak{A}$.

Example 1. Consider the four clauses

$$\begin{array}{ll} \neg\text{boy}(x) \vee \neg\text{girl}(y) \vee \text{loves}(x, y) & \text{boy}(\text{john}) \\ \text{girl}(\text{mary}) & \neg\text{loves}(\text{john}, \text{mary}). \end{array}$$

Here is a resolution proof of $\perp$ from them:

$$\begin{array}{c} \text{boy}(\text{john}) \quad \text{girl}(\text{mary}) \quad \neg\text{boy}(x) \vee \neg\text{girl}(y) \vee \text{loves}(x, y) \\ \neg\text{loves}(\text{john}, \text{mary}) \quad \frac{\text{boy}(\text{john}) \quad \text{girl}(\text{mary}) \quad \neg\text{boy}(x) \vee \neg\text{girl}(y) \vee \text{loves}(x, y)}{\neg\text{boy}(x) \vee \text{loves}(x, \text{mary})} \\ \hline \text{loves}(\text{john}, \text{mary}) \\ \hline \perp. \end{array}$$

Thus, we see that this set of clauses is unsatisfiable, since $\perp$ is true in no structure.

If $\mathcal{C}$ is a set of clauses, and C is any clause, we write $\mathcal{C} \vdash C$ to mean that there is a proof of C from $\mathcal{C}$ using only resolution and factoring. With this notation, we can state the validity of the resolution and factoring rules by saying that $\mathcal{C} \vdash \perp$ implies that $\mathcal{C}$ is unsatisfiable. Surprisingly, the converse of this result is also true.

Theorem 5. *Let $\mathcal{C}$ be a set of clauses. Then $\mathcal{C} \vdash \perp$ if and only if $\mathcal{C}$ is unsatisfiable.*

A proof of this theorem is beyond the scope of these notes, though it may be found in most good texts on theorem-proving or AI. (See for example, Leitsch [12], p. 80.) The importance of this theorem is that it provides us with a finite means of *establishing* entailment and satisfiability for sets of clauses.

4.1.3 Ordered resolution

In fact, the resolution rule can be considerably restricted without compromising Theorem 5. The most important restriction for our purposes is provided by a special kind of ordering on atoms. Let $\mathcal{A}$ be the collection of all atoms (over some signature), and suppose that $\prec$ is a partial order on $\mathcal{A}$. We say that $\prec$ is an *A-ordering* if, for every substitution θ , $A \prec A'$ implies $A\theta \prec A'\theta$. If $\prec$ is an A-ordering, then we can extend $\prec$ to an ordering on *literals* by simply ignoring any negation signs and comparing the atoms in question.

For a fixed A-ordering, we take the *ordered resolution* rule to be exactly the same as the resolution rule (4.6), but subject to the added restriction that, for every literal L in C , $A \not\prec L$, and for every literal L' in D , $A' \not\prec L'$. In other words: ordered resolution requires the literals resolved on to be maximal in their respective clauses. Similarly, we take the *ordered factoring* rule to be exactly the same as the factoring rule (4.7), but subject to the added restriction that the literals unified by θ be maximal in C . If $\mathcal{C}$ is a set of clauses and C a clause, I write $\mathcal{C} \vdash_{\prec} C$ to indicate that there is a proof of C from $\mathcal{C}$ using ordered resolution and ordered factoring (with A-ordering $\prec$).

I provide two example of A-orderings: both will be useful in the sequel. The first is very simple. Let $\prec^2$ denote the A-ordering defined by

$$A \prec^2 A' \text{ iff } A \text{ involves a unary predicate and } A' \text{ a binary predicate.}$$

Trivially, $\prec^2$ is invariant under substitutions, and thus is an A-ordering. My second example of an A-ordering is more complicated and requires some preliminary notation.

Let A be any atom. The *term depth* of A , denoted $d(A)$, is the maximum level of functional embedding of any term in that atom, where unembedded terms are counted as having depth 1. Thus: $d(p(a)) = d(p(x)) = d(p'(a, x)) = 1$, $d(p(f(x))) = d(p(f(a))) = 2$, $d(x, p'(h(x), g(x, f(x)))) = 3$ and so on. In addition, for any variable x , the *depth of x in A* , denoted $d(x, A)$, is the maximal functional depth of any occurrence of x in A (and zero if x does not appear in A). Thus, $d(x, p(y)) = 0$, $d(x, p(x)) = 1$, $d(x, p(f(x))) = 2$,

$d(x, p'(h(x), g(x, f(x)))) = 3$ and so on. Now define the relation $\prec^d$ on the set of atoms by:

$A \prec^d A'$ iff $d(A) < d(A')$ and $d(x, A) < d(x, A')$ for all variables x in A .

Exercise 7. Verify that $\prec^d$ is an A-ordering.

The restriction to ordered resolution and factoring does not destroy completeness:

Theorem 6. *Let $\mathcal{C}$ be a set of clauses and $\prec$ an A-ordering. Then $\mathcal{C} \vdash_{\prec} \perp$ if and only if $\mathcal{C}$ is unsatisfiable.*

For a proof of this theorem, see Leitsch [12], p.99 ff.

4.2 The syllogistic fragment

One of the striking features of the resolution rule is the way in which it preserves membership in certain logical fragments.

