SET CONTAINMENT INFERENCE AND SYLLOGISMS

Paolo Atzeni
D. Stott Parker

March 1988
CSD-880022

Set Containment Inference and Syllogisms †

Paolo Atzeni

IASI-CNR, Viale Manzoni 30, 00185 Roma, ITALY

D. Stott Parker

UCLA Computer Science Dept., Los Angeles, CA, USA 90024-1596

ABSTRACT

Type hierarchies and type inclusion (*isa*) inference are now standard in many knowledge representation schemes. In this paper, we show how to determine consistency and inference for collections of statements of the form

mammal isa vertebrate.

These *containment* statements relate the contents of two sets (or types). The work here is new in permitting statements with negative information: disjointness of sets, or non-inclusion of sets. For example, we permit the following statements also:

mammal isa **non**(*reptile*)
non(*vertebrate*) *isa* **non**(*mammal*)
not(*reptile isa amphibian*)

Binary containment inference is the problem of determining the consequences of positive constraints P and negative constraints **not**(P) on sets, where positive constraints have the form P: $X \subseteq Y$. Negations of these constraints therefore have the form **not**(P): $X \cap$ **non**(Y) $\neq \emptyset$, so positive constraints assert containment relations among sets, and negative constraints assert that two sets have a non-empty intersection.

We show binary containment inference is solved by rules essentially equivalent to Aristotle's *Syllogisms*. Necessary and sufficient conditions for consistency, as well as sound and complete sets of inference rules, are presented for binary containment. The sets of inference rules are compact, and lead to polynomial-time inference algorithms, so permitting negative constraints does not result in intractability for this problem.

† Revision of UCLA Computer Science Department Technical Report CSD-860012, December 1986. Revised July 1987. To appear, *Theoretical Computer Science*.

CONTENTS

1. Introduction

Before plunging into a formal presentation, let us explain why set containment inference is interesting. We are concerned with exploring inference properties of collections of statements like *X isa Y*, where *X* and *Y* are *types*. For example, we can assert

mammal isa vertebrate
reptile isa vertebrate.

Intuitively, types represent sets of individuals, and *isa* represents containment among sets. Statements of this kind form an interesting class of constraints for real knowledge representation problems, since it permits declaration of containment relationships among types. 'Taxonomic' knowledge of this kind is (apparently) basic to human intelligence.

This paper departs from previous work by permitting negation in two ways:

- First, we let *X* and *Y* represent *complements of types*. For example, we permit statements of the form

  mammal isa **non**(*reptile*)

  where **non**(*reptile*) represents the complement of the type *reptile*. This statement asserts that *mammals* are disjoint from *reptiles*.

- Second, we allow negative statements to be made. For example, we permit statements like

  not(*reptile isa amphibian*)

  which asserts that a *reptile* is *not* a type of *amphibian*, or equivalently, that *reptile* must intersect with **non**(*amphibian*). In other words, some non-amphibian reptiles exist.

For example, with these containment statements we can express the following facts about computer components:

	heat_sensitive_device	*isa*	*component*	
	axial_lead_device	*isa*	*component*	
	resistor	*isa*	*axial_lead_device*	
	resistor	*isa*	**non**(*heat_sensitive_device*)	
	half_watt_resistor	*isa*	*resistor*	
	diode	*isa*	*heat_sensitive_device*	
	diode	*isa*	*axial_lead_device*	
	microprocessor	*isa*	*heat_sensitive_device*	
	microprocessor	*isa*	**non**(*axial_lead_device*)	
not(	*heat_sensitive_device*	*isa*	**non**(*axial_lead_device*)	)
not(	*axial_lead_device*	*isa*	**non**(*heat_sensitive_device*)	).

The last two statements say the same thing; namely, that the *heat_sensitive_device* and *axial_lead_device* types *intersect*. For example, *diodes* are in their intersection.

However, neither of these two types is contained in the other, since there are subtypes (*resistor*s and *microprocessor*s) that are contained in one but not the other.

We are interested in developing inference systems for type containment statements. The examples above are assertions that we would like to be able to store in a knowledge base and derive inferences from. For example, we would like to be able to ask

Is *half_watt_resistor* a *heat_sensitive_device* ?

and have a system correctly infer that the answer is *NO*. In general we wish to be able to make queries of the forms

$$X \quad isa \quad Y \,?$$
$$\textbf{not}(\, X \quad isa \quad Y \,)\,?$$

We call this problem the *Binary Containment Inference Problem,* a special case of a general containment inference problem permitting inclusion statements (and their negations) involving more than two types. It is a limited fragment of set theory of practical use.

An initial purpose of this paper was to investigate how *negation* affects inference in specific knowledge domains. In [3] we discussed a restricted positive version of the binary containment inference problem. Generally, of course, permitting negation causes inference to become computationally intractable; combinatorial explosions arise as soon as a negation operator is introduced. We were pleased to discover that, for the binary containment problem, negation does not result in intractability. Thus binary containment assertions are at the same time expressive, yet not general enough a fragment of set theory to cause complexity problems. $O(n^3)$ time, where n is the number of types, is easily sufficient to resolve binary containment queries.

The emphasis of this paper is on developing effective *systems of inference rules* for the binary containment problem. We consider first degeneracy properties (inconsistency, unsatisfiability, triviality) then identify sets of rules for specific subproblems.

Syllogisms turn out to be essentially the rules we need here. There are 24 valid syllogisms. These rules have been used for millenia, and were actually held as synonymous with the word *logic* until the mid-nineteenth century after the work of George Boole. While syllogisms were discarded eventually as being 'less general' than boolean logic, they clearly fit here naturally.

A great deal of related work has appeared in different fields, from cognitive science [8] to database theory [1, 5, 9, 15] to knowledge representation [2, 4, 11, 12, 13, 14]. It is probable that all results in this paper have been discovered by other researchers at one time or another, in one form or another. After all, syllogisms and binary containment inference have been studied over centuries. However we are not aware of a reference covering the results here. Of the references just mentioned, the OMEGA system of Attardi and Simi [2] is the most similar in approach, and in fact produces some of the *isa* inference rules given here, but does not restrict itself to the *binary* containment problem or even to first-order logic. We were motivated by studying

existing knowledge representation systems, which uniformly lacked ability to perform binary containment inference.

We could focus exclusively on logic when studying containment inference, by expressing the problem in monadic predicate logic and then applying, say, resolution proof techniques. The approach of this paper is broader, developing several formal systems to handle containment inference problems. This approach has certain benefits. First, it clarifies the model theory of the binary set containment problem, the problem's relationship to syllogisms, and identifies degenerate cases of constraints precisely. Second, it shows how the twenty-four Aristotelian syllogisms can be compressed into a few rules. Third, it sets the foundation for fast inference algorithms. Finally, it permits generalization to formal systems handling more complex types of 'syllogisms'. For example, DeMorgan studied six different kinds of syllogisms [6], including 'numerical' syllogisms such as

100 Y's exist
70 X's are Y's
40 Z's are Y's

---

at least 10 X's are Z's.

The paper shows how syllogisms correspond to a subset of modern logic that is useful in knowledge representation. The connection between sets, logic, syllogisms, and human intelligence is fascinating, and deserves further investigation.

This paper is organized as follows. In Section 2, we review work on syllogisms and discuss notational conventions to be used in the paper. Section 3 presents terminology and inference rules used in set containment inference, and investigates degenerate containment schemes. Groundwork for subsequent results showing how models of non-degenerate containment schemes can be constructed is presented in Section 4. Sections 5 and 6 then show that these rules are sound and complete, and finally Section 7 relates these results to syllogisms.

2. Syllogisms

Aristotle apparently defined a syllogism to be any valid inference [6], but concentrated on inferences that can be made from four kinds of propositions:

Every *S* is *P*
No *S* is *P* (i.e., Every *S* is not *P*)
Some *S* is *P*
Some *S* is not *P*

A syllogism is composed of three propositions involving three *types* *S*, *M*, and *P*, representing respectively its 'Subject', 'Middle', and 'Predicate'. For example, the following is a syllogism:

Major premise: every P is M
Minor premise: some S is not M

Conclusion: some S is not P

Since the conclusion always involves the subject and predicate, while the premises use the middle type *M* in 4 nontrivial ways (called 'figures' by Aristotle, although he developed only the first 3 figures shown below), there are a total of 4×4×4×4 = 256 possible syllogisms, of which 24 are valid. These 24 are listed below, divided into the four figures:

