Symmetric Relations, Intensional Individuals, and Variable Binding

STUART C. SHAPIRO, MEMBER, IEEE

*A symmetric relation such as "... are adjacent" or "... are related" is characterized by not distinguishing among two or more of its arguments. Such a relation may efficiently be represented as a relation that takes a set as its argument, or as one of its arguments. The semantics of such a representation is, in part, determined by the instantiation (matching or unification) rule used by the reasoning system operating on the representation. Two such rules are discussed. One interprets the relation to be reflexive, the other does not. Since many of these relations are not reflexive, we prefer the latter rule, which forbids two distinct variables from matching the same term. It is argued that this apparently strange restriction is actually reasonable if the rules of the system are interpreted as fully intensional. Under that interpretation, an even stronger version of the instantiation rule emerges, which we name the Unique Variable Binding Rule (**UVBR**). Considering the behavior of the **UVBR** when reasoning about reflexive relations and about nonreflexive relations used reflexively casts light on the implications of a fully intensional knowledge representation scheme. These ideas are illustrated by the output of an intensional, rule-based knowledge representation system that has been modified to allow the choice of using the **UVBR** instead of standard unification.*

I. Introduction

This paper is one in a series advocating the intensional interpretation of knowledge representation systems and examining the implications of that view [1], [5], [6], [9], [10], [12], [15]. In particular, in this paper, we consider the representation and use of such assertions as "Betty, Jane, and Mary resemble each other." This sentence seems already to contain the information that "Betty and Mary resemble each other." So the latter should be obtainable from the former with far less explicit reasoning than inferring it from "Betty and Jane resemble each other" and "Jane and Mary resemble each other." The keys to the fast inference are in using sets as arguments, and in designing an appropriate instantiation rule to make use of them. These ideas are developed in Sections II and III.

Manuscript received April 20, 1985; revised March 31, 1986. This work was supported in part by the Air Force Systems Command, Rome Air Development Center, Griffiss Air Force Base, Rome, NY 13441-5700, and by the Air Force Office of Scientific Research, Bolling AFB, Washington, DC 20332, under Contract F30602-85-C-0008.

The author is with the Department of Computer Science, State University of New York at Buffalo, Buffalo, NY 14260, USA.

Section IV discusses relations such as "are brothers" which are not reflexive, although the instantiation rule we develop in Section III would treat them as such. Therefore, in Section V, we present an alternative rule that forbids two distinct variables in one rule from binding to the same term. In section VI, we discuss the operation of these two rules on negated assertions such as "Betty, Jane, and Mary do not resemble each other."

In Section VII, we present an intensional interpretation of variables that is similar to that of [4] rather than that of [5]; namely, that variables represent arbitrary, intensional individuals. Based on this interpretation, an even stronger version of the modified instantiation rule, called "the Unique Variable Binding Rule" (**UVBR**) is presented, along with the thesis that this rule should be used in intensional knowledge representation/reasoning systems.

In Sections VIII and IX, we discuss the implications of using the **UVBR** in cases of reflexive relations and relations used reflexively. The major implication is that when an individual is related to itself, it is really treated as two intensional individuals which are co-referential.

In the Appendix, we illustrate the differences among the instantiation rules by showing output from a knowledge representation/reasoning system that has been modified to allow the user to choose which rule is to be used.

II. Sets as Arguments

There are many relations and functions that do not distinguish among two or more of their arguments. Symmetric relations, such as "adjacent" are like this; so are most predicate functions, such as "and" and "or," and many other functions, such as "sum." Nevertheless, it is common in knowledge representation and reasoning systems, as well as in non-AI formalizations, to represent these relations and functions by predicates that do distinguish among their arguments. For example,

(1) The sum of 5 and 9 is 14.
(2) Canada and the U.S. are adjacent.
(3) Mary is the female parent of Bill.
(4) Logic gate *g*1 has input wires *iw*1 and *iw*2 and output wire *ow*1.

0018-9219/86/1000-1354$01.00 © 1986 IEEE

might be represented by

(1') *Sum(5, 9, 14)*
(2') *Adjacent(Canada, US)*
(3') *Female(Mary) & Parent-of(Mary, Bill)*
(4') *Gate(g1, iw1, iw2, ow1)*

respectively.

With such representations, the fact that these relations and functions do not distinguish among some or all of their arguments must be represented by explicit rules, or must be programmed explicitly into the inference mechanism. Reasoning about the above examples would require the rules.

(r1) *Sum(x, y, z) => Sum(y, x, z)*
(r2) *Adjacent(x, y) => Adjacent(y, x)*
(r3) *(A & B) => (B & A)*
(r4) *Gate(w, x, y, z) => Gate(w, y, x, z).*

Since the only difference between the antecedents and consequents of these rules is the order of their arguments, they are directly recursive, and would cause many reasoning systems to go into an infinite loop (but see [8], [13]).

Such rules would not be needed if relations and functions that do not distinguish among some of their arguments were represented by predicates that take sets in one of their argument positions. Examples (2)–(4) above could then be represented by

(2") *Adjacent{Canada, US}*
(3") *&{Female(Mary), Parent-of(Mary, Bill)}*
(4") *Gate(g1, {iw1, iw2}, ow1).*

The *Sum* example apparently could not be treated this way, since there would then be no way to represent *Sum(5, 5, 10)* (but see Section VIII). Rules (r2)–(r4) would not be needed because their antecedents and consequents would be exactly the same. This assumes, of course, that the problem of recognizing two explicitly given sets as the same is a simpler problem than that of applying recursive rules such as those shown above. This is certainly true, especially if all sets are canonically represented. In this paper, we will display sets with their elements numerically or lexicographically ordered.

The one style of knowledge representation system that has traditionally used sets as arguments of relations and functions is the semantic network (see the papers in [3], [7]). In semantic networks, arguments are distinguished, not by position, but by keyword, with the keyword being an arc label. To put a set as the argument of a predicate, a semantic network has several arcs with the same label going from the node representing the predicate to each of the members of the set. Arcs that themselves represent relationships are elements of a conjunctive set of assertions. For example, Fred, an albino male penguin, may be represented as a node with Isa arcs to the nodes for "albino," "male," and "penguin." These arcs are not ordered; they are just there. In a linear formalism, this situation is most accurately reflected by the use of sets, either *&{Isa(Fred, albino), Isa(Fred, male), Isa(Fred, penguin)}*, or *Isa(Fred, {albino, male, penguin})*.

