Natural Language Syntax and First Order Inference

David McAllester
Robert Givan

A.I. Memo No. 1176 October, 1989

Abstract:
We have argued elsewhere that first order inference can be made more efficient by using non-standard syntax for first order logic. In this paper we show how a fragment of English syntax under Montague semantics provides the foundation of a new inference procedure. This procedure seems more effective than corresponding procedures based on either classical syntax or our previously proposed taxonomic syntax. This observation may provide a functional explanation for some of the syntactic structure of English.

Keywords: Taxonomy, Knowledge Representation, Theorem Proving, Automated Reasoning, Natural Language, Montague Semantics.

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part the National Science Foundation contract IRI-8819624 and in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-86-K-0124.

©1989 Massachusetts Institute of Technology

1 Introduction

In [McAllester *et al.*, 1989] we argued that first order theorem proving could be made more effective by replacing the classical syntax of first order logic with a syntax based on taxonomic relationships. This claim is supported by a technical theorem stating that a certain quantifier-free fragment of the new taxonomic syntax has a polynomial time decision procedure, and furthermore, this quantifier-free fragment of taxonomic syntax is more expressive than any corresponding quantifier-free fragment of classical syntax. The decision procedure for the quantifier-free fragment of taxonomic syntax can be incorporated into a high-level proof system which is complete for first order inference.

The technical observations about taxonomic syntax can be extended to yet more powerful syntaxes for first order logic. In this paper we investigate a syntax derived from a fragment of English under Montague semantics. This "Montague syntax" for first order logic is quite similar to the taxonomic syntax presented in [McAllester *et al.*, 1989] except that it allows the "quantifiers" *some* and *every* to appear as noun phrase specifiers. The sentence "every man loves some woman" is a well formed formula in Montague syntax for first order logic. Although this sentence appears to have two quantifiers, both of these quantifiers appear as specifiers of noun phrases and neither quantifier involves a bound variable. In Montague syntax for first order logic the noun phrase specifiers "some" and "every" are not considered to be quantifiers and the above sentence is taken to be quantifier-free. We show that a certain case of quantifier-free inference in Montague syntax can be decided in polynomial time. No translation to classical syntax is needed. In fact, the inference procedure relies heavily on the syntactic structure of Montague formulas.

Our previous work on taxonomic syntax and our present work on Montague syntax can be viewed as an extension of work by other researchers on knowledge representation languages, e.g. [Bobrow and Winograd, 1977], [Fahlman, 1979], [Brachman *et al.*, 1983], [Brachman, 1983]. Knowledge representation languages have traditionally been organized around taxonomic relationships between classes. AI researchers often express the intuition that

such taxonomic representations facilitate inference and our previous work on taxonomic syntax for first order inference provides some degree of technical support for this intuition. Our work on alternate syntaxes for first order logic can also be viewed as an extension of work on the use of types, or sorts, in making first order inference more effective [Walther, 1984b], [Walther, 1984a], [Stickel, 1985], [McAllester, 1989]. There also seems to be some relationship between our work on taxonomic syntax and the notion of type used in computer programming languages [Reynolds, 1974], [Burstall, 1984], [Cardelli, 1984]. The current paper was, of course, directly inspired by Montague semantics for English [Dowty *et al.*, 1983]. By restricting this paper to a small formal fragment of English we avoid most of the subtle issues that have traditionally concerned researchers in Montague semantics — we make no attempt to deal with propositional attitudes or the intension/extension distinction.

The decision procedure we give for determining satisfiability of certain classes of Montague formulas is based on an algorithm we call noun phrase subsumption. The noun phrase subsumption algorithm is a modification of the monotone closure procedure described in [McAllester *et al.*, 1989], which can itself be viewed as a generalization of the congruence closure algorithm described in [Downey *et al.*, 1980]. The noun phrase subsumption algorithm can also be viewed as another fast decision procedure for a restricted inference problem in the same spirit as the algorithms described in [Nelson and Oppen, 1979] and [Nelson and Oppen, 1980].

The technical results in this paper can be evaluated along two independent dimensions. First, the results can be evaluated from a purely engineering viewpoint. From this perspective the important contribution is a new inference procedure, noun phrase subsumption, that can be used to improve automated reasoning systems. It should be possible to use the noun phrase subsumption procedure to reduce the amount of user-provided detail needed in the machine verification of mathematical theorems. From this engineering viewpoint the fact that noun phrase subsumption operates on English syntax is irrelevant. In addition to evaluating the engineering implications of the noun phrase subsumption algorithm, one can, of course, evaluate this work in terms of the insight it provides into the observed structure of natural language. The existence and effectiveness of the noun phrase subsumption al-

gorithm suggests that natural language syntax plays an important functional role in human cognition.

2 English Syntax for First Order Logic

In this section we present a formal language based on English syntax. This formal language is really just a syntactic variant of first order predicate calculus — every formula of our new language is equivalent to some formula of first order logic and every formula of first order logic is (essentially) equivalent to a formula in our new logic. We call this new language the Montague syntax for first order logic. The quantifier-free fragment of Montague syntax is more expressive than the quantifier-free fragment of either classical syntax or our earlier taxonomic syntax. In spite of the increased expressive power of the quantifier-free fragment, however, the quantifier-free fragment of Montague syntax retains most of the nice computational properties of the quantifier-free fragment of classical syntax.

Classical syntax involves terms and formulas. In both taxonomic and Montague syntax terms are replaced by class expressions where each class expression denotes a subset of the first order domain. For compatibility with English, Montague syntax allows for two different kinds of class expressions: noun phrase class expressions and verb phrase class expressions. Each class expression, whether noun or verb, denotes a subset of the first order domain. In addition to class expressions and formulas, Montague syntax has a new kind of expression called a maximal noun phrase. If `man` is a noun class token, and hence a noun phrase class expression, then the expressions `(every man)` and `(some man)` are maximal noun phrases. An atomic formula of Montague syntax consists of a maximal noun phrase followed by a verb phrase class expression. The various kinds of expressions of Montague syntax can be defined by mutual recursion as follows:

- A *noun phrase class expression* is any one of the following:
  - A noun class token such as `man`, `bird`, or `natural-number`.

- An application `(R N)` where `R` is a noun relation token, such as `brother-of` or `employee-of`, and `N` is a maximal noun phrase.
- A such-that expression of the form `(N` x `such that` $\Phi(x)$`)` where `N` is an noun phrase class expression, x is a variable, and $\Phi(x)$ is a formula.

- A *maximal noun phrase* is one of the following:
  - A variable. Variables will be denoted with the symbols `x`, `y` and `z`.
  - A constant symbol such as `John`.
  - An expression of the form `(every N)` or `(some N)`, where `N` is an noun phrase class expression.

- A *verb phrase class expression* is an application `(V N)` where `V` is a verb relation token and `N` is a maximal noun phrase.