S11:	every	*S*	is	*P*	if	every	*M*	is	*P*	and	every	*S*	is	*M*.	
S12:	some	*S*	is	*P*	if	every	*M*	is	*P*	and	every	*S*	is	*M*.	*
S13:	some	*S*	is	*P*	if	every	*M*	is	*P*	and	some	*S*	is	*M*.	
S14:	every	*S*	is not	*P*	if	every	*M*	is not	*P*	and	every	*S*	is	*M*.	
S15:	some	*S*	is not	*P*	if	every	*M*	is not	*P*	and	every	*S*	is	*M*.	*
S16:	some	*S*	is not	*P*	if	every	*M*	is not	*P*	and	some	*S*	is	*M*.	
S21:	every	*S*	is not	*P*	if	every	*P*	is not	*M*	and	every	*S*	is	*M*.	
S22:	some	*S*	is not	*P*	if	every	*P*	is not	*M*	and	every	*S*	is	*M*.	*
S23:	some	*S*	is not	*P*	if	every	*P*	is not	*M*	and	some	*S*	is	*M*.	
S24:	every	*S*	is not	*P*	if	every	*P*	is	*M*	and	every	*S*	is not	*M*.	
S25:	some	*S*	is not	*P*	if	every	*P*	is	*M*	and	every	*S*	is not	*M*.	*
S26:	some	*S*	is not	*P*	if	every	*P*	is	*M*	and	some	*S*	is not	*M*.	
S31:	some	*S*	is	*P*	if	every	*M*	is	*P*	and	every	*M*	is	*S*.	*
S32:	some	*S*	is	*P*	if	every	*M*	is	*P*	and	some	*M*	is	*S*.	
S33:	some	*S*	is	*P*	if	some	*M*	is	*P*	and	every	*M*	is	*S*.	
S34:	some	*S*	is not	*P*	if	every	*M*	is not	*P*	and	every	*M*	is	*S*.	*
S35:	some	*S*	is not	*P*	if	every	*M*	is not	*P*	and	some	*M*	is	*S*.	
S36:	some	*S*	is not	*P*	if	some	*M*	is not	*P*	and	every	*M*	is	*S*.	
S41:	every	*S*	is not	*P*	if	every	*P*	is	*M*	and	every	*M*	is not	*S*.	
S42:	some	*S*	is not	*P*	if	every	*P*	is	*M*	and	every	*M*	is not	*S*.	*
S43:	some	*S*	is not	*P*	if	every	*P*	is not	*M*	and	every	*M*	is	*S*.	*
S44:	some	*S*	is not	*P*	if	every	*P*	is not	*M*	and	some	*M*	is	*S*.	
S45:	some	*S*	is	*P*	if	every	*P*	is	*M*	and	every	*M*	is	*S*.	*
S46:	some	*S*	is	*P*	if	some	*P*	is	*M*	and	every	*M*	is	*S*.	

These 24 syllogisms are all valid under the assumption that the sets denoted by the types *S*, *M*, and *P* are nonempty. If this assumption does not hold, the 9 entries above marked with stars (*) are invalid. In other words, although we would normally assume

that whenever

every X is Y

then also

some X is Y,

this inference is invalid when the set denoted by the type X is empty.

The structure of the syllogisms has fascinated philosophers and mathematicians for millenia. At this point the reader may be asking questions such as:

- Is the set of 24 rules (or 15, eliminating starred ones) complete?
- Is there a more compact presentation of these rules?
- Can the rules be used in an efficient inference system?

We answer these questions later in the paper.

Some readers will have noticed that syllogisms involve containment propositions. Specifically, if X and Y are types denoting sets, and **non**(Y) is a type denoting the set complement of Y with respect to some (unspecified) universe:

$$\begin{aligned} \text{every } X \text{ is } Y &\equiv X \subseteq Y \\ \text{some } X \text{ is } Y &\equiv X \cap Y \neq \emptyset \\ \textbf{not} \text{ every } X \text{ is } Y &\equiv X \cap \textbf{non}(Y) \neq \emptyset \\ \textbf{not} \text{ some } X \text{ is } Y &\equiv X \subseteq \textbf{non}(Y) \end{aligned}$$

This collection of syllogistic propositions is thus equivalent to the binary propositions one can make up with the standard set predicates $\subseteq$ and $\cap\neq\emptyset$. In an effort to follow [3], as well as simplify notation ($\cap\neq\emptyset$ is tedious to write), we define the following two predicates and use them for the remainder of the paper:

Definition

(X *isa* Y) if the set denoted by X is a subset of the set denoted by Y.

Definition

(X *int* Y) if the set denoted by X intersects the set denoted by Y. The intersection must be *nonempty*.

Remark

$$X \textit{ int } Y \equiv \textbf{not}(\, X \textit{ isa } \textbf{non}(Y)\,).$$

3. Terminology

3.1 The Set Containment Problem

Our goal in this section is to set up a framework for expressing containment problems. We differentiate between the *scheme* of a containment problem, and its *interpretations* (and *models*). The scheme specifies the structure of the problem, while interpretations of the scheme give specific instances of objects in the types in the scheme.

Definition

A *type scheme* T/U is a collection of *type symbols* $\{U, T_1, \ldots, T_n\}$. The type symbol U is a special symbol and is called the *universe* of the type scheme.

Each type symbol T_i will denote a subset of U. The universe symbol U is needed in order to define what we mean by complements $\mathbf{non}(T_i)$ of types:

Definition

A *type term* X of a type scheme T/U is either

1. a type symbol T_i or U.
2. $\mathbf{non}(Y)$, where Y is a type term of T/U.

Definition

An *assignment* I of a type scheme T/U is a map associating to each type term X a possibly empty subset of a finite *domain* D, subject to the following restrictions:

1. $I(U) = D$
2. $I(X) \subseteq D$
3. $I(\mathbf{non}(\mathbf{non}(X))) = I(X)$
4. $I(\mathbf{non}(X)) \cap I(X) = \varnothing$.

An *interpretation* I is an assignment that satisfies the following additional restriction, where '–' represents set difference:

5. $I(X) = I(U) - I(\mathbf{non}(X))$.

In other words, with an interpretation the type term $\mathbf{non}(X)$ denotes the *complement* under U of the set denoted by X.

From condition 3 in the definition of assignments, it follows that for every type term X,

$$I(X) = I(\mathbf{non}(\mathbf{non}(X))) = I(\mathbf{non}(\mathbf{non}(\mathbf{non}(\mathbf{non}(X))))) = \cdots$$

Therefore, it is possible to consider equivalence classes of type terms under this identity.

Definition

A *type descriptor* of the type scheme $T/U = \{U, T_1, \ldots, T_n\}$ is an equivalence class

$$\{X, \mathbf{non}(\mathbf{non}(X))), \mathbf{non}(\mathbf{non}(\mathbf{non}(\mathbf{non}(X)))), \cdots\}$$

where X is a type term of the form S or $\mathbf{non}(S)$, and S is a type symbol in T/U. It is designated by any element of the class, but usually by X.

The type scheme T/U therefore has the type descriptors U, $\mathbf{non}(U)$, T_1, $\mathbf{non}(T_1)$, . . . , T_n, $\mathbf{non}(T_n)$.

Definition

A type descriptor X is *trivial* in interpretation I if $I(X) = \varnothing$.
The *trivial interpretation* I assigns $I(X) = \varnothing$ for every type descriptor X. That is, an interpretation is trivial if and only if its associated domain is empty.

Definition

A *positive constraint*, or *isa constraint*, P has the form

$$P: \; X_1 \cap \cdots \cap X_p \quad isa \quad Y_1 \cup \cdots \cup Y_q$$

where each X_j, $1 \le j \le p$, and each Y_k, $1 \le k \le q$, is a type descriptor.

The constraint is *satisfied* by the interpretation I if

$$I(X_1) \cap \cdots \cap I(X_p) \subseteq I(Y_1) \cup \cdots \cup I(Y_q).$$

Definition

A *negative constraint*, or *intersection constraint*, has the form $\mathbf{not}(P)$, where P is a positive constraint. It is satisfied iff P is not.

Note that the positive constraint P above is satisfied by the interpretation I iff

$$I(\mathbf{non}(X_1)) \cup \cdots \cup I(\mathbf{non}(X_p)) \cup I(Y_1) \cup \cdots \cup I(Y_q) = I(U)$$

or, equivalently,

$$I(X_1) \cap \cdots \cap I(X_p) \cap I(\mathbf{non}(Y_1)) \cap \cdots \cap I(\mathbf{non}(Y_q)) = \varnothing$$

so the negative constraint $\mathbf{not}(P)$ is equivalent to

$$I(X_1) \cap \cdots \cap I(X_p) \cap I(\mathbf{non}(Y_1)) \cap \cdots \cap I(\mathbf{non}(Y_q)) \neq \varnothing$$

In other words, positive constraints make assertions about *inclusions* among types, while negative constraints make assertions about *intersections* among types.

Definition

A *containment constraint* is a positive constraint or negative constraint.

Definition

A *containment scheme* is a pair $S = (T/U,C)$ where U is a universe symbol, T/U is a type scheme $\{U, T_1, \ldots, T_n\}$, and C is a set of containment constraints on type descriptors in T/U.

Definition

A *model* of a containment scheme $(T/U,C)$ is an interpretation I of T/U that satisfies all constraints in C.

Containment constraints can be 'degenerate'. Consider the three constraints below:

1. X *isa* **non**(X).
2. **non**(X) *isa* X.
3. **not**(X *isa* X).

The first constraint is satisfied only when X denotes $\varnothing$. Similarly, the second is satisfied only when **non**(X) denotes $\varnothing$, so X denotes the same set as U. The first two constraints together can be satisfied only when both X and U denote $\varnothing$. In other words, the first two constraints together imply that there can be only one model: the trivial one.

