MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

Working Paper 222 November 1981

Representing Constraint Systems with Omega

Phyllis E. Koton

Abstract

This paper considers two constraint systems, that of Steele and Sussman, and Alan Borning's Thinglab. Some functional difficulties in these systems are discussed. A representation of constraint systems using the description system Omega is presented which is free of these difficulties.

Representing Constraint Systems with Omega

I. Introduction

The process of deriving consequences from a set of constraints is useful in many applications, and is an easy concept to understand. Constraint systems are becoming increasingly popular, although they were designed as far back as the early sixties, for example Sketchpad [Sutherland 1963] and SIR [Raphael 1964]. More recent systems include Thinglab [Borning 1979] and the constraint system described by Steele & Sussman [Steele & Sussman 1978]. This paper will attempt to explore the relationships between constraint systems and description systems such as Omega [Hewitt 1980].

II. A Brief Overview of Omega

Omega is used to build and reason about a network of descriptions. An Omega statement is made up of descriptions, such as the following:

(an office)

which describes an office (or any office). The inheritance relation "is" is used to relate descriptions, for example,

(Room-812 is (an office))

Descriptions can be made more specific by giving them attributes.

Attributes are of the form:

(attribution =Relation attr-description)

for instance:

(Room-812 is (an office (with floor-number 8)
(with address 545-Tech-Square)))

Omega has four types of attributions which differ in the extent to which they restrict the objects which can fill the role of attr-description. These are, in order of increasing restrictiveness: of, with, must-be*, and with-unique.

The "of" attribution places no restriction on the attr-description. It merely indicates that the attribute "=Relation" exists for the description. For example, a sum of two numbers could be described as

(a sum (of arg1 (a number)) (of arg2 (a number)))

The "of" attribution obeys the Axioms of Monotonicity, Commutativity, Omission, and Strictness.**

The "with" attribution is only slightly stronger. It obeys the Axiom of Merging in addition to the axioms obeyed by the "of" attribution. The Axiom of Merging states that attributions of the same description can be merged.***

The must-be attribution restricts every description which satisfies the relation =Relation to be of the class attr-description.*+ It obeys

---

* Must-be is also called with-every.

** The axioms are given in Appendix A

*** An explanation of why this should not hold for the "of" attribution is given in Appendix A.

*+ Formal definitions of the must-be and with-unique attributions are given in Appendix A.

the same axioms as the "with" attribution. For instance,

(John is (a father (must-be child (a male)))

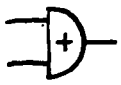
states that all of John's children are males.

Finally, the with-unique attribution restricts there to be exactly one description which satisfies the relation =Relation. It obeys the same axioms as the "with" attribution. For example,

(2 is (a complex-number (with-unique real-part 0)))

III. Representing Constraints in Omega

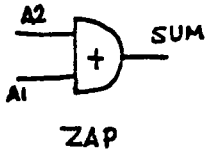
Consider the following example, taken from [Steele & Sussman 1978]. An adder is described as a primitive constraint on three numbers, such that the sum is constrained to be the addend plus the augend, and the addend is constrained to be the difference between the sum and the augend.



The same constraints can be represented in Omega as one statement, which will be called the sum/difference rule:

```
(((a sum (of arg1 a) (of arg2 b)) same c) <==>
 ((a difference (of arg1 c) (of arg2 a)) same b))*
```

Steele & Sussman create an adder, "zap" which has three parts, a1, a2, and s.



---

* Two descriptions d1 and d2 are said to be the same if (d2 is d1) and (d1 is d2).

The same representation in Omega is:

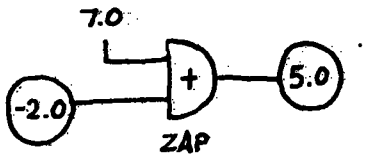
(vp-1 is (an arithmetic-viewpoint))*

((sum-zap same (a sum (of arg1 a1-zap)
(of arg2 a2-zap))) in vp-1)

((a1-zap same (a difference (of arg1 sum-zap)
(of arg2 a2-zap))) in vp-1)

((a2-zap same (a difference (of arg1 sum-zap)
(of arg2 a1-zap))) in vp-1)**

Now, give the sum and a1 of zap a value.