Definition 1. *A clause is said to be Krom if it contains at most two literals.*

Recall the formulas arising from the English fragment Cop. I established earlier that these are all of the forms:

Every A is a B	$\forall x(a(x) \rightarrow b(x))$	No A is a B	$\forall x(a(x) \rightarrow \neg b(x))$
Some A is a B	$\exists x(a(x) \wedge b(x))$	Some A is not a B	$\exists x(a(x) \wedge \neg b(x))$
S is a B	$b(s)$	S is not a B	$\neg b(s)$

To convert to clause form, we simply drop universal quantifiers, replace existential quantifiers with (new) constants, and re-arrange the remaining Boolean formulas in the obvious way:

$\forall x(a(x) \rightarrow b(x))$	$\Rightarrow$	$\neg a(x) \vee b(x)$	$\forall x(a(x) \rightarrow \neg b(x))$	$\Rightarrow$	$\neg a(x) \vee \neg b(x)$
$\exists x(a(x) \wedge b(x))$	$\Rightarrow$	$a(c), b(c)$	$\exists x(a(x) \wedge \neg b(x))$	$\Rightarrow$	$a(c), \neg b(c)$
$b(s)$	$\Rightarrow$	$b(s)$		$\Rightarrow$	$\neg b(s)$

In particular, all the clauses in question are Krom.

Lemma 1. *Let C and C' be Krom clauses which resolve to form the clause C'' . Then C is also Krom.*

Proof. Obvious. □

I am now in a position to prove Theorem 1.

Theorem. *The satisfiability problem for Cop is in PTIME.*

Proof. Let E be a set of Cop-sentences of size n . Let Φ be the first-order translations of E as defined by Cop. By definition, E is satisfiable if and only if Φ is satisfiable; moreover, we have just seen how to compute, in polynomial time, a set $\mathcal{C}$ of clauses such that $\mathcal{C}$ is satisfiable if and only if Φ is. The clauses $\mathcal{C}$ will clearly be function-free, Krom clauses involving at most n unary predicates and at most n individual constants. But there are at most $4n^4 + 2n^2$ distinct such clauses. By Lemma 1, resolution will produce only more clauses of this form, and will therefore reach saturation in time bounded by a polynomial function of n . $\square$

Exercise 8. The above bound on the number of clauses that need to be considered is not optimal: improve on it.

4.3 Syllogisms with relative clauses

In this section, I show that the problem of determining the satisfiability of a set of sentences of Cop+Rel is NP-complete. Thus, adding relative clauses to the syllogistic fragment may increase the complexity of reasoning. Before giving the proof, I rehearse some simple ideas from logic and complexity theory.

First: logic. Let us say that a formula is *purely universal* if it is of the form $\forall x_1 \dots \forall x_n \phi$ with $n \geq 0$ and ϕ quantifier-free. If $\mathfrak{A}$ is a structure with domain A and $\mathfrak{B}$ is the structure obtained by restricting all the interpretations in $\mathfrak{A}$ to some subset $B \subseteq A$, it is usual to say that $\mathfrak{B}$ is a *substructure* of $\mathfrak{A}$, and write $\mathfrak{B} \subseteq \mathfrak{A}$.

Lemma 2. *Let Φ be a set of purely universal, function-free formulas, let $\mathfrak{A} \models \Phi$ and let $\mathfrak{B} \subseteq \mathfrak{A}$. Then $\mathfrak{B} \models \Phi$.*

Proof. Almost trivial. $\square$

Corollary 1. *Let Φ be a satisfiable set of purely universal function-free formulas and let K be the set of individual constants in Φ . (Assume without loss of generality that K is nonempty.) Then Φ is satisfiable in a structure of size $|K|$.*

Next: complexity theory. Let the problem SAT be as follows:

Instances: Any set $\mathcal{D}$ of propositional clauses.

Question: Is $\mathcal{D}$ satisfiable?

It is known that SAT is NP-hard: any problem which can be nondeterministically solved in polynomial time is polynomially reducible to SAT. This fact standardly goes by the name of Cook's theorem, after Cook [3]. To show that any other problem is NP-hard, it evidently suffices to reduce SAT (or any other known NP-hard problem) to it in polynomial time. Since I shall prove some hardness results below, it may be helpful to give a familiar instance of this procedure. Let 3-SAT be the following problem:

Instances: Any set $\mathcal{D}$ of propositional clauses each of which has at most three literals.

Question: Is $\mathcal{D}$ satisfiable?

Lemma 3. *The problem 3-SAT is NP-hard.*

Proof. Let $\mathcal{C}$ be any set of propositional clauses. I describe a procedure for obtaining, in polynomial time, a set $\mathcal{D}$ of clauses, each of which involves at most three literals, such that $\mathcal{C}$ is satisfiable if and only if $\mathcal{D}$ is satisfiable. This reduces the problem 3-SAT to the problem SAT, and hence establishes the NP-hardness of 3-SAT.