The third constraint is the negation of (X *isa* X), which is true of every interpretation for X. Thus, any scheme with the third constraint can satisfy no interpretation, and will have no model.

Definition

A containment scheme is *unsatisfiable* if it has no model; otherwise it is *satisfiable.*

Now consider the following general problem:

Set Containment Problem

Input: a containment scheme $(T/U,C)$, where C is a collection $\{C_j | j = 1, \ldots, m\}$ of containment constraints on the type descriptors of T/U.

Question: Is the containment scheme satisfiable? That is, is there an interpretation for T/U that satisfies each constraint C_j in C?

Not surprisingly, the set containment problem is NP-complete in general. This is shown in Appendix I. However, in the special case where all constraints are positive, the set containment problem is always satisfiable. Specifically, the trivial interpretation (in which $I(U) = \varnothing$) is always a model satisfying C.

In this paper we are interested only in the special case $p = q = 1$, where all constraints are binary. This restricted version is called the *Binary Set Containment Problem.* In this case negative constraints specify binary, non-empty intersections. Therefore, as we noted in Section 2, we can express them as intersection constraints: **not**(X *isa* Y)

becomes (X *int* **non**(Y)). In the remainder of the paper we first consider the special case where all binary constraints are positive, then study the general binary containment problem.

Example

Consider the containment scheme

$$S = (\{pacifist, quaker, republican\}/U, C)$$

where the constraint set C is:

republican	*isa*	**non**(*pacifist*)
republican	*int*	*quaker*
quaker	*isa*	*pacifist.*

The binary set containment problem for this scheme is to determine whether it is satisfiable or not. It turns out this scheme is *unsatisfiable.*

3.2 Containment Inference

We recall some terminology concerning inference rules. The reader unfamiliar with this material may wish to consult texts in database theory, such as [10] and [16]. We remind the reader that classes of constraints studied in database theory do not involve negation, for the most part. That is, collections of data dependencies (functional dependencies, inclusion dependencies, etc.) do not imply that a specific data dependency *does not* hold, but only that one *does* hold. These systems are always satisfiable. Unsatisfiability can occur with containment, as we have already seen. Therefore, the inference problem here is somewhat different than in these texts.

Implication and inference are important concepts in dealing with constraints of the general kind proposed here. If we are given a set of constraints, we are frequently interested in deducing whether other constraints must also hold.

A constraint c is *implied* by a set of constraints C on a scheme S if it holds in all models of S. Given C and c, the *inference problem* is to tell whether C implies c. Algorithms for the solution of the inference problem (called *inference algorithms*) have correctness proofs that are usually based on sound and complete sets of inference rules.

Definition

$C \models c$ if C implies c (that is, c must hold in every model of C).

If C is unsatisfiable, then C has no models, and the definition of $\models$ becomes vacuous. Thus if C is unsatisfiable, then $C \models c$ for every constraint c.

Set Containment Inference Problem

Input: a containment scheme $(T/U,C)$, and a constraint c.

Question: Is it true that $C \models c$?

This problem can be reduced to the complement of the Set Containment Problem mentioned earlier simply by determining unsatisfiability of the scheme with constraints $C \cup \{\textbf{not}\ (c)\}$.

Theorem 1
$C \models c$ **iff** $C \cup \{\textbf{not}\ (c)\}$ is unsatisfiable.

Proof

If $C \models c$ then every model of C satisfies c, and so does not satisfy **not**(c). Hence $C \cup \{\textbf{not}\ (c)\}$ has no model.

Conversely, suppose $C \cup \{\textbf{not}\ (c)\}$ is unsatisfiable. If C is unsatisfiable, then $C \models c$ holds trivially. If C is satisfiable, then none of its models will satisfy **not**(c), implying that all models satisfy c. Again, $C \models c$. □

3.3 Triviality in Containment Schemes

We can show that a type X is trivial in every model of a scheme if, and only if, the scheme implies X *isa* **non**(X). Moreover, a satisfiable scheme has only the trivial model if, and only if, for some type descriptor X we can infer both X *isa* **non**(X) and **non**(X) *isa* X.

Theorem 2.
A type descriptor X is trivial in every model of C **iff** $C \models (X\ isa\ \textbf{non}(X))$.

Proof

If $C \models (X\ isa\ \textbf{non}(X))$, then $(X\ isa\ \textbf{non}(X))$ is satisfied in every model of C. Thus X is trivial in every model. Conversely, if X is trivial in every model of C then $(X\ isa\ \textbf{non}(X))$ is satisfied in every model, and so $C \models (X\ isa\ \textbf{non}(X))$. □

A corollary of Theorem 2 is that a pair of constraints

$$X\ isa\ \textbf{non}(X)$$
$$\textbf{non}(X)\ isa\ X$$

must be implied by any scheme with only the trivial model. (U, and consequently both X and **non**(X), must denote $\varnothing$.)

3.4 Unsatisfiable Containment Schemes

We showed earlier that containment constraints can actually be unsatisfiable, giving as an example **not**(X *isa* X). In fact, this constraint is not only sufficient for

unsatisfiability, it is also necessary:

Theorem 3.
A containment scheme $(T/U,C)$ is unsatisfiable **iff** for some X, $C \models (X\ int\ \mathbf{non}(X))$.

Proof

The constraint $(X\ int\ \mathbf{non}(X))$ has no model for any X. Therefore if the scheme is satisfiable, we cannot have $C \models (X\ int\ \mathbf{non}(X))$ for any X. Conversely, if the scheme is unsatisfiable then the definition of $\models$ becomes vacuous and C implies every constraint, so $C \models (X\ int\ \mathbf{non}(X))$ for some X. □

4. Set Containment Inference

4.1 Inference Rules

Definition

An *inference rule* R, written $R.\ A \vdash a$, is a rule asserting that the constraint a holds whenever the set of constraints A holds.

We define the following inference rules, where X, Y, and Z represent arbitrary type descriptors:

INT0.	$X\ int\ Y \vdash X\ int\ U.$
INT1.	$X\ int\ Y \vdash X\ int\ X.$
INT2.	$X\ int\ Y \vdash Y\ int\ X.$
INT3.	$X\ int\ Y, Y\ isa\ Z \vdash X\ int\ Z.$
INC0.	$X\ int\ \mathbf{non}(X) \vdash Y\ isa\ Z.$
INC1.	$X\ int\ \mathbf{non}(X) \vdash Y\ int\ Z.$
ISA0.	$\vdash X\ isa\ U.$
ISA1.	$\vdash X\ isa\ X.$
ISA2.	$X\ isa\ Y, Y\ isa\ Z \vdash X\ isa\ Z.$
ISA3.	$X\ isa\ Y \vdash \mathbf{non}(Y)\ isa\ \mathbf{non}(X).$
TRIV0.	$X\ isa\ \mathbf{non}(X) \vdash X\ isa\ Y.$

Definition

A *substitution* θ is a map from type descriptors to type descriptors. If a is a constraint, $a\theta$ is the constraint with type descriptors mapped to their corresponding images under θ. Analogously $A\theta$ is the set of images obtained with θ of the constraints in A.

Definition

A constraint c can be *derived in one step* from a set of constraints C if there is a rule $R.\ A \vdash a$, a subset C' of C, and a substitution θ such that $C' = A\theta$ and $c = a\theta$.

Definition

A *derivation* of c from C is a sequence of constraints $c_1, \ldots, c_n$ such that $c = c_n$, and for each i between 1 and n, c_i can be derived from $C \cup \{\ c_j \mid 1 \le j \le i\ \}$ in one step. The inference rules listed above are used in derivations in this paper, except where explicitly stated otherwise.

Definition

$C \vdash c$ if c is in C, or there is a derivation of c from C.

Theorem 4.
The rules above are *sound.* That is, if $C \vdash c$, then also $C \models c$.

Proof

Omitted. □

It is important to have inference rules that are also *complete*, i.e., that allow the derivation of *all* the constraints c such that $C \models c$. Thus, a set of rules is sound and complete when $\vdash$ is equivalent to $\models$. We show later that the rules above are complete.

Definition

A scheme $(T/U,C)$ is *inconsistent* if there is a constraint c such that

$$C \vdash c \quad \text{and} \quad C \vdash \textbf{not}(c).$$

Otherwise the scheme is *consistent.*

Example

The constraint set C considered earlier

republican	*isa*	**non**(*pacifist*)
republican	*int*	*quaker*
quaker	*isa*	*pacifist*

can be shown to be inconsistent with the appropriate inference rules. The inference rule INT3 gives the derivation

republican int quaker, quaker isa pacifist $\vdash$ *republican int pacifist.*

However

republican int pacifist

is equivalent to

not(*republican isa* **non**(*pacifist*)),

contradicting the first constraint in C.

Clearly, an inconsistent scheme is also unsatisfiable. We will see later that the converse also holds.

4.2 Assignment Extension Algorithm

We give first an algorithm useful in proofs later. It takes as input a containment scheme $(T/U,C)$, an assignment I, and a constant t (which I may or may not include in its image, and so may not be in D), and produces an assignment I' which uses t.

