Taxonomic Syntax for First Order Inference

DAVID MCALLESTER and ROBERT GIVAN

Massachusetts Institute of Technology, Cambridge Massachusetts

Abstract: We identify a new polynomial time decidable fragment of first order logic and present a general method for using polynomial time inference procedures in knowledge representation systems. Our results indicate that a non-standard "taxonomic" syntax is essential in constructing natural and powerful polynomial time inference procedures. The central role of taxonomic syntax in our polynomial time inference procedures provides technical support for the often expressed intuition that knowledge is better represented in terms of taxonomic relationships than classical first order formulas. To use our procedures in a knowledge representation system we define a "Socratic proof system" which is complete for first order inference and which can be used as a semi-automated interface to a first order knowledge base.

Categories and Subject Descriptors: F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical logic — *computational logic, mechanical theorem proving*; I.2.3 [**Artificial Intelligence**]: Deduction and Theorem Proving — *deduction*

General Terms: Deduction, Algorithms

Additional Keywords and Phrases: Proof Theory, Machine Inference, Theorem Proving, Automated Reasoning, Polynomial Time Algorithms, Inference Rules, Proof Systems, Mechanical Verification.

This research was supported in part by National Science Foundation Grant IRI-8819624 and in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-85-K-0124 and N00014-89-J-3202.

Author's Address: MIT Artificial Intelligence Laboratory, 545 Technology Square, Cambridge Mass, 02139, DAM@ai.mit.edu

This paper appeared in JACM, vol. 40, no. 2, April 1993. A postscript electronic source for this paper can be found in ftp.ai.mit.edu:/pub/dam/jacm1.ps. A bibtex reference can be found in internet file ftp.ai.mit.edu:/pub/dam/dam.bib.

1 Introduction

Automated reasoning is one of the central problems of artificial intelligence and computer science. Effective automated reasoning systems, if they could ever be constructed, would have a wide range of applications including the verification and synthesis of computer programs, question answering based on declarative knowledge bases, the mechanical verification of new mathematical results, the verification of student arguments in computer aided instruction systems, and probably other unforeseen applications. Unfortunately, many potential applications of automated reasoning have suffered from its apparent computational intractability.

The most successful automated reasoning systems have been based on man-machine interaction. Following [5], we formalize man-machine interactive systems as *Socratic proof systems*. A formal Socratic proof is a series of steps, analogous to the questions used in the Socratic method of teaching, but where a mechanical procedure is used to determine the "acceptability" of individual steps. In this paper we discuss a general framework for the construction of such systems and define a particular formal Socratic proof system based on a new procedure for determining the acceptability of steps. Our objective is to make Socratic proofs as concise and as natural as possible while ensuring that acceptability can still be quickly determined. To make Socratic proofs concise, the step-checking procedure must provide some amount of automated reasoning. To make step-checking fast, however, this automated reasoning must be limited. We achieve natural, concise, and quickly-checkable Socratic proofs by basing our system on a non-standard syntax for first order logic. We formally specify the acceptability of individual Socratic steps in terms of a computationally tractable, and hence semantically incomplete, set of inference rules for this non-standard syntax. The tractable inference rules underlying our Socratic system appear not to have any natural expression in the classical syntax of first order logic.

Many researchers have discussed and built systems that can be viewed as Socratic proof systems. One way of constructing a Socratic system is to simply place a limit on the amount of time allotted for theorem proving in checking individual steps. This has been done for systems based on resolution theorem proving [2]. However, many researchers are uncomfortable with the lack of any natural characterization of exactly when an individual step in these Socratic proofs is acceptable.

Term rewriting systems form another class of Socratic systems with a more natural definition of an acceptable proof step. There currently exists a variety of man-machine interactive systems that use term rewriting combined with various methods of performing mathematical induction [3], [10], [11]. The theorem proving mechanisms employed in these systems are incomplete and often terminate with failure. If a desired theorem cannot be proven in a fully automated way, it can often be broken down into a series of lemmas where each lemma can be proven by the automated procedure and, given these lemmas, the automated procedure can also prove the desired result. This series of lemmas can be viewed as a series of steps in a Socratic proof. Unfortunately, there is usually no guarantee that rewriting terminates in polynomial time, and long computation times can be a problem in practice.

Another approach to the construction of Socratic systems uses some form of decision procedure to verify individual steps of Socratic proofs. Some authors have proposed decision procedures that require more than polynomial time, such as the decision procedure for propositional dynamic logic, or the decision procedure for ground predicate calculus [20], [6]. Others have emphasized polynomial time procedures [17]. Polynomial time procedures are clearly desirable if one wants the acceptability of Socratic proofs to be quickly checkable.

Some research on polynomial time inference procedures has emphasized computationally tractable fragments of first order logic. For example, it is not difficult to give a linear time procedure for determining the satisfiability of a set of propositional Horn clauses [9]. A more complex example involves deciding the satisfiability of a set of ground literals in first order logic with equality. The difficult part of this satisfiability problem is computing all substitutional consequences of equality. A polynomial time decision procedure based on congruence closure was first given by Kozen [13]. Kozen's congruence closure procedure was followed by a series of applications and more efficient implementations [17] [8]. It is also possible to combine the decision procedure for Horn clauses with the decision procedure for ground literals resulting in an efficient procedure for deciding the satisfiability of a set of ground Horn clauses in first order logic with equality.

Polynomial time inference procedures have also been studied within the knowledge representation subfield of artificial intelligence. Knowledge representation languages are sometimes criticized as being merely "pretty versions" of ordinary first order logic. There are two responses to such criticism. First, some of the constructs of knowledge representation languages cannot be easily translated into first order logic.[1] Second, even for languages that can be faithfully translated to first order logic, the non-standard syntax of a knowledge representation language often allows the identification of computationally tractable fragments of the logic which have no natural characterization in classical syntax. Non-standard tractable fragments of logic have been identified in the context of a certain family of knowledge representation languages called frame description languages, or FDLs [4] [16].

Although an FDL is a nonstandard syntax, one can still identify atomic formulas, literals, Boolean combinations and quantified formulas. Earlier work on FDLs has focused primarily on the problem of determining the validity of individual atomic formulas. Because atomic formulas can be more expressive in a nonstandard syntax than in classical syntax, determining the validity of an atomic formula can be nontrivial.[2] In this paper we focus on the problem of determining the satisfiability of a set of ground literals in an expressive nonstandard syntax. For a given syntax, the problem of determining the satisfiability of a set of literals is at least as hard as, and usually much harder than, the problem of determining the validity of individual atomic formulas. Previous work on FDLs is discussed in more detail in Section 3.

The existence of tractable fragments of first order logic has lead some researchers to adopt a two-language approach to knowledge representation — the knowledge base of facts is separated into those facts expressible in a fixed tractable language and those facts not expressible in that language

[1]There are (at least) three features occurring in various knowledge representation languages that are difficult, if not impossible, to translate into first order logic. The first involves heuristic or probabilistic knowledge, also known as defaults and nonmonotonicity [19]. The second is "intensional" propositions such as the proposition P in the sentence "John believes that P" [12]. A third is recursive definitions. Recursive definitions are perhaps best modeled using the μ-calculus [18].

[2]In classical syntax, the only valid atomic formulas are equations of the form $t = t$.

[4]. If a given query is expressed as a formula in the tractable fragment of logic, then a polynomial decision procedure can be used to determine if the query formula follows from the tractable component of the data base. More expressive tractable fragments of logic allow more facts to be included in the tractable component of the data base and make the polynomial time inference procedure more useful. However, since the decision procedure is only applied to formulas in the tractable fragment of the language, the decision procedure can only be applied to a subset of the possible queries. Furthermore, even when the decision procedure can be applied, it can only use as premises those facts in the knowledge base that are also expressed in the tractable language.

In this paper we propose a one-language architecture for knowledge representation. In our one-language architecture we characterize the underlying inference procedure by a set of inference rules. Although the rules are not complete for the full language, the rules can be usefully applied to arbitrary sets of formulas. Furthermore, the rules are designed so that one can determine, in polynomial time, whether or not a given query can be proven from a data base using the given rules. This eliminates the need to separate the data base into two fragments. In a corresponding two-language system, the inference procedure would only be applied to that subset of the data that could be expressed in the decidable sublanguage, i.e., in the subset of the language for which the rules are complete. However, the inference rules can be used to draw conclusions from information not expressible in the decidable sublanguage. The one-language architecture is able to draw more conclusions than the corresponding two-language system.

Polynomial time inference procedures have also been studied in the context of unification theory. Most relevant to the present paper is the study of many sorted unification [22]. Certain first order axioms about taxonomic relationships can be incorporated into the sort structure of many sorted logic and then handled in the unification step of a resolution theorem prover. This allows many inference problems to be represented with fewer clauses and reduces the space of possible proofs that the theorem prover must search. For simple sort systems it is possible to construct efficient sorted unification algorithms. There are two principle differences between the use of sorts in unification and the use of taxonomic formulas described here. First, the sort structures that have been built into unification procedures are simpler than the taxonomic formulas studied here. For example, in the unification algorithms presented in [22] the sorts are represented by symbols under a fixed partial order and complex class expressions are not considered. Second, like earlier work on FDLs, sorted unification also takes a two-language approach to the use of decision procedures. The sort structure is expressed in a separate language from that used to express more complex formulas.

Our one-language architecture for knowledge representation is formalized as a Socratic proof system. Although there may be consequences of a knowledge base that are not "obvious" to the knowledge representation system, the system can always be "convinced" of the fact by presenting it with an appropriate Socratic proof. A Socratic proof is a sequence of steps where a polynomial time decision procedure is used to test the acceptability of individual steps. More powerful decision procedures lead to shorter Socratic proofs. For the Socratic proof described here there is no need for a distinction between a tractable and an intractable language.

The Socratic proof system defined in this paper is similar to the one used in the Ontic system [15]. Ontic provides Socratic completeness relative to Zermelo-Fraenkel set theory, i.e., any formula provable in set theory can be given a Socratic proof in that system. Ontic has been used to verify the Stone Representation Theorem of lattice theory starting with only the axioms of set theory. The use of a powerful inference procedure in verifying individual steps of a Socratic proof greatly reduces the textual length of such proofs. The introduction of our non-standard syntax allows the inference procedure defined here for step-verification to be stronger than the one used in Ontic.

2 Taxonomic Syntax for First Order Logic

The construction of a non-standard syntax for first order logic is motivated by a desire for powerful yet computationally tractable inference rules. By "computationally tractable inference rules" we mean a set of inference rules such that there exists a polynomial time procedure for determining whether or not an arbitrary formula can be derived from arbitrary premises, i.e., the inference relation generated by the rules is polynomial time decidable. Of course, no computationally tractable set of inference rules can be complete for full first order logic. However, there are useful tractable rule sets and the power of tractable rule sets appears to be sensitive to the syntax used for expressing formulas.

In our taxonomic syntax, as in classical syntax, a first order language is defined by a set of constant, function, and predicate symbols where each function and predicate symbol is associated with a specified arity (number of arguments). The models that define the semantics of our taxonomic expressions are identical to the models that define the semantics of classical first order expressions. A model consists of a set D, called the semantic domain of the model, together with an interpretation of every constant, function and predicate symbol. We will use D^n to denote the set of all n-tuples of elements of D. A first order model with semantic domain D interprets each constant symbol as an element of D, each n-ary function symbol as a function from D^n to D and each n-ary predicate symbol as a subset of D^n.

Our taxonomic syntax for first order logic is organized around class expressions and taxonomic relationships between class expressions. Class expressions are analogous to classical first order terms except that class expressions denote sets rather than individuals. Class expressions are constructed from variables, constants, function symbols and predicate symbols in much the same way that terms are constructed from variables, constants, and function symbols. For example, if **A-MAN** is a monadic predicate symbol then **A-MAN** is, all in itself, a class expression. Intuitively, the class expression **A-MAN** denotes the set of all men. If **PARENT-OF** is a binary relation symbol then **PARENT-OF**(**A-MAN**) is a class expression. Intuitively, this class expressions denotes the set of people who are the parent of some man. Formally, class expressions can be defined syntactically as follows.

Definition: A *class expression* is either

- a variable,
- a constant symbol,
- a monadic predicate symbol,
- an application $f(C_1, \cdots, C_n)$ of an n-ary function symbol f to n class expressions $C_1, \cdots, C_n$,

- or an application $R(C_1, \cdots, C_{n-1})$ of an n-ary relation symbol R (with $n > 1$) to $n-1$ class expressions $C_1, \cdots, C_{n-1}$.

Intuitively, the semantics of class expressions corresponds to simply reading class expressions as if they were English noun phrases. For example, if **FATHER-OF** is a monadic function symbol, and **A-MAN** is a monadic predicate symbol, then the class expression **FATHER-OF**(**A-MAN**) denotes the set of individuals that are the father of some man. More formally, this class expression denotes the image of the set denoted by **A-MAN** under the function denoted by **FATHER-OF**. A rigorous treatment of the semantics of class expressions is given below.

The formulas of taxonomic syntax include atomic statements about the taxonomic relationships between class expressions. More specifically, we write $\mathbf{Is}(C,\ W)$ to say that the set denoted by C is a subset of the set denoted by W. We also write $\mathbf{Ex}(C)$ to say that the set denoted by C is non-empty and we write $\mathbf{Det}(C)$ to say that there is at most one element of the set denoted by C. Finally, we write $\mathbf{Int}(C,\ W)$ to say that the set denoted by C has a non-empty intersection with the set denoted by W. Taxonomic formulas are defined syntactically as follows.

A *taxonomic formula* is either

- a classification formula, $\mathbf{Is}(C,\ W)$, where C and W are class expressions,
- an existence formula, $\mathbf{Ex}(C)$, where C is a class expression,
- a determination formula, $\mathbf{Det}(C)$, where C is a class expression,
- an intersection formula, $\mathbf{Int}(C,\ W)$, where C and W are class expressions,
- a Boolean combination of taxonomic formulas,
- or a quantified formula of the form $\forall x \Phi(x)$ or $\exists x \Phi(x)$ where $\Phi(x)$ is a taxonomic formula.

As in the case of class expressions, the semantics of taxonomic formulas roughly corresponds to simply reading these formulas as if they were English sentences. For example, the formula $\mathbf{Is}(x,\ \textit{A-PERSON})$ is true just in case the value of the variable x is an element of the set denoted by the class expression *A-PERSON*. The formula $\mathbf{Is}(y,\ \textit{A-CHILD-OF}(x))$ is true just in case the pair $<x,\ y>$ is contained in the relation denoted by *A-CHILD-OF*. The formula $\mathbf{Is}(z,\ \textit{A-CHILD-OF}(\textit{A-CHILD-OF}(x)))$ is true just in case there exists some member y of the class $\textit{A-CHILD-OF}(x)$ such that z is a member of the class $\textit{A-CHILD-OF}(y)$. The formula $\mathbf{Is}(x,\ \textit{TIMES}(2\ \textit{A-NUMBER}))$ is true just in case x can be written as the product of 2 and some number, i.e., just in case x is an even number.

We now give a rigorous definition of the semantics of taxonomic formulas and class expressions. Note that a first order model does not provide an interpretation of variables; as in classical syntax the semantic value of an expression containing free variables is determined by a variable interpretation, i.e., a mapping from variables to elements of the semantic domain. The semantics of class expressions and taxonomic formulas can be rigorously defined as follows.