III. Symmetric Relations

It need hardly be stressed that a binary relation (between members of the domain *D*) may be represented as a unary relation whose argument is a set of two elements (of *D*) only if the relation is symmetric.[1] If *R* is asymmetric, *(∀(x, y)[R(x, y) => ¯R(y, x)])*, sets cannot be used, because, for example, *R{a, b}* does not distinguish between the two cases *R(a, b)* and *R(b, a)*, only one of which can hold. Similarly, if *R* is nonsymmetric (neither *∀(x, y)[R(x, y) => R(y, x)]*, nor *∀(x, y)[R(x, y) => ¯R(y, x)])*, there is no consistent representation, using sets, of the possible situation, *R(a, b) & ¯R(b, a)*.

If *R* is both symmetric and transitive (and, therefore, also reflexive[2]), there is great representational economy in representing it as a unary relation that takes a set of arbitrary cardinality. *R{a, b, c, d}* can be taken to represent that all of the sixteen binary relationships, *R(a, a)*, *R(a, b)*, ⋯, *R(d, c)*, and *R(d, d)* hold. I say, "can be taken to represent" rather than "represents," because the issue is really decided by the inference mechanism and, basically, by the pattern-matching mechanism. The following questions are among those that need to be answered:

1) If *R{a, b, c, d}* is the only assertion in the system, and the query is *R{b, d}?*, what is the answer?

2) If *R{a, b, c, d}* is the only assertion in the system, and the Wh-question *R{a, x}?* is asked, what are the answers?

3) If *R{a, b, c, d}* is asserted along with the rule *R{x, y} => S(x, y)*, and the query *S(w, z)?* is asked (note that *S* distinguishes between its two arguments), what is the answer?

If we want *R* to be treated as symmetric, transitive, and reflexive, we want the answers to these three questions to be:

1) Yes.

2) *x* can be *a*, *b*, *c*, or *d*.

3) All sixteen relationships, *S(a, a)*, *S(a, b)*, ⋯, *S(d, c)*, *S(d, d)*.

We can enforce that treatment with an inference/pattern-matching mechanism that implements the rule:

R1 If α and β are sets, possibly containing variables but with no variable in both α and β, *R* is a relation, and $R\alpha$ holds, then if there is a subset α_1 of α and a substitution σ which unifies $R\alpha_1$ and $R\beta$, we may conclude $R\beta\sigma$.

In all above three examples, the set α is the set *{a, b, c, d}*. In example (1), β and α_1 are the set *{b, d}*, and σ is the null substitution. In (2), β is *{a, x}*, and there are four possible values for α_1 and σ, *viz.*: *{a}* and *{a/x}*; *{a, b}* and *{b/x}*; *{a, c}* and *{c/x}*; *{a, d}* and *{d/x}*. In (3), there are sixteen possible values for β and σ, each of which produces an instance of the antecedent of the given rule, allowing the mentioned sixteen *S* conclusions.

The extension of rule **R1** to relations of multiple arguments, one or more of which are sets, is obvious but tedious, and so will not be given explicitly here. **R1** does not apply to all functions. For example, although it does apply to &, it does not apply to *OR*. *OR* can take a set of arguments, but *OR{P, Q, R}* does not imply *OR{P, Q}*. A version of **R1**

[1] I mean to exclude the use of sets of sets, such as *{{a}, {a, b}}*, to code ordered pairs, such as *⟨a, b⟩*.

[2] By "reflexive" in this paper, I mean reflexive in its domain, *viz.* *∀(x, y)[R(x, y) => R(x, x) & R(y, y)]* rather than *∀xR(x, x)*.

can apply to the set-oriented logical connectives used in SNePS [11], but a discussion of these connectives would distract the reader too much from the main points of this paper.

Notice that if **R1** is used on a relation *R, R will be treated as a symmetric, transitive, and reflexive relation.* One might, instead, restrict **R1** so that it only applies to such relations, but my main concern in this paper is to investigate instantiation (unification-like) rules that can be applied uniformly, as part of the underlying reasoning mechanism in a knowledge representation/reasoning system. Second-order properties of relations (such as symmetry) can be asserted explicitly in such a system (and SNePS can represent such assertions), so any rule that has such a property as a condition will be an explicit rule of the system rather than an underlying rule of inference used to implement the system. (Compare explicit rules represented as Prolog clauses to the resolution rule of inference used to implement Prolog itself.)

Also note that the transitivity obtained by the use of **R1** does not obviate the need for explicit transitivity rules, but where **R1** applies it provides a faster, more direct inference. The distinction occurs with humans reasoning on natural language examples. An **R1**-like rule may be used to infer from "Bob, Joe, and Harry are brothers" that "Bob and Harry are brothers," but more explicit reasoning is needed to reason from "Bob, Joe, and Harry are brothers" and "Harry, Sam, and Ted are brothers" to "Bob and Sam are brothers."

Representing "Betty, Jane, and Mary resemble each other" as *Resemble{Betty, Jane, Mary}*, and applying **R1**, we get *Resemble{Jane}*. Should this be read as "Jane resembles herself," or as the ungrammatical "Jane resembles each other"? It is at least reasonable to argue that relations like "resembles each other" never apply to only one argument, even though we normally say that they are reflexive. In the next two sections, we develop an alternative rule to **R1** that would not infer *Resemble{Jane}* from *Resemble{Betty, Jane, Mary}*, and in Section IX we return to the representation of "Jane resembles herself."

IV. Symmetric, Nonreflexive Relations

As intended, when rule **R1** is applied to relations represented as unary relations on sets of arbitrary cardinality, the relations are taken to be symmetric, transitive, and reflexive (equivalence relations). When **R1** is applied to relations represented as unary relations on sets of no more than two elements, the relations are treated as symmetric and reflexive. There are, however, many relations that are symmetric and nonreflexive (neither $\forall(x)\, R(x, x)$ nor $\forall(x)^{-} R(x, x)$), and many that are symmetric and irreflexive ($\forall(x)^{-} R(x, x)$). Some of these were mentioned in Section II. *Adjacent(Canada, US)* implies *Adjacent(US, Canada)*, but not *Adjacent(Canada, Canada)*, nor *Adjacent(US, US)*. *Gate(g1, iw1, iw2, ow1)* implies *Gate(g1, iw2, iw1, ow1)*, but not *Gate(g1, iw1, iw1, ow1)* nor *Gate(g1, iw2, iw2, ow1)*. *Sum(5, 9, 14)* implies *Sum(9, 5, 14)*, but not *Sum(5, 5, 14)* nor *Sum(9, 9, 14)*.