At various points the algorithm tests whether $C \vdash c$, given some C and c. The Assignment Extension Algorithm therefore requires an algorithm for deciding implication using the rules. We will show later that efficient algorithms in fact exist,

but for now we require only *some* algorithm. Since the number of constraints derivable from a given set of constraints C is finite, brute force repeated application of rules will suffice here.

Assignment Extension Algorithm

Input:
- a containment scheme $(T/U,C)$, such that $C \nvdash (U \textit{ isa } \mathbf{non}(U))$,
- an assignment I from type descriptors of T/U to subsets of D,
- a symbol t which may or may not be in D.

Output:
- an assignment I' from type descriptors of T/U to subsets of $D \cup \{t\}$.

The algorithm constructs I' from I as follows:

1. *Preserve any previous use of t.*
   For all X with t already in $I(X)$, set
   $$I'(X) = I(X) \cup \{t\} = I(X)$$
   $$I'(\mathbf{non}(X)) = I(\mathbf{non}(X))$$
   Note t cannot be in both $I(X)$ and $I(\mathbf{non}(X))$, for then I would not be an assignment.

2. *Make assignments for trivial types and their complements.*
   For all X (including $\mathbf{non}(U)$) with $I'(X)$ currently undefined and $C \vdash (X \textit{ isa } \mathbf{non}(U))$, set
   $$I'(X) = I(X)$$
   $$I'(\mathbf{non}(X)) = I(\mathbf{non}(X)) \cup \{t\}$$
   We cannot find X at this point such that both X and $\mathbf{non}(X)$ are trivial, since these would imply $C \vdash (U \textit{ isa } \mathbf{non}(U))$.

3. *Propagate t through isa constraints.*
   For all X with t in $I'(X)$ at this point, for all Z with $I'(Z)$ undefined such that $C \vdash X \textit{ isa } Z$, set
   $$I'(Z) = I(Z) \cup \{t\}$$
   $$I'(\mathbf{non}(Z)) = I(\mathbf{non}(Z))$$

4. *Add t to some type permitting the addition.*
   If there is no X such that $I'(X)$ is undefined, then halt: I' is the completed assignment.

   Otherwise select a 'minimal' X such that $I'(X)$ is undefined. We say X is *minimal* if there is no nontrivial Y such that both $C \vdash (Y \textit{ isa } X)$ and $C \nvdash (X \textit{ isa } Y)$. Such an X must exist, since otherwise Step 3 would have already yielded a definition for $I'(X)$. Then set

$$I'(X) = I(X) \cup \{t\}$$
$$I'(\mathbf{non}(X)) = I(\mathbf{non}(X)),$$

and go to Step 3. Step 3 will then propagate definition of I' for supertypes of X.

Clearly this procedure always terminates. Moreover, when it terminates I' is a valid assignment, defined for all type descriptors X of T/U. In particular for every X, the algorithm assigns t to either $I'(X)$ or $I'(\mathbf{non}(X))$, since each step assigns t to either one or the other, and $I'(X)$ is defined exactly once.

The algorithm is not deterministic for some inputs, however. If for example we have a set of constraints including (x *isa* z) and (y *isa* **non**(z)), and begin the algorithm with a symbol t and an assignment I such that $t \in I(x)$ and $t \in I(y)$, then Step 3 of the algorithm can assign t to either $I'(z)$ or $I'(\mathbf{non}(z))$, depending on order of propagation.

The problem in this example is that $C \vdash (x \textit{ isa } \mathbf{non}(y))$, or equivalently $C \vdash \mathbf{not}(x \textit{ int } y)$, but $t \in (I(x) \cap I(y))$. We can avoid this problem by restricting ourselves to assignments that respect intersections among types:

Definition

Given a containment scheme (T/U,C), an assignment I *respects intersections of* C if (V *int* W) $\in C$ implies $I(V) \cap I(W) \neq \varnothing$, and $C \vdash (V \textit{ isa } W)$ implies $I(V) \cap I(\mathbf{non}(W)) = \varnothing$.

Lemma 1.
Let $C \nvdash (U \textit{ isa } \mathbf{non}(U))$, I be an assignment that respects intersections of C, and I' be the result of applying the Assignment Extension Algorithm to I and symbol t. Then I' respects intersections of C, and if $C \vdash (V \textit{ isa } W)$, then $I'(V) \cap \{t\} \subseteq I'(W) \cap \{t\}$.

Proof

For every type descriptor X, the algorithm guarantees that $I'(X) \supseteq I(X)$. Thus if (V *int* W) is in C then $I(V) \cap I(W) \neq \varnothing$, so $I'(V) \cap I'(W) \neq \varnothing$.

Also if $C \vdash (V \textit{ isa } W)$, then $I(V) \cap I(\mathbf{non}(W)) = \varnothing$, so either $t \notin I(V)$ or $t \notin I(\mathbf{non}(W))$. We obtain three cases:

1. $t \in I(V)$, $t \notin I(\mathbf{non}(W))$
   Here $I'(V)$ is defined to include t in Step 1, and there are three possibilities for W: First, if $t \in I(W)$, then $I'(W)$ is defined to include t in Step 1. Second, if $\mathbf{non}(W)$ is trivial, then Step 2 assigns t to $I'(W)$. Otherwise, $I'(W)$ is undefined when the algorithm reaches Step 3 for the first time, and t is propagated into $I'(W)$ since $C \vdash (V \textit{ isa } W)$. Thus in each possibility $I'(W)$ includes t as well, and we are done.

2. $t \notin I(V)$, $t \in I(\mathbf{non}(W))$
   Here Step 1 assigns t to $I'(\mathbf{non}(W))$, and (similar to the previous case) since $C \vdash (\mathbf{non}(W) \textit{ isa } \mathbf{non}(V))$ the algorithm also assigns t to $I'(\mathbf{non}(V))$.

3. $t \notin I(V)$, $t \notin I(\mathbf{non}(W))$
   Also we can assume $t \notin I(\mathbf{non}(V))$, $t \notin I(W)$, that V is nontrivial, and that $t \in I'(V)$ at completion of the algorithm, since otherwise the lemma is vacuously satisfied in Step 1. In this case the algorithm completes Steps 1 and 2 with both $I'(V)$ and $I'(W)$ undefined. Now either Step 3 or Step 4 assign t to $I'(W)$, or somehow they assign t to $I'(\mathbf{non}(W))$. This second possibility cannot arise without violating either that $t \in I'(V)$ at completion of the algorithm, or that V is nontrivial.

In each case we have shown that $I'(V) \cap I'(\mathbf{non}(W)) = \varnothing$, so I' respects intersections of C, and also that $I'(V) \cap \{t\} \subseteq I'(W) \cap \{t\}$. □

Theorem 5.
Let $(T/U,C)$ be a containment scheme such that $C \nvdash (U \textit{ isa } \mathbf{non}(U))$, and let I_0 be an assignment with domain D that respects intersections of C. Then the result I^* of accumulative application of the algorithm to I_0 and the elements of D is a model of C.

Proof

1. I^* is an interpretation. I_0 is an assignment, and we know that when the algorithm is applied to an assignment I and an element t, producing the assignment I', then t belongs to $I'(X) \cup I'(\mathbf{non}(X))$, for every type descriptor X. Since the algorithm is applied to all elements of D, an induction argument shows that $t \in I^*(X) \cup I^*(\mathbf{non}(X))$, for every t in D and every type descriptor X.

2. I^* satisfies all intersections in C: If $(V \textit{ int } W) \in C$ since by hypothesis $I_0(V) \cap I_0(W) \neq \varnothing$, it follows by induction that $I^*(V) \cap I^*(W) \neq \varnothing$.

3. I^* satisfies the *isa* constraints in C: this follows from Lemma 1, again by induction on the cardinality of D, since I_0 respects intersections of C, and every assignment produced by the algorithm also respects intersections of C. Thus if $(V \textit{ isa } W) \in C$ then $I^*(V) \cap I^*(\mathbf{non}(W)) = \varnothing$ and $I^*(V) \subseteq I^*(W)$.

□

Example

We show how the Assignment Extension Algorithm can be used accumulatively as proposed in Theorem 5 to produce a sequence of assignments culminating in a model.