Let $C \in \mathcal{C}$ such that $C = L_1 \vee L_2 \vee D$ where D is a clause having at least two literals. Let p be a new proposition letter (not occurring in $\mathcal{C}$) and set

$$\mathcal{C}' = \mathcal{C} \setminus \{C\} \cup \{p \vee D, \neg p \vee L_1 \vee L_2\}.$$

I claim that $\mathcal{C}$ and $\mathcal{C}'$ are equisatisfiable. Certainly, $\mathcal{C}'$ entails $\mathcal{C}$: hence if $\mathcal{C}'$ is satisfiable, then so is $\mathcal{C}$. Conversely, if $\mathcal{C}$ is satisfiable, extend the satisfying assignment to the new atom p by making p true just in case $L_1 \vee L_2$ is true; the extended assignment must then satisfy $\mathcal{C}'$. Note that we have replaced the clause $C \in \mathcal{C}$ with a shorter clause in $\mathcal{C}'$, while adding no clauses with more than 3 literals. By proceeding in this way, we can eliminate all clauses with more than 3 literals in polynomial time, obtaining an equisatisfiable clause set $\mathcal{D}$. $\square$

I am now in a position to prove Theorem 2.

Theorem. *The problem of determining the satisfiability of a set of sentences in Cop+Rel is NP-complete.*

Proof. To show membership in NP, let E be a finite set of sentences of Cop+Rel, and let Φ be their translations into first-order logic. By definition, E is satisfiable if and only if Φ is satisfiable. Let Φ' be the result of replacing all existentially quantified variables in Φ with new constants as follows:

$$\begin{aligned} \exists x(a(x) \wedge b(x)) &\Rightarrow a(c) \wedge b(c) \\ \exists x(a(x) \wedge \neg b(x)) &\Rightarrow a(c) \wedge \neg b(c) \end{aligned}$$

Clearly, Φ' can be computed in linear time, and is satisfiable if and only if Φ is satisfiable. Thus, it suffices to show that, if Φ' is satisfiable, then it is true in some structure of size bounded by $|\Phi'|$, since such a structure may be guessed and checked in polynomial time. But Φ' is purely universal and function-free, so the result follows by Corollary 1.

To show NP-hardness, we reduce 3-SAT to the problem of determining satisfiability in Cop+Rel. Let $\mathcal{C}$ be a set of propositional clauses, each of which has at most three literals. Without loss of generality, we may assume all the clauses in $\mathcal{C}$ to be of the forms $p \vee q$, $\neg p \vee \neg q$ or $\neg p \vee \neg q \vee r$. We then map each clause in $\mathcal{C}$ to a sentence of Cop+Rel as follows:

$p \vee q$	Every element which is not a q is a p
$\neg p \vee \neg q$	No p is a q
$\neg p \vee \neg q \vee r$	Every p which is a q is an r,

and finally add the Cop+Rel-sentence Some element is an element. Let the resulting set of sentences be E . These sentences translate to first-order logic according to the following table:

Every element which is not a q is a p	$\forall x(\text{element}(x) \wedge \neg q(x) \rightarrow p(x))$
No p is a q	$\forall x(p(x) \rightarrow \neg q(x))$
Every p which is a q is an r	$\forall x(p(x) \wedge q(x) \rightarrow r(x))$
Some element is an element	$\exists x(\text{element}(x) \wedge \text{element}(x)).$

Let Φ be the set of formulas thus obtained. By definition, E is satisfiable if and only if Φ is satisfiable. But it is routine to transform any satisfying assignment for $\mathcal{C}$ into a structure in which Φ is true, and vice versa. This completes the reduction of 3-SAT to Cop+Rel-satisfiability. $\square$

Exercise 9. In the above proof, I claimed:

Without loss of generality, we may assume all the clauses in $\mathcal{C}$ to be of the forms $p \vee q$, $\neg p \vee \neg q$ or $\neg p \vee \neg q \vee r$.

Justify this claim.

4.4 Transitive verbs

I now turn my attention to the fragment Cop+Rel+TV. Again, I establish two results: one yielding an upper bound on the complexity of determining satisfiability, the other a lower bound. Both use the same general approach as for Cop+Rel; however, in both cases, the techniques required are more complicated.

I show first that the satisfiability of a set E of Cop+Rel+TV-sentences can be determined in exponential time. Fix such a set E , and translate into first-order logic in the normal way. An example will help clarify the argument. The sentence

Every stoic whom some cynic despises admires some sceptic
whom every cynic despises

translates to the formula

$$\begin{aligned} \forall x(\text{stoic}(x) \wedge \exists z(\text{cynic}(z) \wedge \text{despise}(z, x)) \rightarrow \\ \exists y(\text{sceptic}(y) \wedge \forall z(\text{cynic}(z) \rightarrow \text{despise}(z, y)) \wedge \text{admire}(x, y))). \end{aligned} \quad (4.8)$$

Now suppose we introduce new unary predicates p and q , and replace (4.8) with the three formulas:

$$\begin{aligned} \forall x(\text{stoic}(x) \wedge p(x) \rightarrow \exists y(\text{sceptic}(y) \wedge q(y) \wedge \text{admire}(x, y))) \\ \forall x(p(x) \leftrightarrow \exists z(\text{cynic}(z) \wedge \text{despise}(z, x))) \\ \forall y(q(y) \leftrightarrow \forall z(\text{cynic}(z) \rightarrow \text{despise}(z, y))). \end{aligned} \quad (4.9)$$

Exercise 10. Verify that: (i) formulas (4.9) entail formula (4.8); and (ii) if $\mathfrak{A}$ satisfies formula (4.8), then, by modifying the interpretations of p and q if necessary, we can ensure that $\mathfrak{A}$ satisfies formulas (4.9). Hence, formula (4.8) is satisfiable if and only if formulas (4.9) are satisfiable.