In Omega,

1. ((sum-zap same 5.0) in vp-1)

2. ((a1-zap same -2.0) in vp-1)

The following descriptions can be derived by the system:

3. ((5.0 same (a sum (of arg1 a1-zap) (of arg2 a2-zap))) in vp-1)

[substitute (1)]

4. ((5.0 same (a sum (of arg1 -2.0) (of arg2 a2-zap))) in vp-1)

[substitute (2)]

5. ((a2-zap same (a difference (of arg1 5.0) (of arg2 -2.0))) in vp-1)

[by the sum/difference rule]

And the system establishes the following relationship:

((a2-zap same 7.0) in vp-1)

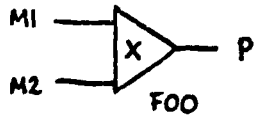
---

* A viewpoint is an Omega mechanism for dealing with change. It will be discussed in a later section.

** Any one of the three descriptions above is sufficient to specify zap (since the others can be derived using the sum/difference rule). All three were given for the purpose of clarity.

IV. Building Compound Constraints

Steele & Sussman next introduce a multiplier constraint, and create an instance of it called foo.



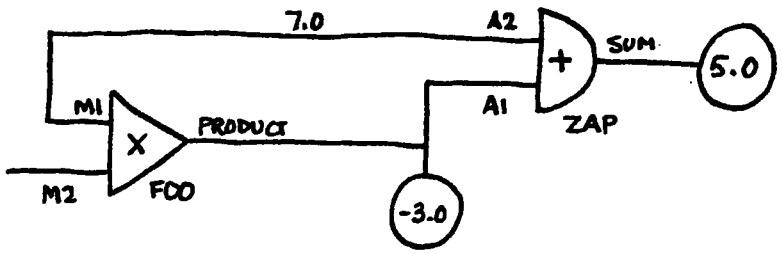
To represent this in Omega, a new rule, called the product/quotient rule is introduced:

```
(((a product (of arg1 a) (of arg2 b)) same c) <==>
 ((a quotient (of arg1 c) (of arg2 a)) same b))
```

Foo can be specified as:

```
(p-foo is (a product (of arg1 m1-foo)
                     (of arg2 m2-foo)))
```

In both Steele & Sussman's system and in Thinglab, a compound constraint can be built out of simpler constraints by joining some of their parts.



The same is true of Omega.

6. ((a2-zap same m1-foo) in vp-1)

7. ((p-foo same a1-zap) in vp-1)

Using constraint information and the values already given, the following information is derived by the system:

8. ((m2-foo same (a quotient (of arg1 p-foo) [product/quotient rule]
(of arg2 m1-foo)) in vp-1)

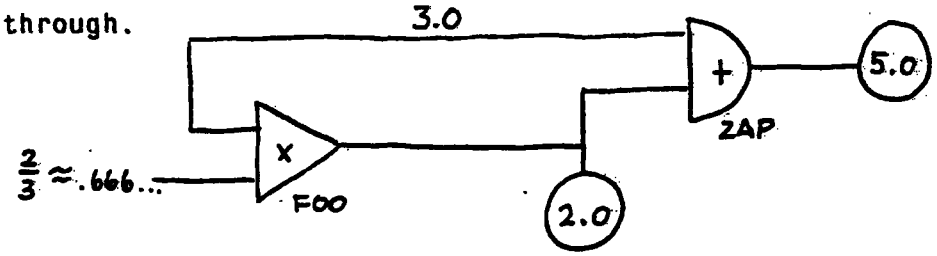
9. ((m2-foo same (a quotient (of arg1 a1-zap)
(of arg2 a2-zap))) in vp-1) [substitute (7)]