Consider the type scheme $S = (\{v,w,x,y\}/U,C)$, where C is the set of constraints

$$\begin{array}{lll} v & \textit{isa} & x \\ v & \textit{isa} & \mathbf{non}(y) \\ x & \textit{int} & y \\ w & \textit{int} & \mathbf{non}(x) \end{array}$$

Noting that this scheme is not inconsistent, define the assignment I_0 as follows: for each type descriptor X in $\{\mathbf{non}(U),v,\mathbf{non}(v),w,\mathbf{non}(w),x,\mathbf{non}(x),y,\mathbf{non}(y)\}$ set

$$I_0(X) \;=\; \{\ \{X,Y\} \ \mid\ C \vdash (X\ \textit{int}\ Y)\ \},$$

and put

$$I_0(U) \;=\; \{\ \{X,Y\} \ \mid\ C \vdash (X\ \textit{int}\ Y)\ \} \;=\; \{\ \{x,y\},\ \{w,\mathbf{non}(x)\}\ \}.$$

This set of two elements is the domain D. The model constructed for S evolves as follows, under two applications of the algorithm:

v	non(v)	w	non(w)	x	non(x)	y	non(y)	U	non(U)
∅	∅	{{w,non(x)}}	∅	{{x,y}}	{{w,non(x)}}	{{x,y}}	∅	{{x,y},{w,non(x)}}	∅
∅	{{x,y},{w,non(x)}}	{{w,non(x)}}	∅	{{x,y}}	{{w,non(x)}}	{{x,y}}	∅	{{x,y},{w,non(x)}}	∅
∅	{{x,y},{w,non(x)}}	{{w,non(x)}}	{{x,y}}	{{x,y}}	{{w,non(x)}}	{{x,y}}	∅	{{x,y},{w,non(x)}}	∅
∅	{{x,y},{w,non(x)}}	{{w,non(x)}}	{{x,y}}	{{x,y}}	{{w,non(x)}}	{{x,y},{w,non(x)}}	∅	{{x,y},{w,non(x)}}	∅

The first sequence is I_0, an assignment satisfying all intersection constraints implied by C.

The second sequence shows the assignment created at Step 3 of the Assignment Extension Algorithm with $t = \{x,y\}$. We have used the facts that $C \vdash (\mathbf{non}(x)\ \textit{isa}\ \mathbf{non}(v))$ and $C \vdash (y\ \textit{isa}\ \mathbf{non}(v))$.

The third sequence shows the final extension of I_0 created by the Assignment Extension Algorithm for $t = \{x,y\}$. (Note this is only one possible extension; the type descriptor $\mathbf{non}(w)$ was selected in Step 4 of the algorithm.)

The fourth sequence shows the subsequent assignment obtained from the Assignment Extension Algorithm for $t = \{w,\mathbf{non}(x)\}$. (The type descriptor y was selected in Step 4 of the algorithm.) This sequence can be verified to be a model I^* of S.

Theorem 6.
A containment scheme $(T/U,C)$ is unsatisfiable iff it is inconsistent.

Proof

The if part follows from soundness of the rules.

With respect to the only if part, we show that if a scheme is consistent, then it is satisfiable. Let $(T/U,C)$ be consistent. If C contains no intersection constraints, then the trivial interpretation is a model for the scheme, and it is therefore satisfiable. Otherwise, we claim that $(U \; isa \; \mathbf{non}(U))$ is not derivable from C: if it were derivable, then for any $(X \; int \; Y)$ in C we derive:

1. $(U \; isa \; \mathbf{non}(U))$
2. $(X \; int \; Y)$
3. $(X \; int \; X)$ (from 2, by INT1)
4. $(X \; isa \; U)$ (by ISA0)
5. $(X \; isa \; \mathbf{non}(U))$ (from 4, 1, by ISA2)
6. $(X \; int \; \mathbf{non}(U))$ (from 3, 5, by INT3)
7. $(\mathbf{non}(U) \; int \; X)$ (from 6, by INT2)
8. $(\mathbf{non}(U) \; int \; U)$ (from 7, 4, by INT3).

This contradicts our supposed consistency. Therefore the Assignment Extension Algorithm can be applied to the scheme. Consider the following assignment:

$$I_0(X) \;=\; \{\, \{X,Y\} \;\mid\; C \vdash (X \; int \; Y) \,\}.$$

The assignment respects the intersection constraints of C: If $(V \; int \; W) \in C$, then explicitly $I_0(V) \cap I_0(W) \neq \emptyset$. Also, if $C \vdash (V \; isa \; W)$ then necessarily $I_0(V) \cap I_0(\mathbf{non}(W)) = \emptyset$; otherwise we would also have $(V \; int \; \mathbf{non}(W)) \in C$, and from these two constraints we can derive via ISA3, INT3 that $C \vdash (V \; int \; \mathbf{non}(V))$, so C is inconsistent.

Therefore, by Theorem 5, accumulative application of the algorithm to I_0 produces a model. Hence the scheme is satisfiable. □

5. Positive Binary Containment

Let us now devote our attention to the containment problem. Consider first the important special case of the set containment inference problem where all constraints are of the form

$$X \ isa \ Y$$

with X and Y type descriptors denoting subsets of U.

In [3], a similar problem was studied and solved. A complete set of inference rules is presented for two containment predicates, *isa* and *dis*. The proposition (X *isa* Y) states that the type X denotes a subset of the type Y, while the proposition (X *dis* Y) states that X and Y denote disjoint sets. In [3] it is shown that, for arbitrary types X, Y, Z, the following rules for *isa* and *dis* are sound and complete:

I1. $\vdash X \ isa \ X.$
I2. $X \ isa \ Y, Y \ isa \ Z \vdash X \ isa \ Z.$
M1. $X \ dis \ Y, Z \ isa \ X \vdash Z \ dis \ Y.$
M2. $X \ dis \ X \vdash X \ isa \ Y.$
D1. $X \ dis \ X \vdash X \ dis \ Y.$

We can use set complementation to simplify these rules. Since, for example,

$$X \ dis \ Y \quad \equiv \quad X \ isa \ \mathbf{non}(Y)$$

we can replace the five rules above with

I1. $\vdash X \ isa \ X.$
I2. $X \ isa \ Y, Y \ isa \ Z \vdash X \ isa \ Z.$
D1. $X \ isa \ \mathbf{non}(X) \vdash X \ isa \ Y.$

provided we let X, Y, Z be arbitrary *type descriptors*. In the statements X *isa* Y, X *dis* Y of [3], X is required to be a type symbol (not a type descriptor). Without this requirement the 5 rules above are incomplete. For example, the rule

$$X \ isa \ Y \ \vdash \ \mathbf{non}(Y) \ isa \ \mathbf{non}(X)$$

is not inferrable from the five rules, but is sound.

Consider, then, the following set of rules:

ISA0. $\vdash X \ isa \ U.$
ISA1. $\vdash X \ isa \ X.$
ISA2. $X \ isa \ Y, Y \ isa \ Z \vdash X \ isa \ Z.$
ISA3. $X \ isa \ Y \vdash \mathbf{non}(Y) \ isa \ \mathbf{non}(X).$
TRIV0. $X \ isa \ \mathbf{non}(X) \vdash X \ isa \ Y.$

Theorem 7.
The set of rules ISA0-3,TRIV0 is sound and complete for positive binary containment inference. That is, if C contains only positive constraints and $\vdash$ represents derivability using these rules, then

$$C \vdash (X \; isa \; Y) \;\; \textbf{iff} \; C \models (X \; isa \; Y).$$

Proof

With respect to completeness, for each containment scheme $S = (T/U,C)$, we show that *isa* constraints not derivable from C by the rules cannot be implied by C. Suppose that $c = (X \; isa \; Y)$ is a constraint implied by C, but not derivable from the rules. We construct a counterexample model I satisfying C but violating c.

First note that c cannot be of the forms $(X \; isa \; U)$ or $(X \; isa \; X)$, since ISA0 and ISA1 preclude these. Also since $(X \; isa \; Y)$ is not derivable from the rules, then $(X \; isa \; \textbf{non}(U))$ and $(X \; isa \; \textbf{non}(X))$ are not either, for otherwise $(X \; isa \; Y)$ would be derivable with TRIV0. Finally $C \nvdash (U \; isa \; \textbf{non}(U))$, since TRIV0 applications would yield $(X \; isa \; Y)$ otherwise.

We construct an assignment I_0, with domain $\{t\}$, as follows:

$$I_0(U) = \{t\}$$
$$I_0(\textbf{non}(U)) = \varnothing$$
$$I_0(X) = \{t\}$$
$$I_0(\textbf{non}(X)) = \varnothing$$
$$I_0(Y) = \varnothing$$
$$I_0(\textbf{non}(Y)) = \{t\}$$
$$I_0(Z) = \varnothing \quad \text{for every } Z \text{ different from } X,Y,U$$
$$I_0(\textbf{non}(Z)) = \varnothing \quad \text{for every } Z \text{ different from } X,Y,U.$$

I_0 is an assignment, since it satisfies the definition given in Section 3. Since $C \nvdash (U \; isa \; \textbf{non}(U))$, Theorem 5 then shows how to construct a model I^* of C, since C contains no intersection constraints, and I_0 respects intersections of C. Also, I^* will not satisfy $(X \; isa \; Y)$, since the algorithm used to construct it preserves any previous use of t, and so $I^*(X) = I^*(\textbf{non}(Y)) = \{t\}$ and $I^*(Y) = \varnothing$. □

6. General Binary Containment

Recall the set containment rules introduced earlier:

INT0.	X *int* $Y \vdash X$ *int* U.
INT1.	X *int* $Y \vdash X$ *int* X.
INT2.	X *int* $Y \vdash Y$ *int* X.
INT3.	X *int* Y, Y *isa* $Z \vdash X$ *int* Z.
INC0.	X *int* **non**(X) $\vdash Y$ *isa* Z.
INC1.	X *int* **non**(X) $\vdash Y$ *int* Z.
ISA0.	$\vdash X$ *isa* U.
ISA1.	$\vdash X$ *isa* X.
ISA2.	X *isa* Y, Y *isa* $Z \vdash X$ *isa* Z.
ISA3.	X *isa* $Y \vdash$ **non**(Y) *isa* **non**(X).
TRIV0.	X *isa* **non**(X) $\vdash X$ *isa* Y.