It is obvious that this process may be repeated as often as we like: even when there are very deeply nested relative clauses, for example, in sentences like

Every stoic whom some cynic despises admires some sceptic whom every cynic
who hates some philosopher despises,

repeated introduction of new unary predicates will produce equisatisfiable formulas in which successive relative clauses have been removed to separate formulas.

Now consider what happens when the formulas (4.9) are converted into clause form. The result from the first formula is:

$$\begin{aligned} & \neg \text{stoic}(x) \vee \neg p(x) \vee \text{sceptic}(f(x)) \\ & \neg \text{stoic}(x) \vee \neg p(x) \vee q(f(x)) \\ & \neg \text{stoic}(x) \vee \neg p(x) \vee \text{admire}(x, f(x)); \end{aligned} \tag{4.10}$$

the others clausify similarly. The crucial observation is that each of the resulting clauses involves at most one literal with a binary predicate. Thus, if E is a set of sentences in Cop+Rel+TV, and Φ is the set of first-order translations of E , then we can equisatisfiably convert Φ , in polynomial time, into a set of clauses $\mathcal{C}$ such that every clause in $\mathcal{C}$ contains at most one literal involving a binary predicate.

Suppose we now saturate $\mathcal{C}$ under resolution, *but we only consider resolutions which eliminate atoms involving binary predicates*; let the resulting set of clauses be $\mathcal{C}^\infty$. Now remove from $\mathcal{C}^\infty$ any clauses containing a binary predicate; let the resulting set of clauses be $\mathcal{C}'$. I claim: (i) $\mathcal{C}$ is satisfiable if and only if $\mathcal{C}'$ is satisfiable; (ii) $\mathcal{C}'$ features only unary (not binary) predicates and only unary function-symbols; (iii) the functional depth of any clause in $\mathcal{C}'$ is at most 1; and (iv) $|\mathcal{C}'|$ is bounded by a quadratic function of $|\mathcal{C}|$.

To see that (i) is true, we note first that $\mathcal{C}$ entails $\mathcal{C}'$; hence, if $\mathcal{C}$ is satisfiable, then so is $\mathcal{C}'$. But now suppose that $\mathcal{C}$ is not satisfiable. Recalling the A-order $\prec^2$ which ranks literals involving binary predicates above those involving unary predicates, Theorem 6 guarantees that there is a proof of $\perp$ using $\prec^2$ -ordered resolution and factoring. But any such proof will simply start by resolving away all the literals involving binary predicates, at which point we will have a proof of $\perp$ from clauses in $\mathcal{C}'$, whence $\mathcal{C}'$ is not satisfiable. This establishes claim (i). Claims (ii) and (iii) are obvious given the clauses we start with in $\mathcal{C}$. Claim (iv) follows from the fact that each clause in $\mathcal{C}$ contains at most one binary predicate.

By (i), E is satisfiable if and only if $\mathcal{C}'$ is satisfiable. By (ii) we can apply the splitting rule to every clause in $\mathcal{C}'$ to obtain clauses involving only one variable. (The splitting rule allows us to replace a clause $\pi \vee \psi$ nondeterministically by π

or ψ provided that π and ψ have no variables in common.) By (iii) the resulting clauses will have functional depth no greater than 1. By (iv) the number of backtracking choices generated by the splitting rule is at most exponential in $|\Phi'|$.

Having split the clauses in $\mathcal{C}'$, we obtain a collection of clauses $\mathcal{C}''$, such that every $C \in \mathcal{C}''$ has the following properties:

1. C contains only unary predicates, unary function-symbols and constants;
2. C contains at most one variable;
3. if C is not ground, then every literal in C contains the variable of C ;
4. the functional depth of any atom occurring in C is at most 1.

It is obvious the first three of these properties are preserved by resolution and factoring. The fourth property, however, is not. In fact, the following example shows that applications of the resolution rule in general increase functional depth:

$$\frac{p(x) \vee q(fx) \qquad \neg r(gx') \vee s(x') \vee \neg q(x')}{p(x) \vee \neg r(gfx) \vee s(fx)};$$

here, both premises have depth 2, but the conclusion has depth 3.

Recall, however, the A-ordering $\prec^d$, and consider what happens when *ordered* resolution under $\prec^d$ is applied to a pair of clauses in our set $\mathcal{C}''$. A little thought shows that the functional depth of the conclusion is bounded by the functional depth of the premises. A formal proof of this claim can of course be given; but it suffices here to illustrate with a representative example:

$$\frac{p(x) \vee q(fx) \qquad \neg r(x') \vee s(x') \vee \neg q(x')}{p(x) \vee \neg r(fx) \vee s(fx)}.$$

Thus, $\prec^d$ -ordered resolution and factoring preserve the four properties of $\mathcal{C}''$ listed above. But if the number of constants, function-symbols and predicates in $\mathcal{C}''$ is bounded by n , the total number of such clauses is bounded by a function $2^{p(n)}$, where p is a fixed polynomial. Thus, $\prec^d$ -ordered resolution and factoring on $\mathcal{C}''$ reaches saturation after an exponential number of steps. This gives us a procedure for checking the satisfiability of $\mathcal{C}'$ and hence of E in exponential time.

I have now shown that satisfiability for the fragment Cop+Rel+TV is in EXPTIME. That is: we have an *upper* bound on complexity. I now turn to find a matching lower bound. Recall that, to say that a problem is EXPTIME-hard is simply to say that any problem which is in EXPTIME can be polynomially reduced to it. It follows that, just as with NP-hardness, one way to show that a problem is EXPTIME-hard is to provide a polynomial reduction of a *known* EXPTIME-hard problem to the problem in question. This is the strategy which I adopt.

