Computational Complexity of Controlled Natural Languages (Extended abstract)

Ian Pratt-Hartmann

University of Manchester,
Manchester M13 9PL, UK,
ipratt@cs.man.ac.uk

1 Introduction

A *controlled natural language* is a precisely delineated fragment of some natural language (usually English), developed for the purpose of supporting some technical activity—such as process specification [5, 6], hardware specification [4], database querying [1] or data-schema specification [2, 12]. The intention is that the controlled natural language should provide an easy-to-use interface to some underlying logical formalism, within which certain procedures—such as query-answering, model-checking or determining satisfiability or entailment—can then be executed. The question therefore arises as to how the computational complexity of these logical procedures depends on the grammar of the controlled natural language through which their input is channelled.

In this talk, I shall investigate the complexity of determining logical relationships within controlled natural languages featuring a variety of grammatical constructions. The constructions considered here are largely motivated by *Attempto Controlled English* [3]; however, the analysis is intended to apply to (just about) any conceivable controlled natural language. Most of the results mentioned in this talk have already been published elsewhere: its primary contributions are to organize these results into a coherent framework, and to make them accessible to the controlled natural language community.

Formally, we take a *language* to be a mapping from strings (over some alphabet) to sets of formulas (in some logic). A string mapped to a non-empty set of formulas is a *sentence* of the language in question, and the formulas to which it is mapped are the possible *meanings* of that sentence. In the special case where no sentence is given more than one meaning, the logical concepts of satisfiability and entailment carry over naturally from logic to language: a set of sentences E is *satisfiable* if the formulas Φ to which they translate are satisfiable (in the usual logical sense); and E *entails* a sentence e if the formula to which e translates is entailed (in the usual logical sense) by Φ. For any language defined in this way we may ask: what is the computational complexity of determining satisfiability and entailment in that language? In the sequel, we provide answers to these questions for a range of such languages.

2 Languages with the copula

We begin with the simplest possible controlled natural languages: those whose sentences are all of the forms Some p $\mathsf{is\ a}$ q, Every p $\mathsf{is\ a}$ q or No p $\mathsf{is\ a}$ q. Here, p and q are taken from a countably infinite set of count-nouns, such as artist, $\mathsf{beekeeper}$, $\mathsf{carpenter}$ *etc*. We call this this fragment of English $\mathcal{S}^-$. Ignoring some minor grammatical details, $\mathcal{S}^-$ may be defined using a *semantically annotated context-free grammar*, thus:

$$\begin{array}{ll}
\mathrm{S}/\lambda y_1 \lambda y_2.(y_1\ y_2) \rightarrow \mathrm{NP,\ VP} & \mathrm{Det}/\lambda x_1 \lambda x_2.((\exists\ x_1)\ x_2) \rightarrow \mathsf{some} \\
\mathrm{NP}/\lambda y_1 \lambda y_2.(y_1\ y_2) \rightarrow \mathrm{Det,\ N'} & \mathrm{Det}/\lambda x_1 \lambda x_2.((\forall\ x_1)\ x_2) \rightarrow \mathsf{all} \\
\mathrm{VP}/\lambda y_1.y_1 \rightarrow \mathsf{is,\ a,}\ \ \mathrm{N'} & \mathrm{Det}/\lambda x_1 \lambda x_2.((\forall\ x_1)\ (\lambda x.(\neg\ (x_2\ x)))) \rightarrow \mathsf{no} \\
\mathrm{N'}/\lambda y_1.y_1 \rightarrow \mathrm{N} & \mathrm{N}/p_i \rightarrow \mathsf{p}_i\ (i = 1, 2, \ldots).
\end{array}$$

The semantic annotations in the rule-heads are expressions of the simply-typed lambda calculus with constants. We denote the type of domain objects by e and the type $\{\top, \bot\}$ of truth-values by t. If τ_1 and τ_2 are types, then $\langle \tau_1\ \tau_2 \rangle$ is the type of functions from τ_1 to τ_2. The symbol $\forall$ is the obvious logical constant of type $\langle\langle e\ t\rangle\ \langle\langle e\ t\rangle\ t\rangle\rangle$ (similarly, *mutatis mutandis*, for $\exists$ and $\neg$), and the symbols p_1, p_2, ... are non-logical constants (*Urelemente*) of type $\langle e\ t\rangle$, representing the meanings of the count nouns p_1, p_2, ... Strings are parsed in the normal way; and during parsing, the meaning of a phrase is computed by applying the semantic annotation on the relevant rule to the already-computed meanings of the non-terminals on its right-hand side, in left-to-right order. It is routine to verify that the above grammar produces (following β-reduction and conversion to first-order syntax) the familiar first-order translations for sentences of $\mathcal{S}^-$.