Other symmetric, nonreflexive relations are what we might call "almost transitive." An almost transitive relation *R* obeys the rule, $\forall(x, y, z)[xRy \,\&\, yRz \,\&\, x \neq z \Rightarrow xRz]$. Examples are "are brothers" and "are related." If Bob, Joe, and Harry are brothers, then certainly Bob and Harry are brothers, but we would not want to say that Bob and Bob are brothers, even though Bob and Joe are brothers and Joe and Bob are brothers.

For the reasons cited in Sections II and III, we want to represent symmetric relations, even symmetric, nonreflexive relations, as relations that take sets as arguments. For example, we want to use the representations *Adjacent{Canada, US}*, *Gate(g1, {iw1, iw2}, ow1)*, *Brothers {Bob, Joe, Harry}*. However, rule **R1** would treat these irreflexive relations as reflexive.

Experience shows that symmetric nonreflexive relations and symmetric almost transitive relations are used more often in AI reasoning and expert systems than symmetric reflexive relations or equivalence relations. We here, therefore, developed a modification of **R1** that treats symmetric relations as nonreflexive, rather than reflexive. Additional motivations arise from more fundamental knowledge representation issues and will be discussed in Section VI.

Treating *R* as a symmetric, almost transitive relation, *R{a, b, c, d}* would be taken to represent that the twelve binary relationships, *R(a, b)*, *R(a, c)*, $\cdots$, *R(d, b)*, and *R(d, c)* hold. In this case, we would want the three questions of Section III to be answered as:

1) Yes.
2) *x* can be *b*, *c*, or *d*.
3) The twelve relationships, *S(a, b)*, *S(a, c)*, $\cdots$, *S(d, b)*, *S(d, c)*.

This treatment of question (3) shows that the modification of **R1** must forbid two different variables from matching the same term. Answer (2) shows that a variable must also not match a term also contained in the same relationship. Our formal revision of **R1** will be presented in Section V.

Some example rules using symmetric nonreflexive relations are:

Brothers like each other: *Brothers{x, y}* => *Likes(x, y)* (notice that *Likes* is nonsymmetric, and we do not need to add *& Likes(y, x)* because the rule will apply two, though not four, ways to *Brothers{Bob, Joe}*).

In a neurologic diagnosis system [16], we want to say that if one nerve tract is malfunctioning, and it is adjacent to another, then the other should be examined: *&{Malfunctioning(x), Tract(x)} => [&{Adjacent{x, y}, Tract(y)} => Should(examine, y)]*.

In a fault-diagnosis system, we want to say that if the voltage of the two inputs to an AND gate are high and the output is low, the AND gate is faulty: *&{AND-gate(x), Gate(x, {y, z}, w)} => [&{High(y), High(z), Low(w)} => Faulty(x)]*.

Notice that in all these examples, sets are used as arguments when the order of the elements of the sets is irrelevant, but that it is important that the variables in the sets not be substituted for by the same term. In the first example, no one is his own brother, so the rule should not be used to infer that someone likes himself. In the second example, a tract is not adjacent to itself, and, moreover, a tract already known to be malfunctioning need not be examined further. In the third example, an AND gate is not necessarily faulty because its output wire is low and one of its input wires is high; the other input wire might be low.

V. A Rule for Symmetric, Nonreflexive Relations

As mentioned above, the proper modification of rule **R1** to treat relations with sets as arguments as symmetric, nonreflexive, and almost transitive is to prevent two different variables in the same relation from matching the same term, and to prevent any variable in a relation from matching another term in it. This modification is incorporated in the rule:

R2 If α and β are sets, possibly containing variables but with no variable in both α and β, R is a relation, and $R\alpha$ holds, then if there is a subset α_1 of α and a substitution σ which unifies $R\alpha_1$ and $R\beta$ such that the cardinality of $\beta\sigma$ is the same as the cardinality of β, we may conclude $R\beta\sigma$.

The cardinality restriction enforces both the restriction that two different variables cannot match the same term and that no single variable can match a term also in the same relation, because in either of these two cases, $\beta\sigma$ would have a smaller cardinality than β. This assumes that the representation of sets and their elements are such that the cardinality count will not be confused by the same term being represented in two different ways.

VI. Negated Relations

If $R\beta$ is an instance of $R\alpha$, then so is $\neg R\beta$ an instance of $\neg R\alpha$. This raises the question of the semantics of negative assertions such as $\neg$*Resemble*{*Betty, Jane, Mary*}. We will discuss this in two parts: the negation of a relation applied to a set of two elements; the negation of a relation applied to a set of more than two elements.

Maida and Shapiro [6] discuss the relation *EQUIV*{*x, y*}, which means that the two intensional entities, *x* and *y*, are co-referential. The proposition that the Morning Star is not Mars, would be represented by $\neg$*EQUIV*{*Mars, MorningStar*}, but, by **R1**, taking {*Mars, MorningStar*} for α, {*MorningStar*} for α_1, {*x, y*} for β, and {*MorningStar/x, MorningStar/y*} for σ, this would imply that the Morning Star is not co-referential with itself (for purposes of some rule with $\neg$*EQUIV*{*x, y*} in its antecedent). This problem was the actual motivation for the work reported in this paper. **R2** solves this problem since the cardinality of $\beta\sigma$ is 1 while the cardinality of β is 2, and this is disallowed by **R2**.

A more serious problem is exemplified by $\neg$*Resemble*{*Betty, Jane, Mary*}. If *Resemble*{*Betty, Jane, Mary*} means that "Betty, Jane, and Mary resemble each other" it should be the case that $\neg$*Resemble*{*Betty, Jane, Mary*} means that "Betty, Jane, and Mary do not resemble each other." The question is, does it follow from "Betty, Jane, and Mary do not resemble each other" that "Betty and Jane do not resemble each other?" Logically, it does not, since the complement of a transitive relation is not necessarily transitive. For example, from *x is different from y* and *y is different from z* it does not follow that *x is different from z*. On the other hand, we have already pointed out that *R*{*x, y*} & *R*{*y, z*} is treated differently from *R*{*x, y, z*}.