10. ((m2-foo same (a quotient (of arg1 -2.0)
(of arg2 a2-zap))) in vp-1) [substitute (2)]

```
11.     ((m2-foo same (a quotient (of arg1 -2.0)
                                  (of arg2 (a difference
                                             (of arg1 sum-zap)
                                             (of arg2 a1-zap)))))
                                                   in vp-1)
                                        [sum/difference rule on a2-zap in (10)]

12.     ((m2-foo same (a quotient (of arg1 -2.0)
                                  (of arg2 (a difference
                                             (of arg1 5.0)
                                             (of arg2 -2.0)))))
                                                   in vp-1)
                                                   [substitute (1) and (2)]

13.     ((m2-foo same (a quotient (of arg1 -2.0)
                                  (of arg2 7.0))) in vp-1)

        ((m2-foo same -0.28) in vp-1)
```

V. Changing Information and Viewpoints

In Steele & Sussman's system, when a premise is changed, all information which depended on it is withdrawn and new conclusions are reached. Their system maintains the constraint dependencies on which their conclusions are based. It cannot, however, remember the previous values of changed parameters. For example, to change the a1 of zap to 2.0, the new value is inserted in the system and changes are propagated through.



Omega does not permit changing a description. Information can be added that specifies a description in more detail, but if we want to change some information a new viewpoint must be created that has the new information in the appropriate field. Other information that was not dependent on the changed information is inherited to the new viewpoint.

```
(vp-2 is (an arithmetic viewpoint
                (with previous-viewpoint vp-1)
                (with modification
                       (an adder-arg-modification
                          (with adder zap)
                          (with arg a1)
                          (with new-description 2.0)))))
```

Since viewpoints are descriptions we can explicitly describe the changes between vp-2 and vp-1 in creating the description of vp-2. This helps the system determine what information should be inherited by a new viewpoint and why, in addition to providing a record of changes in the database [Barber 1981, p.7].

The description of zap in the new viewpoint contains the new information along with information inherited from the old viewpoint.

```
((a2-zap is (a difference (of arg1 5.0))
                          (of arg2 2.0)) in vp-2)
```

The following information can be derived by the system:

```
((a2-zap same 3.0) in vp-2)

((m2-foo same 0.66) in vp-2)
```

There are several differences between the viewpoint mechanism and the approaches to changed information used by Steele & Sussman or Borning. First, the viewpoint mechanism allows Omega to have a complete record of all changes made to the system. This includes dependency information as well as previous values of objects. Although saving the information takes up space, it is very useful to have a record of changes for explaining why changes took place. In some applications, a complete record of values is essential (for example, if the system is being used to model a financial portfolio). Steele & Sussman's system retains only dependency information, but no history of previous values. Thinglab does not even maintain dependency information.

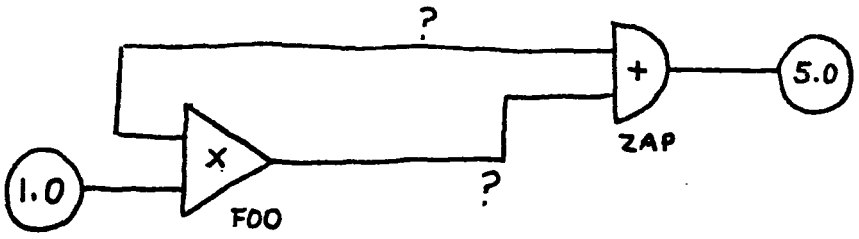
A second difference is that when changes are made in Thinglab and Steele & Sussman's system, both old and new information exists in the database while incremental deductions are made. Some of the old information is consistent with the new information and some is inconsistent. At any given time, the information in the system might be inconsistent [Barber 1981, p.8] because of the incremental nature of the deductions. Omega maintains information consistency by keeping old and new information in separate viewpoints.

Viewpoints also allow the system to reason about proposed changes. For example, to compare the results of several potential changes, we could create a new viewpoint for each proposed change and investigate its consequences, without altering our current version of the description.