> **Definition:** Let $\mathcal{M}$ be a first order model with semantic domain D and let ρ be a mapping from variables to elements of D. For any class expression C we define the semantic interpretation of C, denoted $\mathcal{M}(C, \rho)$, to be a subset of D determined by the following conditions.
>
> - If v is a variable then $\mathcal{M}(v, \rho)$ is the singleton set containing $\rho(v)$.
> - If c is a constant then $\mathcal{M}(c, \rho)$ is the singleton set containing the element of D that $\mathcal{M}$ assigns to c.
> - If P is a monadic predicate symbol then $\mathcal{M}(P, \rho)$ equals the subset of D that $\mathcal{M}$ assigns to the predicate P.
> - If f is an n-ary function symbol, and $C_1, \cdots, C_n$ are class expressions, then $\mathcal{M}(f(C_1, \cdots, C_n), \rho)$ is the set of all y such that there exist elements $x_1 \ldots, x_n$ in $\mathcal{M}(C_1, \rho), \cdots, \mathcal{M}(C_n, \rho)$ respectively such that y is the value of the function that $\mathcal{M}$ assigns to f when applied to the tuple $<x_1, \cdots, x_n>$.

- If R is an n-ary relation symbol for $n > 1$, and $C_1, \cdots, C_{n-1}$ are class expressions, then $\mathcal{M}(R(C_1, \cdots, C_{n-1}), \rho)$ is the set of all y such that there exist elements $x_1, \ldots, x_{n-1}$ in $\mathcal{M}(C_1, \rho)$, $\cdots$, $\mathcal{M}(C_{n-1}, \rho)$ respectively such that the tuple $<x_1, \cdots, x_{n-1}, y>$ is a member of the relation that $\mathcal{M}$ assigns to the symbol R.

For any taxonomic formula Φ we define the semantic interpretation of Φ, denoted $\mathcal{M}(\Phi, \rho)$, to be either **T** or **F** as determined by the following conditions.

- $\mathcal{M}(\mathbf{Is}(C_1,\ C_2), \rho)$ is **T** if and only if $\mathcal{M}(C_1, \rho)$ is a subset of $\mathcal{M}(C_2, \rho)$.
- $\mathcal{M}(\mathbf{Ex}(C), \rho)$ is **T** if and only if $\mathcal{M}(C, \rho)$ is non-empty.
- $\mathcal{M}(\mathbf{Det}(C), \rho)$ is **T** if and only if $\mathcal{M}(C, \rho)$ has at most one member, i.e., is either empty or a singleton.
- $\mathcal{M}(\mathbf{Int}(C_1,\ C_2), \rho)$ is **T** if and only if the set $\mathcal{M}(C_1, \rho)$ has a non-empty intersection with the set $\mathcal{M}(C_2, \rho)$.
- Boolean combinations and quantified formulas have their standard interpretation.

Much of the standard terminology of classical syntax can be carried over to our taxonomic syntax. Any formula other than a Boolean combination or quantified formula will be called an *atomic formula*, i.e., atomic formulas are taken to be classification formulas, existence formulas, determination formulas, and intersection-formulas. A *literal* is defined to be either an atomic formula or the negation of an atomic formula. A *ground expression* is a formula or class expression that does not contain any variables (either free or bound). A formula is called *satisfiable* if there exists a first order model and a variable interpretation under which that formula is true. A formula that is true under all first order models and all variable interpretations will be called *valid*. A formula is valid if and only if its negation is not satisfiable. We say that a pair $<\mathcal{M}, \rho>$ *satisfies* a set of formulas Σ if $\mathcal{M}(\Psi, \rho)$ is **T** for every element Ψ of Σ. If there exists a pair $<\mathcal{M}, \rho>$ that satisfies Σ then Σ is called *satisfiable*. We write $\Sigma \models \Phi$ if Φ is true under all interpretations of Σ, i.e., if $\mathcal{M}(\Phi, \rho)$ equals **T** for any pair $<\mathcal{M}, \rho>$ that satisfies Σ.

Note that classical terms are a subset of class expressions — class expressions that do not contain predicate or relation symbols, i.e., classes built purely from constants and function symbols, will be called *terms*. Note that under the semantics of taxonomic syntax, terms always denote singleton sets. So if s and t are terms then the formulas $\mathbf{Is}(s,\ t)$ and $\mathbf{Is}(t,\ s)$ are both equivalent to the classical equation $s = t$.

Every atomic formula of classical syntax can be translated directly into a classification formula of taxonomic syntax. As just noted, a classical equation $s = t$ is equivalent to $\mathbf{Is}(s,\ t)$. If R is an n-ary predicate symbol then the classical atomic formula $R(s_1, \cdots, s_{n-1}, w)$ is equivalent to the taxonomic atomic formula $\mathbf{Is}(w,\ R(s_1, \cdots, s_{n-1}))$. Although every classical atomic formula is equivalent to a taxonomic classification formula, there is no corresponding inverse translation from taxonomic atomic formulas to classical atomic formulas. Consider the taxonomic formula $\mathbf{Is}(P,\ Q)$ where P and Q are monadic predicate symbols. It is possible to show that any classical formula equivalent to $\mathbf{Is}(P,\ Q)$ must involve quantifiers — if Φ is a quantifier-free classical formula there exists a model and a variable interpretation in which Φ is true but $\mathbf{Is}(P,\ Q)$ is false. Intuitively, classical quantifier-free formulas can only mention a finite subset of the semantic domain while the atomic formula $\mathbf{Is}(P,\ Q)$ places a constraint on all domain elements. These observations imply that taxonomic ground literals are strictly more expressive than classical ground literals. However, taxonomic atomic formulas can be translated into (quantified) classical formulas. When the full quantified language is considered, our taxonomic syntax is expressively equivalent to classical syntax.

3 Frame Description Languages

Of course there other ways of defining a non-standard syntax for first order logic. Our taxonomic syntax for first order logic is related to a large family of knowledge representation languages known as frame description languages (FDLs) [4], [16], [21], [7]. Each FDL is similar to our taxonomic syntax in that it provides a simple recursive definition of a particular set of class expressions built from constant, function, predicate, and relation symbols.[3] The class expressions of a particular FDL can be considerably different from the class expressions of our taxonomic syntax. For example, all FDLs discussed in the knowledge representation literature include intersection operations on class expressions — given any two class expressions C_1 and C_2 the class expression $\mathbf{AND}(C_1, C_2)$ denotes the intersection of the sets denoted by C_1 and C_2. Various other ways of constructing class expressions are allowed depending on the particular FDL in question. There is no simple relationship between the expressive power of quantifier-free taxonomic syntax as defined here and the FDLs that have been discussed in the literature. For example, the class expression $\forall R.C$ as defined in [7] cannot be expressed in the quantifier-free fragment of our taxonomic syntax. Conversely, the class expression $R(C\ W)$ of our taxonomic syntax cannot be expressed in any of the languages discussed in [7].

For a given FDL one can define at least three decision problems of increasing difficulty which we will call the atomic formula validity problem, the atomic formula entailment problem, and the literal conjunction satisfiability problem. The atomic formula validity problem is just the problem of determining if a single ground classification formula is valid. The atomic formula entailment problem is the problem of determining if a given ground classification formula follows from a finite data base containing ground classification formulas, i.e., does a given classification formula follow from a finite conjunction of other classification formulas. Finally, the literal conjunction satisfiability problem is the problem of determining if a finite conjunction of ground classification formulas and negations of ground classification formulas is satisfiable. For a fixed FDL these three problems are of increasing difficulty — the atomic formula validity problem is just a special case of the atomic formula entailment problem, which is itself essentially a special case of the literal conjunction satisfiability problem.

The knowledge representation literature has focused almost exclusively on the atomic formula validity problem. The atomic formula validity prob-

[3]Within the knowledge representation literature an FDL is not viewed as an alternative syntax for full first order logic. Rather, the formulas of an FDL are restricted to include only classification formulas between ground class expressions. Under this restriction, these languages are far less expressive than full first order logic.

lem for classification formulas is equivalent to the problem of determining whether one class expression necessarily denotes a subset of the set denoted by a second class expression. This particular problem is known in the knowledge representation literature as the *subsumption problem* — the problem of determining whether one class subsumes another class. Several FDLs have been found in which the atomic formula validity problem (the subsumption problem) is non-trivial yet polynomial time decidable [16]. However, we do not know of any published polynomial decision procedures for the atomic formula entailment problem of an FDL. The literal conjunction satisfiability problem has been similarly ignored. In practice one must draw conclusions from a data base of given facts, so a solution to the atomic formula entailment problem is more directly applicable than a solution to the atomic formula validity problem.

The ground classification formulas of the taxonomic syntax defined in this paper constitute a particular FDL. The atomic formula validity problem of this FDL is trivial — a ground classification formula is valid if and only if the two class expressions involved are identical. Despite the trivial nature of the atomic formula validity problem, it is quite difficult to construct decision procedures for the atomic formula entailment problem and literal conjunction satisfiability problem. We show in this paper that, for our taxonomic syntax, the literal conjunction satisfiability problem, and hence the atomic formula entailment problem, is polynomial time decidable.

Given the difficulty we have encountered in constructing a polynomial time decision procedure for the literal conjunction satisfiability problem of our taxonomic syntax, we feel that our syntax represents a distinguished compromise between tractability and expressive power. Our taxonomic syntax is also distinguished by its close relationship to classical syntax — the class expressions of our syntax are constructed from constants, functions, predicates and relations using application as the only method for constructing new classes.

4 Literal Conjunction Satisfiability

(1) $\mathbf{Is}(C,\ C)$

(2) $$\frac{\mathbf{Is}(C,\ W),\ \mathbf{Is}(W,\ Z)}{\mathbf{Is}(C,\ Z)}$$

(3) $$\frac{\mathbf{Ex}(C),\ \mathbf{Is}(C,\ t)}{\mathbf{Is}(t,\ C)}$$

(4) $$\frac{\mathbf{Is}(C_1,\ W_1),\cdots,\mathbf{Is}(C_n,\ W_n)}{\mathbf{Is}(R(C_1,\ldots,C_n),\ R(W_1,\ldots,W_n))}$$

(5) $\mathbf{Ex}(t)$

(6) $$\frac{\neg\mathbf{Is}(C,\ W)}{\mathbf{Ex}(C)}$$

(7) $$\frac{\mathbf{Ex}(C),\ \mathbf{Is}(C,\ W)}{\mathbf{Ex}(W)}$$

(8) $$\frac{\begin{array}{l}\mathbf{Ex}(C_1)\\ \cdots\\ \mathbf{Ex}(C_n)\end{array}}{\mathbf{Ex}(f(C_1,\cdots,C_n))}$$

(9) $$\frac{\mathbf{Ex}(R(s_1,\ldots,s_n))}{\mathbf{Ex}(s_i)}$$

(10) $$\frac{\neg\mathbf{Det}(C)}{\mathbf{Ex}(C)}$$

(11) $$\frac{\mathbf{Int}(C,\ W)}{\mathbf{Ex}(W)}$$

(12) $\mathbf{Det}(t)$

(13) $$\frac{\mathbf{Det}(W),\ \mathbf{Is}(C,\ W)}{\mathbf{Det}(C)}$$

(14) $$\frac{\mathbf{Det}(C_1),\cdots,\mathbf{Det}(C_n)}{\mathbf{Det}(f(C_1,\cdots,C_n))}$$

(15) $$\frac{\mathbf{Ex}(C)}{\mathbf{Int}(C,\ C)}$$

(16) $$\frac{\mathbf{Int}(C,\ W),\ \mathbf{Is}(C,\ Z)}{\mathbf{Int}(Z,\ W)}$$

(17) $$\frac{\mathbf{Int}(C_1,\ W_1),\cdots,\mathbf{Int}(C_n,\ W_n)}{\mathbf{Int}(f(C_1,\ldots,C_n),\ f(W_1,\ldots,W_n))}$$

(18) $$\frac{\mathbf{Int}(C,\ W)}{\mathbf{Int}(W,\ C)}$$

(19) $$\frac{\mathbf{Int}(C,\ W),\ \mathbf{Det}(C)}{\mathbf{Is}(C,\ W)}$$

Figure 1: The inference rules for taxonomic literals. In these rules C, W, and Z range over arbitrary class expressions, t ranges over terms (class expressions built purely from constants and function symbols), R ranges over both function symbols and relation symbols, and f ranges over function symbols.

It is a well known that the satisfiability of a finite set of classical ground literals is polynomial time decidable. In this section we show that this result can be extended to our taxonomic syntax. The significance of this extension lies in the significantly greater expressive power of taxonomic ground literals.

The decision procedure for the classical ground literal satisfiability problem is based on congruence closure for reasoning about equality [13], [8], [17]. Our decision procedure for taxonomic syntax can be viewed as a (non-trivial) adaptation of congruence closure. The congruence closure procedure can be viewed as an implementation of the four basic inference rules for equality — reflexivity, transitivity, symmetry, and substitutivity (congruence). These four basic inference rules are semantically complete for deriving ground equations from ground equations. An analogous set of inference rules for taxonomic literals is given in figure 1. In the transition to taxonomic syntax, the four simple rules for equality have been replaced by 19 rules! Given the simplicity of our taxonomic class expressions, and the clear analogy between classification formulas and classical equalities, it is surprising that so many inference rules are needed. Before presenting the polynomial time decision procedure based on these rules we will present a series of examples of satisfiability problems in an attempt to provide an intuitive justification for the large number of inference rules.

To gain better insight into the need for a large rule set, we will investigate some ways one might attempt to reduce the number of rules required. The most obvious way of reducing the number of rules is to simplify the language by eliminating existence, determined, and intersection formulas. All literals involving atomic formulas other than classification formulas can be replaced by classification formula literals. For example, $\mathbf{Det}(P)$ can be replaced by $\mathbf{Is}(P,\ c)$ where c is a new constant symbol. The most difficult literal to replace is $\neg\mathbf{Int}(P,\ Q)$ which can be replaced by $\mathbf{Is}(f(P),\ a)$, $\mathbf{Is}(f(Q),\ b)$ and $\neg\mathbf{Is}(a,\ b)$ where f is a new function symbol and a and b are new constant symbols.

Definition: A finite set Σ of taxonomic ground literals will be called a *clean premise set* if Σ consists only of classification formulas and negations of classification formulas.

Lemma: If Σ is a finite set of taxonomic ground literals then one can compute, in linear time in the size of Σ, a clean premise set Σ' such that Σ' is satisfiable if and only if Σ is satisfiable.

The classification formulas of taxonomic syntax are analogous to the equalities of classical syntax. In fact, the first four inference rules in figure 1 correspond to the four basic rules of equality — rules 1 through 4 are identical to the rules for equality except that the taxonomic symmetry rule (rule 3) can only be applied to a formula $\mathbf{Is}(C,\ t)$ when C is known to be non-empty and t is a term.[4] This restriction guarantees that the symmetry rule is only applied when the two class expressions involved both denote singleton sets. By restricting the symmetry rule to apply only when both class expressions are terms, one can give a version of rules 1 through 4 that only involves classification formulas. One might hope, by analogy with equality, that this version of rules 1 through 4 would be complete for clean premise sets. Unfortunately, there appears not to be any complete version of rules 1 through 4. To appreciate the difficulties involved, consider the following set of three classification formulas.

$$\mathbf{Is}(a,\ R(P)),\ \mathbf{Is}(P,\ b),\ \mathbf{Is}(P,\ c)$$

In these literals a, b, and c are constant symbols, P is a monadic predicate symbol, and R is a binary relation symbol. These literals together imply $\mathbf{Is}(b,\ c)$. To see this note that the first literal implies that $R(P)$ denotes a non-empty set. The semantics of class expressions is such that this can only happen when P denotes a non-empty set. But the second two literals imply that if P is non-empty then it must contain a single element which is the value assigned to both b and c so we must have $\mathbf{Is}(b,\ c)$. We have not found any version of rules 1 through 4 that can, in themselves, derive $\mathbf{Is}(b,\ c)$ from the above three literals. However, rules 1 through 9 are sufficient to derive $\mathbf{Is}(b,\ c)$. Given the first literal above, inference rules 5 and 7 can be used to derive $\mathbf{Ex}(R(P))$. Inference rule 9 can then be used to derive $\mathbf{Ex}(P)$. Given the literal $\mathbf{Is}(P,\ b)$, the symmetry rule (rule 3) can now be used to derive $\mathbf{Is}(b,\ P)$. Finally, given $\mathbf{Is}(P,\ c)$, transitivity (rule 2) can be used to derive $\mathbf{Is}(b,\ c)$. Note the importance of inference rule 9 in this derivation. Inference rule 9 is a source of complexity for the literal conjunction satisfiability

[4] Recall that terms are class expressions constructed entirely from constants and function symbols.

problem. Rule 9 provides a way of proving formulas of the form $\mathbf{Ex}(C)$ even when there is no term t such that one can derive $\mathbf{Is}(t,\ C)$.