- A *formula* is either an expression of the form `(NP VP)`, where `NP` is a maximal noun phrase and `VP` is a verb phrase class expression, or a Boolean combination of formulas.

Formulas of the form `(NP VP)` are called atomic formulas—these and their negations `(NOT (NP VP))` will be called literals. For example, if `man` and `woman` are noun class tokens, and `loves` is a transitive verb token, then the expressions

```
((every man) (loves (some woman)))

((some man) (loves (every woman)))

((some (employee-of (some company))) (loves (every woman)))
```

are all atomic formulas of Montague syntax for first order logic. Furthermore, these formulas are taken to be quantifier-free. We consider the essence of quantification to be the presence of bound variables. Such-that class expressions allow for the introduction of bound variables in our Montague syntax. For example, consider the statement that every man loves himself. This would be expressed in classical syntax as

$$\forall x \, \texttt{man}(x) \rightarrow \texttt{loves}(x, x).$$

In Montague syntax this statement can be expressed as the formula

```
((every man) (is (some (man x such that (x (loves x))))))
```

An expression of Montague syntax will be called quantifier-free if it does not contain any such-that expressions.

For the sake of readability it is often convenient to use the tokens "**a**" and "**an**" as synonyms for the token "**some**". For example, the above formulas could be written as

```
((every man) (loves (a woman)))

((a man) (loves (every woman)))

((an (employee-of (a company))) (loves (every woman))).
```

The formal semantics given in the next section assigns each of the above formulas a meaning that is consistent with reading these formulas as English.

Just as in first order logic, we assume that equality is included as a fixed logical constant. In Montague syntax we use the token "IS" to denote the equality relation. The token IS is a verb relation token. Given this verb relation token Montague syntax includes the atomic formulas

```
((every dog) (is (a mammal)))

((every poodle) (is (a dog)))

((some mammal) (is (a (child-of (a reptile))))).
```

Under the semantics given in the following section, all these formulas are assigned meanings consistent with reading them as English.

To ensure that first order Montague syntax is as expressive as classical first order logic, a special logical constant THING is included as a noun class token. In any semantic interpretation the class token THING denotes the entire universe of discourse of that interpretation (the semantics of the constant THING is given in more detail in the next section). Using this constant we can construct atomic formulas that refer to all things. For example, consider the formula (John (hates (every THING))).

Unfortunately, some formulas of Montague syntax are ambiguous when read as English sentences. Our formal semantics assigns every formula a unique unambiguous meaning. The meaning assigned by the formal semantics is not always the same as the most common reading of the corresponding English sentence. For example, because "**a**" and "**some**" are synonymous, the formulas

```
((some dog) (is (a mammal)))
```

and `((a dog) (is (a mammal)))`

are identical. The second formula, being identical to the first, has a different formal meaning from the formula `((every dog) (is (a mammal)))`. This example shows that first order Montague syntax, as a formal language, is not intended to be completely faithful to the meaning of English, although the correspondence is quite good in general. Independent of any relationship to English, first order Montague syntax can be viewed as an engineering tool for improving the effectiveness of automated reasoning by making the quantifier-free fragment of the language more expressive while preserving its computational tractability.

3 Formal Semantics

Our formal semantics for the Montague syntax is a simplification of Montague's original semantics [Dowty *et al.*, 1983]. Just as in classical syntax, a model of first order Montague syntax is a first order structure, i.e., a domain D together with an interpretation of various tokens as elements of D, subsets of D, or relations on D. Each first order structure assigns, to each constant such as `John`, a meaning which is an element of the domain of that structure. Noun class tokens, such as `man` and `mammal`, have the same semantics as monadic predicates in classical syntax: each first order structure assigns to each noun class expression a meaning which is a particular subset of its domain. Noun relation tokens, and verb relation tokens, both have the same semantics as binary relation symbols in classical syntax: each model assigns to each noun and verb relation token a particular relation, i.e., set of pairs, over the domain of that structure. For example, the noun relation token `brother-of` and the verb relation token **loves** both denote relations

on the domain of the structure. We require that in every model of first order Montague syntax the verb relation token `IS` denotes the identity relation on the first order domain and the noun class token `THING` denotes the entire domain.

In the following specification of the semantics of Montague syntax we assume that a fixed first order structure and a particular variable interpretation have been specified. A variable interpretation over a first order structure M assigns each variable a meaning which is a member of the domain of M. Given a first order structure and a variable interpretation we can assign a meaning to each class expression, where the meaning is a subset of the semantic domain, and assign a truth value to each formula. We do not give an independent meaning to maximal noun phrases, although we do give meaning to all class expressions and formulas that involve maximal noun phrases. The reason for not giving an independent meaning for maximal noun phrases is discussed below.

First we give the meaning of class expressions. A class token (either a noun token or a verb token) denotes the set that is assigned to it by the first order structure. To define the meaning of other class expressions some additional terminology is needed. We view relations as operators that take a single argument and return a set. If `R` is a relation token, either a noun relation token or a verb relation token, and d is an element of the semantic domain, then we will use the notation (`R` d) to denote the set of domain elements d' such that the pair $\langle d', d \rangle$ is a member of the relation denoted by `R`. The meaning of a class expression (`R N`) where `R` is a relation token (either verb or noun) and `N` is a maximal noun phrase depends on the kind of maximal noun phrase involved. If `N` is a constant or variable denoting domain element d, then the class expression (`R N`) denotes the set (`R` d). For example, the class expression (`brother-of John`) denotes the set of all brothers of John, and the class expression (`loves Mary`) denotes the set of all people who love Mary. A class expression of the form (`R (some C)`) denotes the union of all sets (`R` d) where d is a member of the set denoted by `C`. If `C` denotes the empty set, then so does (`R (some C)`). For example, the class (`brother-of (some person)`) denotes the set of all people that are the brother of some person, and (`loves (some person)`) denotes the class of all people who love some person. A class expression of the form (`R (every`

C)) denotes the intersection all sets of the form (R d) where d is in the set denoted by C. If C denotes the empty set then (R (every C)) denotes the entire semantic domain. For example, if d is a member of the class denoted by (brother-of (every person)) then d must be a brother of every person. Similarly, if d is a member of the class (loves (every person)), then d must love every person. Finally, consider a class expression of the form (N x such that $\Phi(x)$). This expression denotes the set of all elements d of the class denoted by N such that the formula $\Phi(x)$ is true when x is interpreted as d. For example, the phrase (man x such that (x (loves x))) denotes the class of all men who love themselves.

Given that every class expression denotes a well defined subset of the first order domain, we can readily define the meaning of any Montague formula. Every Montague formula is either an atomic formula or a Boolean combination of atomic formulas. Boolean combinations are given their standard meaning, and so it suffices to assign meaning to the atomic formulas. An atomic formula is always an expression of the form (NP VP) where NP is a maximal noun phrase and VP is a verb phrase class expression. The meaning of such a formula expression depends on the kind of maximal noun phrase involved. If NP is a variable or constant symbol then the formula (NP VP) is true just in case the object denoted by NP is a member of the class denoted by VP. A formula of the form ((every N) VP) is true just in case every member of the class denoted by N is a member of the class denoted by VP, i.e., just in case the class denoted by N is a subset of the class denoted by VP. A formula of the form ((some N) VP) is true just in case some member of the class denoted by N is also a member of the class denoted by VP, i.e., just in case the class denoted by N intersects the class denoted by VP.