At this point the reader might wonder what is the difference between viewpoints and situational variables [McCarthy 1969]. One difference is that systems which use situational variables have no inheritance mechanisms [Hewitt, personal communication]. Furthermore, in Omega viewpoints are descriptions and thus may be described explicitly [Barber 1981, p.4].

VI. Dealing with Circularities

Now consider a simple problem which cannot be solved by local constraint propagation in either Steele and Sussman's system or in Thinglab.



Although there are enough "knowns" in the system to solve the problem, neither the adder nor the multiplier alone has enough information to make any deductions. There are two factors which prevent Thinglab and Steele & Sussman's system from solving this network. First, their systems are dependent on having a direction of computational flow in the network. The presence of the loop in the constraints prevents the system from finding this flow [Steele & Sussman p. 24]. Second, their systems are unable to make use of abstract information present in the network (for example, that the product of any number and one is the number), since they can only propagate concrete values.

The two systems had to expand their basic constraint propagation schemes to handle circularities. Thinglab solves problems of this type using a technique called "relaxation," which makes successive approximations of the values until all constraints are satisfied. Steele & Sussman employ multiple, redundant descriptions to re-organize the information in such a way that the loops are bypassed.

Omega can use abstract information such as the rule of multiplicative identity which is embedded in the system. This rule gives Omega a

more global knowledge. In addition to knowing the value at a1-zap, it knows the relation of the value at a1-zap to the value at m1-foo. This allows Omega to derive the solution to the above problem using only its rules of inference and some rules of arithmetic expressed as Omega descriptions.

The network, expressed in Omega, is:

1. ((sum-zap same (a sum (of arg1 a1-zap) (of arg2 a2-zap))) in vp-3)

2. ((p-foo same (a product (of arg1 m1-foo) (of arg2 m2-foo))) in vp-3)

3. ((p-foo same a1-zap) in vp-3)

4. ((a2-zap same m1-foo) in vp-3)

5. ((sum-zap same 5.0) in vp-3)

6. ((m2-foo same 1.0) in vp-3)

Using the sum/difference rule, the product/quotient rule, and its rules of inference, the system can derive the following descriptions:

7. ((p-foo same (a product (of arg1 a2-zap)
(of arg2 1.0))) in vp-3)

[substitute (4) and (6) in (2)]

8. ((p-foo same a2-zap) in vp-3) [multiplicative identity rule]

9. ((sum-zap same (a sum (of arg1 p-foo)
(of arg2 a2-zap))) in vp-3)

[substitute (3) in (1)]

10. ((sum-zap same (a sum (of arg1 a2-zap)
(of arg2 a2-zap))) in vp-3) [by (8)]

11. ((sum-zap same (a product (of arg1 2) (of arg2 a2-zap))) in vp-3)

[A + A rule]

12. ((5.0 same (a product (of arg1 2) (with arg2 a2-zap))) in vp-3)

[substitute (5)]

13. ((a2-zap same (a quotient (with arg1 5) (with arg2 2))) in vp-3)
[product/quotient rule]

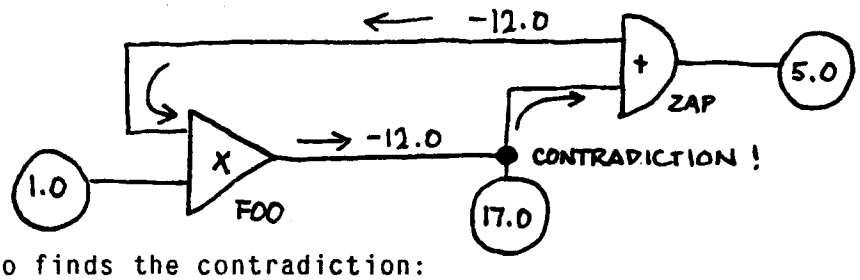
14. ((a2-zap same 5/2) in vp-3)

15. ((p-foo same 5/2) in vp-3) [by (8)]

16. ((a1-zap same 5/2) in vp-3) [by (3)]

VI. Contradictions

A critical property of constraint systems is the ability to recognize contradictions. Sussman & Steele give the example of setting the product of foo to 17.0 in the previous network:



Omega also finds the contradiction:

```
(vp-4 is (an arithmetic-viewpoint
                (with previous-viewpoint vp-3)
                (with modification
                        (an adder-arg-modification
                           (with adder zap)
                           (with arg a1)
                           (with new-value 17.0)))))
```

1. ((a1-zap same 17.0) in vp-4)

2. ((sum-zap same 5.0) in vp-4)

3. ((m1-foo same 1.0) in vp-4)

4. ((sum-zap same (a sum (with arg1 a1-zap) (with arg2 a2-zap))) in vp-4)

5. ((a1-zap same (a product (with arg1 m1-foo)(with arg2 a2-zap))) in vp-4)

6. ((sum-zap same (a sum (with arg1 17.0) (with arg2 a2-zap))) in vp-4)

7. ((5.0 same (a sum (with arg1 17.0) (with arg2 a2-zap))) in vp-4)

8. ((a2-zap same (a difference (with arg1 5) (with arg2 17.0))) in vp-4)

9. ((a2-zap same -12.0) in vp-4)

10. ((a1-zap same (a product (with arg1 1) (with arg2 a2-zap))) in vp-4)

11. ((17.0 same (a product (with arg1 1) (with arg2 a2-zap))) in vp-4)

12. ((a2-zap same (a quotient (with arg1 17.0) (with arg2 1))) in vp-4)

13. ((a2-zap same 17.0) in vp-4)

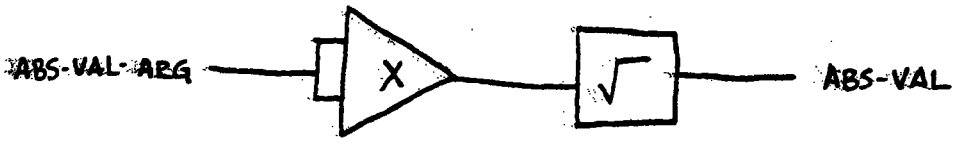
14. ((a2-zap in vp-4) same 17.0)

15. ((a2-zap in vp-4) same -12.0)

16. ((17.0 same -12.0) in vp-4) contradiction

VIII. Antecedent and Consequent Reasoning

Omega uses both antecedent and consequent reasoning to solve problems. Given a statement of the form (a ==> b), antecedent reasoning says "if a is asserted, then assert b". Consequent reasoning says "if goal b, then establish subgoal a." Steele & Sussman's system and Thinglab only use antecedent reasoning. This reduces the number of different types of problems that they can solve. For example, "absolute value" is a constraint between two numbers easily represented in any of the three systems.



In Omega:

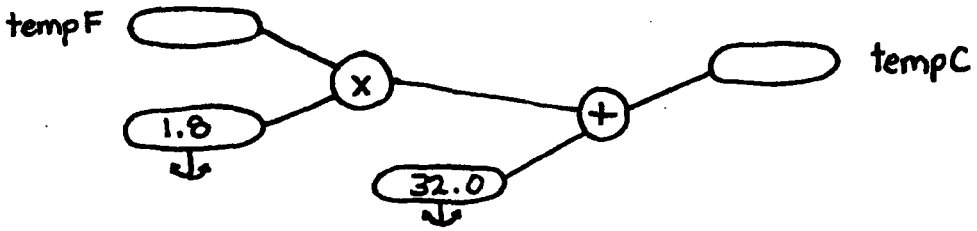
```
((an abs-val (of arg abs-val-arg)) same
 (a sqrt (of arg1 (a product (of arg2 abs-val-arg)
                             (of arg2 abs-val-arg)))))

((an abs-val-arg (of arg x)) is (or x (minus x)))
```

If we present a number, say 4. as arg1 of the absolute value

constraint, all systems will determine that 4 is the value that should be assigned to abs-val. If we change arg1 to -4, the system again can determine abs-val. If, however, we present 4 as the value of abs-val, and ask the system for the value of arg1, the problem becomes less trivial. Thinglab and Steele & Sussman's system, if they came up with any answer at all, would probably report that arg1 is 4 (by propagating known values backwards through the sqrt and squarer constraints). This is only part of the answer; furthermore it is possible that we intended the value of arg1 to be negative. Omega will return the correct solution, (or (-4) 4).