Theorem 8.
The rules INT0-3,INC0-1,ISA0-3,TRIV0 are sound and complete for general binary containment inference. That is, if C is set of binary constraints, c is a binary constraint, and $\vdash$ represents derivability using these rules, then

$$C \vdash c \quad \textbf{iff} \quad C \models c.$$

Proof

Because the rules are sound, if $C \vdash c$ then $C \models c$. We must show that if $C \nvdash c$ then $C \not\models c$.

Suppose that $C \nvdash c$. Then C is consistent, since otherwise the inconsistency rules INC1,INC2 would derive c. Now consider $C' = C \cup \{\textbf{not}(c)\}$. By Lemma 2 (in Appendix II), since C is consistent and $C \nvdash c$, then C' is also consistent. Therefore by Theorem 6, C' is satisfiable; but any model of C' satisfies C and does not satisfy c, and so $C \not\models c$. □

The rule INT0 is redundant. This is shown by the derivation

1. (X *int* Y)
2. (Y *isa* U) (by ISA0)
3. (X *int* U) (from 1, 2, by INT3).

However, we keep INT0 here for the following reason:

Corollary 1.
The rules INT0-2,INC0-1 are sound and complete for binary containment inference with only intersection constraints.

7. Syllogisms

We show in this section that the 24 syllogisms listed earlier are essentially the rules we need for general set containment inference. However, although these 24 rules are elegant, they are also somewhat verbose. We show that we can reduce the syllogism rules to a set of 5 simple rules (involving really only two basic syllogisms, S11 and S13). We then show that these 5 rules correspond directly to the sound and complete inference rules discussed before.

Let X, Y, Z represent arbitrary type descriptors. Consider the following rules:

R1: every X is Z **if** every Y is Z **and** every X is Y.
R2: some X is Z **if** every Y is Z **and** some X is Y.
R3: some X is X.
R4: some X is Y **if** some Y is X.
R5: every X is **non**(Y) **if** every Y is **non**(X).

R1 and R2 are syllogisms mentioned earlier. Rule R3 is equivalent to the assumption that types are nonempty; this is actually necessary only where the existential quantifier "some" implies actual existence of some object, as it often does in natural language. (Recall that nine of the twenty-four syllogisms require types to be nonempty.) R4 and R5 state that both intersection and disjointness of types are symmetric relations.

Theorem 9.
All valid syllogisms follow from the rules R1-R5.

Proof

A simple case analysis shows this, and is instructive about the structure of the 24 syllogisms. Let S*ij*(*M*/**non**(*M*)) denote the *ij*-th syllogism with type descriptor *M* replaced by **non**(*M*), etc. We simply list the rules and the 'variable substitutions' needed to derive each syllogism.

S11: R1	S21: S11(*P*/**non**(*P*)) & R5	S31: S13 & R3 & R4	S41: S11(**non**(*S*)/*P*,*P*/*S*) & R5
S12: R1 & R3	S22: S21 & R3	S32: S13 & R4	S42: S41 & R3
S13: R2	S23: S13(*P*/**non**(*P*)) & R5	S33: S32(*P*/*S*,*S*/*P*)	S43: R5 & S34
S14: S11(*P*/**non**(*P*))	S24: S21(*M*/**non**(*M*))	S34: S31(*P*/**non**(*P*))	S44: R5 & S35
S15: S12(*P*/**non**(*P*))	S25: S22(*M*/**non**(*M*))	S35: S32(*P*/**non**(*P*))	S45: S12(*S*/*P*,*P*/*S*) & R3
S16: S13(*P*/**non**(*P*))	S26: S23(*M*/**non**(*M*))	S36: S33(*P*/**non**(*P*))	S46: S13(*S*/*P*,*P*/*S*)

□

Consider the following non-triviality assumptions:

INT1. $X \neq \mathbf{non}(U) \vdash X \; int \; X$.
ISA0. $\vdash X \; isa \; U$.
ISA1. $\vdash X \; isa \; X$.

These assumptions subsume the compressed syllogism rule R3.

Corollary 2.
Under the non-triviality assumptions above, the compressed syllogism rules R4,R2,R1,R5 are equivalent to the following rules:

INT2. $X\ int\ Y \vdash Y\ int\ X$.
INT3. $X\ int\ Y, Y\ isa\ Z \vdash X\ int\ Z$.
ISA2. $X\ isa\ Y, Y\ isa\ Z \vdash X\ isa\ Z$.
ISA3. $X\ isa\ Y \vdash \mathbf{non}(Y)\ isa\ \mathbf{non}(X)$.

Thus in any satisfiable containment scheme, for which the inconsistency rules INC0, INC1 are inapplicable, we may use interchangeably the syllogisms or the rules listed in the Corollary.

8. Concluding Remarks

This paper generalizes the results in [3] to consider negation in various ways. The results are encouraging, in that only a few rules are needed for complete binary containment inference. As long as we are interested only in binary properties of containment among sets, this gives us a complete inference system for set theory.

Perhaps the first work to be done is in developing algorithms using the inference systems presented here. Algorithms are beyond the scope of what we wished to present, but all of the systems of rules developed in this paper can be incorporated in inference systems that run in polynomial time. A simple upper bound is $O(n^3)$ where n is the number of types. Improved bounds will follow where more is known about the type structure. For instance, few real type hierarchies seem to be very deep. Even the standard biological taxonomy of living creatures is only about 10 levels deep.

When we begin to consider more complex forms of knowledge about types, such as sentences like

$$(\mathit{amphibian} \cap \mathbf{non}(\mathit{tailed})) \subseteq (\mathit{frog} \cup \mathit{toad})$$

the containment problem becomes NP-complete, and the inference problem co-NP-complete. Still, it would be interesting to extend the binary rules in this paper for the more general inference problem.

Other directions for further research lie in exploring graphical representations of the constraints, as in [3], and in providing efficient algorithms for containment problems. Also, there are a variety of ways to generalize the problems discussed here that the reader has no doubt already considered. These include investigating alternative types of 'syllogisms', restricting values of U in interpretations (for example, specifying $I(U)$ in advance), and so forth. The general area of containment inference is a new area in which many problems wait to be studied.

Appendix I: NP-Completeness of the Set Containment Problem

NP-completeness of the Set Containment Problem can be shown by reducing the well-known SAT problem [7] directly to it.

To show the problem is in NP, notice that a set containment scheme $(T/U,C)$ has a model if and only if it has a model over a domain D with cardinality at most N_{int}, the number of intersection constraints in C. (Given a model I having domain D with cardinality greater than N_{int}, a model I' restricting I to a subdomain of cardinality N_{int} can be constructed by repeatedly discarding members of D that are not solely used to satisfy some intersection constraint.) Thus we can solve the satisfiability problem for set containment schemes by nondeterministically guessing an interpretation I having domain of size N_{int}, then checking that I satisfies the scheme.

To make the reduction from SAT to set containment, suppose we are given a propositional formula in conjunctive form,

$$f = D_1 \wedge \cdots \wedge D_m$$

on variables $V_1, \ldots, V_n$. Each disjunct D_i is given by

$$D_i = (L_{i1} \vee \cdots \vee L_{ir_i}),$$

where each L_{ij} is either some variable V or its complement $\neg V$.

We construct a corresponding set containment scheme $S = (T/U,C)$ by putting $T/U = \{U,V_1, \ldots, V_n\}$, and defining

$$C = \{U \; int \; U\} \cup \{ C_i \mid 1 \le i \le m \}$$

as follows.

For each literal L_{ij} $1 \le i \le m$, $1 \le j \le r_i$, in the disjuncts $D_i = (L_{i1} \vee \cdots \vee L_{ir_i})$, define

$$K_{ij} = \begin{cases} V_k & \text{if } L_{ij} = V_k \\ \mathbf{non}(V_k) & \text{if } L_{ij} = \neg V_k \end{cases}$$

and let C_i be the constraint

$$C_i\colon \; (K_{i1} \cup \cdots \cup K_{ir_i}) = U.$$

for $1 \le i \le m$. Since any constraint

$$\mathbf{non}(X_1) \cup \cdots \cup \mathbf{non}(X_p) \cup Y_1 \cup \cdots \cup Y_q = U$$

is equivalent to

$$X_1 \cap \cdots \cap X_p \;\; isa \;\; Y_1 \cup \cdots \cup Y_q,$$

C consists of the positive constraints C_i, $1 \le i \le m$, and the negative constraint $U \; int \; U$. This defines S.