We leave it to the reader to verify that, if we restrict our attention to languages with only constants and unary and binary relation symbols, then every classical first order formula can be translated to a logically equivalent formula of Montague syntax and every formula of Montague syntax can be translated to an equivalent formula in classical first order syntax. Thus first order Montague syntax is really just a syntactic variant of first order logic. Binary relation symbols, in the presence of equality, are in some sense sufficient to express arbitrary first order facts. Thus Montague syntax as defined here seems sufficiently expressive. However, the usefulness of Montague syntax in

speeding up automated reasoning can be enhanced by allowing for function symbols and for both functions and relations of unbounded arity (number of arguments). This extension of Montague syntax is discussed briefly in a later section.

The above semantics assigns a meaning to class expressions (as sets) and to formulas (as truth values) but does not assign any independent meaning to maximal noun phrases. This was done to simplify the conceptual complexity of the semantics. As Montague observed, it is possible to give maximal noun phrases an independent meaning.[1] In this way one can give the quantifier-free fragment of the language a purely compositional semantics, i.e., a semantics in which every phrase is assigned a meaning and the meaning of any phrase is determined by the meanings assigned to its immediate subphrases. This completely compositional semantics is not required for an understanding of the noun phrase subsumption procedure presented below. [2]

4 Simplified Montague Syntax

The distinction between nouns and verbs in our formal Montague syntax is unmotivated. Noun relation symbols are semantically indistinguishable from verb relation symbols and noun phrase class expressions are semantically indistinguishable from verb phrase class expressions. To simplify the discussion of the computational properties of Montague syntax, we will ignore the dis-

[1]The meaning assigned to maximal noun phrases is higher-order: a maximal noun phrase denotes a predicate on sets. For example, the maximal noun phrase (**every man**) denotes the predicate $\lambda s.(\mathtt{man} \subseteq s)$, i.e., the predicate that takes a set s and returns true if the class **man** is a subset of the class s. The maximal noun phrase (**some man**) denotes the predicate $\lambda s.s \cap \mathrm{man} \neq \emptyset$ and the maximal noun phrase **John** denotes the predicate $\lambda s.\mathtt{John} \in s$. A formula (**NP VP**) is true just in case the higher order predicate denoted by the maximal noun phrase **NP** is true when applied to the class denoted by **VP**. Now consider a class expression of the form (**R D**) where **D** is a maximal noun phrase that denotes the higher-order predicate D. The class denoted by (**R D**) can be defined as the set of all domain elements d' such that the predicate D is true of the set $\{d : \langle d', d\rangle \in \mathtt{R}\}$.

[2]In our experience, thinking about the compositional semantics seems to interfere with a clear understanding of the computational properties of first order Montague syntax. For example, the meaning of maximal noun phrases is second order, but first order Montague syntax is really just a syntactic variant of first order predicate calculus.

tinction between nouns and verbs. Ignoring this distinction there are three syntactic categories: class expressions, maximal noun phrases, and formulas. Dropping the distinction between nouns and verbs leads to a simplified syntax in which the expressions of each kind can be defined by mutual recursion as follows:

- A *class expression* is either a class token, an application `(R N)` of a relation token `R` to a maximal noun phrase `N`, or a such-that class expression of the form `(N` x `such that` $\Phi(x)$`)` where `N` is a class expression, x is a variable, and $\Phi(x)$ is a formula.
- A *maximal noun phrase* is either a variable, a constant, or an expression of the form `(some N)` or `(every N)` where `N` is a class expression.
- A *formula* is either an atomic formula of the form `(NP C)` where `NP` is a maximal noun phrase and `C` is a class expression, or is a Boolean combination of such atomic formulas. A *literal* is an atomic formula or its negation.

The simplified syntax is "free-er" than the Montague syntax given above: every atomic formula of Montague syntax is also an atomic formula of the simplified syntax but many formulas of the simplified syntax are not legal in the unsimplified Montague syntax. For example, the formulas `((every dog) mammal)` and `((every (loves John)) (loves Mary))` are legal in the simplified syntax but not legal in the unsimplified syntax. However, since the unsimplified syntax is simply a subset of the simplified syntax, any inference algorithm that runs on the simplified syntax will be able to handle the unsimplified syntax as a special case. For this reason we restrict the discussion of computational properties to a discussion of the simplified syntax.

From an engineering perspective the simplified syntax seems clearly superior to the unsimplified syntax. The simplified syntax allows verb phrase class expressions to be used directly in constructing maximal noun phrases such as `(some (loves Mary))`. This ability to use verb phrase class expressions in maximal noun phrases apparently implies that the quantifier-free fragment of the simplified syntax is more expressive than the quantifier-free fragment of the unsimplified syntax. Furthermore, this increase in expressive

power apparently has no computational cost. Any inference algorithm on the unsimplified syntax can be used to perform inference on the simplified syntax. More specifically, all formulas of the simplified syntax can be translated into formulas of the unsimplified syntax by treating every relation token in the simplified syntax as a noun relation token and translating every atomic formula `(NP C)` in the simplified syntax into the formula `(NP (is (a C)))` in the unsimplified syntax.

Our first order Montague syntax and the associated inference procedure given below may provide a functional explanation for some of the syntactic structure of maximal noun phrases. However, as the above simplification of the language indicates, it does not provide any functional explanation for the distinction between nouns and verbs. Nouns and verbs do appear to play different functional roles in natural language. For example, the fact that verbs can be tensed (past, present or future) while nouns can not indicates that nouns and verbs play different functional roles. Not being linguists, it would be foolish for us to attempt an analysis of all the potential semantic differences between nouns and verbs in natural language. We will simply say that we expect that there is some semantic difference between nouns and verbs in natural language and that this difference is not being captured in our first order Montague syntax. Perhaps some additional semantic complexity of verb phrases could be incorporated into a yet more sophisticated formal language with yet more expressive power in its quantifier-free fragment.

5 The Noun Phrase Subsumption Procedure

We now define a decision procedure for determining the satisfiability of a set of quantifier-free Montague literals using techniques similar to those used in [McAllester *et al.*, 1989]. Unfortunately, this satisfiability problem is NP complete; the proof of NP hardness is given in a later section. The NP hardness of the general problem arises from the fact that, for a given class expression appearing in the input, we may not know whether or not there exist elements of that class. If, for each class expression, we know whether or not elements of that class exist then the satisfiability problem becomes polynomial time decidable.