Although existence formulas appear to be essential in any complete inference process, determined and intersection formulas are not. It turns out that inference rules 1 through 9 are complete for determining the satisfiability of clean premise sets. Note that rules 1 through 9 are self-contained in the sense that they only involve classification formulas and existence-formulas. Rules 5 and 6 introduce existence formulas, rules 7 through 9 propagate existence formulas, and rule 3 uses existence formulas in deriving new classification formulas.

> **Intractable Completeness Theorem:** If Σ is a clean premise set then Σ is unsatisfiable if and only if Σ contains a formula of the form $\neg\mathbf{Is}(C,\ W)$ such that the formula $\mathbf{Is}(C,\ W)$ can be derived from Σ using inference rules 1 through 9.

For reasons explained below, we do not give a direct proof of this completeness theorem. Rather, we prove that inference rules 10 through 19 are redundant in the sense that any classification formula provable using the entire rule set is provable from rules 1 through 9 alone. A completeness proof for the entire rule set will then establish completeness for rules 1 through 9. The completeness proof for the entire rule set is not any simpler than a direct proof of the completeness of rules 1 through 9. However, a direct proof of completeness for rules 1 through 9 alone would not provide a polynomial time decision procedure (hence the name *intractable* completeness theorem). Rules 10 through 19 play an essential role in the polynomial time decision procedure discussed below.

It should be pointed out that, although inference rules 1 through 9 are complete for determining satisfiability, they are not complete in the normal sense. Suppose we are trying to determine if $\mathbf{Is}(P,\ Q)$ follows from a clean premise set Σ. It is possible that $\mathbf{Is}(P,\ Q)$ follows but cannot be derived using the above rules. However, if we add $\neg\mathbf{Is}(P,\ Q)$ to Σ then rule 6 can be used to derive $\mathbf{Ex}(P)$, a formula not necessarily derivable from Σ. The additional formula $\mathbf{Ex}(P)$ may lead to a derivation of $\mathbf{Is}(P,\ Q)$ showing that

$\mathbf{Is}(P,\ Q)$ does indeed follow from Σ. The construction of a concrete example of this phenomenon is left as an exercise for the reader.

We now show that inference rules 10 through 19 are redundant (for clean premise sets) relative to rules 1 through 9. If Σ is a clean premise set then one can prove the following facts by simultaneous induction on the length of derivations from Σ using the full rule set.

- If $\mathbf{Det}(C)$ is derivable from Σ using the full rule set then there exists some term t such that the formula $\mathbf{Is}(C,\ t)$ is derivable from Σ using rules 1 through 9.
- If $\mathbf{Int}(C,\ W)$ is derivable from Σ using the full rule set then there exists some class expression Z such that $\mathbf{Ex}(Z)$, $\mathbf{Is}(Z,\ C)$ and $\mathbf{Is}(Z,\ W)$ are all derivable from Σ using rules 1 through 9.
- If $\mathbf{Is}(C,\ W)$ is derivable from Σ using the full rule set then $\mathbf{Is}(C,\ W)$ is derivable from Σ using rules 1 through 9.
- If $\mathbf{Ex}(C)$ is derivable from Σ using the full rule set then $\mathbf{Ex}(C)$ is derivable from Σ using rules 1 through 9.

As mentioned above, inference rules 10 through 19 play an important role in the polynomial time decision procedure for determining satisfiability. The need for inference rules 10 through 19 is best demonstrated through an example. Consider the following set of literals.

- $\mathbf{Is}(a,\ P)$, $\mathbf{Is}(g^6(P),\ P)$
- $\mathbf{Is}(g(a),\ Q)$, $\mathbf{Is}(g^7(Q),\ Q)$
- $\mathbf{Is}(f(P),\ b)$, $\mathbf{Is}(f(Q),\ c)$, $\neg\mathbf{Is}(b,\ c)$

In these literals a, b and c denote constant symbols, f and g denote monadic function symbols, P and Q denote monadic predicate symbols, and $g^n(a)$ is an abbreviation for $g(g(\cdots g(a)))$ with n applications of g. It turns out that this set of seven literals is unsatisfiable. More specifically, inference rules 1 through 5 can be used to derive $\mathbf{Is}(b,\ c)$, contradicting the last literal above. We will refer to rule 1 as reflexivity, rule 2 as transitivity, rule 3 as symmetry and rule 4 as monotonicity. To derive $\mathbf{Is}(b,\ c)$ first consider the pair of literals $\mathbf{Is}(a,\ P)$ and $\mathbf{Is}(g^6(P),\ P)$. From the literal $\mathbf{Is}(a,\ P)$, repeated use of monotonicity allows us to derive $\mathbf{Is}(g^6(a),\ g^6(P))$. From the literal $\mathbf{Is}(g^6(P),\ P)$, transitivity now allows us to derive $\mathbf{Is}(g^6(a),\ P)$. In general, given the literal $\mathbf{Is}(g^6(P),\ P)$, and any literal of the form $\mathbf{Is}(W,\ P)$, monotonicity and transitivity allow us to derive $\mathbf{Is}(g^6(W),\ P)$. So from $\mathbf{Is}(g^6(a),\ P)$ we can derive $\mathbf{Is}(g^6(g^6(a)),\ P)$. And more generally, for any natural number n, monotonicity and transitivity allow us to derive $\mathbf{Is}(g^{6n}(a),\ P)$. Similarly, from the literals $\mathbf{Is}(g(a),\ Q)$ and $\mathbf{Is}(g^7(Q),\ Q)$ monotonicity and transitivity allow us to derive $\mathbf{Is}(g^{7m+1}(a),\ Q)$ for any natural number m. 36 is the first natural number that can be written both as $6n$ and as $7m+1$. Monotonicity and transitivity allow us to derive both $\mathbf{Is}(g^{36}(a),\ P)$ and $\mathbf{Is}(g^{36}(a),\ Q)$. Now monotonicity also allows us to derive $\mathbf{Is}(f(g^{36}(a)),\ f(P))$ and $\mathbf{Is}(f(g^{36}(a)),\ f(Q))$. Given the literals $\mathbf{Is}(f(P),\ b)$ and $\mathbf{Is}(f(Q),\ c)$, transitivity now allows us to derive $\mathbf{Is}(f(g^{36}(a)),\ b)$ and $\mathbf{Is}(f(g^{36}(a)),\ c)$. Symmetry, together with rule 5, now allows us to derive $\mathbf{Is}(b,\ f(g^{36}(a)))$. Finally, transitivity can be used to derive $\mathbf{Is}(b,\ c)$.

The derivation of $\mathbf{Is}(b,\ c)$ from the above literals using rules 1 through 9 requires the construction of a large class expression, $g^{36}(a)$. This class expression is much larger than any class expression appearing in the given literals. This shows a fundamental difference between rules 1 through 9 for taxonomic formulas and the four basic rules for equalities underlying congruence closure. The four rules underlying congruence closure are “local”. More precisely, a derivation from a set of equalities Σ will be called local if *every formula in the derivation is an equality between terms appearing as a subexpression of equations in* Σ. If Σ is an unsatisfiable set of equalities and negations of equalities (analogous to a clean premise set) then there exists a *local* derivation of inconsistency. The locality of the equality rules allows for the construction of a polynomial time decision procedure. Inference rules 1 through 9 for taxonomic syntax are not local in this sense. However, the full set of inference rules is local in the same sense that the four basic equality rules are local — derivations can be restricted to the class expressions that actually appear in the given set of literals. In the following definitions we avoid using the standard symbol $\vdash$ which we reserve as notation for the full first order inference relation discussed in a later section.

> **Definition:** We write $\Sigma \vdash\!\circ\ \Psi$ if Ψ can be derived from Σ using the inference rules of figure 1. We write $\Sigma \vdash\!\circ\ \mathbf{F}$ if Σ contains a formula of the form $\neg\Psi$ such that $\Sigma \vdash\!\circ\ \Psi$.
>
> **Definition:** We write $\Sigma \vdash\!\!\!\dashv \Psi$ if Ψ can be derived from Σ using the inference rules of figure 1 *such that every class expression appearing in the derivation of Ψ also appears as a subexpression of some formula in Σ*. We write $\Sigma \vdash\!\!\!\dashv \mathbf{F}$ if Σ contains a formula of the form $\neg\Psi$ such that $\Sigma \vdash\!\!\!\dashv \Psi$.

Now let Σ be any finite set of taxonomic ground literals (clean or unclean). The following three statements are the main results of this section.

> **Tractability Lemma:** One can determine whether or not $\Sigma \vdash\!\!\!\dashv \mathbf{F}$ is time polynomial in the size of Σ.

> **Tractable Completeness Theorem:** $\Sigma \vdash\!\!\!\dashv \mathbf{F}$ if and only if Σ is unsatisfiable.

> **Locality Corollary:** $\Sigma \vdash\!\!\!\dashv \mathbf{F}$ if and only if $\Sigma \vdash\!\circ\ \mathbf{F}$.

The tractability lemma follows directly from the definition of $\vdash\!\!\!\dashv$. If there are n class expressions appearing in Σ then there are only order n^2 literals that can be constructed from these class expressions. Since the relation $\vdash\!\!\!\dashv$ restricts all derivations to these order n^2 literals, the inference relation $\vdash\!\!\!\dashv$ is polynomial time decidable.

The locality corollary follows directly from the completeness theorem. To see this note that, because $\vdash\!\!\!\dashv$ is simply a restriction of $\vdash\!\circ$, if $\Sigma \vdash\!\!\!\dashv \mathbf{F}$ then $\Sigma \vdash\!\circ\ \mathbf{F}$. Conversely, if $\Sigma \not\vdash\!\!\!\dashv \mathbf{F}$ then, by the completeness theorem, Σ must be satisfiable. The soundness of $\vdash\!\circ$ then implies that $\Sigma \not\vdash\!\circ\ \mathbf{F}$.

Before considering a proof of the completeness theorem we return to a discussion of the following set of literals.

- $\mathbf{Is}(a,\ P)$, $\mathbf{Is}(g^6(P),\ P)$
- $\mathbf{Is}(g(a),\ Q)$, $\mathbf{Is}(g^7(Q),\ Q)$
- $\mathbf{Is}(f(P),\ b)$, $\mathbf{Is}(f(Q),\ c)$, $\neg\mathbf{Is}(b,\ c)$

The last three literals imply that P and Q do not intersect. Using only rules 1 through 9 it is possible to prove $\mathbf{Is}(g^{36}(a),\ P)$ and $\mathbf{Is}(g^{36}(a),\ Q)$. But $g^{36}(a)$ is a large class expression that does not appear in the above literals. Rules 1 through 9 are not, in themselves, local. Under the full rule set, however, if one can prove $\mathbf{Int}(P,\ Q)$ then one can derive a contradiction from the last three literals. Furthermore, under the full rule set one can derive $\mathbf{Int}(g(P),\ Q)$. Starting with this intersection statement, repeated use of rules 17 and 16 allows one to derive all formulas of the form $\mathbf{Int}(g^n(P),\ g^m(Q))$ where $0 \leq n \leq 6$ and $0 \leq m \leq 7$. This includes the formula $\mathbf{Int}(P,\ Q)$. All class expressions in these derivations appear in the above literals so the derivation is local. The general proof of the tractable completeness theorem is quite difficult and is given in appendix A.

Note that the locality corollary is a purely syntactic statement — it expresses the equivalence of two syntactic relations. We have found a purely syntactic proof of this purely syntactic statement. Although we do not give our syntactic proof, a syntactic proof of a similar theorem is given in section 7. Given a syntactic proof of locality it would suffice to prove semantic completeness for $\vdash\!\circ$ rather than $\vdash$. Unfortunately, it does not appear that a semantic proof of completeness for $\vdash\!\circ$ would be much simpler than the proof of completeness for $\vdash$.

5 Socratic Proofs

Although the inference rules given in the previous section are complete for determining the satisfiability of a finite conjunction of ground literals, they do not provide any method of reasoning about Boolean or quantified formulas and are not complete for first order inference. Any knowledge representation system that is able to store arbitrary first order facts must provide additional inference mechanisms if some form of first order completeness is desired. In this section we show how the polynomial time decision procedure discussed in the previous section can be incorporated into a *Socratic proof system* that is complete for first order inference.

Intuitively, a Socratic proof is a series of statements, each of which obviously follows from the earlier statements in the proof, and the last of which is a desired fact or theorem. Formally, we define the notion of an "obvious statement" by a set of inference rules called *rules of obviousness.* A statement is obvious if it can be derived from previous statements in the Socratic proof using the rules of obviousness. The rules of obviousness are selected so that one can determine, in polynomial time, whether or not a given statement is obvious. The formal definition of an obvious statement provides a formal definition of correctness for Socratic proofs — a Socratic proof is correct if every step is obvious in this technical sense. The polynomial time decision procedure for obviousness provides a polynomial time procedure for determining the correctness of Socratic proofs.

This simple notion of a Socratic proof raises a technical difficulty. Consider a Socratic proof, a series of formulas, $\Phi_1, \Phi_2, \cdots, \Phi_n$, where each Φ_i is obvious provided that one has established the preceeding formulas $\Phi_1, \cdots, \Phi_{i-1}$. The problem is that we have defined obviousness in terms of inference rules — a formula is obvious if it can be derived using the rule of obviousness. This implies that Φ_1 can be derived using the rules of obviousness and that Φ_2 can be derived from Φ_1. But this implies that Φ_2 can itself be derived using the given rules. In fact, each Φ_i can be derived directly. So for any correct Socratic proof of the form $\Phi_1, \Phi_2, \cdots, \Phi_n$, the final formula Φ_n must itself be obvious (derivable from the rules of obviousness). Our simple specification of a Socratic proof is degenerate.

To prevent this degeneracy we specify that the individual statements of a Socratic proof be sequents of the form $\Sigma \vdash \Phi$ rather than formulas — our Socratic proof system is a kind of natural deduction system. A sequent of the form $\Sigma \vdash \Phi$ is obvious if Φ can be derived from the premise set Σ using the rules of obviousness. In addition to the rules of obviousness, there are Socratic proof rules that allow for the derivation of non-obvious sequents. For example, if the sequents $\Sigma \cup \{\Psi\} \vdash \Phi$ and $\Sigma \cup \{\neg\Psi\} \vdash \Phi$ are both obvious, then one can derive $\Sigma \vdash \Psi$, even if this last sequent is not obvious. The inference rules that allow for the derivation of non-obvious sequents will be called Socratic inference rules. The overall Socratic proof system is defined by two sets of inference rules — the rules of obviousness and the Socratic rules for deriving non-obvious sequents.

The length of Socratic proofs can be reduced by giving powerful rules of obviousness and thus increasing the set of obvious sequents. The rules of obviousness, however, are constrained by the requirement that they be computationally tractable — they must define a polynomial time decidable inference relation. Although the rules of obviousness must be tractable, they need not be complete in any semantic sense. Completeness is reserved for the Socratic proof system as a whole. Although the inference rules given in the previous section are adequate for determining satisfiability of finite conjunctions of taxonomic ground literals, a more powerful set of rules is used to define obviousness in our Socratic proof system. The inference relation defined by this more powerful set of rules appears not to have any natural semantic characterization — there appears not to be any natural semantics under which the more powerful rules are sound and complete.