The logic K^U is the modal logic K together with an additional universal modality, whose semantics are given by

$\models_w U\phi$ if and only if $\models_{w'} \phi$ for all worlds w' .

The satisfiability problem for K^U is EXPTIME-hard. (I omit the proof here: it is an easy adaptation of the corresponding result for propositional dynamic logic; see, e.g. Harel *et al.* [9]: 216 ff.)

It suffices, therefore, to reduce this problem to satisfiability in Cop+Rel+TV. Let ϕ be a formula of K^U . For convenience, we take V to be the dual modality to U . For every proper or improper subformula ψ of ϕ , let A_ψ be a noun. Let **Es** and **Rs** be verbs. Now define, for each such ψ a set of formulas T_ψ inductively as follows:

$$\begin{aligned} T_p &= \emptyset \text{ if } p \text{ is atomic} \\ T_{\psi \wedge \pi} &= T_\psi \cup T_\pi \cup \{ \text{Every } A_\psi \text{ which is an } A_\pi \text{ is an } A_{\psi \wedge \pi}, \\ &\quad \text{Every } A_{\psi \wedge \pi} \text{ is an } A_\psi, \text{Every } A_{\psi \wedge \pi} \text{ is an } A_\pi \} \\ T_{\neg \psi} &= T_\psi \cup \{ \text{Every element which is not an } A_\psi \text{ is an } A_{\neg \psi}, \\ &\quad \text{No } A_\psi \text{ is an } A_{\neg \psi} \} \\ T_{\diamond \psi} &= T_\psi \cup \{ \text{Every element which Rs some } A_\psi \text{ is an } A_{\diamond \psi}, \\ &\quad \text{Every } A_{\diamond \psi} \text{ Rs some } A_\psi \} \\ T_{V\psi} &= T_\psi \cup \{ \text{Every element which Es some } A_\psi \text{ is an } A_{V\psi}, \\ &\quad \text{Every } A_{V\psi} \text{ Es some } A_\psi \}. \end{aligned}$$

Now let S_ϕ be the collection of Cop+Rel+TV-sentences

$$\{ \text{Every } A_\psi \text{ is an element} \mid \psi \text{ a subformula of } \phi \} \cup \{ \text{Some } A_\phi \text{ is an } A_\phi, \text{Every element Es every element} \}.$$

It is routine to show that ϕ is satisfiable if and only if $T_\phi \cup S_\phi$ is satisfiable.

Thus, I have proved Theorem 3:

Theorem. *The satisfiability problem for Cop+Rel+TV is EXPTIME-complete.*

Finally, I return to the issue of the faithfulness to English usage. I noted above that the grammar for Cop+Rel+TV generates some strings whose status as English sentences is at best marginal, such as, for example, centre-embedded constructions. The question naturally arises as to whether the presence of such sentences affects the complexity result just obtained. After all, it is at least conceivable that, when these sentences are removed, the complexity of determining satisfiability in the resulting fragment becomes *easier*.

In fact, it does not. To see why, consider the above proof of EXPTIME-hardness. I showed that any instance of the EXPTIME-hard problem of K^U -satisfiability can be reduced in polynomial time to an instance of Cop+Rel+TV-satisfiability. But look at the sentence forms which I used in that reduction. Here is the complete list:

Every A which is a B is a C	Every A is a B
Every A which is not a B is a C	No A is a B
Every A which Rs some B is a C	Every A Rs some B
Some A is a B	Every A Rs every B .

There is nothing faintly suspect about any of these sentences: no linguistically motivated tightening of the grammar will exclude them. And in that case, the reduction of the above proof will work unproblematically. Thus, when determining the computational complexity of fragments of natural language, we can often afford to be quite relaxed about the occasional leak in the grammar, even if it is a bad one. With luck, it will be a simple matter to establish that plugging that leak cannot affect the relevant complexity result, as I have just done for Theorem 3.

Chapter 5

Fragments with Anaphora

There are still many simple arguments that cannot be captured by the fragment Cop+Rel+TV. Here is one:

Every artist despises some bureaucrat
Every bureaucrat admires every artist who despises him
Every artist despises some bureaucrat who admires him.

(Let us assume that the pronouns above are resolved intrasententially.) So the next question is what happens to the computational complexity of the satisfiability problem when pronouns (he, him) and reflexives (himself) are admitted.

Let the fragment Cop+Rel+TV+RA (RA for “restricted anaphora”) be defined by adding the following grammar rules to those defining Cop+Rel+TV:

Syntax	Formal lexicon
NP → Reflexive	Reflexive → itself (him/herself)
NP → Pronoun	Pronoun → it (he/she/him/her)