To simplify the satisfiability problem we first observe that without loss of generality we can assume that every occurrence of a constant symbol c in Σ is in a class expression of the form (IS c) — any formula or class expression that contains a constant symbol c is semantically equivalent the expression that results from replacing c by (some (is c)). This simplification allows us to assume without loss of generality that in any formula of the form (NP C), and any class expression of the form (R NP), where R is a relation other than IS, the maximal noun phrase NP is either of the form (some s) or (every s) for some class expression s.

It is useful to introduce abbreviations for several formulas that are used commonly in this paper. First, we use the notation $s \subseteq t$ where s and t are class expressions as an abbreviation for the formula ((every s) t). We use the notation $s \sqcap t$ where s and t are class expressions as an abbreviation for ((some s) t). Formulas of the first type express subset relationships and formulas of the second type express the statement that two classes intersect. Given the above simplification for constant symbols, every atomic formula is of one of these two types. We use the notation $\exists s$ where s is a class expression as an abbreviation for the formula $s \sqcap s$. Formulas of the form $\exists s$ express the statement that there exist elements of the set denoted by s. Finally we use the notation $s = t$ where s and t are class expressions as an abbreviation for the conjunction $(s \subseteq t) \wedge (t \subseteq s)$.

Definition: We say that a set of formulas Σ *determines existentials* if, for every class expression s that appears in any formula in Σ, Σ contains either the formula $\exists s$ or the formula $\neg\exists s$.

Montague Syntax Quantifier-Free Decidability Theorem: The satisfiability of a set of quantifier-free Montague literals that determines existentials is polynomial time decidable.

The above theorem implies that one can determine whether an arbitrary set Σ of quantifier-free Montague literals is satisfiable by searching for a superset of Σ that determine existentials and is satisfiable. If there are n class expressions that appear in formulas in Σ then one need search at most 2^n different extensions of Σ. In practice, the noun phrase subsumption procedure, on which the above theorem is based, can be applied even when Σ does

not determine existentials; if Σ does not determine existentials the procedure is not guaranteed to be complete. If a complete procedure is desired, one can search the space of possible extensions of Σ using the noun phrase subsumption procedure at each node of the search tree.

To prove the above theorem we now observe that Σ can always be transformed into an equi-satisfiable set Σ' which contains no literals of the form $s \sqcap t$ where s and t are different class expressions. We will call such literals *positive intersection literals.* This transformation can be achieved by simply replacing any positive intersection literal with three literals $w \subseteq s$, $w \subseteq t$ and $\exists w$ where w is a new class token. Any model of Σ' is also a model of Σ, and any model of Σ yields a model of Σ'. For the remainder of this section we assume that Σ contains no positive intersection literals. Negative intersection literals, i.e. literals of the form $\neg(s \sqcap t)$, may still be present.

The noun phrase subsumption procedure is based on the notion of restricted inference. The inference process involves the inference rules given in figure 1. These rules introduce a new formula, Ds, read as "determined s", where s is a class expression. The formula Ds is true just in case the set denoted by s contains at most one member.

The noun phrase subsumption procedure is an inference process involving the inference rules in figure 1. If Σ is a set of Montague literals we write $\Sigma \vdash \Phi$ if any one of the following conditions hold:

- Φ can be proven from Σ using the above rules of inference.
- Φ is of the form $s = t$ and $\Sigma \vdash s \subseteq t$ and $\Sigma \vdash t \subseteq s$.
- Φ is of the form $s \sqcap t$ and there exists some w such that $\Sigma \vdash \exists w$, $\Sigma \vdash w \subseteq s$ and $\Sigma \vdash w \subseteq t$

Clearly, the last two conditions above could have been incorporated into the inference rules in table 1. However, there are no occurrences of equality or intersection formulas in the antecedents of the inference rules. This implies that the noun phrase subsumption procedure can be run to completion without deriving equality or intersection formulas. After the procedure has

(1) $\exists$`THING`

(2) $s \subseteq$ `THING`

(3) $\dfrac{s \subseteq t}{$`(R (some` s`)) ⊆ (R (some` t`))`$}$

(4) $\dfrac{s \subseteq t}{$`(R (every` t`)) ⊆ (R (every` s`))`$}$

(5) $\dfrac{r \subseteq s,\ s \subseteq t}{r \subseteq t}$

(6) $t \subseteq t$

(7) $\exists$`(is` c`)`

(8) D`(is` c`)`

(9) $\dfrac{\exists$`(R (some` s`))`$}{\exists s}$

(10) $\dfrac{\exists r,\ r \subseteq t}{\exists t}$

(11) $\dfrac{Dt,\ r \subseteq t}{Dr}$

(12) $\dfrac{\neg(r \subseteq t)}{\exists r}$

(13) $\dfrac{\exists s,\ Dt,\ s \subseteq t}{t \subseteq s}$

(14) $\dfrac{\exists r,\ r \subseteq s,\ r \subseteq t}{$`(R (every` s`)) ⊆ (R (some` t`))`$}$

(15) $\dfrac{Ds,\ \exists s}{s =$ `(IS (every` s`))`$}$

(16) $\dfrac{\exists$`(IS (every` s`))`$,\ \exists s}{Ds}$

(17) $s =$ `(IS (some` s`))`

(18) $\dfrac{\neg\exists s}{$`THING ⊆ (R (every` s`))`$}$

(19) $\dfrac{Dt,\ s \subseteq t}{$`(R (some` s`)) ⊆ (R (every` t`))`$}$

Figure 1: The inference rules for quantifier-free literals. In these rules the letters r, s, and t range over class expressions, c ranges over constant symbols, and R ranges over relation symbols. An inference rule that concludes an equation $s = t$ is an abbreviation for two inference rules, one that concludes $s \subseteq t$ and another that concludes $t \subseteq s$.

been run to completion, we can easily check whether a given equality or a given intersection formula has been implicitly deduced. The notation $\Sigma \vdash \mathbf{F}$ abbreviates the statement that there exists some formula Ψ such that $\Sigma \vdash \Psi$ and $\Sigma \vdash \neg\Psi$.

The inference rules allow for the derivation of an infinite number of formulas. To guarantee that the inference process terminates it must be restricted in some way. This is done by requiring that every formula derived by the inference process only mention class expressions that explicitly appear in some formula in Σ. The notation $\Sigma \vdash\!\!\!\dashv \Psi$ is defined by analogy with $\Sigma \vdash \Psi$ except that *every class expression appearing in any derivation must either be the special class expression* `THING` *or must appear in some formula in* Σ. The notation $\Sigma \vdash\!\!\!\dashv \mathbf{F}$ is defined analogously to $\Sigma \vdash \mathbf{F}$. The inference rules can only derive atomic formulas (rules 15 and 17 are each abbreviations for a pair of rules that derive subset formulas). Since there are only quadratically many atomic formulas that involve class expressions that appear in Σ, one can construct a polynomial-time procedure for determining if $\Sigma \vdash\!\!\!\dashv \mathbf{F}$. The following section contains a proof that for any set Σ of quantifier-free taxonomic literals satisfying the above simplifying assumptions, if $\Sigma \not\vdash\!\!\!\dashv \mathbf{F}$ then Σ is satisfiable.