We claim that S is satisfiable if and only if the original formula f is satisfiable. If f is satisfiable, then there is a truth assignment *truth* satisfying f. Letting $I(U) = \{t\}$ (a singleton set),

$$I(V_k) = \begin{cases} \{t\} & \text{if } \mathit{truth}(V_k) = \textbf{true} \\ \varnothing & \text{if } \mathit{truth}(V_k) = \textbf{false} \end{cases}$$

for $1 \leq k \leq n$. This interpretation can be seen to satisfy all the constraints in C since the corresponding truth assignment satisfies f, and U *int* U is satisfied iff $I(U)$ is nonempty.

Conversely, if S has a model, then as shown above since it has one intersection constraint it must have a model I that assigns $I(U) = \{t\}$, a domain of cardinality one. Reversing the argument above, we find I induces a truth assignment satisfying f.

Appendix II: Completeness Results for Binary Containment

We prove several technical lemmas underlying completeness of the inference rules presented in this paper. These lemmas are also useful for developing inference algorithms.

Lemma 2.
If C is consistent and $C \nvdash c$, then $C' = C \cup \{\mathbf{not}(c)\}$ is also consistent.

Proof

Assume, by way of contradiction, that C' is inconsistent. Then for every Z the constraint $(Z\ \mathit{int}\ \mathbf{non}(Z))$ is derivable from C'. Let $(V\ \mathit{int}\ \mathbf{non}(V))$ be a constraint derivable from C' whose derivation does not contain any other constraint of the form $(Z\ \mathit{int}\ \mathbf{non}(Z))$. So, this derivation does not use the rules INC0, INC1.

By Lemma 4 below, there are type descriptors W_1, W_2 such that $C' \vdash (W_1\ \mathit{int}\ W_2)$ using INT1 or INT2, $C' \vdash (W_1\ \mathit{isa}\ V)$, and $C' \vdash (W_2\ \mathit{isa}\ \mathbf{non}(V))$.

We consider two cases:

1. $c = (X\ \mathit{int}\ Y)$, $\mathbf{not}(c) = (X\ \mathit{isa}\ \mathbf{non}(Y))$.
   In this case, since $(W_1\ \mathit{int}\ W_2)$ cannot match $\mathbf{not}(c)$, it must be the case that $C \vdash (W_1\ \mathit{int}\ W_2)$. Now, by Lemma 3 below, there exist $Z_0, \ldots, Z_n$ where $Z_0 = W_1$, $Z_n = V$, such that for each i, $(Z_{i-1}\ \mathit{isa}\ Z_i) \in C'$ or $(\mathbf{non}(Z_i)\ \mathit{isa}\ \mathbf{non}(Z_{i-1})) \in C'$. Similarly there exist $Z'_0, \ldots, Z'_n$ where $Z'_0 = W_2$, $Z'_n = \mathbf{non}(V)$, such that for each j, $(Z'_{j-1}\ \mathit{isa}\ Z'_j) \in C'$ or $(\mathbf{non}(Z'_j)\ \mathit{isa}\ \mathbf{non}(Z'_{j-1})) \in C'$.

   Since C is consistent, $C \nvdash (V\ \mathit{int}\ \mathbf{non}(V))$, and so it is impossible to derive both $(W_1\ \mathit{isa}\ V)$ and $(W_2\ \mathit{isa}\ \mathbf{non}(V))$ from C. So, at least one of the sequences of *isa* constraints above must include $(X\ \mathit{isa}\ \mathbf{non}(Y))$ or $(Y\ \mathit{isa}\ \mathbf{non}(X))$ or both.

   There are now several cases to consider. We show every case yields the conclusion $C \vdash (X\ \mathit{int}\ Y)$, contradicting our hypothesis that $C \nvdash c$. For simplicity we state first two rules of inference derivable from the existing rules that can be used repeatedly in the different cases:

   Rule1. $W_1\ \mathit{int}\ W_2,\ W_1\ \mathit{isa}\ X,\ W_2\ \mathit{isa}\ Y \ \vdash\ X\ \mathit{int}\ Y.$
   Rule2. $X\ \mathit{int}\ X,\ \mathbf{non}(Y)\ \mathit{isa}\ Y \ \vdash\ X\ \mathit{int}\ Y.$

   i. One sequence contains $(X\ \mathit{isa}\ \mathbf{non}(Y))$ or $(Y\ \mathit{isa}\ \mathbf{non}(X))$ alone.
      We can assume that $(X\ \mathit{isa}\ \mathbf{non}(Y))$ appears in the first sequence, and appears only once. This causes no loss of generality, since the proof for $(Y\ \mathit{isa}\ \mathbf{non}(X))$ is identical if we interchange X and Y, and the proof for the second sequence is identical if we interchange V and $\mathbf{non}(V)$, and W_1 and W_2. So the following are derivable from C:

1. W_1 *int* W_2
2. W_1 *isa* X 2a. **non**(Y) *isa* V
3. W_2 *isa* **non**(V)

From 2a and 3 we derive (W_2 *isa* Y) by ISA3 and ISA2, and from this and 1, 2, we derive (X *int* Y) by Rule1.

ii. One sequence contains both (X *isa* **non**(Y)) and (Y *isa* **non**(X)).
Without loss of generality, we can assume that each constraint appears only once and that (X *isa* **non**(Y)) precedes (Y *isa* **non**(X)). In this case, the following constraints are derivable from C:

1. W_1 *int* W_2
2. W_1 *isa* X 2a. **non**(Y) *isa* Y 2b. **non**(X) *isa* V
3. W_2 *isa* **non**(V)

From 2b and 3 derive (W_2 *isa* X) by ISA3 and ISA2, and from this and 1, 2, we derive (X *int* X) by Rule1. Then Rule2 yields (X *int* Y).

iii. Both sequences contain a single constraint, say (X *isa* **non**(Y)).
In this case the following constraints are derivable from C:

1. W_1 *int* W_2
2. W_1 *isa* X 2a. **non**(Y) *isa* V
3. W_2 *isa* X 3a. **non**(Y) *isa* **non**(V)

From 1,2,3 we derive (X *int* X) by Rule1. From 2a and 3a we derive (**non**(Y) *isa* Y) by ISA3 and ISA2. Then Rule2 yields (X *int* Y).

iv. Both sequences contain a single constraint, one with (X *isa* **non**(Y)), and the other with (Y *isa* **non**(X)).
We may assume the following constraints are derivable from C:

1. W_1 *int* W_2
2. W_1 *isa* X 2a. **non**(Y) *isa* V
3. W_2 *isa* Y 3a. **non**(X) *isa* **non**(V)

From 1,2,3 we derive (X *int* Y) using Rule1.

v. One sequence contains (X *isa* **non**(Y)), . . . ,(Y *isa* **non**(X)), and the other contains (X *isa* **non**(Y)).
We may assume the following constraints are derivable from C:

1. W_1 *int* W_2
2. W_1 *isa* X 2a. **non**(Y) *isa* Y 2b. **non**(X) *isa* V
3. W_2 *isa* X 3a. **non**(Y) *isa* **non**(V)

From 1,2,3 we derive (X *int* X) by Rule1. From this and 2a Rule2 yields (X *int* Y).

vi. One sequence contains (X *isa* **non**(Y)), . . . ,(Y *isa* **non**(X)), and the other contains (Y *isa* **non**(X)).

We may assume the following constraints are derivable from C:

1. W_1 *int* W_2
2. W_1 *isa* X 2a. **non**(Y) *isa* Y 2b. **non**(X) *isa* V
3. W_2 *isa* Y 3a. **non**(X) *isa* **non**(V)

From 1,2,3 we derive (X *int* Y) by Rule1.

vii. Both sequences contain (X *isa* **non**(Y)), . . . ,(Y *isa* **non**(X)).
We may assume the following constraints are derivable from C:

1. W_1 *int* W_2
2. W_1 *isa* X 2a. **non**(Y) *isa* Y 2b. **non**(X) *isa* V
3. W_2 *isa* X 3a. **non**(Y) *isa* Y 3b. **non**(X) *isa* **non**(V)

From 1,2,3 we derive (X *int* X) by Rule1. From this and 2a Rule2 yields (X *int* Y).

viii. One sequence contains (X *isa* **non**(Y)), . . . ,(Y *isa* **non**(X)), and the other contains (Y *isa* **non**(X)), . . . ,(X *isa* **non**(Y)).
We may assume the following constraints are derivable from C:

1. W_1 *int* W_2
2. W_1 *isa* X 2a. **non**(Y) *isa* Y 2b. **non**(X) *isa* V
3. W_2 *isa* Y 3a. **non**(X) *isa* X 3b. **non**(Y) *isa* **non**(V)

From 1,2,3 we infer (X *int* Y) by Rule1.

2. c = (X *isa* Y), **not**(c) = (X *int* **non**(Y)).
Here $C' \vdash (W_1 \textit{ int } W_2)$ using INT1 and INT2, and $C' \vdash (W_1 \textit{ isa } V)$, $C' \vdash (W_2 \textit{ isa } \textbf{non}(V))$. Now, since C' contains exactly the same *isa* constraints as C, $C \vdash (W_1 \textit{ isa } V)$, $C \vdash (W_2 \textit{ isa } \textbf{non}(V))$, so by ISA3 and ISA2 we obtain $C \vdash (W_1 \textit{ isa } \textbf{non}(W_2))$. Therefore ($W_1$ *int* W_2) is not in C, since C is consistent. So, (X *int* **non**(Y)) $\vdash$ (W_1 *int* W_2) using INT1 and INT2. This gives four cases:

i. $W_1 = X$, $W_2 = \textbf{non}(Y)$.

ii. $W_1 = \textbf{non}(Y)$, $W_2 = X$.

iii. $W_1 = X$, $W_2 = X$.

iv. $W_1 = \textbf{non}(Y)$, $W_2 = \textbf{non}(Y)$.