IX. Reasoning about Procedures

Steele & Sussman's system and Thinglab can execute procedures implicitly by propagating constraints, however they have only limited ability to reason about the procedures they use. Consider the following example from Thinglab, a farenheit to centigrade temperature converter.



Thinglab and Steele & Sussman's system can provide information such as "what is the centigrade temp if the farenheit temp is 32 degrees?" but cannot answer questions such as "if the farenheit temperature increases, does the centigrade temperature increase?" Omega can reason about procedures explicitly, using rules such as:

((tempF in vp-1) > (tempF in vp-0)) ==> ((tempC in vp-1) > (tempC in vp-0))

In other words, if the farenheit temperature at 2:00 pm is greater than the

farenheit temperature at 3:00 pm, the system can deduce that the centigrade tempertaure has also increased.

X. Conclusions

All representation languages are not created equal. Omega has certain features which makes it more powerful for dealing with constraint systems than some other languages. This is not to imply, however, that Omega is the only knowledge representation language suitable for this task. FRL [Goldstein 77] and KRL [Fikes 80] are two other languages which have many features useful for this application. A comparison of the relative powers of Omega, KRL, and FRL for dealing with constraint systems is a topic for further research.

Functionally, Omega has advantages over Sussman & Steele's system and Thinglab. This paper gives an example of a problem that Omega can solve that the other systems cannot without stepping outside their basic propagation mechanisms. Omega uses both antecedent and consequent reasoning, while the other systems are restricted to antecedent reasoning. The viewpoint mechanism allows Omega to reason explicitly about changing processes, while the other systems can only execute processes by propagating constraints.

XI. Acknowledgments

Many of the topics presented in this paper were pointed out to me by Carl Hewitt, Gerald Barber, and Giuseppe Attardi. Steele & Sussman's paper and Borning's work were very useful for pointing out the issues involved in constraint networks.

Appendix A

Basic Omega Axioms

Some of the following is taken from [Hewitt 80].

Extensionality

```
((=description1 is =description2) <==>
 (for-all =d ((=d is =description1) ==> (=d is =description2))))
```

from which can be derived

Reflexivity of Inheritance

```
(=description is =description)
```

Transitivity of Inheritance

```
(((=description1 is =description2) ∧
  (=description2 is =description3)) ==>
  (=description1 is =description3))
```

Commutativity

Commutativity says that the order in which attributions are listed are irrelevant.

```
        ((a =description1
               <<=attributions1>>
               =attribution2
               <<=attributions3>>
               =attribution4
               <<=attributions5>>)
   same  (a =description1
               <<=attributions1>>
               =attribution4
               <<=attributions3>>
               =attribution2
               <<=attributions5>>))
```

For example,

```
((a class (with lecturer KRD) (with lecturer PSZ)) same
 (a class (with lecturer PSZ) (with lecturer KRD)))
```

Omission

Omission says that attributions of a description can be deleted to form a more general description.

```
((a =description1
        <<attributions1>>
        =attribution2
        <<attributions3>>) is
  (a =description1
        <<=attributions1>>
        <<=attribution3>>))
```

For example,

```
((a class (with lecturer KRD) (with lecturer PSZ)) is
 (a class (with lecturer KRD)))
```

Merging

The Axiom of Merging says that attributions of the same concept can be merged.

```
(((=description1 is (a =description2 <<=attributions1>>)) ^
  (=description1 is (a =description2 <<=attributions2>>)))
   ==>
    (=description1 is (a =description2 <<=attributions1>>
                                       <<=attributions2>>)))
```

For example, if

```
(6.891 is (a class (with lecturer KRD)))
(6.891 is (a class (with lecturer PSZ)))
(6.891 is (a class (with lecturer HES)))
```

then

```
(6.891 is (a class (with lecturer KRD)
                   (with lecturer PSZ)
                   (with lecturer HES)))
```