6 Correctness of the Procedure

Suppose that Σ is a set of quantifier-free Montague literals that determines existentials, contains no positive intersection literals, and such that every appearance of any constant symbol `c` in Σ is contained in a class expression of the form `(IS` *c*`)`. This section gives a proof that $\Sigma \not\vdash\!\!\!\dashv \mathbf{F}$ if and only if Σ is satisfiable, thus proving the polynomial time decidability theorem stated in the previous section. If $\Sigma \vdash\!\!\!\dashv \mathbf{F}$ then the soundness of the individual inference rules guarantees that Σ is unsatisfiable. If $\Sigma \not\vdash\!\!\!\dashv \mathbf{F}$ the proof is more difficult. In this case we must construct a model of Σ.

Assume that $\Sigma \not\vdash\!\!\!\dashv \mathbf{F}$. As in most formal completeness proofs, we will construct a formal model of Σ where the elements of the domain are constructed from the class expressions that appear in Σ. The definition of the semantic

domain of the model involves two complications. First, we must construct equivalence classes of class expressions. If $\Sigma \vdash s = t$, and $\Sigma \vdash Ds$, then s and t must denote the same singleton set. In this case the single object in the set denoted by s is (essentially) the equivalence class of all class expressions that are provably equal to s. The second complication involves the need for both "minimal" and "maximal" elements of every class. If $\Sigma \not\vdash$ (R (some s)) $\subseteq t$ then we must guarantee that s contains a maximal element d such that (R d) is a "large" set, and in particular, that (R d) includes something not in the set denoted by t. Let $|s|$ be the equivalence class of the class expression s. We use the notation "some-$|s|$" to denoted the pair of the token "some" and the class $|s|$. The pair "some-$|s|$" will be the desired maximal element of the class denoted by s. If $\Sigma \not\vdash t \subseteq$ (R (every s)) then we must guarantee that s contains some minimal element d such that (R d) is a small set, and in particular, that the set denoted by t contains something not in (R d). By analogy with maximal elements, we use the notation "every-$|s|$" to denote the formal object that will be the minimal element of the set denoted by s.

We say that a class expression s is a *domain expression* if it is either the special class THING or appears in Σ and $\Sigma \vdash \exists s$. If s is a domain expression then we use the notation $|s|$ to denote the set of all domain expressions t such that $\Sigma \vdash s = t$. Inference rules 6 and 11 guarantee that the sets of the form $|s|$ form a partition of the domain expressions into equivalence classes. The semantic domain D of our model will consist of the maximal elements "every-$|s|$" where s is a domain expression and the minimal elements "some-$|s|$" where s is a domain expression such that $\Sigma \not\vdash Ds$. Inference rule 11 guarantees that if $\Sigma \vdash Ds$ and $\Sigma \vdash s = t$ then $\Sigma \vdash Dt$. This implies that the choice of whether or not to include the domain element "some-$|s|$" in the semantic domain is independent of the choice of the representative s of the class $|s|$.

Given this semantic domain D, we must define an interpretation for the class tokens and relation symbols in Σ such that each literal of Σ is satisfied. The model we construct will satisfy a certain *denotation invariant* — the set denoted by a class expression s that appears in Σ will consist of all domain elements "some-$|t|$" and "every-$|t|$" such that $\Sigma \vdash t \subseteq s$. We define the interpretation of constant symbols, class tokens, and relation symbols using this desired domain invariant as a guide. We use the notation "Q-$|s|$" to

mean either the object "**some-**$|s|$" or the object "**every-**$|s|$". The denotation of a class token C is defined to be the set of all domain members of the form "Q-$|s|$" such that $\Sigma \vdash s \subseteq C$. This definition immediately guarantees the denotation invariant for class tokens. Inference rule 2, together with the definition of the denotation of class tokens, guarantees that the special class token `THING` denotes the entire semantic domain D.

We define the denotation of a constant symbol c that appears in Σ to be the domain member "**every-**$|$`(IS c)`$|$". The simplifying assumptions given above guarantee that the class expression `(IS` c`)` appears in Σ and inference rule 7 guarantees that this class expression is a domain expression. Inference rules 7, 8, 11 and 13 guarantee that the denotation invariant holds for classes of the form `(IS` c`)` where c is a constant symbol. We interpret each constant symbol that does not appear in Σ as an arbitrary element of the semantic domain.

We will now define the interpretation of relation symbols other than `IS`. (The relation `IS` always denotes the identity relation.) For any domain element "Q-$|s|$" we must define the set `(R` "Q-$|s|$"`)`. Intuitively, the set `(R` "Q-$|s|$"`)`, where Q is either **some** or **every**, should be the set of domain members "Q'-$|t|$" such that $\Sigma \vdash t \subseteq$ `(R (`Q s`))`. This intuitive definition fails because the class expression `(R (`Q s`))` need not appear in Σ. To remedy this situation we define a new relation $\vdash_o$.

- We write $\Sigma \vdash_o t \subseteq$ `(R (some` s`))` if any one of the following conditions hold:
  - There is some expression `(R (some` w`))` in Σ such that $\Sigma \vdash w \subseteq s$ and $\Sigma \vdash t \subseteq$ `(R (some` w`))`
  - There is some expression `(R (every` w`))` in Σ such that $\Sigma \vdash s \sqcap w$ and $\Sigma \vdash t \subseteq$ `(R (every` w`))`.
- We write $\Sigma \vdash_o t \subseteq$ `(R (every` s`))` if any one of the following conditions hold:
  - There is some expression `(R (every` w`))` in Σ such that $\Sigma \vdash s \subseteq w$ and $\Sigma \vdash t \subseteq$ `(R (every` w`))`

- There is some expression (R (**some** w)) in Σ such that $\Sigma \vdash Dw$, $\Sigma \vdash w = s$ and $\Sigma \vdash t \subseteq$ (R (**some** w))

One can check that if $\Sigma \vdash t \subseteq$ (R (Q s)) then $\Sigma \vdash\!\circ\ t \subseteq$ (R (Q s)). Conversely, if (R (Q s)) appears in Σ, and $\Sigma \vdash\!\circ\ t \subseteq$ (R (Q s)) then $\Sigma \vdash t \subseteq$ (R (Q s)). The difference between the two relations is restricted to expressions of the form (R (Q s)) that do not appear in Σ. The reader can also check that if $\Sigma \vdash \exists s$ and $\Sigma \vdash\!\circ\ t \subseteq$ (R (**every** s)) then $\Sigma \vdash\!\circ\ t \subseteq$ (R (**some** s)).