Whatever the intuition, **R1** and **R2** will both take $\neg R${*a, b, c*} as implying $\neg R${*a, b*}, etc. Therefore, just as the semantics of *Ra*, where *a* is a set, is "The elements of *a* all have the relation *R* to each other," the semantics of $\neg$*Ra* is "None of the elements of *a* have the relation *R* to any of the others," *if* **R1** *or* **R2** *is used as the rule of instantiation.*

VII. An Intensional Interpretation

In previous papers [6], [12], we argued that symbols of a knowledge representation system represent intensions rather than extensions, and that the Uniqueness Principle holds in propositional semantic networks. The Uniqueness Principle states that every concept represented in the network is represented by one and only one node. Two distinct nodes always represent two different intensions, or concepts, although they may be asserted to be co-referential by an instance of the *EQUIV* equivalence relation. *EQUIV* is not *Equal*. In particular, the entire network, being intensional, is an opaque context, and a property may not be transferred from one node to an *EQUIV* one without an explicit assertion that the property is referentially transparent.

A variable may be considered to represent an intension; namely, the arbitrary individual satisfying the appropriate conditions of the antecedent of the rule within which the variable appears (or satisfying the restriction of its quantifier, if restricted quantifiers are used) (cf. [4]). For example, in the rule

&{*Malfunctioning(x), Tract(x)*}
=> [&{*Adjacent*{*x, y*}, *Tract(y)*}
=> Should(examine, y)]

x represents an arbitrary malfunctioning tract, and *y* represents an arbitrary tract adjacent to it. In the rule,

&{*AND-gate(x), Gate(x,* {*y, z*}*, w)*}
=> [&{*High(y), High(z), Low(w)*}
=> Faulty(x)]

x represents an arbitrary AND gate, *y* and *z* its arbitrary input wires, and *w* its arbitrary output wire.[3] It is important to realize that, in the kind of semantic network being discussed here, the variables *x* and *y* of the first rule would be different nodes from the *x* and *y* used in the second rule, but that all instances of a single variable in one rule are precisely the same node. Moreover, since semantic networks allow structure sharing, every rule about AND gates can share the very same node representing the predicate *AND-gate(x)*, and, by the Uniqueness Principle, this should be the case. That means that the node used for this *x* is the one and only node in the network representing an arbitrary AND gate, and all rules about AND gates are propositions about this arbitrary AND gate. Thus the one and only arbitrary AND gate serves the same role in this representation scheme as the typical AND gate would in some other representation systems.

Considering the discussion that has gone on so far, the interpretation of these rules, and the interpretation of variables as arbitrary intensional entities, it should be clear that distinct variables used in a single rule are meant to represent distinct concepts. *Therefore, it would be an abuse of the intended meaning of the rule to instantiate it so that two distinct concepts were collapsed into one.* This is the

[3] Cf. [4] for a discussion of the way arbitrary individuals *y, z,* and *w* are dependent on arbitrary individual *x*.

primary theoretical justification for the part of rule **R2** requiring that two variables not match the same term. Removed from the context of relations having sets as arguments, we will call it the Unique Variable Binding Rule:

UVBR No instance of a rule is allowed in which two distinct variables of the rule are replaced by the same term, nor in which any variable is replaced by a term already occurring in the rule.

The central thesis of this paper is that the Unique Variable Binding Rule should be used in AI rule-based systems that adopt the principle of intensional representation and the Uniqueness Principle.

In [2], a network knowledge representation system, NETL, was presented that used typical individuals as the main method for storing general knowledge. A version of **UVBR** also arose, in a slightly different guise, in the NETL book. The issue presented there was: if Hates(TYPICAL-ELEPHANT, TYPICAL-ELEPHANT) is stored in the system, does that mean that: 1) every elephant hates itself; 2) every elephant hates every elephant including itself; or 3) every elephant hates every elephant excluding itself? It was decided that Hates(TYPICAL-ELEPHANT, TYPICAL-ELEPHANT) should represent (1), every elephant hates itself. In order to represent (3), a new representational technique was introduced; namely, a chain of typical individuals (called *OTHER-nodes in [2, pp. 153–159]). Let TYPICAL-ELEPHANT be the typical elephant, TYPICAL-ELEPHANT1 be any elephant except TYPICAL-ELEPHANT, TYPICAL-ELEPHANT2 be any elephant except TYPICAL-ELEPHANT or TYPICAL-ELEPHANT1, etc., for as many elephants as needed for any general statement. Thus to represent (1), put Hates (TYPICAL-ELEPHANT, TYPICAL-ELEPHANT) in the network. To represent (3), put Hates(TYPICAL-ELEPHANT, TYPICAL-ELEPHANT1) in the network. To represent (2), put them both in the network.

The interpretation of variables being presented in the present paper, along with the **UVBR**, includes the NETL chain of typical individuals as a special case. The statement that every elephant hates itself is represented by the rule *Elephant(x) => Hates(x, x)*. The statement that every elephant hates every other elephant is represented by *&{Elephant(x), Elephant(y)} => Hates(x, y)*. The statement that every elephant hates every elephant including itself is represented by *Elephant(x) => &{Hates(x, x), Elephant(y) => Hates(x, y)}*.

VIII. Nonreflexive Relations Used Reflexively

An apparent problem is the reflexive use of nonreflexive relations, or (which we shall discuss first) cases where a single constant appears in two or more argument positions of a relation that distinguishes its arguments. It might appear that, because of the **UVBR**, the general rules designed to deal with these relations would not work properly in these cases. The argument to be made in this section is that if one uses intensional representation, these cases never occur! Instead, different intensional constants are involved in the relations, but these different constants are co-referential.

Consider the rule, "If someone likes someone, she buys her presents." This can be represented[4] as *Likes(x, y) => Buys(x, y, presents)*. As should now be clear, this rule will not apply to the situation expressed by "Jane likes herself" if we represent that by *Likes(Jane, Jane)*. We probably do want to conclude that "Since Jane likes herself, she buys herself presents." Notice, however, that we can go on:

> Since Jane likes herself, she buys herself presents, and since she knows herself so well they are always just right and never unduly duplicate things she already has, which is good, because otherwise she would have to tell herself to take them back.

What seems to be going on here is that Jane is filling two different roles, that of the gift giver, and that of the gift receiver. To say that Jane is filling two roles is just another way of saying that there are two different intensions involved, Jane the gift giver, and Jane the gift receiver, and that they are both co-referential with Jane. (Also see [14] for a linguistic discussion of this phenomenon.)