The Axiom of Merging does not hold for the "of" attribution. For example, if x=7 we could say of x:

```
(x is (a sum (of arg1 2) (of arg2 3)))
(x is (a sum (of arg1 4) (of arg2 1)))
```

Then by the axiom of deletion, we could deduce the following:

```
(x is (a sum (of arg1 2)))
(x is (a sum (of arg2 1)))
```

But from the above, we would not want to be able to conclude that

```
(x is (a sum (of arg1 2) (of arg2 1))) !
```

Monotonicity of Attributes

```
((=description1 is =description2) ==>
 ((a =concept (with =attribute =description1))
 is
  (a =concept (with =attribute =description2))))
```

For example, if

```
(PSZ is (a knowledge-base-expert))
(6.891 is  (a class (with lecturer PSZ)))
```

then

```
(6.891 is (a class (with lecturer (a knowledge-base-expert))))
```

Strictness

The Axiom of Strictness serves to get rid of garbage in the description system.

```
((a Concept (of Relation nothing)) is nothing)
```

Fusion of Attributes

if

```
(x is (a Concept (with Rel description1))) ∧
(x is (a Concept (must-be Rel description2))
```

then

```
(x is (a Concept (with Rel (and description1 description2))))
```

Fusion, cont'd.

```
if      (x is (a Concept (with Rel description1))) ∧
        (x is (a Concept (must-be Rel description2))
then
         (x is (a Concept (with-unique Rel (and description1 description2))))
```

Distributivity Axiom for Viewpoints

```
        (((=description1 is =description2) in vp)
        <==>
         ((=description1 in vp) is (=description2 in vp)))
```

This axioms states that an inheritance relation holds in a viewpoint iff the descriptions are relativized to that viewpoint.

Appendix B

Axioms of Arithmetic

These axioms are used in the examples above. Some of these axioms were taken from a 6.036 problem set.

Sum-Difference Rule

```
(((a sum (with arg1 =b)
         (with arg2 =c)) same =a)
<==>
 ((a difference  (with arg1 =a)
                 (with arg2 =b)) same =c))
```

Product-Quotient Rule

```
(((a product (with arg1 =b)
             (with arg2 =c)) same =a)
<==>
 ((a quotient (with top =a)
              (with bottom =b)) same =c))
```

Rule of Multiplicative Identity

```
((a product (with arg1 =a)
            (with arg2  1)) same =a)
```

A + A Rule

```
((a sum (with arg1 =a)
        (with arg2 =a)) same
 (a product (with arg1 =a)
            (with arg2  2)))
```

References

[Barber 1981] Barber, Gerald. "Reasoning About Change in Knowledgeable Office Systems (draft)." (April 1981).

[Borning 1979] Borning, Alan. "Thinglab -- A Constraint-Oriented Simulation Laboratory." Ph.D. Thesis. Stanford U. (March 1979).

[Hewitt 1980] Hewitt, Carl, Giuseppe Attardi, and Maria Simi. "Knowledge Embedding in the Description System Omega" in Proceedings of the First Annual National Conference on Artificial Intelligence. Stanford U. (August, 1980).

[McCarthy 1969] McCarthy, J. and P.J. Hayes. "Some Philosophical Problems from the Standpoint of Artificial Intelligence." Machine Intelligence 4, pp.463-502. (Edinburgh University Press, 1969).

[Raphael 1964] Raphael, Bertram. "SIR: A Computer Program for Semantic Information Retrieval." Ph.D. Thesis. MIT (Cambridge, 1964).

[Steele & Sussman 1978] Steele, Guy Lewis, Jr. and Gerald Jay Sussman. "Constraints." MIT AI Lab Memo 502 (Cambridge, 1978).

[Sutherland 1963] Sutherland, Ivan E. "Sketchpad: A Man-Machine Graphical Communication System." Ph.D. Thesis. MIT (Cambridge, 1963).