We now define the set (R "Q-$|s|$") to be the set of all domain elements "**some-**$|t|$" and "**every-**$|t|$" such that $\Sigma \vdash\!\circ\ t \subseteq$ (R (Q s)). We must check that this definition is well formed, i.e., that the definition is independent of the choice of s and t used as the representatives of the equivalence classes $|s|$ and $|t|$. Fortunately, the transitivity of the subset relation guarantees that if t' is equivalent to t and s' is equivalent to s then $\Sigma \vdash\!\circ\ t' \subseteq$ (R (Q s')) if and only if $\Sigma \vdash\!\circ\ t \subseteq$ (R (Q s)).

This completes the definition of a first order structure — we have defined a semantic domain and assigned an appropriate meaning to all constant symbols, class tokens, and relation tokens. We will now prove that every class expression that appears in Σ satisfies the desired denotation invariant.

> **Class Denotation Lemma:** For any class expression s that appears in Σ, the denotation of s equals the set of domain elements "Q-$|t|$" such that $\Sigma \vdash t \subseteq s$.

We prove this lemma by induction on the structure of class expressions. Every class expression appearing in Σ is either a class token, an expression of the form (IS c) for some constant symbol c, or an expression of the form (R (Q s)) for some relation token R (including equality), specifier Q, and class expression s. We have already argued that the denotation invariant holds for class tokens and class expressions of the form (IS c) for constant symbols c. Now we assume that s satisfies the denotation invariant and consider an expression in Σ of the form (R (Q s)). It now suffices to show that (R (Q s)) satisfies the denotation invariant, i.e., the set denoted by (R (Q s)) is the

set of domain elements "Q'-$|t|$" such that $\Sigma \vdash t \subseteq$ (R (Q s)). There are now four cases depending on whether R is the special relation IS and on whether the specifier Q is **some** or **every**.

Case 1: R is IS and Q is **some**. We must show that "Q'-$|t|$" is in the set denoted by (IS (**some** s)) if and only if $\Sigma \vdash t \subseteq$ (IS (**some** s)). The semantics of IS and **some** imply that the set denoted by (IS (**some** s)) is the same as set denoted by s. Inference rule 17 guarantees that $\Sigma \vdash t \subseteq$ (IS (**some** s)) if and only if $\Sigma \vdash t \subseteq s$. So it suffices to show that "Q'-$|t|$" is in the set denoted by s if and only if $\Sigma \vdash t \subseteq s$. But this is precisely the statement of the denotation invariant for s which we have assumed.

Case 2: R is IS and Q is **every**. We must show that "Q'-$|t|$" is in the set denoted by (IS (**every** s)) if and only if $\Sigma \vdash t \subseteq$ (IS (**every** s)). First, suppose that $\Sigma \vdash \exists s$ and $\Sigma \vdash Ds$. This case is similar to case 1 above —the denotation invariant for s, together with inference rule 13, imply that the set denoted by s contains the single element "**every-**$|s|$" so the set denoted by (IS (**every** s)) equals the set denoted by s. Inference rule 15 guarantees that $\Sigma \vdash t \subseteq$ (IS (**every** s)) if and only if $\Sigma \vdash t \subseteq s$. The denotation invariant for (IS (**every** s)) now follows directly from the denotation invariant for s.

Now suppose that $\Sigma \vdash \exists s$ but $\Sigma \nvdash Ds$. In this case the denotation invariant for s implies that s contains (at least) the two elements "**every-**$|s|$" and "**some-**$|s|$". Thus the class (IS (**every** s)) denotes the empty set. It now suffices to show that there is no domain class t such that $\Sigma \vdash t \subseteq$ (IS (**every s**)). If $\Sigma \vdash t \subseteq$ (IS (**every s**)) then inference rules 10 and 16 guarantee that $\Sigma \vdash Ds$, contradicting the assumptions of this case.

Finally, suppose that $\Sigma \nvdash \exists s$. Since Σ determines existentials, we must have $\Sigma \vdash \neg\exists s$. In this case the denotation invariant ensures that s denotes the empty set — if "Q'-$|t|$" were in the class denoted by s then t must be a domain expression and so $\Sigma \vdash \exists t$ and the denotation invariant for s implies $\Sigma \vdash t \subseteq s$ so by inference rule 10 we have $\Sigma \vdash \exists s$. But if s denotes the empty set then the class (IS (**every** s)) denotes the entire semantic domain. To show that the denotation invariant holds for (IS (**every** s)) we must show that $\Sigma \vdash t \subseteq$ (IS (**every s**)) for an arbitrary domain class t. This last statement is guaranteed by inference rules 2 and 18.

Case 3: R is a relation token and Q is **some**. First, suppose that "Q'-$|t|$" is in the set denoted by (R (**some** s)). We must show that $\Sigma \vdash t \subseteq$ (R (**some** s)). In this case there must be some element "Q''-$|s'|$" in the set denoted by s such that (R "Q''-$|s'|$") contains "Q'-$|t|$". By the induction hypothesis we must have $\Sigma \vdash s' \subseteq s$. By the definition of the meaning of R, we must have $\Sigma \vdash\!\circ\ t \subseteq$ (R (Q'' s')). Since s' is a domain class we must have $\Sigma \vdash \exists s'$. As noted above, the definition of $\vdash\!\circ$ implies that if $\Sigma \vdash \exists s'$ and $\Sigma \vdash\!\circ\ t \subseteq$ (R (**every** s')) then $\Sigma \vdash\!\circ\ t \subseteq$ (R (**some** s')). Thus we must have $\Sigma \vdash\!\circ\ t \subseteq$ (R (**some** s')) (it is possible that $\Sigma \vdash\!\circ\ t \subseteq$ (R (**some** s')) even if "**some-**$|s'|$" is not a domain member.) By the definition of of $\vdash\!\circ$ there must exist some expression (R (Q''' w)) that appears in Σ such that $\Sigma \vdash t \subseteq$ (R (Q''' w)) and such that (R (Q''' w)) satisfies one of the two ways of establishing $\Sigma \vdash\!\circ\ t \subseteq$ (R (**some** s')). Let (R (Q''' w)) be an expression that satisfies one of these two cases. We leave it to the reader to verify that in each case the expression (R (Q''' w)) ensures that $\Sigma \vdash\!\circ\ t \subseteq$ (R (**some** s)) and thus that $\Sigma \vdash t \subseteq$ (R (**some** s)).

Now suppose that $\Sigma \vdash t \subseteq$ (R (**some** s)) for some domain class t and consider a domain element of the form "Q-$|t|$". We must show that "Q-$|t|$" is a member of the set denoted by (R (**some** s)). Since t is a domain class we have $\Sigma \vdash \exists t$. Inference rules 9 and 10 now guarantee that $\Sigma \vdash \exists s$. Now suppose that $\Sigma \vdash Ds$. In that case the definition of $\vdash\!\circ$ ensures that $\Sigma \vdash\!\circ\ t \subseteq$ (R (**every** s)). Since s satisfies the denotation invariant, and $\Sigma \vdash \exists s$, the element "**every-**$|s|$" must be in the class denoted by s. Finally, since $\Sigma \vdash\!\circ\ t \subseteq$ (R (**every** s)), we have that the set (R "**every-**$|s|$") contains "Q-$|t|$" and thus "Q-$|t|$" is in the set denoted by (R (**some** s)). Now suppose that $\Sigma \nvdash Ds$. In this case the fact that $\Sigma \vdash \exists s$ and the denotation invariant for s guarantee that the set denoted by s includes the element "**some-**$|s|$". But the fact that $\Sigma \vdash t \subseteq$ (R (**some** s)) immediately implies that "Q-$|t|$" is in the set (R "**some-**$|s|$") and thus "Q-$|t|$" is in the set denoted by (R (**some** s)).