If this analysis is correct, the correct way to represent "Jane likes herself" is *&{EQUIV{Jane, Jane1}, Likes(Jane, Jane1)}*. The rule about buying presents applies to this instance of *Likes*, and implies *Buys(Jane, Jane1, presents)*, which shows Jane in her two roles.

A similar analysis applies to cases like applying the rule for computing the area of a rectangle to a particular rectangle that just happens to be a square. Just because the length and the width happen to co-refer to the same value does not mean that they are not distinct intensional entities. At very bottom, the elementary school rule that 2 times 2 is 4 mentions two intensions, the multiplier and the multiplicand, which are co-referential. Thus we are not really multiplying a number by *itself*, but are multiplying two *different occurrences* of a number by each other.

Therefore, we can revise our conclusion of Section II that sets cannot be used for the representation of *Sum*, "since there would then be no way to represent *Sum(5, 5, 10)*." The representation would in fact be *∀(x, y)[EQUIV{x, y, 1} => ∃(z)&[EQUIV{z, 2}, Sum({x, y}, z)]]*. That is, if you add any occurrences of 1 together, you get an occurrence of 2.

IX. Reflexive Relations Revisited

Finally, let us consider the use of **UVBR** with reflexive relations. There apparently are two kinds: ones that distinguish their arguments, such as "≤"; ones that do not, such as *Similar*. However, the analysis of the previous section holds, and shows that, in order to compare an individual to itself, it is necessary to consider it as two intensional individuals. Thus to say that every number is ≤ itself, we say that for every two occurrences of it, one is ≤ the other, i.e., *∀x[Number(x) => ∀y[EQUIV(x, y) => x ≤ y]]*, and to say that everything is similar to itself, we say, *∀(x, y)[EQUIV(x, y) => Similar{x, y}]*. So, the representation of "Jane resembles herself" discussed at the end of Section III is not *Resemble{Jane}*, but *∀x[EQUIV{Jane, x} => Resemble{Jane, x}]*.

If this proliferation of co-referential entities seems very

[4]We will drop the Person predicate for space reasons, and ignore tense, aspect, and number because they are independent of the concerns of this paper.

cumbersome, it is probably because it is, and to overcome this, humans have a number of special case terms. So, instead of adding a number to itself, we *double* it, and instead of a rectangle with equal length and width, we speak of a *square*.

X. Implementation Status

The SNePS Semantic Network Processing System [11] is a combined knowledge representation/reasoning system that has used **R1**. Recently, SNePS was modified to allow a choice of **R1** or **UVBR**. The Appendix contains transcripts of SNePS runs showing the difference between **R1** and **UVBR** on the major examples of this paper.

XI. Conclusions

This paper is one in a series advocating the intensional interpretation of knowledge representation systems and examining the implications of that view [1], [5], [6], [9], [10], [12], [15]. Here, we have been concerned with the interpretation, as intensional individuals, of variables in rules of rule-based systems. Under this interpretation, it is inappropriate to use an instance of a rule that collapses two or more variables into the same term, because that would violate the meaning intended by whomever entered that rule in the system. The rule expressing this restriction on variable binding was termed the Unique Variable Binding Rule (**UVBR**).

It turned out that the **UVBR** is a vital part of a rule (**R2** of Section V) for instantiating symmetric, nonreflexive relations, and symmetric, almost transitive relations without treating them as reflexive. Rule **R2** is needed to take advantage of the benefits of representing such relations as relations with sets as one or more of their arguments. These benefits include an almost exponential savings in representation space (e.g., one relation on a set of four elements instead of twelve relations) and the automatic inference of the relation's symmetric and transitive implications by the selection of subsets.

The **UVBR** further supports the intensional interpretation of knowledge representation systems, because in those cases where nonreflexive relations are used reflexively, it requires that the different occurrences of an individual in different argument positions be considered different, but co-referential, intensional individuals.

Appendix

In order to experiment with the **UVBR**, a modification was made to the SNePS semantic network processing system [11], so that if the switch **DuplBind** (a global Lisp variable) is set to ***t***, **R1** is used, but if **DuplBind** is set to ***nil***, **UVBR** is used.