Now define the language $\mathcal{S}$ to comprise all the sentences of $\mathcal{S}^-$ together with Some p $\mathsf{is\ not\ a}$ q, Every p $\mathsf{is\ not\ a}$ q and No p $\mathsf{is\ not\ a}$ q, to which it assigns the expected meanings. (The relevant defining grammar rule is easy to formulate.) The language $\mathcal{S}$ is, in effect, the language of the classical syllogistic. We can increase expressive power further by allowing the (slightly artificial) construction non- in noun-phrases, with the interpretation that a non-p is simply anything which is not a p. This gives us, amongst other things, the sentence-forms $\mathsf{Some\ non}$-p $\mathsf{is\ not\ a}$ q and $\mathsf{Every\ non}$-p $\mathsf{is\ a}$ q, which are not logically equivalent to any $\mathcal{S}$-sentences. We call this language $\mathcal{S}^\dagger$.

The satisfiability problem for $\mathcal{S}^\dagger$ is essentially the same as 2-SAT (the satisfiability problem for propositional clauses with at most two literals). Thus, it is routine to show:

Theorem 1. *The problem of determining the satisfiability of a set of sentences in any of the languages $\mathcal{S}^-$, $\mathcal{S}$ or $\mathcal{S}^\dagger$ is* NLOGSPACE*-complete.*

Let us consider the addition of adjectives. We define the language $\mathcal{S}^-\mathcal{A}$ by augmenting the grammar rules for $\mathcal{S}^-$ with

$$\begin{array}{ll}
\mathrm{N'}/\lambda y_1 \lambda y_2 \lambda x.(\wedge\ (y_1\ x)\ (y_2\ x)) \rightarrow \mathrm{A,\ N'} & \mathrm{VP}/\lambda y_1.y_1 \rightarrow \mathsf{is,}\ \ \mathrm{A} \\
\mathrm{A}/a_i \rightarrow \mathsf{a}_i, &
\end{array}$$

where a_1, a_2, ... are adjectives, having meanings a_1, a_2, ... of type $\langle e\ t\rangle$. Thus, $\mathcal{S}^-\mathcal{A}$ includes sentences such as Every tall intelligent artist is a beekeeper or No carpenter is tall, with adjectives taken to have intersective semantics. The languages $\mathcal{SA}$ and $\mathcal{S}^\dagger\mathcal{A}$ may be defined analogously, using the additional rule

$$\mathrm{VP}/\lambda y\lambda x.(\neg\ (y_1\ x)) \rightarrow \textsf{is},\ \textsf{not},\ \ \mathrm{A}.$$

The satisfiability problem for $\mathcal{SA}$ is essentially the same at the satisfiability problem for propositional Horn clauses. Thus, it is routine to show:

Theorem 2. *The problem of determining the satisfiability of a set of sentences in either of the languages $\mathcal{S}^-\mathcal{A}$ or $\mathcal{SA}$ is* PTIME*-complete. The problem of determining the satisfiability of a set of sentences in the language $\mathcal{S}^\dagger\mathcal{A}$ is* NPTIME*-complete.*

Next, we consider languages with relative clauses. We define $\mathcal{S}^-\mathcal{W}$, $\mathcal{SW}$ and $\mathcal{S}^\dagger\mathcal{W}$ by adding to $\mathcal{S}^-$, $\mathcal{S}$ and $\mathcal{S}^\dagger$ the grammar rules

$$\mathrm{N}'/\lambda y_1\lambda y_2\lambda x.(\wedge\ (y_1\ x)\ (y_2\ x)) \rightarrow \mathrm{N},\ \textsf{which},\ \textsf{is},\ \textsf{a},\ \mathrm{N}$$
$$\mathrm{N}'/\lambda y_1\lambda y_2\lambda x.(\wedge\ (y_1\ x)\ (\neg\ (y_2\ x))) \rightarrow \mathrm{N},\ \textsf{which},\ \textsf{is},\ \textsf{not},\ \textsf{a},\ \mathrm{N}.$$