Case 4: R is a relation token and Q is **every**. We must show that "Q'-$|t|$" is in the set denoted by (R (**every** t)) if and only if $\Sigma \vdash t \subseteq$ (R (**every** t)). If $\Sigma \nvdash \exists s$ the proof is identical to the last part of the proof of case 2. Suppose $\Sigma \vdash \exists s$. In this case the denotation invariant guarantees that the set denoted by s contains the element "**every-**$|s|$". Now if "Q'-$|t|$" is in the set denoted

by (R (every s)), then "Q'-$|t|$" must be in the set (R "every-$|s|$"). But this implies that $\Sigma \vdash\!\!\circ\ t \subseteq$ (R (every s)) and since (R (every s)) appears in Σ, this implies that $\Sigma \vdash t \subseteq$ (R (every s)).

Now suppose $\Sigma \vdash t \subseteq$ (R (every s)). To show that "Q'-$|t|$" is a member of the set denoted by (R (every s)) let "Q''-$|s'|$" be an arbitrary member of the set denoted by s. We must show that "Q'-$|t|$" is a member of the set (R "Q''-$|s'|$"). The denotation invariant for s implies that $\Sigma \vdash s' \subseteq s$. But in this case the definition of $\vdash\!\!\circ$ guarantees that $\Sigma \vdash\!\!\circ\ t \subseteq$ (R (Q'' s')) and thus "Q'-$|t|$" is in the set (R "Q''-$|s'|$") as desired.

We now conclude our proof of the noun phrase subsumption inference procedure's completeness, by showing that the model defined above satisfies every literal φ in Σ. φ must be of the form $s \subseteq t$, $\neg(s \subseteq t)$, $\exists s$, or $\neg(s \sqcap t)$ (formulas of the form $\neg\exists s$ are a special case of negative intersection formulas and we have assumed that Σ does not contain any positive intersection formulas other than formulas of the form $\exists s$). First, consider a literal in Σ of the form $s \subseteq t$. The denotation invariant (and the transitivity inference rule) implies that the class denoted by s must be a subset of the class denoted by t. Now consider a formula in Σ of the form $\neg(s \subseteq t)$. Inference rule 12 guarantees that $\Sigma \vdash \exists s$. Thus the semantic domain includes the object "every-$|s|$". But since $\Sigma \not\vdash \mathbf{F}$, we must have $\Sigma \not\vdash s \subseteq t$. Thus by the class denotation invariant, "every-$|s|$" must be a member of the set denoted by s that is not a member of the set denoted by t, and thus the formula $s \subseteq t$ must be false in the defined model. Now consider a formula in Σ of the form $\exists s$. The class denotation invariant, and definition of the semantic domain immediately imply that the set denoted by s includes the object "every-$|s|$" and thus the formula $\exists s$ is true in the defined model. Finally, consider a formula in Σ of the form $\neg(s \sqcap t)$. Suppose this formula were false in the defined model, i.e., there exists a domain element that is in both the set denoted by s and the set denoted by t. Let "Q-$|w|$" be a domain element that is in both s and t. The definition of the semantic domain implies that $\Sigma \vdash \exists w$. The denotation invariant for s and t implies that $\Sigma \vdash w \subseteq s$ and $\Sigma \vdash w \subseteq t$. But the definition of the relation $\vdash$ implies that in this case we have $\Sigma \vdash s \sqcap t$ and hence $\Sigma \vdash \mathbf{F}$ which we have assumed is not so.

We have now proven that the model we defined is indeed a model of Σ,

and therefore that Σ is satisfiable anytime the noun subsumption inference procedure detects no contradiction in Σ.

7 NP Completeness of the General Case

Now consider a set Σ of quantifier-free Montague literals that does not determine existentials. The results of the preceding section show that the problem of determining the satisfiability of Σ is in NP — one can verify that Σ is satisfiable by first guessing which existential statements are true in some model and then using the noun phrase subsumption procedure to verify that the guessed existential statements do have a model. We now show that determining the satisfiability of a set Σ that does not determine existentials is NP-hard.

The proof of NP-hardness is by reduction of a special case of monotone 3-SAT. More specifically, we start with a set of propositional clauses where each clause either contains three negative literals or two positive literals. We leave it to the reader to verify that satisfiability of an arbitrary 3SAT problem can be reduced to satisfiability of this special case. For each proposition symbol P in our restricted 3SAT problem we introduce a class token P' and we reduce the set of clauses to a set Σ of Montague literals as follows:

For each clause of the form $P \vee Q$ we add the literal (R (**every** P')) $\subseteq$ (G (**some** Q')) where R and G are new relation tokens. Any model of this literal must satisfy either $\exists P'$ or $\exists Q'$ — if both P' and Q' are assigned the empty set then (R (**every** P')) denotes the unversal set, which must be non-empty, while (G (**some** Q')) denotes the empty set. Conversely, for any interpretation of the classes P' and Q' as sets, if at least one of the two sets is non-empty then one can ensure that the above literal is satisfied by making R the empty relation and G the universal relation.

Now for any class tokens s, t and w we define $[\exists s \rightarrow (t \subseteq w)]$ to be the two literals $t \subseteq$ (H (**every** s)) and (H (**some** s)) $\subseteq w$, where H is a new relation token specific to this constraint. Any model of these literals must satisfy the constraint that if s denotes a non-empty class then the class denoted by t

must be a subset of the class denoted by s. Conversely, for any assignment of sets to the classes s, t, and w satisfying the desired constraint, there exists an interpretation of H satisfying the above literals — if s is assigned the empty class then the above literals are satisfied by any interpretation of H; if s is non-empty then t must be a subset of w and the above literals are satisfied by interpreting H as the relation that maps every domain element to the set w.

Finally, for any clause of the form $\neg P \vee \neg Q \vee \neg U$ we add the literals that constitute the constraints

$$[\exists P' \rightarrow (s \subseteq w_1)]$$

$$[\exists Q' \rightarrow (w_1 \subseteq w_2)]$$

$$[\exists U' \rightarrow (w_2 \subseteq t)]$$

$$\neg(s \subseteq t)$$

where s, t, w_1, and w_2 are new class tokens specific to this clause. Any model of the above formulas must assign one of the classes P', Q', or R' the empty set. Conversely, for any interpretation of P', Q', and R' as sets at least one of which is empty, there exist interpretations of s, t, w_1 and w_2 as sets that satisfy the above constraints.