This Appendix contains a script of a run of SNePS showing the difference between **R1** and **UVBR**. SNePS is implemented in Franz Lisp and, for this demonstration, was run, compiled, on a VAX 11/750 under the UNIX™ operating system. The SNePS prompt is "*". Lines beginning that way are input to SNePS. Lines beginning with ";" are comments. Other lines are SNePS output. Text beginning with a ";" in mid-line is a comment added to the script for this paper.

```
; Define the arc labels we will use.
; For each label R, R- labels the converse arc.
;
* (define member member- class class- object object-
*         property property- rel rel- argument argument-
*         a1 a1- a2 a2- a3 a3-)

(member member-)        ; points to an object being classified
(class class-)          ; points to the class of an object
(object object-)        ; points to an object
(property property-)    ; points to the property of an object
(rel rel-)              ; points to a relation
(argument argument-)    ; points to a set of arguments
(a1 a1-)                ; points to the first argument
(a2 a2-)                ; points to the second argument
(a3 a3-)                ; points to the third argument
(defined)
exec: 0.16 sec    gc: 0.00 sec


; Example 1
;
; Bob, Joe, and Harry are brothers.
;
* (desc                    ; desc produces a Lispish description.
*  (build rel      brothers
*         argument (Bob Joe Harry)))

(m1 (argument (Harry) (Joe) (Bob)) (rel (brothers)))
(dumped)
exec: 0.10 sec    gc: 0.00 sec

; If x and y are brothers, then x likes y.
;
* (desc
*  (build
*     avb (*x *y)                          ; avb is the universal quantifier.
*     ant (build rel      brothers         ; ant points to the antecedent.
*                 argument (*x *y))
*      cq (build rel likes a1 *x a2 *y)))  ; cq points to the consequent.
                                           ; actual variable nodes are v1, v2, etc.
(m4 (cq (m3 (a2 (v2)) (a1 (v1)) (rel (likes))))
    (ant (m2 (argument (v2) (v1)) (rel (brothers))))
    (avb (v2) (v1)))
(dumped)
exec: 0.31 sec    gc: 0.00 sec


; Save old nodes for a later, second run of the deduction.
;
* ((*nodes oldnodes) - oldnodes)

(m4 likes m3 m2 v2 y v1 x brothers Harry Joe Bob m1 oldnodes)
exec: 0.08 sec    gc: 0.00 sec

; First, use R1.
;
* (^(setq DuplBind t))
(t)
exec: 0.03 sec    gc: 0.00 sec
```

```
; Who likes whom?
;
* (desc (deduce rel likes a1 %x a2 %y))

(m5  (a2 (Harry)) (a1 (Harry)) (rel (likes))) ; Harry likes himself.
(m6  (a2 (Joe))   (a1 (Harry)) (rel (likes))) ; Harry likes Joe.
(m7  (a2 (Bob))   (a1 (Harry)) (rel (likes))) ; Harry likes Bob.
(m8  (a2 (Harry)) (a1 (Joe))   (rel (likes))) ; Joe likes Harry.
(m9  (a2 (Joe))   (a1 (Joe))   (rel (likes))) ; Joe likes himself.
(m10 (a2 (Bob))   (a1 (Joe))   (rel (likes))) ; Joe likes Bob.
(m11 (a2 (Harry)) (a1 (Bob))   (rel (likes))) ; Bob likes Harry.
(m12 (a2 (Joe))   (a1 (Bob))   (rel (likes))) ; Bob likes Joe.
(m13 (a2 (Bob))   (a1 (Bob))   (rel (likes))) ; Bob likes himself.
(dumped)                                      ; Note that each person likes himself.
exec: 3.83 sec    gc: 0.00 sec


; Now, clear the working storage used for this deduction.
;
* (clear-infer)
(cleared)
exec: 0.05 sec    gc: 0.00 sec


; Erase the nodes built by the first run of the deduction.
;
* (erase *nodes - *oldnodes)

(m13 (a2 (Bob))   (a1 (Bob))   (rel (likes)))
(m12 (a2 (Joe))   (a1 (Bob))   (rel (likes)))
(m11 (a2 (Harry)) (a1 (Bob))   (rel (likes)))
(m10 (a2 (Bob))   (a1 (Joe))   (rel (likes)))
(m9  (a2 (Joe))   (a1 (Joe))   (rel (likes)))
(m8  (a2 (Harry)) (a1 (Joe))   (rel (likes)))
(m7  (a2 (Bob))   (a1 (Harry)) (rel (likes)))
(m6  (a2 (Joe))   (a1 (Harry)) (rel (likes)))
(m5  (a2 (Harry)) (a1 (Harry)) (rel (likes)))
(nodes deleted)
exec: 0.20 sec    gc: 0.00 sec


; Change the setting of DuplBind to use UVBR.
;
* (^(setq DuplBind nil))
nil
exec: 0.05 sec    gc: 0.00 sec


; Now, who likes whom?
;
* (desc (deduce rel likes a1 %x a2 %y))

(m14 (a2 (Joe))   (a1 (Harry)) (rel (likes))) ; Harry likes Joe.
(m15 (a2 (Bob))   (a1 (Harry)) (rel (likes))) ; Harry likes Bob.
(m16 (a2 (Harry)) (a1 (Joe))   (rel (likes))) ; Joe likes Harry.
(m17 (a2 (Bob))   (a1 (Joe))   (rel (likes))) ; Joe likes Bob.
(m18 (a2 (Harry)) (a1 (Bob))   (rel (likes))) ; Bob likes Harry.
(m19 (a2 (Joe))   (a1 (Bob))   (rel (likes))) ; Bob likes Joe.
(dumped)                                      ; Note that now, no one likes himself.
exec: 2.50 sec    gc: 2.40 sec


; Again clear memory for the next example.
* (clear-infer)
(cleared)
exec: 0.05 sec    gc: 0.00 sec
;
* (erase *nodes - *oldnodes)

(m19 (a2 (Joe))   (a1 (Bob))   (rel (likes)))
(m18 (a2 (Harry)) (a1 (Bob))   (rel (likes)))
(m17 (a2 (Bob))   (a1 (Joe))   (rel (likes)))
(m16 (a2 (Harry)) (a1 (Joe))   (rel (likes)))
(m15 (a2 (Bob))   (a1 (Harry)) (rel (likes)))
(m14 (a2 (Joe))   (a1 (Harry)) (rel (likes)))
(nodes deleted)
exec: 0.13 sec    gc: 0.00 sec


; Example 2
;
; If a tract (of the central nervous system) is malfunctioning,
; adjacent tracts should be examined.
;
* (desc
*  (build avb  $x
*         @ant ((build object *x property Malfunctioning)
*               (build member *x class Tract))
*         cq   (build avb  $y
*                     @ant ((build rel Adjacent argument (*x *y))
*                           (build member *y class Tract))
*                     cq   (build rel Should a1 examine a2 *y))))

(m26 (cq
      (m25 (cq   (m24 (a2 (v4)) (a1 (examine)) (rel (Should))))
           (@ant (m23 (class (Tract))      (member (v4)))
                 (m22 (argument (v4) (v3)) (rel (Adjacent))))
           (avb  (v4))))
     (@ant (m21 (class (Tract))            (member (v3)))
           (m20 (property (Malfunctioning)) (object (v3))))
     (avb  (v3)))
(dumped)
exec: 0.63 sec    gc: 0.00 sec


; 16R is a tract.
;
* (desc (build member 16R class Tract))

(m27 (class (Tract)) (member (|16R|)))
(dumped)
exec: 0.10 sec    gc: 0.00 sec

; 13RC8-C2 is a tract.
;
* (desc (build member 13RC8-C2 class Tract))

(m28 (class (Tract)) (member (|13RC8-C2|)))
(dumped)
exec: 0.10 sec    gc: 0.00 sec


; They are adjacent.
;
* (desc (build rel Adjacent argument (16R 13RC8-C2)))

(m29 (argument (|13RC8-C2|) (|16R|)) (rel (Adjacent)))
(dumped)
exec: 0.10 sec    gc: 0.00 sec
```

```
; 16R is malfunctioning.
;
* (desc (build object 16R property Malfunctioning))

(m30 (property (Malfunctioning)) (object (|16R|)))
(dumped)
exec: 0.11 sec    gc: 0.00 sec

; Save the current nodes for use in the next example.
;
* (*nodes = oldnodes)
(m30 m29 13RC8-C2 m28 16R m27 m26 m25 Should examine m24 m23