Theorem 3. *The problem of determining the satisfiability of a set of sentences in any of the languages $\mathcal{S}^-\mathcal{W}$, $\mathcal{SW}$ or $\mathcal{S}^\dagger\mathcal{W}$ is* NPTIME*-complete.*

Notice that these rules do not permit nesting of relative clauses, thus avoiding ambiguous and unnatural noun-phrases such as artist who is not a beekeeper who is not a carpenter. In fact, allowing embedded relative clauses does not change the complexity results reported in Theorem 3. Adjectives can be added to these languages as well, resulting in languages $\mathcal{S}^-\mathcal{AW}$, $\mathcal{SAW}$ and $\mathcal{S}^\dagger\mathcal{AW}$, defined in the (more or less) obvious way. It is easily seen that this does not increase the complexity of satisfiability either.

More difficult to analyse is the effect of adding numerical quantifiers. Define the language $\mathcal{S}^-\mathcal{Q}$ to feature sentences of the forms At least C p are q or At most C p are q, where C is a string of decimal digits representing a natural number; and define $\mathcal{SQ}$ and $\mathcal{S}^\dagger\mathcal{Q}$ analogously. (We ignore the issue of plural inflections.)

Theorem 4 ([9]). *The problem of determining the satisfiability of a set of sentences in any of the languages $\mathcal{S}^-\mathcal{Q}$, $\mathcal{SQ}$ or $\mathcal{S}^\dagger\mathcal{Q}$ is* NPTIME*-complete.*

Adding adjectives and relative clauses to these languages can be shown not to affect the complexity of satisfiability.

3 Languages with transitive verbs

The languages considered so far are too trivial to be of much practical use, since they feature no relations of arity greater than 1. Accordingly, let us define the language $\mathcal{R}^-$ by augmenting $\mathcal{S}^-$ with sentences involving transitive verbs, such as Every boy loves some girl or No boy loves no girl. Helping ourselves to a countable set of transitive verbs r_1, r_2, ..., and corresponding binary predicates r_1, r_2, ..., this can be achieved by means of the additional grammar rules

$$\mathrm{VP}/\lambda y_1 \lambda y_2.(y_1\ y_2) \rightarrow \mathrm{V},\ \mathrm{NP} \qquad \mathrm{V}/\lambda x_1 \lambda x_2.(x_1\ \lambda x_3.((r_i\ x_2)\ x_3)) \rightarrow \mathsf{r}_i.$$

We can add expressive power by allowing verb-level negation. A rough-and-ready attempt at this would be to take the rules for $\mathcal{S}$ and $\mathcal{R}^-$ together with

$$\mathrm{S}/\lambda y_1 \lambda y_2.(y_1\ y_2) \rightarrow \mathrm{NP},\ \mathrm{NegP} \qquad \mathrm{NegP}/\lambda y_1 \lambda x.(\neg\ (y_1\ x)) \rightarrow \mathsf{does},\ \mathsf{not},\ \mathrm{VP}.$$

Let us call this language $\mathcal{R}$. These rules are very leaky. For one thing, they ignore the need for the negative polarity determiner any in No boy loves any girl; in addition, they accept strange sentences such as No boy does not love some girl (which is assigned the same meaning as Every boy loves some girl). However, these details are easily corrected, and anyway have no effect on the complexity of the satisfiability problem.

Theorem 5 ([10]). *The problem of determining the satisfiability of a set of sentences in either of the languages $\mathcal{R}^-$ or $\mathcal{R}$ is* NLOGSPACE*-complete.*

Adding the non-construction, however, produces an unexpected jump in complexity. Let $\mathcal{R}^\dagger$ be the language defined in the same way as $\mathcal{R}$, but allowing 'negated' subjects and objects of transitive verbs, such as Every non-artist admires some non-beekeeper.

Theorem 6 ([10]). *The problem of determining the satisfiability of a set of sentences in the language $\mathcal{R}^\dagger$ is* EXPTIME*-complete.*

Relative clauses have a similar effect in the presence of transitive verbs. Define the language $\mathcal{R}^-\mathcal{W}$ by adding suitable rules for relative clauses to $\mathcal{R}^-$. Thus, $\mathcal{R}^-\mathcal{W}$ contains sentences such as Every artist who admires every carpenter admires some beekeeper. Define $\mathcal{R}\mathcal{W}$ and $\mathcal{R}^\dagger\mathcal{W}$ analogously.