The additional rules of obviousness are given in figure 2. These rules allow for inference involving both Boolean and quantified formulas. First consider just the Boolean rules, rules 20 through 30. Rules 20 through 30 only involve disjunction and negation — all other Boolean operations can be viewed as abbreviations for expressions involving disjunction and negation. Rule 29 allows the distinguished formula **F** to be derived whenever one can derive both Φ and $\neg\Phi$. Rule 30 allows any formula to be derived from **F**. In practice the inference process can be terminated whenever one can derive **F**. Rules 20 through 30 are incomplete for Boolean inference. Rules 20 through 29 characterize a limited form of Boolean inference known as Boolean constraint propagation [15]. These rules can also be viewed as a characterization of propositional unit resolution. Intuitively, each rule expresses a local relationship between a Boolean formula and its immediate subformulas.

The incompleteness of rules 20 through 30 for Boolean inference results from an inability to perform case analysis. For the standard representation of $P \rightarrow Q$ in terms of disjunction and negation, the rules allow one to derive

(20) $$\frac{\Psi}{\neg\neg\Psi}$$

(21) $$\frac{\neg\neg\Psi}{\Psi}$$

(22) $$\frac{\Phi}{\Psi \vee \Phi}$$

(23) $$\frac{\Phi}{\Phi \vee \Psi}$$

(24) $$\frac{\begin{array}{l}\Phi \vee \Psi \\ \neg\Phi\end{array}}{\Psi}$$

(25) $$\frac{\begin{array}{l}\Phi \vee \Psi \\ \neg\Psi\end{array}}{\Phi}$$

(26) $$\frac{\begin{array}{l}\neg\Phi \\ \neg\Psi\end{array}}{\neg(\Psi \vee \Phi)}$$

(27) $$\frac{\neg(\Phi \vee \Psi)}{\neg\Phi}$$

(28) $$\frac{\neg(\Phi \vee \Psi)}{\neg\Psi}$$

(29) $$\frac{\begin{array}{l}\Psi \\ \neg\Psi\end{array}}{\mathbf{F}}$$

(30) $$\frac{\mathbf{F}}{\Phi}$$

(31) $$\frac{\forall x \Phi(x)}{\Phi(t)}$$

Figure 2: Additional Rules of Obviousness. In the universal instantiation rule (rule 31), t must be the specified focus term.

Q from P and $P \to Q$. However, the rules do not allow one to derive Q from $P \to Q$ and $\neg P \to Q$ (unless one can derive either P or $\neg P$). This incompleteness can be overcome with the single Socratic rule of case analysis which states that if one can derive $\Sigma \cup \{\Psi\} \vdash \Phi$ and $\Sigma \cup \{\neg\Psi\} \vdash \Phi$ then one can derive $\Sigma \vdash \Phi$.

Now consider the rule of universal instantiation (rule 31). First note that in taxonomic syntax universal formulas can only be instantiated with terms. Instantiating a universal formula with a class expression can result in unsound inference. For example, the formula $\forall x \mathbf{Det}(x)$ is semantically valid, but instantiating this formula with a monadic predicate P results in $\mathbf{Det}(P)$, which is not valid. Although the restriction to terms guarantees soundness, it does not guarantee tractability. Without additional restrictions the addition of the universal instantiation inference rule causes the rules of obviousness to become intractable.[5] To avoid intractability the rule of universal instantiation is restricted using the notion of a *focus term*. A focus term is a term that is explicitly specified in a step of a Socratic proof. If the sequent $\Sigma \vdash \Phi$ is introduced as an obvious sequent in a Socratic proof then this step of the Socratic proof must be accompanied by an explicit specification of a focus object t. The sequent $\Sigma \vdash \Phi$ is acceptable as an obvious sequent if Φ can be derived from Σ using the rules of obviousness where the universal instantiation rule (rule 31) is restricted to the specified focus object.

The rule of universal instantiation could have been incorporated into the Socratic proof rules for deriving non-obvious sequents rather than the rules of obviousness. There are two reasons for incorporating universal instantiation into the rules of obviousness. First, even this restricted version of universal instantiation significantly increases the power of the rules of obviousness. In practice, a data base of facts can be implicitly present in the antecedent of every sequent of a Socratic proof. So, in practice, the antecedent set Σ of a sequent $\Sigma \vdash \Phi$ can contain a very large number of general facts. Many of the these general facts will be universally quantified formulas of

[5]The rules of obviousness given in the previous section subsume the classical rules for equality on terms. Adding an unrestricted rule of universal instantiation results in a set of rules that is complete for equational reasoning. It is well known that semantic entailment between universally quantified equations is undecidable.

the form $\forall x \Phi(x)$. By including the rule of universal instantiation in the rules of obviousness, all facts of the form $\forall x \Phi(x)$ can be applied to the given focus term in determining the obviousness of an individual sequent $\Sigma \vdash \Phi$.[6] If the universal instantiation rule was relegated to the Socratic proof rules then each application of a lemma of the form $\forall x \Phi(x)$ would have to be done explicitly as a separate step in the Socratic proof. The second reason for including universal instantiation in the rules of obviousness is simply to demonstrate that it is possible to include rules for quantifiers in the rules of obviousness while preserving tractability. The proof that rules 1 through 31 are computationally tractable is given in a later section.

Figure 3 contains the Socratic proof rules used in our Socratic proof system. A Socratic proof is a series of lines where each line contains a sequent of the form $\Sigma \vdash \Phi$. The sequents of a Socratic proof are divided into two kinds: obvious sequents and non-obvious sequents. Each obvious sequent must be explicitly associated with a focus term — an obvious sequent $\Sigma \vdash \Phi$ is acceptable if the formula Φ is derivable from Σ using inference rules 1 through 31 where universal instantiation (rule 31) is restricted to the specified focus term. A non-obvious sequent must be derived from earlier sequents using one of the Socratic proof rules shown in figure 3.

The first Socratic inference rule (rule S1) will be called Socratic case analysis. Socratic case analysis, together with the rules of obviousness 20 through 31, provides complete Boolean inference. The second Socratic rule (rule S2) will be called Socratic transitivity. Socratic transitivity is needed to combine obvious sequents. Suppose $\Sigma \vdash \Phi$ is obvious under focus term t_1 and that $\Sigma \cup \{\Phi\} \vdash \Psi$ is obvious under focus term t_2. Because these two obvious sequents involve different focus terms, there may not exist any single focus term under which $\Sigma \vdash \Psi$ is obvious. The Socratic transitivity rule, however, allows one to derive the non-obvious sequent $\Sigma \vdash \Psi$ from

[6]It is possible to allow for more than one focus term. Universal instantiation is then allowed on any of the given focus terms. The use of a set of focus terms greatly increases the power of the rules of obviousness at some cost in computational tractability. In the presence of more than one focus object the cost of determining the obviousness is exponential in the level of quantifier nesting in the given sequent. In practice the level of quantifier nesting remains small. For a fixed level of quantifer nesting the cost remains polynomial in the number of focus objects — the order of the polynomial being determined by the level of quantifier nesting. A more detailed discussion of sets of focus terms can be found in [15].

(S1) $$\frac{\Sigma \cup \{\Psi\} \vdash \Phi \qquad \Sigma \cup \{\neg\Psi\} \vdash \Phi}{\Sigma \vdash \Phi}$$

(S2) $$\frac{\Sigma \vdash \Psi \qquad \Sigma \cup \{\Psi\} \vdash \Phi}{\Sigma \vdash \Phi}$$

(S3) $$\frac{\Sigma \vdash \Phi(x)}{\Sigma \vdash \forall x \Phi(x)}$$

(S4) $$\frac{\Sigma \vdash \neg\mathbf{Is}(x,\ C)}{\Sigma \vdash \neg\mathbf{Ex}(C)}$$

(S5) $$\frac{\Sigma \cup \{\mathbf{Is}(x_1,\ C), \mathbf{Is}(x_2,\ C)\} \vdash \mathbf{Is}(x_1,\ x_2)}{\Sigma \vdash \{\mathbf{Det}(C)\}}$$

(S6) $$\frac{\Sigma \cup \{\mathbf{Is}(x,\ C),\ \mathbf{Is}(x,\ W)\} \vdash \mathbf{F}}{\Sigma \vdash \neg\mathbf{Int}(C,\ W)}$$

(S7) $$\frac{\Sigma \cup \{\mathbf{Is}(x_1,\ C_1), \ldots, \mathbf{Is}(x_n,\ C_n)\} \vdash \neg\mathbf{Is}(t,\ R(x_1, \ldots, x_n))}{\Sigma \vdash \neg\mathbf{Is}(t,\ R(C_1, \ldots, C_n))}$$

(S8) $$\frac{\Sigma \cup \{\mathbf{Is}(x,\ C)\} \vdash \mathbf{Is}(x,\ W)}{\Sigma \vdash \mathbf{Is}(C,\ W)}$$

Figure 3: The Socratic Proof Rules. In these rules C, $C_1, \cdots, C_n$ and W are class expressions, t is a term, R is a predicate or function symbol, and $x, x_1, \cdots, x_n$ are variables that do not appear free in Σ, C, $C_1, \cdots, C_n$, W or t.

the obvious sequents $\Sigma \vdash \Phi$ and $\Sigma \cup \{\Phi\} \vdash \Psi$. The remaining Socratic rules, rules S3 through S8, are forms of universal generalization. The need for universal generalization rules in the Socratic proof system is discussed below.

It is worth noting that from Socratic case analysis (S1), Socratic transitivity (S2), and rule 31 of the rules of obviousness, one can derive a Socratic refutation rule — if there exists a derivation of the sequent $\Sigma \cup \{\neg\Phi\} \vdash \mathbf{F}$ then there exists a derivation of the sequent $\Sigma \vdash \Phi$. To see this, suppose we are given the (possibly non-obvious) sequent $\Sigma \cup \{\neg\Phi\} \vdash \mathbf{F}$. By the second rule for contradictions (rule 31), the sequent $\Sigma \cup \{\neg\Phi, \mathbf{F}\} \vdash \Phi$ is obvious. The Socratic transitivity rule (rule S2) can be applied to these two sequents to give $\Sigma \cup \{\neg\Phi\} \vdash \Phi$. The sequent $\Sigma \cup \{\Phi\} \vdash \Phi$ is obvious. These last two sequents can be combined using Socratic case analysis to give $\Sigma \vdash \Phi$.

The completeness of the Socratic proof system can be proven by a standard Herbrand construction. We will say that a set of formulas Σ is consistent if there is no derivation of the sequent $\Sigma \vdash \mathbf{F}$. The Herbrand construction is used to prove refutation completeness — if Σ is consistent then Σ is satisfiable, or equivalently, if Σ is not satisfiable then there exists a derivation of $\Sigma \vdash \mathbf{F}$. Given the derived refutation rule, refutation completeness implies completeness in the normal sense — if $\Sigma \models \Phi$ then $\Sigma \cup \{\neg\Phi\}$ is unsatisfiable so there must exist a derivation of $\Sigma \cup \{\neg\Phi\} \vdash \mathbf{F}$ and therefore a derivation of $\Sigma \vdash \Phi$. The proof of refutation completeness is given in appendix B.

The Socratic rules of universal generalization (rules S3 through S8) play an important role in the construction of the Herbrand model given in appendix B. In the proof of refutation completeness we are given a consistent set of formulas Σ and we construct a Herbrand model of Σ. Suppose that Σ contains $\mathbf{Ex}(C)$. A Herbrand model of Σ must contain a witness for $\mathbf{Ex}(C)$, i.e., a term t such that the model satisfies $\mathbf{Is}(t,\ C)$. Let x be a variable that does not appear free in Σ. If Σ is consistent then there must not exist a derivation of the sequent $\Sigma \vdash \neg\mathbf{Is}(x,\ C)$, otherwise rule S4 would ensure that there exists a derivation of $\Sigma \vdash \neg\mathbf{Ex}(C)$ and Σ would be inconsistent. But if there is no derivation of $\Sigma \vdash \neg\mathbf{Is}(x,\ C)$ then there cannot be any derivation of $\Sigma \cup \{\mathbf{Is}(x,\ C)\} \vdash \mathbf{F}$ (otherwise the refutation rule would give $\Sigma \vdash \neg\mathbf{Is}(x,\ C)$). So $\Sigma \cup \{\mathbf{Is}(x,\ C)\}$ must be consistent. In summary, if Σ is consistent and contains $\mathbf{Ex}(C)$ then rule S4 ensures that $\Sigma \cup \mathbf{Is}(x,\ C)$ is also consistent for any variable x that does not appear free in Σ. In this way a consistent set of formulas can be extended in a way that provides witnesses (such as the variable x) for existential statements (such as $\mathbf{Ex}(C)$). The rules of universal generalization (S3 through S8) justify a consistency-preserving extension process that provides witnesses for existential statements.

6 Accessing a Large Knowledge Base

Suppose that one is trying to verify a new fact using a large library of definitions and previously verified lemmas. How can a verification system automatically identify those lemmas in the lemma library that are relevant to this new verification? Automtically identifying relevant lemmas is one of the classical problems of automated reasoning. We propose sidesteping this problem by finding a way of efficiently applying *all* the lemmas in a large lemma library. If the decision procedure for the tractable rule set is sufficiently efficient in practice, then it can be practical to determine whether or not a sequent $\Sigma \vdash \Phi$ is obvious (under a given focus object) even if Σ is quite large — even if Σ contains a large lemma library.

Of course we do not want to write Socratic proofs in which, at each step of the proof, we have to explicitly write all the lemmas in some large lemma library. It is much more convenient to write proofs in the presence of "implicit premises" which are automatically added to the premise set of each sequent in the proof. Whether or not a sequent typed by a system user is obvious depends on the particular lemmas automatically added to the premise set of the sequent. We say that the obviousness of a typed sequent is relative to the knowledge base — more knowledge means that more user input sequents will be obvious. As a simple example, consider a knowledge base (a set of formulas) that contains the lemmas

$$\forall x \mathbf{Is}(\texttt{ADAM},\ \texttt{AN-ANCESTOR-OF}(x))$$

and

$$\forall x \mathbf{Is}(\texttt{AN-ANCESTOR-OF}(\texttt{AN-ANCESTOR-OF}(x)),\ \texttt{AN-ANCESTOR-OF}(x)).$$

Now consider the sequent

$$\mathbf{Is}(\texttt{GOD},\ \texttt{AN-ANCESTOR-OF(ADAM)}) \vdash \mathbf{Is}(\texttt{GOD},\ \texttt{AN-ANCESTOR-OF(JOHN)}).$$

This sequent is not obvious (and not valid). However if a knowledge base including the above axioms is added to the premise set the sequent becomes obvious under the focus object `JOHN`.

Consider a sequent of the form $\Sigma \vdash \Phi$ as typed by a user in some verification system. In addition to extending Σ to include all lemmas from a lemma library, it is possible to extend Σ to include all formulas Ψ such that there is some earlier line in the proof of the form $\Sigma' \vdash \Psi$ where Σ' is a subset of Σ. This allows for proofs of the form

Line Number	Sequent	Justification
1.	$\Sigma \vdash \Phi_1$	Focus t_1
2.	$\Sigma \vdash \Phi_2$	Focus t_2
$\vdots$		
n.	$\Sigma \vdash \Phi_n$	Focus t_n

where each Φ_i for $i < j$ is added as an implicit premise in line j.

The Socratic proof system described here is similar to the one used in the Ontic system described in [15]. The Ontic system provides Socratic completeness relative to Zermelo-Fraenkel set theory, i.e., any formula provable in set theory can (in principle) be given a Socratic proof in the Ontic system. The Ontic system has been used to verify the Stone Representation Theorem of lattice theory starting with only the axioms of set theory. In the verification of this proof, a lemma library with hundreds of lemmas was automatically added to the premise set of each sequent. The decision procedure for determining the obviousness of individual sequents was found to be efficient enough to make the verification of obviousness relative to a large lemma library possible.