In cases i and ii, $C \vdash (W_1 \textit{ isa } \textbf{non}(W_2))$ yields $C \vdash (X \textit{ isa } Y)$, contradicting the hypothesis that $C \nvdash c$.
In cases iii and iv, $C \vdash (W_1 \textit{ isa } \textbf{non}(W_2))$ yields respectively $C \vdash (X \textit{ isa } \textbf{non}(X))$ and $C \vdash (\textbf{non}(Y) \textit{ isa } Y)$, from either of which TRIV0 derives $C \vdash (X \textit{ isa } Y)$, and again contradiction.

□

Lemma 3.
If $X \neq Y$, $C \nvdash (U\ isa\ Y)$, and $C \nvdash (X\ isa\ \mathbf{non}(U))$, then $(X\ isa\ Y)$ is derivable from C without using INC0 or INC1 if and only if there exist type descriptors $Z_0, \ldots, Z_n$ such that $X = Z_0$, $Y = Z_n$, and for each i between 1 and n either $(Z_{i-1}\ isa\ Z_i)$ is in C or $(\mathbf{non}(Z_i)\ isa\ \mathbf{non}(Z_{i-1}))$ is in C.

Proof

If the derivation of $(X\ isa\ Y)$ from C does not use INC0, INC1, it can be reduced to a derivation that does not involve intersection constraints, since the only way to derive *isa* constraints from them is by means of INC0.

So, we have a derivation that uses rules ISA0-3. We show that this derivation can be transformed into a derivation that has no applications of ISA0 or ISA1, and has all applications of ISA3 before applications of ISA2. Every step using ISA1 can be eliminated, since no constraint of the form $(Z\ isa\ Z)$ can give any effective contribution to the derivation. With respect to ISA0, since $C \nvdash (U\ isa\ Y)$ and $C \nvdash (X\ isa\ \mathbf{non}(U))$, no constraint of the form $(Z\ isa\ U)$ can be used in a non-redundant derivation. Therefore any steps using ISA0 can be eliminated. Then, any constraint derived by means of ISA3 to the result of an application of ISA2 can be obtained by applying ISA3 first and ISA2 to the results of the applications of ISA3: any derivation

1. $V\ isa\ Z$
2. $Z\ isa\ W$
3. $V\ isa\ W$ (from 1, 2, by ISA2)
4. $\mathbf{non}(W)\ isa\ \mathbf{non}(V)$ (from 3, by ISA3)

can be replaced by the derivation

1. $V\ isa\ Z$
2. $Z\ isa\ W$
3. $\mathbf{non}(Z)\ isa\ \mathbf{non}(V)$ (from 1, by ISA3)
4. $\mathbf{non}(W)\ isa\ \mathbf{non}(Z)$ (from 2, by ISA3)
5. $\mathbf{non}(W)\ isa\ \mathbf{non}(V)$ (from 4, 5, by ISA2).

This means that any derivation of *isa* can be transformed to a derivation using rule ISA3 to the constraints in C and then ISA2 as many times as needed. Then, an induction on the number of applications of ISA2 completes the proof. □

Lemma 4.
$(X\ int\ Y)$ is derivable from C without using INC0 or INC1 if and only if there exist type descriptors W_1, W_2 such that $(W_1\ int\ W_2)$ is either in C or is derivable from C using only INT1 and INT2, and $(W_1\ isa\ X)$ and $(W_2\ isa\ Y)$ are derivable from C without using INC0 or INC1.

Proof

Given a derivation of (X *int* Y) as in the hypotheses, it is possible to transform it into a derivation that presents all the steps deriving *isa* constraints first, then all the steps deriving intersection constraints, since no *isa* constraint is derived from intersection constraints.

Now if this derivation is nonredundant, in each of the steps generating intersection constraints one intersection constraint is used and one generated. Therefore, since each derived constraint (except the last one, X *int* Y) must be used in a subsequent step, the whole sequence uses only one intersection constraint in C. Let this constraint be (V_1 *int* V_2). Now by induction on the total number of intersection constraints appearing in the derivation, we prove the following claim:

Claim Let (X *int* Y) be any intersection constraint derived from a set C of *isa* constraints and (V_1 *int* V_2). Then $C \vdash (V_i \; isa \; X)$ with i=1 or 2, and $C \vdash (V_j \; isa \; Y)$ with j=1 or 2.

Basis $X = V_1$, $Y = V_2$.
Trivial.

Induction (X *int* Y) follows in the derivation from (Z_1 *int* Z_2) and possibly an *isa* constraint, and $C \vdash (V_i \; isa \; Z_1)$, $C \vdash (V_j \; isa \; Z_2)$. We distinguish various cases, according to the last rule used in deriving (X *int* Y).

1. INT0: $X = Z_1$, $Y = U$.
   (V_j *isa* Y) derives from ISA0, and (V_i *isa* X) is the same as (V_i *isa* Z_1).
2. INT1: $X = Y = Z_1$.
   (V_i *isa* X) and (V_j *isa* Y) are both the same as (V_i *isa* Z_1).
3. INT2: $X = Z_2$, $Y = Z_1$.
   (V_i *isa* X) and (V_j *isa* Y) are the same as (V_j *isa* Z_2) and (V_i *isa* Z_1), respectively.
4. INT3: $X = Z_1$, and (Z_1 *int* Y) is derived from (Z_1 *int* Z_2) and (Z_2 *isa* Y).
   (V_i *isa* X) is the same as (V_i *isa* Z_1), and (V_j *isa* Y) follows from (V_j *isa* Z_2) and (Z_2 *isa* Y).

The lemma follows directly from this claim. Put $W_1 = V_i$, $W_2 = V_j$. Then (V_1 *int* V_2) $\vdash$ (W_1 *int* W_2) using only INT1 and INT2, and also $C \vdash (W_1 \; isa \; X)$ and $C \vdash (W_2 \; isa \; Y)$.

□

Acknowledgement

Richard Huntsinger, Karen Lever, and Tom Verma gave the paper a careful reading and suggested many improvements in the presentation. The anonymous referees made a number of important suggestions, improving both the organization and clarity of the paper.

References

1. Arisawa, H. and T. Miura, On the Properties of Extended Inclusion Inference, *Proc. 12th Intnl. Conf. on Very Large Data Bases*, Kyoto, September 1986, 449-456.

2. Attardi, G. and M. Simi, Consistency and Completeness of OMEGA, a Logic for Knowledge Representation, *Proc. 7th IJCAI*, Vancouver, August 1981, 504-510.

3. Atzeni, P. and D. Stott Parker, Formal Properties of Net-based Knowledge Representation Schemes, *Proc. 2nd Conference on Data Engineering*, Los Angeles, CA, February 1986.

4. Cerro, L.F. del and E. Orlowska, DAL – A Logic for Data Analysis, *Theoretical Computer Science* **36**, 1985, 251-264.

5. Cosmadakis, S.S. and P.C. Kanellakis, Two Applications of Equational Theories to Database Theory, *Proc. Conf. on Rewriting Techniques and Applications*, New York, 1985, 107-123.

6. Gardner, M., *Logic Machines and Diagrams,* University of Chicago Press, 1982.

7. Garey, M.R. and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness,* W.H. Freeman and Co., San Francisco, 1979.

8. Johnson-Laird, P.N. and B.B. Bara, Syllogistic Inference, *Cognition* **16**, 1984, 1-61.

9. Kanellakis, P.C., S.S. Cosmadakis, and M.Y. Vardi, Unary Inclusion Dependencies have Polynomial Time Inference Problems, *Proc. 15th ACM Symp. on Theory of Computing*, Boston, April 1983, 264-277.

10. Maier, D., *The Theory of Relational Data Bases,* Computer Science Press, Rockville, MD, 1983.

11. Papalaskaris, M.A. and L.K. Schubert, Parts Inference: Closed and Semi-closed Partitioning Graphs, *Proc. 7th IJCAI*, Vancouver, August 1981, 304-309.

12. Pawlak, Z., Rough Sets, *International J. Computer & Information Sciences* **11**, 1982, 341-356.

13. Schubert, L.K., Problems with Parts, *Proc. 6th IJCAI*, Tokyo, August 1979, 778-784.

14. Schubert, L.K., M.A. Papalaskaris, and J. Taugher, Determining Type, Part, Color, and Time Relationships, *IEEE Computer*, October 1983, 53-60.

15. Spyratos, N., The Partition Model: A Deductive Database Model, Report No. 286, INRIA, April 1984, to appear in ACM TODS, March 1987.

16. Ullman, J.D., *Principles of Data Base Systems, 2nd Ed.,* Computer Science Press, Rockville, MD, 1984.