Theorem 7 ([8]). *The problem of determining the satisfiability of a set of sentences in any of the languages $\mathcal{R}^-\mathcal{W}$, $\mathcal{R}\mathcal{W}$ or $\mathcal{R}^\dagger\mathcal{W}$ is* EXPTIME*-complete.*

Numerical quantifiers have a greater effect on the complexity of satisfiability. Let the language $\mathcal{R}^-\mathcal{Q}$ be obtained by augmenting $\mathcal{R}^-$ with numerical quantification. Thus, $\mathcal{R}^-\mathcal{Q}$ contains sentences such as At most 13 artists admire at least 4 beekeepers. Define $\mathcal{R}\mathcal{Q}$ and $\mathcal{R}^\dagger\mathcal{Q}$ analogously.

Theorem 8 ([9]). *The problem of determining the satisfiability of a set of sentences in any of the languages $\mathcal{R}^-\mathcal{Q}$, $\mathcal{R}\mathcal{Q}$ or $\mathcal{R}^\dagger\mathcal{Q}$ is* NEXPTIME*-complete.*

Adding adjectives to most of the above languages involving transitive verbs can be shown not to affect the complexity of satisfiability.

4 Languages with other constructions

Languages involving ditransitive verbs can be defined in exactly the same way as for transitive verbs. For example, let $\mathcal{D}^-$ is defined analogously to $\mathcal{R}^-$, but admits sentences such as Every artist introduces some beekeeper to some carpenter. Only one result has been obtained in this case:

Theorem 9 ([11]). *The problem of determining the satisfiability of a set of sentences in the language $\mathcal{D}^-$ is in* PTIME.

Finally, we consider languages featuring bound-variable anaphora (subject to various restrictions). In [7], a very simple controlled natural language involving transitive verbs, relative clauses and restricted anaphora is presented, and shown to have a NEXPTIME-complete satisfiability problem. The satisfiability problem for the same language, but with ditransitive verbs, is shown in [11] to be undecidable.

References

1. I. Androutsopoulos, G.D. Ritchie, and P. Thanisch. Natural language interfaces to databases—an introduction. *Natural Language Engineering*, 1(1):29–81, 1995.
2. N. E. Fuchs, U. Schwertel, and S. Torge. Controlled natural language can replace first-order logic. In *14th IEEE International Conference on Automated Software Engineering*, pages 295–298. IEEE Computer Society Press, 1999.
3. Norbert E. Fuchs, Uta Schwertel, and Rolf Schwitter. Attempto Controlled English—not just another logic specification language. In Pierre Flener, editor, *Logic-Based Program Synthesis and Transformation*, volume 1559 of *Lecture Notes in Computer Science*, pages 1–20, Berlin, June 1999. Eighth International Workshop LOPSTR'98, Springer.
4. Alexander Holt and Ewan Klein. A semantically-derived subset of English for hardware verification. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 451–456. Association for Computational Linguistics, June 1999.
5. Benjamin Macias and Stephen Pulman. A method for controlling the production of specifications in natural language. *The Computer Journal*, 38(4):310–318, 1995.
6. R. Nelken and N. Francez. Automatic translation of natural-language system specifications into temporal logic. *Lecture Notes in Computer Science*, 1102:360–371, 1996.
7. I. Pratt-Hartmann. A two-variable fragment of English. *Journal of Logic, Language and Information*, 12:13–45, 2003.
8. Ian Pratt-Hartmann. Fragments of language. *Journal of Logic, Language and Information*, 13:207–223, 2004.
9. Ian Pratt-Hartmann. On the computational complexity of the numerically definite syllogistic and related logics. *Bulletin of Symbolic Logic*, 14(1):1–28, 2008.
10. Ian Pratt-Hartmann and Lawrence S. Moss. Logics for the relational syllogistic. ArXiv preprint server, `http://arxiv.org/abs/0808.0521`, 2008.
11. Ian Pratt-Hartmann and Allan Third. More fragments of language. *Notre Dame Journal of Formal Logic*, 47(2):151–177, 2006.
12. Sunil Vadera and Farid Meziane. From English to formal specifications. *The Computer Journal*, 37(9):753–763, 1994.