7 A Decision Procedure for Obviousness

We now show that the rules of obviousness (rules 1 through 31) are tractable. The decision procedure given here is similar to the decision procedure discussed in Section 4. The basic idea is to run the inference rules on a restricted set of "local" formulas. Provided that there is only a small (polynomial) number of local formulas, this inference process can be run to completion in a small (polynomial) amount of time. After defining a polynomial time inference procedure of this form, we prove that the restricted inference process underlying the procedure is complete relative to the rules of obviousness — if the restricted inference procedure fails to find a proof then there is no proof.

We start by defining the notion of a local expression. In the following definitions Σ is a finite set of formulas, t is a term, and Φ is a formula. We are interested in determining whether or not the sequent $\Sigma \vdash \Phi$ is obvious under the focus term t.

Definition: A set Υ of formulas and class expressions will be called *downward closed* under focus term t provided it satisfies the following conditions.

- If $\Psi_1 \vee \Psi_2 \in \Upsilon$ then $\Psi_1 \in \Upsilon$ and $\Psi_2 \in \Upsilon$.
- If $\neg\Psi \in \Upsilon$ then $\Psi \in \Upsilon$.
- If $\forall x \Phi(x) \in \Upsilon$ then $\Phi(t) \in \Upsilon$.
- Any class expression appearing as a subexpression of an atomic formula in Υ is also in Υ.

Definition: A formula or class expression will be called *local* relative to Σ, t and Φ if it is a member of the least set of formulas and class expressions that contains Φ and all members of Σ and that is downward closed relative to t.

Lemma: The number of expressions local to Σ, t and Φ is linear in the total written length of Σ, t and Φ.

The conditions in the definition of downward closed correspond to the ways in which the inference rules can derive information from the truth or falsity of a formula. A truth value for a disjunction can be relevant to determining truth values of the two disjuncts. A universal formula can be used to derive the instantiation of that formula with the given focus object. It is important to note, however, that the conditions in the definition of downward closed are not themselves inference rules — these rules are used only in determining the set of local expressions.

> **Definition:** We write $\Sigma, t \vdash\!\circ\ \Psi$ if Ψ can be derived from Σ using rules 1 through 31 under focus term t.
>
> **Definition:** We write $\Sigma, t \vdash\!\!\!\dashv \Phi$ if $\Sigma, t \vdash\!\circ\ \Phi$ and every formula in the derivation of Φ from Σ is either a local formula, the negation of a local formula, or an atomic formula involving only local class expressions.

The definition of $\Sigma, t \vdash\!\!\!\dashv \Phi$ is similar to the definition of $\Sigma \vdash\!\!\!\dashv \Phi$ given in section 4. The definitions differ in two ways. First, the set of local class expressions is defined slightly differently to take into account quantified formulas and the focus term t. Second, in the definition of $\Sigma \vdash\!\!\!\dashv \Phi$ given earlier, proofs are allowed to contain any formulas as long as all class expressions appearing in those formulas are local. In fact, for the earlier definition, the inference rules can only derive atomic formulas. So the earlier definition is equivalent to the statement that $\Sigma \vdash\!\!\!\dashv \Phi$ just in case there exists a derivation of Φ from Σ such that every derived formula is an atomic formula involving only local class expressions. The introduction of inference rules for formulas other than atomic formulas forces the introduction of local formulas as well as local class expressions. The above definition of $\Sigma, t \vdash\!\!\!\dashv \Phi$ takes into account the local formulas as well as the local class expressions.

The following statements are analogous to those in section 4.

> **Obviousness Tractability Lemma:** One can determine whether or not $\Sigma, t \vdash\!\!\!\dashv \Phi$ is polynomial time in the total written size of Σ, t, and Φ.

Obviousness Locality Theorem: $\Sigma, t \ \vdash\!\!| \ \Phi$ if and only if $\Sigma, t \vdash\!\circ \ \Phi$.

As in section 4, the tractability lemma follows directly from the definition of $\vdash\!\!|$. There is a linear number of local formulas and class expressions. An atomic formula can involve at most two class expressions. Therefore, there is only a quadratic number of formulas that can appear in local derivations. The proof of the locality theorem is more difficult. In section 4 the locality of the inference rules was proven as a corollary of the semantic completeness theorem for local inference. Unfortunately, we do not know of any semantics under which rules 1 through 31 are complete. In this section we sketch a purely syntactic proof of the above locality theorem for rules 1 through 31.

To prove the above locality theorem it suffices to show that if $\Sigma, t \ \not\vdash\!\!| \ \Phi$ then $\Sigma, t \not\vdash\!\circ \ \Phi$. Given $\Sigma, t \not\vdash\!\!| \ \Phi$ we show that it is possible to incrementally "grow" the set of local expressions in such a way that any given expression is eventually considered to be local and so that Φ never becomes provable. If $\Sigma, t \vdash\!\circ \ \Phi$ then there must exist some finite proof of Φ and our growth process would eventually include all the formulas in that proof. If the growth process preserves the invariant that Φ is not provable, then we must have $\Sigma, t \not\vdash\!\circ \ \Phi$.

Definition: A *locality set* for Σ, t and Φ is any set of formulas and class expressions that contains Φ, every member of Σ, and is downward closed relative to t.

Definition: A *label formula* of a set Υ is either a member of Υ, the negation of a member of Υ, or an atomic formula constructed purely from class expressions that are members of Υ.

Definition: If Υ is a locality set for Σ, t and Φ, then for any formula Ψ we write $\Sigma, t \vdash\!\!|_{\Upsilon} \ \Psi$ if $\Sigma, t \vdash\!\circ \ \Psi$ and every formula in the derivation of Ψ is a label formula of Υ.

Given $\Sigma, t \not\vdash\!\!| \ \Phi$, we immediately have $\Sigma, t \not\vdash\!\!|_{\Upsilon} \ \Phi$ where Υ is the least locality set for Σ, t, and Φ. To grow Υ while preserving the non-derivability of Φ we define the notion of a one step extension of Υ.

Definition: A *one step extension* of a set Υ relative to a focus term t is an expression α that is either

- a constant symbol,
- a variable,
- a monadic predicate symbol,
- an application $R(C_1, \ldots, C_n)$ where R is either a relation or function symbol and each C_i is a class expression in Υ,
- an atomic formula such that every class expression in α is a member of Υ,
- the negation of a formula in Υ,
- a disjunction of two formulas in Υ,
- a formula of the form $\forall x \Phi(x)$ where $\Phi(t)$ is a member of Υ.

Lemma: If Υ is a locality set for Σ, t, and Φ, and α is a one step extension of Υ relative to t, then $\Upsilon \cup \{\alpha\}$ is also a locality set for Σ, t and Φ.

Lemma: For any expression β and any locality set Υ for Σ, t and Φ there exists a finite series $\alpha_1, \alpha_2, \cdots, \alpha_n$ such that each α_i is a one step extension of $\Upsilon \cup \{\alpha_1, \cdots, \alpha_{i-1}\}$ and the set $\Upsilon \cup \{\alpha_1, \cdots, \alpha_n\}$ contains β.

Extension Theorem: If Υ is a locality set for Σ, t and Φ such that $\Sigma, t \not\vdash_{\Upsilon} \Phi$, and α is a one step extension of Υ, then $\Sigma, t \not\vdash_{\Upsilon \cup \{\alpha\}} \Phi$.

The above three results imply the obviousness locality theorem. The proof of the extension theorem is long but not conceptually deep (the completeness theorem in appendix A is shorter but conceptually deeper). To prove the extension theorem we define a *new label formula* to be a label formula of $\Upsilon \cup \{\alpha\}$ that is not a label formula of Υ. We define a *newly derivable formula* to be a formula Ψ such that $\Sigma, t \vdash_{\Upsilon \cup \{\alpha\}} \Psi$ but $\Sigma, t \not\vdash_{\Upsilon} \Psi$. It suffices to show that every newly derivable formula is a new label formula, and hence no old label formula is newly derivable.

The proof that every newly derivable formula is a new label formula is discussed in appendix C. The proof involves precisely characterizing all the newly derivable formulas. Proving that this characterization is complete, i.e., that the inference rules only generate new formulas of the given types, requires a large case analysis that examines the interaction of each inference rule with each type of newly derivable formula. Appendix C contains the characterization of the newly derivable label formulas. However, the long case analysis required to show that this characterization is complete is not given here.

It is possible to construct a general theory of local inference relations, i.e., inference relations that can be proven to be tractable using a generalization of the technique described above. In research reported elsewhere we have constructed an automated procedure for verifying the tractability of a large class of rule sets [14]. The large case analysis necessary to establish the tractability of rules 1 through 31 has been machine verified.

8 Conclusion

Automated reasoning is a classical problem of artificial intelligence. Many potential applications, such as software verification, automatic programming, and intelligent data bases have suffered from the apparent computational intractability of automated reasoning. Polynomial time inference procedures provide one approach to improving the efficiency of inference systems.

We have presented an example of a general approach to the construction and use of polynomial time inference procedures. A polynomial time inference procedure can be specified by a set of inference rules that generates a polynomial time decidable inference relation. It seems that the construction of a powerful natural tractable rule set requires the introduction of a non-standard syntax for first order logic. We have shown how a polynomial time inference procedure can be incorporated into a Socratic proof system that is complete for first order inference. The length of proofs in such a Socratic proof system is sensitive to the power of the underlying polynomial time inference procedure. More powerful tractable rule sets should reduce the length of Socratic proofs. Other non-standard syntactic variants of first order logic are possible and such syntactic variants may result in yet more powerful tractable rule sets.

Appendix A: Proof of the Tractable Completeness Theorem

In this appendix we prove the tractable completeness theorem of section 4. The theorem states that the restricted inference relation $\vDash$ for rules 1 through 19 is complete for determining the satisfiability of a set of ground literals in taxonomic syntax. The first step in understanding the proof of the completeness theorem is to become familiar with the rule set. Each rule fits into an overall pattern based on rules 1 through 4. The overall pattern is based on the fact that determined, existence and intersection formulas can all be associated with a notion of "witness". A witness for an existence formula $\mathbf{Ex}(C)$ is a term t such that one can derive $\mathbf{Is}(t,\ C)$. Recall that terms always denote singleton sets, so that if one can derive $\mathbf{Is}(t,\ C)$ then C must be non-empty. A witness for a determined formula $\mathbf{Det}(C)$ is a term t such that one can derive $\mathbf{Is}(C,\ t)$. Finally, a witness for an intersection formula $\mathbf{Int}(C,\ W)$ is a class expression Z such that one can derive $\mathbf{Ex}(Z)$, $\mathbf{Is}(Z,\ C)$ and $\mathbf{Is}(Z,\ W)$. It was noted in section 4 that, for clean premise sets, derivable determined formulas and derivable intersection formulas always have witnesses. If, in addition, every existence formula had a witness, then the four basic inference rules would suffice provided the symmetry rule was restricted to pairs of terms. Unfortunately, inference rules 6 and 9 can introduce existence formulas that do not have witnesses.

The first four inference rules are called reflexivity, transitivity, symmetry, and monotonicity respectively. Rules five and six are called the first and second existence introduction rules. Rule 7 is called the existence transitivity rule and rule 8 is called the existence monotonicity rule. If every existence formula had a witness, then rules 7 and 8 would correspond to transitivity and monotonicity respectively. Rule 9 will be called the painful rule for reasons discussed below. Rules 10 and 11 are called the first and second extraneous existence introduction rules. Rules 10 and 11 are not needed for clean premise sets — they are used to introduce existence formulas from negative determined literals and negative intersection literals in unclean premise sets. Rule 12 is called the determined introduction rule. Rules 13 and 14 are called the determined transitivity and determined monotonicity rules respectively. Rule 15 is called the intersection introduction rule. Rules 16 and 17 are called the intersection transitivity and intersection monotonicity rules respectively. Rule 18 is called the intersection symmetry rule — it compensates for the lack of symmetry in the intersection transitivity rule. Rule 19 is called the feedback rule. Rule 19 allows formulas other than classification formulas to derive new classification formulas. If all determined and intersection formulas had witnesses, then rule 19 would be derivable from the symmetry rule (rule 3).

Unfortunately, the completeness proof is quite difficult. To see why, consider a set Σ of taxonomic ground literals (clean or unclean) such that $\Sigma \not\vdash \mathbf{F}$. We must show that Σ is satisfiable. As in almost all completeness proofs, this is done by constructing a model $\mathcal{M}$ of Σ. In most completeness proofs, the semantic domain of $\mathcal{M}$ consists of a "Herbrand universe" — a set of equivalence classes of terms. If every existential formula $\mathbf{Ex}(C)$ had a witness, i.e., a term t such that one could derive $\mathbf{Is}(t,\ C)$, then one might be able to construct a model for Σ whose domain was equivalence classes of terms. But the painful inference rule (rule 9) introduces existential statements that have no witnesses. One might expect that this problem could be overcome by using equivalence classes of provably non-empty class expressions rather than equivalence classes of terms. But this approach does not work either. The problem is again rule 9. Consider the pair of literals $\mathbf{Is}(P,\ f(P))$ and $\neg\mathbf{Is}(f(P),\ P)$ where P is a monadic predicate symbol and f is a monadic function symbol. These literals state that P is a proper subset of $f(P)$. These literals are satisfiable. For example, P can be interpreted as the positive integers and f as the function that subtracts one. Since these literals are satisfiable, any general completeness proof must provide a way of constructing a model that satisfies them. The second literal implies $\mathbf{Ex}(f(P))$ and rule 9 then implies $\mathbf{Ex}(P)$. Consider an element x_0 of the set denoted by P in any model of these literals. Since the model satisfies $\mathbf{Is}(P,\ f(P))$, the element x_0 must be a member of the class denoted by $f(P)$. But this implies that there a "predecessor" x_1 in P such that x_0 equals $f(x_1)$. By a similar argument x_1 must have a predecessor x_2 and so on. One can show that, in any model of these two literals, P must denote an infinite set. But no simple Herbrand construction yields a model in which P denotes an infinite set. Our completeness proof is not based on a Herbrand construction.

Before defining the (non-Herbrand) model construction used in our proof, it is useful to enumerate some of the properties that we would like the model to have. The inference rules are based on the derivation of atomic formulas

— there are no inference rules for deriving negative literals. To prove completeness of a rule set of this form, the model should exhibit certain "default properties". Intuitively, any atomic formula not derivable from the inference rules should be false in the model. Consider the proof of completeness for the four basic inference rules for classical equality. In the standard proof of completeness of equational reasoning, any equation not provable from the premise set is false in the constructed model. We might say that equations *default to false*. Under our taxonomic model construction process, existence, determined, and intersection formulas default to false — any such atomic formula that is not provably true is false in the constructed model. Unfortunately, the fact that existence formulas default to false does not permit arbitrary classification formulas to default to false. Consider a class expression C such that one cannot derive $\mathbf{Ex}(C)$. If we cannot derive $\mathbf{Ex}(C)$ then $\mathbf{Ex}(C)$ defaults to false. This means that classification formulas cannot always default to false. Suppose that we ca not derive either $\mathbf{Ex}(C)$ or $\mathbf{Is}(C,\ W)$. In this case $\mathbf{Ex}(C)$ defaults to false, which forces $\mathbf{Is}(C,\ W)$ to be true. So $\mathbf{Is}(C,\ W)$ does not default to false. However, our model construction process is designed so that if one *can* derive $\mathbf{Ex}(C)$, then classification formulas of the form $\mathbf{Is}(C,\ W)$ default to false just like all the other atomic formulas. We have not been successful at basing a completeness theorem on any other form of default conditions — attempts to have all classification formulas default to false and to have existence formulas default to true have not been successful.