We leave it to the reader to verify that the set of literals generated by this reduction is satisfiable if and only if the original restricted 3SAT problem is satisfiable.

8 Open Technical Problems

We have not yet investigated generalizing Montague syntax to three place and higher arity relations. In Montague syntax, a three place relation would be viewed as an operator that takes two arguments and returns a set. The English verb "give" can be viewed as denoting a three place relation and one could extend Montague syntax to allow the verb phrase class expression (**gave (a man) (a book)**) and the formula (**John (gave (a man) (a book))**). The meaning of the class expression (**gave (some** s**) (every** t**)**)

would be the union over d in the meaning of s of the intersection over d' in the meaning of t of the set (**gave** d d'). This definition preserves the left to right order of the quantifiers so that the sentence (**John (gave (some man) (every book))**) has the expected meaning. Preserving the order of the quantifiers also seems to assign the preferred meaning to the sentence (**John (gave (every man) (a book))**). In any case, we do not require that the meaning of the formal expressions of Montague syntax always agree with the meaning of the corresponding English expressions. Consider a set Σ of literals in Montague syntax extended to allow for three place relations such that Σ determines existentials. Does there exist a polynomial time decision procedure for determining the satisfiability of Σ? It seems likely that if we restrict the language to three place relations, or relations of any fixed arity, then the noun phrase subsumption procedure can be modified to yield a polynomial time decision procedure. However, the case of unbounded arity seems less clear. In classical syntax operators of more than two arguments can always be replaced by operators of two arguments and an additional pairing function for forming tuples. Unfortunately, this transformation does not seem to apply to Montague syntax. Let **pair** be a three place relation that takes two arguments x and y and returns a set containing the single element which is the pair of x and y. In Montague syntax the expression (**pair (every** s**) (every** t**)**) would denote the empty set whenever s or t denotes a set of more than one member.

Another open technical question involves the introduction of function symbols. Our previous work on taxonomic syntax [McAllester *et al.*, 1989] shows that the satisfiability problem for quantifier-free taxonomic literals remains polynomial time decidable in the presence of function symbols. It seems likely that this result would carry over to Montague literals that determine existentials and have bounded arity. We have not yet investigated the addition of function symbols.

9 Summary

We have presented an inference algorithm for noun phrase subsumption that gives a polynomial time decision procedure for the satisfiability of a set of

Montague literals that determines existentials. This shows that, although Montague literals are quite expressive, they retain much of the computational tractability of the literals of classical predicate calculus with equality. Furthermore, since the noun phrase subsumption procedure is based on the restricted application of inference rules, this procedure can be applied to an arbitrary set of formulas. In [McAllester *et al.*, 1989] we argue that a similar procedure for taxonomic syntax can be used to reduce the amount of detail that need be provided by a human user in the machine verification of mathematical theorems. A similar argument can be made for the noun phrase subsumption procedure.

The pragmatic value of the noun phrase subsumption procedure remains an open empirical question: further experimentation is needed to determine if noun phrase subsumption really does improve the performance of automated reasoning systems. In the near future we do not expect to have a definitive answer to the question of the utility of noun phrase subsumption. A powerful argument for the utility of noun phrase subsumption might be that, at some future time, all competitive high-performance mathematical verification systems rely on the procedure.

If noun phrase subsumption is useful, this utility may provide a functional explanation for the syntactic structure of English noun phrases. The inference rules that define the noun phrase subsumption procedure are stated directly in terms of Montague syntax. Defining the same procedure in terms of classical syntax seems to require the implicit translation of classical formulas into Montague syntax equivalents. Even if the procedure were defined on classical syntax, it seems that the efficiency of the procedure could be greatly improved by translating the classical syntax into Montague syntax and running the inferences directly on Montague syntax. In short, Montague syntax seems to be needed for the effective use of the noun phrase subsumption procedure. If the noun phrase subsumption procedure is pragmatically important, then the syntactic structure of English noun phrases has a functional motivation.

References

[Bobrow and Winograd, 1977] D. Bobrow and T. Winograd. An overview of KRL, a knowledge representation language. *Cognitive Science*, 1(1):3–46, 1977.

[Brachman *et al.*, 1983] R. Brachman, R. Fikes, and H. Levesque. Krypton: A functional approach to knowledge representation. *IEEE Computer*, 16:63–73, 1983.

[Brachman, 1983] R. J. Brachman. What is-a is and isn't: An analysis of taxonomic links in semantic networks. *IEEE Computer*, 16(10):30–36, October 1983.

[Burstall, 1984] Rod Burstall. Programming with modules as typed functional programming. In *International Conference on 5th Generation Computing Systems*, Tokyo, Japan, November 1984.

[Cardelli, 1984] Luca Cardelli. The semantics of multiple inheritance. In *Procedings of the Conference on the Semantics of Datatypes*, pages 51–66. Springer-Verlag, June 1984.

[Downey *et al.*, 1980] Peter J. Downey, Ravi Sethi, and Robert E. Tarjan. Variations on the common subexpression problem. *JACM*, 27(4):758–771, October 1980.

[Dowty *et al.*, 1983] D. Dowty, R. E. Wall, and S. Peters. *Introduction to Montague Semantics.* D. Reidel, 1983.

[Fahlman, 1979] Scott E. Fahlman. *NETL: A System for Representing Real World Knowledge.* MIT Press, 1979.

[McAllester *et al.*, 1989] D. McAllester, R. Givan, and T. Fatima. Taxonomic syntax for first order inference. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, pages 289–300, 1989.

[McAllester, 1989] David A. McAllester. *Ontic: A Knowledge Representation System for Mathematics.* MIT Press, 1989.

[Nelson and Oppen, 1979] Greg. Nelson and Derek Oppen. Simplification by cooperating decision procedures. *ACM Trans. Prog. Lang. and Syst.*, 1:245–257, October 1979.

[Nelson and Oppen, 1980] Greg Nelson and Derek Oppen. Fast decision procedures based on congruence closure. *JACM*, 27(2):356–364, April 1980.

[Reynolds, 1974] John C. Reynolds. Towards a theory of type structure. In *Proceedings Colloque sur la Programmation*. Springer-Verlag, 1974.

[Stickel, 1985] Mark E. Stickel. Automated deduction by theory resolution. *Journal of Automated Reasoning*, 1:333–355, 1985.

[Walther, 1984a] Christoph Walther. A mechanical solution of Schubert's steamroller by many-sorted resolution. In *Proceedings of AAAI-84*, pages 330–334, 1984.

[Walther, 1984b] Christoph Walther. Unification in many-sorted theories. In T. O'Shea, editor, *ECAI 84: Advances in Artificial Intelligence*, pages 593–602, North-Holland, 1984. ECCAI, Elsevier Scientific Publishers B. V.