The content lexicon (for nouns and verbs) and the wh-movement rule are carried over from Cop+Rel+TV. Furthermore, we take pronouns and reflexives, which are assumed always to have antecedents in the sentences in which they occur, to be subject to the usual rules of binding theory, and in addition to obey a further restriction explained below. I shall not rehearse binding theory here, referring the reader instead to a standard text, such as Cowper ([4]: 171). For present purposes, we can use our linguistic intuitions to determine which NPs a given reflexive or pronoun can take as antecedent. I also forego a formal account of the semantics for Cop+Rel+TV+RA, in order not to be detained by the technicalities of handling bound-variable anaphora within the framework of Montague semantics. A full semantic analysis of Cop+Rel+TV+RA (with some inessential variations) is given in Pratt-Hartmann [16]; for an approachable general account of bound-variable anaphora and Montague semantics, see Heim and Kratzer ([10]: Chh. 9, 11). Accordingly, I simply assume in the

present paper some mechanism for producing faithful first-order translations of sentences of Cop+Rel+TV+RA along the lines outlined for the other fragments considered above. For example, I expect the above argument to be translated as follows:

$$\frac{\begin{array}{l} \forall x(\text{artist}(x) \rightarrow \exists y(\text{bureaucrat}(y) \wedge \text{despise}(x, y))) \\ \forall x(\text{bureaucrat}(x) \rightarrow \forall y(\text{artist}(y) \wedge \text{despise}(y, x) \rightarrow \text{admire}(x, y))) \end{array}}{\forall x(\text{artist}(x) \rightarrow \exists y(\text{bureaucrat}(y) \wedge \text{admire}(y, x) \wedge \text{despise}(x, y)))}.$$

I now come to the additional restriction on pronoun resolution mentioned above. By way of introduction, consider the sentence

Every artist who employs a caretaker despises every bureaucrat who admires himself.

The phrase-structure of this sentence is shown in Figure 5.1; its translation into first-order logic is

$$\forall x_1(\text{artist}(x_1) \wedge \exists x_2(\text{caretaker}(x_2) \wedge \text{employ}(x_1, x_2)) \rightarrow \forall x_3(\text{bureaucrat}(x_3) \wedge \text{admire}(x_3, x_3) \rightarrow \text{despise}(x_1, x_3))).$$

That the arguments of the predicate *admire* in this formula are identical is due to the use of the reflexive *himself*, which, according to binding theory, must coindex with the NP headed by *bureaucrat*. As we see from Figure 5.1, this NP is the closest NP to the reflexive *himself* in the phrase-structure of the sentence.

By contrast, the sentence

Every artist who employs a caretaker despises every bureaucrat who admires him

exhibits an anaphoric ambiguity, according to whether the antecedent of the pronoun *him* is the NP headed by *artist* or the NP headed by *caretaker*. (The NP headed by *bureaucrat* is not available as a pronoun antecedent here.) The translations of these two readings into first-order logic, are, respectively,

$$\forall x_1(\text{artist}(x_1) \wedge \exists x_2(\text{caretaker}(x_2) \wedge \text{employ}(x_1, x_2)) \rightarrow \forall x_3(\text{bureaucrat}(x_3) \wedge \text{admire}(x_3, x_1) \rightarrow \text{despise}(x_1, x_3))) \quad (5.1)$$

and

$$\forall x_1 \forall x_2(\text{artist}(x_1) \wedge \text{caretaker}(x_2) \wedge \text{employ}(x_1, x_2) \rightarrow \forall x_3(\text{bureaucrat}(x_3) \wedge \text{admire}(x_3, x_2) \rightarrow \text{despise}(x_1, x_3))). \quad (5.2)$$

We see from Figure 5.1 that the NP headed by *artist* is closer to the pronoun, as measured along the edges of the phrase-structure, than is the NP headed by *caretaker*. Given that the NP headed by *bureaucrat* is disallowed as an antecedent to the pronoun in this sentence, the first reading is thus the one in which the pronoun takes its *closest allowed* antecedent.

We notice something else about this first reading. Consider again its translation into first-order logic (5.1). Although there are three variables in the formula, corresponding to the three nouns in the sentence, the variables x_2 and x_3 never occur free in the same subformula. Hence, we can replace x_3 by x_2 , to get the equivalent formula:

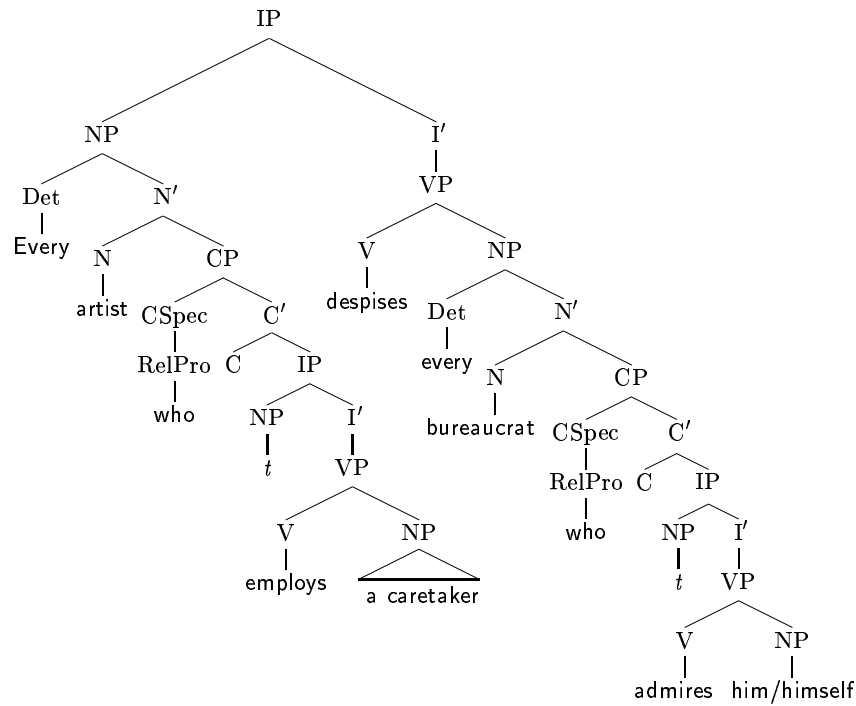


Figure 5.1: Typical phrase-structure in the fragment Cop+Rel+TV+RA

Every thing bees every thing which ays it	$\forall x \forall y (a(x, y) \rightarrow b(y, x))$
Every thing which ays some thing bees itself	$\forall x (\exists y a(x, y) \rightarrow b(x, x))$
Every thing which some thing ays bees itself	$\forall y (\exists x a(x, y) \rightarrow b(y, y))$
Every thing which bees itself ays every thing	$\forall x (b(x, x) \rightarrow \forall y a(x, y))$
Every thing ays every thing which bees itself	$\forall x (b(x, x) \rightarrow \forall y a(y, x))$
Nothing bees some thing which it ays	$\forall x \forall y (a(x, y) \rightarrow \neg b(x, y))$
Every thing bees every thing which it ays	$\forall x \forall y (a(x, y) \rightarrow b(x, y))$
Every thing ays every thing	$\forall x \forall y a(x, y)$
Every thing ays some thing	$\forall x \exists y a(x, y)$
Every thing ays every thing which it does not bee	$\forall x \forall y (\neg b(x, y) \rightarrow a(x, y))$
Every thing which bees some thing which it cees ays it	$\forall x \forall y ((b(x, y) \wedge c(x, y)) \rightarrow a(x, y)).$

Table 5.1: Expressiveness of Cop+Rel+TV+RA

$$\forall x_1 (\text{artist}(x_1) \wedge \exists x_2 (\text{caretaker}(x_2) \wedge \text{employ}(x_1, x_2)) \rightarrow \\ \forall x_2 (\text{bureaucrat}(x_2) \wedge \text{admire}(x_2, x_1) \rightarrow \text{despise}(x_1, x_2))).$$

The reader is invited to verify that no such move is available for (5.2). This observation generalizes: it is shown in Pratt-Hartmann [16], Theorem 1, that every sentence in Cop+Rel+TV+RA translates to a formula which may be written with exactly two variables. (The details of the proof are somewhat tedious.)

Despite this limitation, it should come as no surprise that Cop+Rel+TV+RA is more expressive than Cop+Rel+TV. Table 5.1 lists some Cop+Rel+TV+RA-sentences and the formulas they translate to. In giving these translations, I have suppressed reference to the unary predicate corresponding to the noun *thing* for readability. (This can be shown not to affect anything of substance.) Notice the use of ‘donkey-anaphora’ in the last row of this table. This type of anaphora is permitted in Cop+Rel+TV+RA, subject of course to the restriction that pronouns always take their closest allowed antecedent. In fact, these examples suffice to establish the following theorem:

Theorem 7. *The problem of determining the satisfiability of a set of sentences in Cop+Rel+TV+RA is NEXPTIME-complete.*

Finally, consider what happens when we relax the artificial restriction on anaphora resolution in Cop+Rel+TV+RA, allowing a pronoun to take any allowed antecedent in the sentence in which it occurs. As we have seen, this relaxation results in ambiguous sentences, for example

Every artist who employs a caretaker despises every bureaucrat who admires him.

In order to eliminate this ambiguity, I suppose such sentences to come complete with (allowable) indexation patterns indicating the antecedents of any pronouns. For instance, the above sentence would be replaced by the two sentences:

Every artist_{*i*} who employs a caretaker_{*j*} despises every bureaucrat_{*k*} who
 admires him_{*i*}
 Every artist_{*i*} who employs a caretaker_{*j*} despises every bureaucrat_{*k*} who
 admires him_{*j*}.

Let Cop+Rel+TV+GA (GA for “general anaphora”) be the fragment of ‘English’ thus obtained.

I can now formulate the satisfiability problem for Cop+Rel+TV+GA as before, and ask what its complexity is. Given that the three-variable fragment of first-order logic is undecidable, it is no surprise that the same fate befalls Cop+Rel+TV+GA.

Theorem 8. *The problem of determining the satisfiability of a set of sentences in Cop+Rel+TV+GA is undecidable.*

Actually, a slightly stronger result is shown in Pratt-Hartmann [15], Corollary 1: the entailment problem for the positive fragment of Cop+Rel+TV+GA (no negation or negative determiners) is also undecidable. Again, the details are tedious, and I omit them here.

Finally, I mention that we can improve a little on our result for Cop+Rel+TV+RA. The hardness proof in Theorem 7 proceeds (as expected) by reduction of a known NEXPTIME-hard problem to satisfiability in Cop+Rel+TV+RA. Careful analysis of that reduction shows that Donkey sentences are not actually required. We therefore have

Corollary 2. *Donkeys do not eat MIPS.*

Chapter 6

Concluding Remarks

The technical content of these lectures is easily summarized. The following table lists various English fragments and gives the complexity class of the corresponding satisfiability problem.

Fragment	Computational complexity
Cop	PTIME
Cop+Rel	NP-complete
Cop+Rel+TV	EXPTIME-complete
Cop+Rel+TV+RA	NEXPTIME-complete
Cop+Rel+TV+GA	undecidable.