The partial default properties of classification formulas are not a serious problem regarding completeness for determining satisfiability. Suppose Σ contains a negative literal of the form $\neg\mathbf{Is}(C,\ W)$. In this case we want the formula $\mathbf{Is}(C,\ W)$ to default to false — if Σ contains $\neg\mathbf{Is}(C,\ W)$ then $\mathbf{Is}(C,\ W)$ had better be false in the constructed model. We have assumed that $\Sigma \not\vdash \mathbf{F}$. Since Σ contains $\neg\mathbf{Is}(C,\ W)$, we must have that $\Sigma \not\vdash \mathbf{Is}(C,\ W)$. Since Σ contains $\neg\mathbf{Is}(C,\ W)$, the second existence introduction rule (rule 6) ensures that one can derive $\mathbf{Ex}(C)$. But our model construction process will ensure that if one can derive $\mathbf{Ex}(C)$ then the formula $\mathbf{Is}(C,\ W)$ defaults to false. Given that $\Sigma \not\vdash \mathbf{Is}(C,\ W)$ we will have that $\mathbf{Is}(C,\ W)$ is false in the constructed model.

Although the partial default properties of classification formulas are not a problem in determining the consistency of a set of ground literals, they are a problem for the classical notion of completeness. A failure to derive $\mathbf{Is}(C,\ W)$ does not imply that there exists a model of Σ in which $\mathbf{Is}(C,\ W)$ is false. In fact, rules 1 through 19 are not complete in this traditional sense.

Our model is constructed from the atomic formulas (positive literals) derivable from Σ and observes the default properties for non-derivable formulas discussed above. We do not construct a Herbrand model. The problem with Herbrand constructions involves formulas of the form $\mathbf{Is}(W,\ f(C_1, \cdots, C_n))$. If we can derive this formula then, for every element x in the set denoted by W there must be elements $y_1, \cdots, y_n$ in the sets denoted by $C_1, \cdots, C_n$ respectively such that $f(y_1, \cdots, y_n)$ equals x.[7] The existence of appropriate "predecessor" domain elements for each element of the set denoted by W cannot be guaranteed in any natural Herbrand universe. Our model construction must include a predecessor construction process for handling provable formulas of the form $\mathbf{Is}(W,\ f(C_1, \cdots, C_n))$. This predecessor construction process can itself generate an infinite domain. For example, if we can prove $\mathbf{Is}(P,\ f(P))$ then every element of P must have a predecessor in P and our predecessor construction process will construct an infinite number of elements of P.

The predecessor construction process is only needed for function symbols. Consider a derivable formula of the form $\mathbf{Is}(W,\ R(C_1, \cdots, C_n))$ and consider an element x in the set denoted by W. As in the case of function symbols, x must have predecessors $y_1, \cdots, y_n$ in the sets denoted by $C_1, \cdots, C_n$ respectively such that x is an element of $R(y_1, \cdots, y_n)$. However, if R is a relation symbol then $R(y_1, \cdots, y_n)$ is a set and can include the entire set denoted by W. This allows all elements of W to have the same predecessor tuple. In our model construction process any single tuple of "generic" elements of $C_1, \cdots, C_n$ serves as a single predecessor tuple for all elements of the class $R(C_1, \cdots, C_n)$. The existence of a single predecessor tuple for all elements of the class $R(C_1, \cdots, C_n)$ eliminates the need for a predecessor construction process for relation symbols.

[7] If $y_1, \cdots, y_n$ are elements of the semantic domain of a model $\mathcal{M}$, and f is an n-ary function symbol, then by abuse of notation we use the expression $f(y_1, \cdots, y_n)$ to denote the element of the semantic domain of $\mathcal{M}$ that results from applying the function denoted by f to the domain elements $y_1, \cdots, y_n$. A similar convention is used for an $(n+1)$-ary relation symbol R and the expressions of the form $R(y_1, \cdots, y_n)$ — the expression $R(y_1, \cdots, y_n)$ denotes the set of all domain elements x such that $<y_1, \cdots, y_n, x>$ is in the relation denoted by R.

Our semantic domain is the result of a predecessor construction process. Before discussing this process we consider a way of classifying domain elements into types. Consider an arbitrary model $\mathcal{M}$. The type of a domain element x in $\mathcal{M}$ is defined to be the set of class expressions such that x is an element of the set denoted by the class expression. Intuitively, the type of an element is the set of class expressions that contain it. In general, a *type* is defined to be any set of class expressions. A type τ will be said to be *inhabited* in the model $\mathcal{M}$ if there exists a domain element x such that τ is precisely the type of x, i.e., τ is the set of all class expressions C such that x is an element of the class denoted by C. Not all types need be inhabited in $\mathcal{M}$. For example, suppose that C and W are class expressions that denote disjoint sets in $\mathcal{M}$. In this case no domain element will have a type that includes both C and W.

So far we have ignored the fact that we are trying to prove completeness for a *restricted* inference process. The fact that $\Sigma \not\vdash \mathbf{F}$ implies that there is no *local* derivation of an inconsistency. Local derivations are restricted to formulas containing class expressions that actually appear as subexpressions of formulas in Σ. These class expressions will be called *local*. Our classification of domain elements into types will be based only on local class expressions — types will be subsets of the local class expressions. The desired default properties of the model dictate which subsets of the local class expressions should be allowed to be inhabited.

> **Definition:** A class expression will be called *local* if it appears in Σ (either as a member or as a subexpression of a member).
>
> **Definition:** A Σ-*inhabitable type* is a set τ of local class expressions satisfying the following properties.
>
> - If C is in τ and $\Sigma \vdash \mathbf{Is}(C,\ W)$ then W is in τ.
> - $\Sigma \vdash \mathbf{Ex}(C)$ for every C in τ.
> - For all C and W in τ, $\Sigma \vdash \mathbf{Int}(C,\ W)$.

The first property must be satisfied by any inhabited type in any model of Σ. The second property is forced by the desire for existence formulas to default to false — if we cannot derive $\mathbf{Ex}(C)$ then C should be empty and so C should not be a member of any inhabited type. The third property is dictated by the desire for intersection formulas to default to false — if we cannot derive $\mathbf{Int}(C, W)$ then no inhabited type should contain both C and W.

Roughly speaking, the semantic domain of our model will correspond to the Σ-inhabitable types — each Σ-inhabitable type will be inhabited by at least one domain element. At an intuitive level, the Σ-inhabitable types correspond to the term equivalence classes of a Herbrand model — each Σ-inhabitable type specifies a kind of domain element. Note that the empty set is Σ-inhabitable. Our semantic domain will include elements that are not members of any set denoted by a local class expression.

> **Definition:** If $\Sigma \vdash \mathbf{Ex}(C)$ then we define C^* to be the set of all local class expressions W such that $\Sigma \vdash \mathbf{Is}(C, W)$.
>
> **Lemma:** If $\Sigma \vdash \mathbf{Ex}(C)$ then C is an element of C^* and C^* is a Σ-inhabitable type.
>
> **Proof:** If $\Sigma \vdash \mathbf{Ex}(C)$ then C must be a local class expression. The reflexivity rule (rule 1) guarantees that C is an element of C^*. The transitivity rule guarantees that C^* satisfies the first condition on Σ-inhabitable types. The existence transitivity rule, and the fact that $\Sigma \vdash \mathbf{Ex}(C)$, guarantees the second condition. Given $\mathbf{Ex}(C)$, $\mathbf{Is}(C, W)$, and $\mathbf{Is}(C, Z)$ one can derive $\mathbf{Int}(W, Z)$. This guarantees the third condition and so C^* is a Σ-inhabitable type.
>
> **Lemma:** If $\Sigma \vdash \mathbf{Ex}(C)$ then C^* is the least Σ-inhabitable type that contains C.
>
> **Proof:** The first condition on the definition of Σ-inhabitable types, and the definition of C^*, guarantee that any Σ-inhabitable type that contains C must also contain C^*.

In general, there can be many Σ-inhabitable types that contain C other than the type C^*. For example, if $\Sigma \vdash \mathbf{Int}(C, W)$ then $C^* \cup W^*$ is also a Σ-inhabitable type that contains C. In certain cases, however, C^* is the only such Σ-inhabitable type.

Definition: A local class expression C will be called *singular* if $\Sigma \vDash \mathbf{Ex}(C)$ and $\Sigma \vDash \mathbf{Det}(C)$.

Lemma: If C is singular then C^* is the only Σ-inhabitable type that contains C.

Proof: Consider a Σ-inhabitable type τ that contains C. The type C^* is the least type containing C so C^* must be a subset of τ. To show that τ is a subset of C^*, consider a class expression W in τ. The definition of Σ-inhabitable ensures that $\Sigma \vDash \mathbf{Int}(C,\ W)$. But since $\Sigma \vDash \mathbf{Det}(C)$, the feedback rule (rule 19) ensures that $\Sigma \vDash \mathbf{Is}(C,\ W)$. So W is a member of C^*.

Each domain element is to inhabit a particular Σ-inhabitable type and each Σ-inhabitable type is to be inhabited by at least one domain element. If the above lemma about singular classes had failed, our model construction would not work. If there is to be at least one domain element inhabiting every Σ-inhabitable type, and the set denoted by C is to contain exactly one element, then there must be exactly one type that contains C, the type C^*.

Definition: A Σ-inhabitable type τ will be called *singular* if it contains a singular class expression.

Lemma: If $\Sigma \vDash \mathbf{Ex}(C)$ then C is singular if and only if C^* is singular.

Proof: If C is singular, then since C is a member of C^*, C^* is singular. Conversely, if C^* is singular, then there exists some W such that $\sigma \vDash \mathbf{Is}(C,\ W)$ and $\Sigma \vDash \mathbf{Det}(W)$. But in this case the determined transitivity rule (rule 13) ensures that $\Sigma \vDash \mathbf{Det}(C)$ so C is singular.

We now turn to the actual construction of a semantic domain. The semantic domain elements are pairs of the form $<\tau, \alpha>$ where τ is the Σ-inhabitable type to be inhabited by this domain element. Each type τ will have a distinguished "generic inhabitant" which is the pair $<\tau, 0>$. If τ is a singular type, then the generic inhabitant of τ is the only inhabitant of τ. If τ is not singular then in order to guarantee that determined formulas default to false we add a second inhabitant of the type τ which is the pair $<\tau, 1>$. Other domain elements are constructed by a predecessor generation process. Suppose that τ contains a class expression of the form $f(C_1, \ldots, C_n)$. If the pair $<\tau, \alpha>$ is to inhabit the type τ, then the pair $<\tau, \alpha>$ must be a member of the class $f(C_1, \cdots, C_n)$. This means that the domain element $<\tau, \alpha>$ must have predecessor elements in the classes denoted by $C_1, \cdots, C_n$. If the type τ is singular then generic inhabitants of $C_1^*, \cdots, C_n^*$ serve as the predecessor tuple to the single inhabitant of τ. It is possible to show that if each C_i is a singular class expression, and the type τ contains $f(C_1, \cdots, C_n)$ then τ is a singular type. Thus if τ is not singular, then some C_i is not singular. Let C_i be the first such non-singular class. If τ is not singular, and contains the class expression $f(C_1, \cdots, C_n)$, then the predecessor tuple of $<\tau, \alpha>$ is

$$<C_1^*, 0>, \cdots, <C_{i-1}^*, 0>, \;\; <C_i^*, f(C_1^*, \cdots, C_n^*) \mapsto <\tau, \alpha>>, \;\; <C_{i+1}^*, 0>, \cdots, <C_n^*, 0>.$$

Note that the information about where f should map this predecessor tuple is contained in the i'th component — all other components are simply generic inhabitants. The expression $f(C_1^*, \cdots, C_n^*) \mapsto <\tau, \alpha>$ in the i'th component of this predecessor tuple is just a notation for the tuple containing the function symbol f, the types $C_1^*, \cdots, C_n^*$ and the domain element $<\tau, \alpha>$, i.e., it is just a finite representation of how the function f should behave when given this argument as the i'th component. The semantic domain of our model can be defined rigorously as follows.

Definition: Let D be the least set containing

- all pairs of the form $<\tau, 0>$ where τ is a Σ-inhabitable type,
- all pairs of the form $<\tau, 1>$ where τ is a non-singular Σ-inhabitable type,
- and all pairs of the form $<C_i^*, f(C_1^*, \cdots, C_n^*) \mapsto <\tau, \alpha>>$ where C_i^* is a non-singular type, C_i is the first non-singular class expression among $C_1, \cdots, C_n$, and τ contains the class expression $f(C_1, \cdots, C_n)$.

Lemma: If τ is a singular type then D contains only a single pair of the form $<\tau, \alpha>$.

Note that although there are only finitely many Σ-inhabitable types (for a finite set Σ), the predecessor generation process can cause D to be infinite.

This is ary since there exist finite sets of taxonomic ground literals that only admit infinite models.

To complete the definition of the model of Σ we must give the interpretation of the constant, function, predicate, and relation symbols.

Definition: Let $\mathcal{M}$ be the first order model with domain D (as defined above) and which interprets constant, function predicate and relation symbols as follows.

- A constant c is interpreted to be the pair $<c^*, 0>$.
- A monadic predicate symbol P is interpreted to be the set of all pairs $<\tau, \alpha>$ where the type τ contains the symbol P.
- A n-ary relation symbol R for $n > 1$ is interpreted as the set of n-tuples $<<C_1^*, 0>, \ldots, <C_{n-1}^*, 0>, <\tau, \alpha>>$ such that τ contains the class expression $R(C_1, \ldots, C_{n-1})$.
- An n-ary function symbol f is interpreted as the function that maps $<<\sigma_1, \beta_1>, \ldots, <\sigma_n, \beta_n>>$ to $<\tau, \alpha>$ provided one of the following two conditions hold.
  - Some β_i is the specification $f(\sigma_1, \cdots, \sigma_n) \mapsto <\tau, \alpha>$.
  - No β_i is an appropriate specification, and τ is the union of all types of the form $f(C_1, \ldots, C_n)^*$ where $f(C_1, \cdots, C_n)$ is a local class expression such that $\Sigma \vDash \mathbf{Ex}(f(C_1, \cdots, C_n))$ and each class expression C_i is a member of the corresponding type σ_i. In this case $<\tau, \alpha>$ is the generic inhabitant of type τ, i.e., α is 0.

Well-Formedness Lemma: The above definition interprets each n-ary function symbol as a unique function from D^n to D.

Proof: There are two cases in the definition of the interpretation of a function symbol f. In the first case we must show that it is not possible for distinct β_i and β_j to give incompatible values for f applied to this argument tuple. This follows from the definition of D. If β_i is the specification $f(\sigma_1, \cdots, \sigma_n) \mapsto <\tau, \alpha>$ then the definition of D ensures that σ_i is the first non-singular type among $\sigma_1, \cdots, \sigma_n$. This condition uniquely determines the index i of any appropriate specification. If the second condition in the definition of the interpretation of f holds, the output type τ is taken to be the union of all types of the form $f(C_1, \cdots, C_n)^*$ where $f(C_1, \cdots, C_n)$ is a local class expression such that $\Sigma \vdash \mathbf{Ex}(f(C_1, \cdots, C_n))$ and each class expression C_i is a member of the corresponding type σ_i. In this case we must show that τ is a Σ-inhabitable type. By an earlier lemma, each set of the form $f(C_1, \cdots, C_n)^*$ is a Σ-inhabitable type. To be a Σ-inhabitable type, the set τ must satisfy three conditions. Any union of Σ-inhabitable types satisfies the first two conditions — every element of the union is provably non-empty, and any local class expression that provably denotes a superset of a member of the union is also a member of the union. To show that the union τ satisfies the third condition on Σ-inhabitable types, consider any two class expressions Z_1 and Z_2 in τ. We must show $\Sigma \vdash \mathbf{Int}(Z_1,\ Z_2)$. The expressions Z_1 and Z_2 must be members of types of the form $f(C_1, \cdots, C_n)^*$ and $f(W_1, \cdots, W_n)^*$ respectively where each C_i and W_i are members of the argument type σ_i. Since C_i and W_i are members of the same Σ-inhabitable type σ_i we must have $\Sigma \vdash \mathbf{Int}(C_i,\ W_i)$. But, by the intersection monotonicity rule (rule 17) we then have $\Sigma \vdash \mathbf{Int}(f(C_1, \cdots, C_n),\ f(W_1, \cdots, W_1))$. By the intersection transitivity rule (rule 16) we then have $\Sigma \vdash \mathbf{Int}(Z_1,\ Z_2)$. ∎

The above well-formedness lemma depends critically on the intersection monotonicity rule (rule 17). If this rule were not included in the rule set then the union used to define function application would not produce a Σ-inhabitable type, and hence would not produce a valid domain element. The intersection monotonicity rule ensures that the set of Σ-inhabitable types is rich enough to contain appropriate values for function applications.