     Adjacent m22 v4 Tract m21 Malfunctioning m20 v3 oldnodes
     m4 likes m3 m2 v2 y v1 x brothers Harry Joe Bob m1)
exec: 0.05 sec    gc: 0.00 sec

; Use R1.
;
* (^(setq DuplBind t))
(t)
exec: 0.05 sec    gc: 0.00 sec

; What should be examined?
;
* (desc (deduce rel Should a1 examine a2 %x))

(m31 (a2 (|16R|))      (a1 (examine)) (rel (Should))) ; Examine 16R.
(m32 (a2 (|13RC8-C2|)) (a1 (examine)) (rel (Should))) ; Examine 13RC8-C2.
(dumped) ; It is not necessary to examine a tract already known to be malfunctioning.
exec: 1.98 sec    gc: 0.00 sec

; Clear storage for next run.
;
* (clear-infer)
(cleared)
exec: 0.03 sec    gc: 0.00 sec

* (erase *nodes - *oldnodes)
(m32 (a2 (|13RC8-C2|)) (a1 (examine)) (rel (Should)))
(m31 (a2 (|16R|))      (a1 (examine)) (rel (Should)))
(nodes deleted)
exec: 0.20 sec    gc: 0.00 sec

; Use UVBR.
;
* (^(setq DuplBind nil))
nil
exec: 0.01 sec    gc: 0.00 sec

; Again, what should be examined?
;
* (desc (deduce rel Should a1 examine a2 %x))

(m33 (a2 (|13RC8-C2|)) (a1 (examine)) (rel (Should))) ; Examine 13RC8-C2.
(dumped)                          ; Now we weren't told to examine 16R.
exec: 1.80 sec    gc: 0.00 sec

; Clear storage again.
;
* (clear-infer)
(cleared)
exec: 0.03 sec    gc: 0.00 sec

; Example 3
;
; If an AND gate has high inputs and a low output, it is faulty.
;
* (desc (build avb  ($x $y $z $w)
*              &ant ((build member *x class AND-gate)
*                    (build rel Gate a1 *x a2 (*y *z) a3 *w))
*              cq   (build &ant ((build object *y property High)
*                                (build object *z property High)
*                                (build object *w property Low))
*                         cq   (build object *x property Faulty))))

(m41 (cq
      (m40 (cq   (m39 (property (Faulty)) (object (v5))))
           (&ant (m38 (property (Low))    (object (v8)))
                 (m37 (property (High))   (object (v7)))
                 (m36 (property (High))   (object (v6))))))
     (&ant (m35 (a3 (v8)) (a2 (v7) (v6)) (a1 (v5)) (rel (Gate)))
           (m34 (class (AND-gate)) (member (v5))))
     (avb  (v8) (v7) (v6) (v5)))
(dumped)
exec: 0.81 sec    gc: 0.00 sec

; A1 is an AND gate.
;
* (desc (build member A1 class AND-gate))

(m42 (class (AND-gate)) (member (A1)))
(dumped)
exec: 0.08 sec    gc: 0.00 sec

; A1's inputs are Alinp1 and Alinp2, and its output is Aloutp.
;
* (desc (build rel Gate a1 A1 a2 (Alinp1 Alinp2) a3 Aloutp))

(m43 (a3 (Aloutp)) (a2 (Alinp2) (Alinp1)) (a1 (A1)) (rel (Gate)))
(dumped)
exec: 0.13 sec    gc: 0.00 sec

; Alinp1 is high.
;
* (desc (build object Alinp1 property High))

(m44 (property (High)) (object (Alinp1)))
(dumped)
exec: 0.10 sec    gc: 0.00 sec

; Alinp2 is low.
;
* (desc (build object Alinp2 property Low))

(m45 (property (Low)) (object (Alinp2)))
(dumped)
exec: 0.11 sec    gc: 0.00 sec

; Aloupt is low.
;
* (desc (build object Aloutp property Low))

(m46 (property (Low)) (object (Aloutp)))
(dumped)
exec: 0.11 sec    gc: 0.00 sec
```

```
; Save current nodes
* (*nodes = oldnodes)

(m46 m45 m44 Alinp2 Alinp1 Aloutp m43 A1 m42 m41 m40
     Faulty m39 Low m38 m37 High m36 Gate m35 AND-gate
     m34 v8 w v7 z v6 v5 m33 m30 m29 13RC8-C2 m28 16R
     m27 m26 m25 Should examine m24 m23 Adjacent m22 v4
     Tract m21 Malfunctioning m20 v3 oldnodes m4 likes
     m3 m2 v2 y v1 x brothers Harry Joe Bob m1)
exec: 0.06 sec    gc: 0.00 sec

; Use R1.
;
* (^(setq DuplBind t))
(t)
exec: 0.03 sec    gc: 0.00 sec

; What is faulty?
;
* (desc (deduce object %x property Faulty))

(m47 (property (Faulty)) (object (A1))) ; A1 is faulty.
(dumped)                                ; because Alinp1 was counted as both inputs.
exec: 3.35 sec    gc: 0.00 sec

; Clear storage.
;
* (clear-infer)
(cleared)
exec: 0.03 sec    gc: 0.00 sec
;
* (erase *nodes - *oldnodes)

(m47 (property (Faulty)) (object (A1)))
(nodes deleted)
exec: 0.33 sec    gc: 0.00 sec

; Use UVBR.
;
* (^(setq DuplBind nil))
nil
exec: 0.05 sec    gc: 0.00 sec

; Now, what is faulty?
;
* (desc (deduce object %x property Faulty))

(dumped)            ; No answer means nothing was found.
exec: 2.86 sec    gc: 0.00 sec

; Clear storage.
;
* (clear-infer)
(cleared)
exec: 0.03 sec    gc: 0.00 sec

; Example 4
;
; Clyde is an elephant.
;
* (desc (build member Clyde class elephant))

(m48 (class (elephant)) (member (Clyde)))
(dumped)
exec: 0.15 sec    gc: 0.00 sec

; Jumbo is an elephant.
;
* (desc (build member Jumbo class elephant))

(m49 (class (elephant)) (member (Jumbo)))
(dumped)
exec: 0.11 sec    gc: 0.00 sec

; Elephants hate each other.
;
* (desc (build avb  ($x $y)
*              @ant ((build member *x class elephant)
*                    (build member *y class elephant))
*              cq   (build rel hates a1 *x a2 *y)))

(m53 (cq (m52 (a2 (v10)) (a1 (v9)) (rel (hates))))
     (@ant (m51 (class (elephant)) (member (v10)))
           (m50 (class (elephant)) (member (v9))))
     (avb (v10) (v9)))
(dumped)
exec: 0.46 sec    gc: 0.00 sec

; Save current nodes.
;
* (*nodes = oldnodes)
(m53 hates m52 m51 m50 v10 v9 Jumbo m49 Clyde elephant
     m48 m46 m45 m44 Alinp2 Alinp1 Aloutp m43 A1 m42 m41
     m40 Faulty m39 Low m38 m37 High m36 Gate m35 AND-gate
     m34 v8 w v7 z v6 v5 m33 m30 m29 13RC8-C2 m28 16R m27
     m26 m25 Should examine m24 m23 Adjacent m22 v4 Tract
     m21 Malfunctioning m20 v3 oldnodes m4 likes m3 m2 v2
     y v1 x brothers Harry Joe Bob m1)
exec: 0.05 sec    gc: 0.00 sec

; Use R1.
;
* (^(setq DuplBind t))
(t)
exec: 0.03 sec    gc: 0.00 sec

; Who hates whom?
;
* (desc (deduce rel hates a1 %x a2 %y))

(m54 (a2 (Clyde)) (a1 (Clyde)) (rel (hates))) ; Clyde hates himself.
(m55 (a2 (Jumbo)) (a1 (Clyde)) (rel (hates))) ; Clyde hates Jumbo.
(m56 (a2 (Clyde)) (a1 (Jumbo)) (rel (hates))) ; Jumbo hates Clyde.
(m57 (a2 (Jumbo)) (a1 (Jumbo)) (rel (hates))) ; Jumbo hates himself.
(dumped)
exec: 2.68 sec    gc: 0.00 sec

; Clear storage.
;
* (clear-infer)
(cleared)
exec: 0.05 sec    gc: 0.00 sec
;
* (erase *nodes - *oldnodes)

(m57 (a2 (Jumbo)) (a1 (Jumbo)) (rel (hates)))
(m56 (a2 (Clyde)) (a1 (Jumbo)) (rel (hates)))
(m55 (a2 (Jumbo)) (a1 (Clyde)) (rel (hates)))
(m54 (a2 (Clyde)) (a1 (Clyde)) (rel (hates)))
(nodes deleted)
```

```
exec: 0.51 sec    gc: 0.00 sec

; Use UVBR.
;
* (^(setq DuplBind nil))
nil
exec: 0.03 sec    gc: 0.00 sec
; And now, who hates whom?
;
* (desc (deduce rel hates a1 %x a2 %y))

(m58 (a2 (Jumbo)) (a1 (Clyde)) (rel (hates))) ; Clyde hates Jumbo.
(m59 (a2 (Clyde)) (a1 (Jumbo)) (rel (hates))) ; Jumbo hates Clyde.
(dumped)                                      ; Now, no one hates himself.
exec: 1.90 sec    gc: 0.00 sec
```