So one could go on. There are many more fragments of English I could have analysed, of all levels of complexity. The utility of such an analysis from the point of view of natural language engineering, at least in principle, should be obvious. Moreover, the techniques required to carry out analyses of other fragments will be, in essence, those used above: a specification of the fragment in terms of syntax rules, a formal semantics mapping the fragment into first-order logic (or some other formal logic), and the deployment of standard methods of computational complexity theory on the resulting fragment of formal logic.

More generally, my analysis allows us to view the relationship between traditional logic and mathematical logic in a more conciliatory light than has sometimes been the case (Englebretsen [5], Sommers [19]). It would be wrong to think of the logic of the *Principia Mathematica* as being so pitilessly superior to that of the *Prior Analytics* that we can simply forget about the latter. Yes, first-order logic is more expressive than the language of the syllogism; but expressiveness is a double-edged sword, because it correlates, loosely at least, with computational complexity. Indeed, the very recent history of logic, especially within Computer Science, is dominated by the search for logical fragments of limited expressive power whose satisfiability problems are decidable. The kinds of fragments which have drawn most attention, for example various prefix classes (see Börger, Grädel and Gurevich [2] for a survey), the guarded fragment (Andréka, van Ben-

them and N  meti [1], Gr  del [7]) and the two-variable fragment (Mortimer [14], Gr  del and Otto [8]), owe their salience to purely logic-internal considerations. But there is every reason to consider also those logics arising from fragments of natural languages. The syllogistic is one such logic. And if that logic is too inexpressive to be of much practical use, perhaps its natural generalizations are not. I have presented a selection of such generalizations in this paper.

I finish with a parting shot at some alternative approaches. I hope that the foregoing analysis will help lay to rest some appealing but ultimately confused ideas concerning the value of natural-language-friendly logic. According to its proponents, we obtain a better (i.e. more efficient) method of assessing the validity of arguments couched in natural language if we reason within a logical calculus whose syntax is closer to that of natural language than is—say—first-order logic. This idea is attractive because it suggests an ecological dictum: treat the syntax of natural language with the respect it is due, and your inference processes will run faster. Writers apparently expressing support for such views include Fitch [6], Hintikka [11], Suppes [20], Purdy [18] and (perhaps) McAllister and Givan [13]. The observations of this paper lend no support to such views, and indeed cast doubt on them. There is no reason, having identified a fragment of a natural language, why satisfiability within that fragment should not be decided by first translating into first-order logic and then using procedures appropriate to the fragment of first-order logic so obtained. Indeed, from a complexity-theoretic point of view, there is every reason to believe that, for all but the most impoverished fragments, reasoning using schemata based on the syntax of natural language will confer no advantage whatever.

Bibliography

- [1] Hajnal Andréka, Johan van Benthem, and István Németi. Modal languages and bounded fragments of predicate logic. *Journal of Philosophical Logic*, 27(3):217–274, 1998.
- [2] Egon Börger, Erich Grädel, and Yuri Gurevich. *The Classical Decision Problem*. Perspectives in Mathematical Logic. Springer-Verlag, Berlin, 1997.
- [3] S.A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd IEEE Symp. on the Foundations of Computer Science*, pages 151–158, 1971.
- [4] Elizabeth A. Cowper. *A Concise Introduction to Syntactic Theory*. University of Chicago Press, Chicago, 1992.
- [5] George Englebretsen. *Three Logicians*. Van Gorcum, Assen, 1981.
- [6] Frederic B. Fitch. Natural deduction rules for English. *Philosophical Studies*, 24:89–104, 1973.
- [7] E. Grädel. On the restraining power of guards. *Journal of Symbolic Logic*, 64:1719–1742, 1999.
- [8] Erich Grädel and Martin Otto. On logics with two variables. *Theoretical Computer Science*, 224(1–2):73–113, 1999.
- [9] David Harel, Dexter Kozen, and Jerzy Tiuryn. *Dynamic Logic*. MIT Press, Cambridge, MA., 2000.
- [10] I. Heim and A. Kratzer. *Semantics in Generative Grammar*. Blackwell, Oxford, 1998.
- [11] Jaakko Hintikka. Quantifiers vs quantification theory. *Inquiry*, 5:153–77, 1974.
- [12] A. Leitsch. *The Resolution Calculus*. Springer, Berlin, 1997.
- [13] David A. McAllester and Robert Givan. Natural language syntax and first-order inference. *Artificial Intelligence*, 56:1–20, 1992.

- [14] M. Mortimer. On languages with two variables. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 21:135–140, 1975.
- [15] Ian Pratt-Hartmann. On the semantic complexity of some fragments of English. Technical Report UMCS-00-5-1, University of Manchester Department of Computer Science, Manchester, 2000.
- [16] Ian Pratt-Hartmann. A two-variable fragment of English. *Journal of Logic, Language and Information*, 12:13–45, 2003.
- [17] Ian Pratt-Hartmann. Fragments of language. *Journal of Logic, Language and Information*, 13:207–223, 2004.
- [18] William C. Purdy. A logic for natural language. *Notre Dame Journal of Formal Logic*, 32(3):409–425, 1991.
- [19] Fred Sommers. *The Logic of Natural Language*. Clarendon Press, Oxford, 1982.
- [20] Patrick Suppes. Logical inference in English: a preliminary analysis. *Studia Logica*, 38:375–391, 1979.