We now prove the main lemma of our model construction.

> **Definition:** The *syntactic type* of a domain element $<\tau, \alpha>$ is the type τ.
>
> **Definition:** The *semantic type* of a domain element x is the set of local class expressions C such that x is a member of the set denoted by C in the model $\mathcal{M}$ defined above.
>
> **Lemma:** For any element x of the domain of $\mathcal{M}$, the semantic type of x equals the syntactic type of x.

Proof:. We prove, by structural induction on class expressions, that for any local class expression C, a domain element $<\tau, \alpha>$ is a member of the set denoted by C if and only if C is a member of τ. For constant symbols and monadic predicate symbols the result follows directly from the definition of the interpretation that $\mathcal{M}$ assigns to constant and function symbols. Applications of relations and function symbols are discussed below.

Consider a local class expression of the form $R(C_1, \cdots, C_n)$ where the result holds for each C_i, and consider a domain element $<\tau, \alpha>$. Note that since $R(C_1, \cdots, C_n)$ is local, each C_i must also be local. Now suppose that $R(C_1, \cdots, C_n)$ is a member of τ. Since $R(C_1, \cdots, C_n)$ is a member of a Σ-inhabitable type, we must have $\Sigma \vdash \mathbf{Ex}(R(C_1, \cdots, C_n))$. By the painful rule (rule 9) we must have $\Sigma \vdash \mathbf{Ex}(C_i)$ for each C_i. This implies that C_i^* is a Σ-inhabitable type that contains C_i. By the induction hypothesis the domain elements $<C_1^*, 0>, \cdots, <C_n^*, 0>$ are members of the sets denoted by $C_1, \cdots, C_n$ respectively. The interpretation of the relation symbol R now ensures that $<\tau, \alpha>$ is a member of the class denoted by $R(C_1, \cdots, C_n)$. Conversely, suppose that $<\tau, \alpha>$ is a member of the class denoted by $R(C_1, \cdots, C_n)$. The interpretation of R implies that τ contains a type of the form $R(W_1, \cdots, W_n)$ where each W_i is such that $<W_i^*, 0>$ is a member of the set denoted by C_i. The induction hypothesis implies that C_i is a member of W_i^*. So we have $\Sigma \vdash \mathbf{Is}(W_i,\ C_i)$. The monotonicity rule now ensures that $\Sigma \vdash \mathbf{Is}(R(W_1, \cdots, W_n),\ R(C_1, \cdots, C_n))$. Since $R(W_1, \cdots, W_n)$ is a member of τ, and τ is a Σ-inhabitable type, we have that $R(C_1, \cdots, C_n)$ is a member of τ.

Now consider a local class expression of the form $f(C_1, \cdots, C_n)$ where the result holds for each C_i. There are two subcases. First, suppose that each C_i is singular. In this case the existence and determined monotonicity rules (rules 8 and 14) ensure that $f(C_1, \cdots, C_n)$ is singular. So $f(C_1, \cdots, C_n)^*$ is the only Σ-inhabitable type containing $f(C_1, \cdots, C_n)$ and $<f(C_1, \cdots, C_n)^*, 0>$ is the only domain element of this syntactic type. In this case we must show that $f(C_1, \cdots, C_n)$ denotes the singleton set containing $<f(C_1, \cdots, C_n)^*, 0>$. The induction hypothesis, and the definition of the semantic domain D, implies that, for each C_i, the set denoted by C_i consists of the single element $<C_i^*, 0>$. The interpretation of f then implies that $f(C_1, \cdots, C_n)$ denotes the set containing the single element $<\tau, 0>$ where τ is the union of all types of the form $f(W_1, \cdots, W_n)^*$ where W_i is a member of C_i^* for each W_i. This union includes $f(C_1, \cdots, C_n)^*$, and the monotonicity rule ensures that each set $f(W_1, \cdots, W_n)^*$ is contained in $f(C_1, \cdots, C_n)^*$, so this union equals $f(C_1, \cdots, C_n)^*$.

Now suppose that some C_i is non-singular. Consider a domain element $<\tau, \alpha>$. We must show that $<\tau, \alpha>$ is a member of the class denoted by $f(C_1, \cdots, C_n)$ if and only if τ contains $f(C_1, \cdots, C_n)$. As before, since $f(C_1, \cdots, C_n)$ is local, each C_i must be local. First suppose that $f(C_1, \cdots, C_n)$ is a member of τ. Since $f(C_1, \cdots, C_n)$ is a member of a Σ-inhabitable type, we must have $\Sigma \vdash \mathbf{Ex}(f(C_1, \cdots, C_n))$. By the painful rule (rule 9) we must have $\Sigma \vdash \mathbf{Ex}(C_i)$ for each C_i. This implies that C_i^* is a Σ-inhabitable type that contains C_i. Let C_j be the first non-singular type among $C_1, \cdots, C_n$. The definition of D ensures that D contains the element $<C_j^*, f(C_1^*, \cdots, C_n^*) \mapsto <\tau, \alpha>>$. By the induction hypothesis the domain elements,

$$<C_1^*, 0>, \cdots, <C_{i-1}^*, 0>, \;\; <C_i^*, f(C_1^*, \cdots, C_n^*) \mapsto <\tau, \alpha>>, \;\; <Ci+1^*, 0>, \cdots, <C_n^*, 0>.$$

are members of the sets denoted by $C_1, \cdots, C_n$ respectively. The interpretation of f then ensures that $<\tau, \alpha>$ is a member of the set denoted by $f(C_1, \cdots, C_n)$. Finally, suppose that $<\tau, \alpha>$ is in the set denoted by $f(C_1, \cdots, C_n)$. In this case the sets denoted by $C_1, \cdots, C_n$ must contain elements $<\sigma_1, \beta_1>, \ldots, <\sigma_n, \beta_n>$ respectively such that one of the following two conditions hold.

- Some β_i is the specification $f(\sigma_1, \cdots, \sigma_n) \mapsto <\tau, \alpha>$.
- No β_i is an appropriate specification and τ is the union of all types of the form $f(Z_1, \ldots, Z_n)^*$ where $f(Z_1, \cdots, Z_n)$ is a local class expression such that $\Sigma \vdash \mathbf{Ex}(f(Z_1, \cdots, Z_n))$ and each class expression Z_i is a member of the corresponding type σ_i.

.

In either case, the induction hypothesis implies that, for each C_i, we have that C_i is a member of σ_i. In the first case, the definition of D ensures that each σ_i is of the form W_i^* such that $f(W_1, \cdots, W_n)$ is a member of τ. Since C_i is a member of σ_i, which equals W_i^*, we have $\Sigma \vdash \mathbf{Is}(W_i,\ C_i)$. The monotonicity rule then ensures that $f(C_1, \cdots, C_n)$ is a member of τ. In the second case, note that since σ_i contains C_i, and since σ_i is a Σ-inhabitable type, we have $\Sigma \vdash \mathbf{Ex}(C_i)$. By the existence monotonicity rule (rule 8) we have $\Sigma \vdash \mathbf{Ex}(f(C_1, \cdots, C_n))$. This implies that the set $f(C_1, \cdots, C_n)^*$ is included in the union that defines τ. So $f(C_1, \cdots, C_n)$ is a member of τ. ■

The satisfiability of Σ is a corollary of the equivalence of syntactic and semantic types.

> **Satisfiability Corollary:** $\mathcal{M}$ is a model of Σ and hence Σ is satisfiable.

Proof: We must show that every formula in Σ is true in $\mathcal{M}$. If $\mathbf{Ex}(C)$ is an element of Σ then C^* is a Σ-inhabitable type that includes C so, by the equivalence of semantic and syntactic types, $\mathbf{Ex}(C)$ is true in $\mathcal{M}$. If $\neg\mathbf{Ex}(C)$ is in Σ then, since Σ is consistent, $\Sigma \not\vdash \mathbf{Ex}(C)$ and so no Σ-inhabitable type can contain C. This implies that $\mathbf{Ex}(C)$ is false in $\mathcal{M}$. Now suppose that $\mathbf{Det}(C)$ is an element of Σ. There are two cases. First if $\Sigma \vdash \mathbf{Ex}(C)$ then C is a singular type and only one domain element has a syntactic type that contains C, so $\mathbf{Det}(C)$ is true in $\mathcal{M}$. If $\Sigma \not\vdash \mathbf{Ex}(C)$ then C denotes the empty set in $\mathcal{M}$ so $\mathbf{Det}(C)$ is again true in $\mathcal{M}$. Now suppose Σ contains $\neg\mathbf{Det}(C)$. In this case the first extraneous existence introduction rule (rule 10) ensures that $\Sigma \vdash \mathbf{Ex}(C)$. Since $\Sigma \not\vdash \mathbf{F}$ we have $\Sigma \not\vdash \mathbf{Det}(C)$ and thus C is not singular. The definition of the semantic domain ensures that there are at least two domain elements whose syntactic types include C. Thus $\mathbf{Det}(C)$ is false in $\mathcal{M}$. Now suppose that Σ contains $\mathbf{Int}(C,\ W)$. The second extraneous existence introduction rule (rule 11) ensures that we have $\Sigma \vdash \mathbf{Ex}(C)$ and $\Sigma \vdash \mathbf{Ex}(W)$. In this case $C^* \cup W^*$ is a Σ-inhabitable type that includes both C and W so the literal $\mathbf{Int}(C,\ W)$ is true in $\mathcal{M}$. Now suppose that $\neg\mathbf{Int}(C,\ W)$ is a member of Σ. In this case we have $\Sigma \not\vdash \mathbf{Int}(C,\ W)$ so no Σ-inhabitable type can include both C and W. Thus $\mathbf{Int}(C,\ W)$ is false in $\mathcal{M}$. Now suppose that $\mathbf{Is}(C,\ W)$ is a member of Σ. In this case, every Σ-inhabitable type that includes C must also include W, so $\mathbf{Is}(C,\ W)$ is true in $\mathcal{M}$. Finally, suppose that $\neg\mathbf{Is}(C,\ W)$ is a member of Σ. In this case the second existence introduction rule (rule 6) ensures that $\Sigma \vdash \mathbf{Ex}(C)$. But we have $\Sigma \not\vdash \mathbf{Is}(C,\ W)$. In this case C^* is a Σ-inhabitable type that includes C but does not include W. So $\mathbf{Is}(C,\ W)$ is false in $\mathcal{M}$. ■

Appendix B: Socratic Completeness

This appendix gives a sketch of a proof that the Socratic proof system defined by the rules of obviousness (rules 1 through 31) and the Socratic proof rules (rules S1 through S8) is complete for first order inference. This proof is based on a standard Herbrand construction analogous to similar well known constructions for other first order inference systems [1]. Consider a set of formulas Σ and a particular formula Φ such that $\Sigma \models \Phi$, i.e., every model of Σ is a model of Φ. We must show that in this case there exists a Socratic derivation of the sequent $\Sigma \vdash \Phi$. It was shown in section 5 that if there exists a Socratic derivation of $\Sigma \cup \{\neg\Phi\} \vdash \mathbf{F}$ then there exists a Socratic derivation of $\Sigma \vdash \Phi$. So it suffices to show that there exists a Socratic derivation of $\Sigma \cup \{\neg\Phi\} \vdash \mathbf{F}$. More generally, we show that for any unsatisfiable set of formulas Γ there exists a Socratic derivation of $\Gamma \vdash \mathbf{F}$. Actually, we prove the contrapositive, that if there is no derivation of $\Gamma \vdash \mathbf{F}$ then Γ is satisfiable.

Suppose that there is no derivation of $\Gamma \vdash \mathbf{F}$. We show that Γ is satisfiable by constructing a Herbrand model of Γ. This Herbrand model must assign a well defined truth value to every sentence (closed formula) of taxonomic syntax. However, the formula set Γ need not determine a truth value for every sentence. Before constructing a Herbrand model, we extend Γ to a consistent set of formulas that assigns a truth value to every formula. In addition to assigning truth values to every formula, we ensure that every true existential statement has a witness. For example, if $\mathbf{Ex}(C)$ is determined to be true, then there will exist some term t such that $\mathbf{Is}(t,\ C)$ is also assigned true. In summary, we extend Γ to a larger set so that every formula is assigned a truth value and every true existential statement has a witness.

For simplicity we assume that the set of constant, function and predicate symbols in the language is countable and that there is a countably infinite set of variables. In this case one can enumerate all taxonomic formulas in an infinite sequence Θ_1, Θ_2, Θ_3 ....[8] Given that there is no derivation of $\Gamma \vdash \mathbf{F}$, one can then construct an infinite sequence of sets of formulas Ω_0, Ω_1, Ω_2, Ω_3 ... such that each Ω_i is a consistent extension of Γ, i.e., Ω_i contains Γ

[8] The completeness proof can be modified to handle uncountable languages, in which case one constructs a transfinite enumeration of formulas.

and there is no Socratic derivation of $\Omega_i \vdash \mathbf{F}$. Furthermore, Ω_i determines a well defined truth value for Θ_i and if Θ_i is some form of existential formula then Ω_i ensures the existence of a witness for that formula. The sequence of extensions $\Omega_0, \Omega_1, \Omega_2, \Omega_3, \cdots$, can be defined by stating that Ω_0 equals Γ and for $i > 0$, Ω_i is constructed from Ω_{i-1} as follows.

1. If Θ_i is a Boolean combination of other formulas then if $\Omega_{i-1} \vdash \Theta_i$ is derivable, then Ω_i equals Ω_{i-1}, otherwise Ω_i equals $\Omega_{i-1} \cup \{\neg\Theta_i\}$.

2. If Θ_i is of the form $\forall x \Phi(x)$ then, if $\Omega_{i-1} \vdash \forall x \Phi(x)$ is derivable then Ω_i equals Ω_{i-1}, otherwise Ω_i equals $\Omega_{i-1} \cup \{\neg\forall x \Phi(x), \neg\Phi(x)\}$ where x is a variable that does not appear free in Θ_i or Ω_{i-1}.

3. If Θ_i is of the form $\mathbf{Ex}(C)$ then, if $\Omega_{i-1} \vdash \neg\mathbf{Ex}(C)$ is derivable then Ω_i equals Ω_{i-1}, otherwise Ω_i equals $\Omega_{i-1} \cup \{\mathbf{Ex}(C), \mathbf{Is}(x,\ C)\}$ where x is a variable that does not appear free in Θ_i or Ω_{i-1}.

4. If Θ_i is of the form $\mathbf{Det}(C)$ then, if $\Omega_{i-1} \vdash \mathbf{Det}(C)$ is derivable then Ω_i equals Ω_{i-1}, otherwise Ω_i equals $\Omega_{i-1} \cup \{\neg\mathbf{Det}(C),\ \mathbf{Is}(x_1,\ C),\ \mathbf{Is}(x_2,\ C),\ \neg\mathbf{Is}(x_1,\ x_2)\}$ where x_1 and x_2 are variables that do not appear free in Θ_i or Ω_{i-1}.