Acknowledgment

The ideas presented in this paper have been clarified by discussions with W. Rapaport and other members of the SNePS Research Group of SUNY at Buffalo. Examples showing the need for **R2** and the **UVBR** came from the work of Z. Xiang, J. Geller, M. R. Taie, and others.

References

[1] R. J. Brachman, "What's in a concept: structural foundations for semantic networks," *Int. J. Man-Machine Studies*, vol. 9, pp. 127–152, 1977.

[2] S. Fahlman, *NETL: A System for Representing and Using Real-World Knowledge*. Cambridge, MA: MIT Press, 1979.

[3] N. V. Findler, Ed., *Associative Networks: The Representation and Use of Knowledge by Computers*. New York: Academic Press, 1979.

[4] K. A. Fine, "A defense of arbitrary objects," *Proc. Aristotelian Soc.* (suppl.), vol. 58, pp. 55–77, 1983.

[5] D. J. Israel, "Interpreting network formalisms," in *Computational Linguistics*, N. Cercone, Ed. Oxford, UK: Pergamon, 1983, pp. 1–13.

[6] A. S. Maida and S. C. Shapiro, "Intensional concepts in propositional semantic networks," *Cogn. Sci.*, vol. 6, pp. 291–330, Oct.–Dec. 1982. Reprinted in *Readings in Knowledge Representation*, R. J. Brachman and H. J. Levesque, Eds. Los Altos, CA: Kaufman, 1985, pp. 169–189.

[7] G. McCalla and N. Cercone, Eds., "Special Issue on Knowledge Representation," *Computer*, vol. 16, pp. 12–123, Oct. 1983.

[8] D. P. McKay and S. C. Shapiro, "Using active connection graphs for reasoning with recursive rules," in *Proc. 7th Int. Joint Conf. on Artificial Intelligence*, pp. 368–374, 1981.

[9] W. J. Rapaport, "Belief representation and quasi-indicators," Dep. Comput. Sci., SUNY at Buffalo, Tech. Rep. 215, 1984.

[10] W. J. Rapaport and S. C. Shapiro, "Quasi-indexical reference in propositional semantic networks," in *Proc. Coling-84*, pp. 65–70, 1984.

[11] S. C. Shapiro, "The SNePS semantic network processing system," in *Associative Networks: The Representation and Use of Knowledge by Computers*, N. V. Findler, Ed. New York: Academic Press, 1979, pp. 179–203.

[12] ——, "What do semantic network nodes represent?" Dep. Comput. Sci., SUNY at Buffalo, SNeRG Tech. Note 7, 1981.

[13] S. C. Shapiro and D. P. McKay, "Inference with recursive rules," in *Proc. 1st Annu. Nat. Conf. on Artificial Intelligence*, pp. 151–153, 1980.

[14] L. Talmy, "Rubber-sheet cognition in language," in *Papers from the 13th Regional Meet.*, Chicago Linguistic Society, 1977.

[15] W. A. Woods, "What's in a link: Foundations for semantic networks," in *Representation and Understanding: Studies in Cognitive Science*, D. G. Brobow and A. M. Collins, Eds. New York: Academic Press, 1975, pp. 35–82.

[16] Z. Xiang, S. N. Srihari, S. C. Shapiro, and J. G. Chutkow, "Analogical and propositional representation of structure in neurological diagnosis," in *Proc. 1st Conf. on Artificial Intelligence Applications*, pp. 127–132, 1984.