5. If Θ_i is of the form $\mathbf{Int}(C,\ W)$ then, if $\Omega_{i-1} \vdash \neg\mathbf{Int}(C,\ W)$ is derivable then Ω_i equals Ω_{i-1}, otherwise Ω_i equals $\Omega_{i-1} \cup \{\mathbf{Int}(C,\ W), \mathbf{Is}(x,\ C), \mathbf{Is}(x,\ W)\}$ where x is a variable that does not appear free in Θ_i or Ω_{i-1}.

6. If Θ_i is of the form $\mathbf{Is}(t,\ f(C_1, \cdots, C_n))$ where t is a term and f is a function symbol, then if $\Omega_{i-1} \vdash \neg\mathbf{Is}(t,\ f(C_1, \cdots, C_n))$ is derivable then Ω_i equals Ω_{i-1}, otherwise Ω_i equals $\Omega_{i-1} \cup \{\mathbf{Is}(t,\ f(C_1, \cdots, C_n)),\ \mathbf{Is}(x_1,\ C_1),\ \cdots,\ \mathbf{Is}(x_n,\ C_n),\ \mathbf{Is}(t,\ f(x_1, \cdots, x_n))\}$ where $x_1, \cdots, x_n$ are variables that do not appear free in Θ_i or Ω_{i-1}.

7. If Θ_i is of the form $\mathbf{Is}(C,\ W)$, but not of the form of case 6 then, if $\Omega_{i-1} \vdash \mathbf{Is}(C,\ W)$ is derivable then Ω_i equals Ω_{i-1}, otherwise Ω_i equals $\Omega_{i-1} \cup \{\neg\mathbf{Is}(C,\ W), \mathbf{Is}(x,\ C), \neg\mathbf{Is}(x,\ W)\}$ where x is a variable that does not appear free in Θ_i or Ω_{i-1}.

There is a direct relationship between cases 2 through 7 above and the Socratic inference rules S3 through S8. For each of the cases 2 through 7 the corresponding rule in S3 through S8 guarantees that if there no derivation of $\Omega_{i-1} \vdash \mathbf{F}$ then there is no derivation of $\Omega_i \vdash \mathbf{F}$. We leave it the reader to verify that for each Ω_i there does not exist a derivation of $\Omega_i \vdash \mathbf{F}$ but there does exist a derivation of either $\Omega_i \vdash \Theta_i$ or $\Omega_i \vdash \neg\Theta_i$. Cases 2 through 7 guarantee the existence of witnesses for all existential claims. For example, if Θ_i is $\mathbf{Ex}(C)$ and there is a derivation of $\Omega_i \vdash \Theta_i$, then there exists a variable x such that there is a derivation of $\Omega_i \vdash \mathbf{Is}(x,\ C)$.

Now let Ω be the set of all formulas Ψ such that there exists some Ω_i such that there is a derivation of $\Omega_i \vdash \Psi$. Note that Ω contains the original set Γ. Because all formulas are elements of the sequence $\Theta_1, \Theta_2, \cdots$, the set Ω is complete in the sense that for any formula Ψ either $\Psi \in \Omega$ or $\neg\Psi \in \Omega$. Furthermore, there cannot be any derivation of $\Omega \vdash \mathbf{F}$ because, since any finite derivation will involve only a finite subset of Ω, if there exists a derivation of $\Omega \vdash \mathbf{F}$ then there must be some Ω_i such that there exists a derivation of $\Omega_i \vdash \mathbf{F}$. Furthermore, the set Ω is closed under all of the rules of obviousness (rules 1 through 31) where the rule of universal instantiation (rule 31) is no longer restricted to focus terms. Finally, cases 2 through 7 above guarantee the existence of important existential witnesses.

One can now define a first order structure whose domain consists of equivalence classes of terms. More specifically, for any variable t we define $|t|$ to be the set of terms t' such that the formula $\mathbf{Is}(t,\ t')$ is a member of Ω. The rules of obviousness for classification formulas ensure that these sets form equivalence classes. We take the domain of the first order structure to be the collection of equivalence classes of the form $|t|$. It is now possible to define an interpretation of the variables, constants, functions, relations, and predicate symbols such that the semantic value of a class expression C equals the elements of the form $|t|$ such that the formula $\mathbf{Is}(t,\ C)$ is a member of Ω. Given this first order model, one can define a variable interpretation ρ that maps each variable x to the domain element $|x|$. Under this model and variable interpretation one can show that a formula Ψ is semantically true just in case Ψ is a member of Ω. This shows that the original set Γ is satisfiable.

Appendix C: Proof of the Extension Theorem

This appendix contains a proof of the extension theorem stated in section 7. The extension theorem states that if Υ is a locality set for Σ, t, and Φ such that $\Sigma, t \not\vdash_{\Upsilon} \Phi$, then for any one-step extension α of Υ we have that $\Sigma, t \not\vdash_{\Upsilon \cup \{\alpha\}} \Phi$. The following definitions will be useful in proving this result.

> **Definition:** A *new label formula* is a label formula of $\Upsilon \cup \{\alpha\}$ that is not a label formula of Υ. A *previously derivable formula* is formula Ψ such that $\Sigma, t \vdash_{\Upsilon} \Psi$. A *newly derivable formula* is a formula Ψ that was not previously derivable but such that $\Sigma, t \vdash_{\Upsilon \cup \{\alpha\}} \Psi$.

Since Φ is a member of Υ, Φ is a label formula of Υ. To prove that $\Sigma, t \not\vdash_{\Upsilon \cup \{\alpha\}} \Phi$ it suffices to show that every newly derivable formula is a new label formula. This is done by proving that every new label formula falls into one of several very specific types. A characterization of the newly derivable formulas can be viewed as an invariant on the inference process — the inference process maintains the invariant that every newly derivable formula falls into one of a specified set of formula types.

The structure of the newly derivable label formulas depends on the structure of the extension expression α. If α is an atomic formula then α is a label formula of (but not a member of) Υ. In this case the only new label formula is $\neg\alpha$ which is not newly derivable, and there are no newly derivable formulas. If α is a quantified formula of the form $\forall x \Phi(x)$ then both α and $\neg\alpha$ are new label formulas. However, neither of these formulas are newly derivable so there are no newly derivable formulas. If α is a negation of the form $\neg\Psi$ then α is a label formula of (but not a member of) Υ. In this case the only new label formula is $\neg\neg\Psi$ and $\neg\neg\Psi$ is newly derivable if and only if Ψ was previously derivable. In this case $\neg\neg\Psi$ is the only possible newly derivable formula. If α is a disjunction $\Psi_1 \vee \Psi_2$ then the only new label formulas are $\Psi_1 \vee \Psi_2$ and $\neg(\Psi_1 \vee \Psi_2)$. The label formula $\Psi_1 \vee \Psi_2$ is newly derivable provided one of Ψ_1 and Ψ_2 was previously derivable. The formula $\neg(\Psi_1 \vee \Psi_2)$ is newly derivable provided both $\neg\Psi_1$ and $\neg\Psi_2$ were previously derivable.

Now we consider the case where the extension expression α is a class expression. If α is a class expression then the new label formulas are all atomic formulas involving the new local class expression α, e.g., formulas of the form $\mathbf{Ex}(\alpha)$, $\mathbf{Det}(\alpha)$, $\mathbf{Is}(\alpha,\ C)$, $\mathbf{Is}(C,\ \alpha)$, $\mathbf{Int}(C,\ \alpha)$, and $\mathbf{Int}(\alpha,\ C)$ where C is a class expression in $\Upsilon \cup \{\alpha\}$. We must show that if α is a class expression then every newly derivable formula is an atomic formula involving α. If α is a monadic predicate symbol then the only newly derivable formula is $\mathbf{Is}(\alpha,\ \alpha)$. If α is a constant symbol or a variable then the only newly derivable formulas are $\mathbf{Is}(\alpha,\ \alpha)$, $\mathbf{Ex}(\alpha)$, and $\mathbf{Det}(\alpha)$. Now suppose that α is an application $R(C_1, \cdots, C_n)$ where each class expression C_i is a member of Υ. In this case one can show that every newly derivable formula is of one the types listed below. In each of these types the newly derivable formula is a new label formula, so the extension theorem follows from the statement that every newly derivable formula is of one of these types.

- The formula $\mathbf{Is}(R(C_1, \cdots, C_n),\ R(C_1, \cdots, C_n))$.
- A formula of the form $\mathbf{Is}(Z,\ R(C_1, \cdots, C_n))$ where there exists an expression $R(W_1, \cdots, W_n)$ in Υ such that the formulas $\mathbf{Is}(Z,\ R(W_1, \cdots, W_n))$, $\mathbf{Is}(W_1,\ C_1)$, $\ldots$, $\mathbf{Is}(W_n,\ C_n)$ were previously derivable.
- A formula of the form $\mathbf{Is}(R(C_1, \cdots, C_n),\ Z)$ where there exists an expression $R(W_1, \cdots, W_n)$ in Υ such that the formulas $\mathbf{Is}(C_1,\ W_1)$, $\ldots$, $\mathbf{Is}(C_n,\ W_n)$, $\mathbf{Is}(R(W_1, \cdots, W_n),\ Z)$ were previously derivable.
- A formula of the form $\mathbf{Is}(Z,\ R(C_1, \cdots, C_n))$ where there exists an expression $R(W_1, \cdots, W_n)$ in Υ such that the formulas $\mathbf{Is}(Z,\ R(W_1, \cdots, W_n))$, $\mathbf{Det}(R(W_1, \cdots, W_n))$, $\mathbf{Int}(W_1,\ C_1)$, $\ldots$, $\mathbf{Int}(W_n,\ C_n)$ were previously derivable.
- The formula $\mathbf{Ex}(R(C_1, \cdots, C_n))$ where R is a function symbol and the formulas $\mathbf{Ex}(C_1)$, $\ldots$, $\mathbf{Ex}(C_n)$ were previously derivable.
- The formula $\mathbf{Ex}(R(C_1, \cdots, C_n))$ where there exists a class expression $R(W_1, \cdots, W_n)$ in Υ such that $\mathbf{Ex}(R(W_1, \cdots, W_n))$, $\mathbf{Is}(W_1,\ C_1)$, $\ldots$, $\mathbf{Is}(W_n,\ C_n)$ were previously derivable.
- The formula $\mathbf{Det}(R(C_1, \cdots, C_n))$ where R is a function symbol and the formulas $\mathbf{Det}(C_1)$, $\ldots$, $\mathbf{Det}(C_n)$ were previously derivable.

- The formula $\mathbf{Det}(R(C_1, \cdots, C_n))$ where there exists a class expression $R(W_1, \cdots, W_n)$ in Υ such that $\mathbf{Det}(R(W_1, \cdots, W_n))$, $\mathbf{Is}(C_1,\ W_1)$, $\ldots$, $\mathbf{Is}(C_n,\ W_n)$ were previously derivable.

- The formula $\mathbf{Int}(R(C_1, \cdots, C_n),\ R(C_1, \cdots, C_n))$ where $\mathbf{Ex}(R(C_1, \cdots, C_n))$ is newly derivable.

- A formula of the form $\mathbf{Int}(R(C_1, \cdots, C_n),\ Z)$ or $\mathbf{Int}(Z,\ R(C_1, \cdots, C_n))$ where there exists a class expression $R(W_1, \cdots, W_n)$ in Υ such that the formulas $\mathbf{Int}(Z,\ R(W_1, \cdots, W_n))$, $\mathbf{Is}(W_1,\ C_1)$, $\ldots$, $\mathbf{Is}(W_n,\ C_n)$ were previously derivable.

- A formula of the form $\mathbf{Int}(R(C_1, \cdots, C_n),\ Z)$ or $\mathbf{Int}(Z,\ R(C_1, \cdots, C_n))$ where R is a function symbol and there exists an expression $R(W_1, \cdots, W_n)$ in Υ such that the formulas $\mathbf{Is}(R(W_1, \cdots, W_n),\ Z)$, $\mathbf{Int}(W_1,\ C_1)$, $\ldots$, $\mathbf{Int}(W_n,\ C_n)$ were previously derivable.

The inference process maintains the invariant that every newly derivable formula is of one of the above types. To prove this one can consider each inference rule and consider each way that the rule might be applied to both newly derivable and previously derivable formulas. If we assume that each newly derivable formula is of one of the above types, then one can show that no matter how the rules are applied to both previously derivable and newly derivable formulas, every derivable formula is either previously derivable or is of one of the above types.

The large case analysis involved in considering each inference rule and each way that the rule might be applied is not given here. Proofs of this type, i.e., proofs involving a large invariant and a large case analysis, are difficult to verify and subject to error. Unfortunately, we have not been able to find any simpler proof. On the other hand, it is possible to construct a general theory of "local" inference rules — a rule set is local if it satisfies an appropriate generalization of the extension theorem given here. It is also possible to construct a mechanical procedure that can determine in a large number of cases whether or not a given rule set is local. Such a procedure is described in [14]. It is straightforward to verify that when the extension α is an application term then only inference rules that can derive new label formulas are rules 1 through 19. The fact that rules 1 through 19 maintain the invariant given above has been mechanically verified using the general procedure for determining locality. The formula types in the above list were mechanically generated by this procedure.

References

[1] John Bell and Moshe Machover. *A Course in Mathematical Logic.* North-Holland, 1977.

[2] R. S. Boyer Bledsoe, W. W. and W. H. Henneman. Computer proofs of limit theorems. *Artificial Intelligence*, 3:27–60, 1972.

[3] Robert S. Boyer and J Struther Moore. *A Computational Logic.* ACM Monograph Series. Academic Press, 1979.

[4] Ronald Brachman and James Schmolze. An overview of the kl-one knowledge representation system. *Computational Intelligence*, 9(2):171–216, 1985.

[5] J. M. Crawford and Benjamin Kuipers. Towards a theory of access-limited logic for knowledge representation. In *First International Conference on Principles of Knowledge PUBLISHER = Morgan Kaufmann Publishers, Representation and Reasoning*, pages 67–78, 1989.

[6] Martin Davis. Obvious logical inferences. In *Proceedings of AAAI-81*, pages 530–531. Morgan Kaufmann Publishers, 1981.

[7] Francesco Donini, Maurizio Lenzerini, Daniele Nardi, and Werner Nutt. The complexity of concept languages. In *Proceedings of KR91*, pages 151–162. Morgan Kaufmann Publishers, 1991.

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

[9] William Downing and Jean H. Gallier. Linear time algorithms for testing the satisfiability of propositional horn formulae. *Journal of Logic Programming*, 1(3):267–284, 1984.

[10] G. Sivikamur Kapur, D. and H. Zhang. Rrl: A rewrite rule laboratory. In *8th CADE*. Springer Verlag, 1986.

[11] Jussi Ketonen. Ekl - a mathematically oriented proof checker. In *7th CADE*, pages 65–79. Springer-Verlag, 1984.

[12] Kurt Konolidge. *A Deduction Model of Belief*. Morgan Kaufmann, 1986.

[13] Dexter C. Kozen. Complexity of finitely presented algebras. In *Proceedings of the Ninth Annual ACM Symposium on the Theory of Computation*, pages 164–177, 1977.

[14] D. McAllester. Automatic recognition of tractability in inference relations. Memo 1215, MIT Artificial Intelligence Laboratory, February 1990. To appear in JACM.

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

[16] Bernhard Nebel. Terminological reasoning is inherently intractable. *Artificial Intelligence*, 43:235–249, 1990.

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

[18] D. Park. Finiteness is μ-ineffable. *Theoretical Computer Science*, 3(2):173–181, 1976.

[19] Judea Pearl. *Probilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[20] V. Pratt. On specifying verifiers. In *Proceedings of the 7th Annual ACM Symposium on Principles of Programming Languages*, 1980.

[21] M. Schmidt-Schaub and G. Smalka. Attributive concept descriptions with complements. *Artificial Intelligence*, 47, 1991.

[22] Christoph Walther. Many sorted unification. *JACM*, 35(1), 1988.