The Structure-Mapping Engine: Algorithm and Examples

Brian Falkenhainer
Qualitative Reasoning Group
Department of Computer Science

Kenneth D. Forbus
Qualitative Reasoning Group
Department of Computer Science

Dedre Gentner
Psychology Department

Abstract

This paper describes the *Structure-Mapping Engine* (SME), a program for studying analogical processing. SME has been built to explore Gentner's *Structure-mapping theory* of analogy, and provides a "tool kit" for constructing matching algorithms consistent with this theory. Its flexibility enhances cognitive simulation studies by simplifying experimentation. Furthermore, SME is very efficient, making it a useful component in machine learning systems as well. We review the Structure-mapping theory and describe the design of the engine. We analyze the complexity of the algorithm, and demonstrate that most of the steps are polynomial, typically bounded by $\mathcal{O}(N^2)$. Next we demonstrate some examples of its operation taken from our cognitive simulation studies and work in machine learning. Finally, we compare SME to other analogy programs and discuss several areas for future work.

This paper appeared in
Artificial Intelligence, **41**, 1989, pp 1-63.
For more information, please contact
forbus@ils.nwu.edu

1 Introduction

In analogy, a given situation is understood by comparison with another similar situation. Analogy may be used to guide reasoning, to generate conjectures about an unfamiliar domain, or to generalize several experiences into an abstract schema. Consequently, analogy is of great interest to both cognitive psychologists and artificial intelligence researchers. Psychologists wish to clarify the mechanisms underlying analogy in order to understand human learning and reasoning. Artificial Intelligence researchers wish to emulate analogical processing on computers to produce more flexible reasoning and learning systems.

This paper describes the *Structure-Mapping Engine* (SME), a program built to explore the computational aspects of Gentner's *Structure-mapping theory* of analogical processing [27,29]. SME has been used both as a cognitive simulation of human analogical processing and as a component in a larger machine learning system.

SME is both flexible and efficient. It constructs all consistent ways to interpret a potential analogy and does so without backtracking. SME provides a "tool kit" for building matchers satisfying the structural consistency constraint of Gentner's theory. The rest of the constraints definining a matcher are specified by a collection of rules, which indicate local, partial matches and estimate how strongly they should be believed. The program uses these estimates and a novel procedure for combining the local matches to efficiently produce and evaluate all consistent global matches.

Cognitive simulation studies can offer important insights for understanding the human mind. They serve to verify psychological theories and supply a detailed vocabulary for describing cognitive processes. Cognitive simulations can provide "idealized subjects", whose prior knowledge and set of available processes is completely known to the experimenter. Unfortunately, cognitive simulations tend to be complex and computationally expensive (c.f. [2,67]). Complexity can obscure the relationship between the theory and the program. While all design decisions affect a program's performance, not all of them are directly motivated by the theory being tested. To assign credit properly (or to model performance in detail) requires exploring a space of similar architectures. Such explorations are very difficult if the major way to change the program's operation is surgery on the code. Complex programs also tend to be computationally expensive, which usually means fewer experiments are performed and fewer possibilities are explored. While there have been several important AI programs that study computational aspects of analogy (e.g., [5,73,74]), they were not designed to satisfy the above criteria.

Over the last decade there have been a variety of programs that simulate different aspects of analogical processing (as reviewed in Section 5). However, the progress to date has been disappointingly slow. Often papers describe programs that work on only a handful of carefully chosen examples, and do not specify the algorithms in a replicable fashion. We believe the difficulty has been in part the lack of a good problem decomposition. Without some theoretically motivated decomposition of analogy, it is easy to merge distinct problems, and become lost in the space of possible mechanisms. Our decomposition, described in the next section, is psychologically motivated. Roughly, SME focuses on the *mapping* process in analogy, leaving the *access* and *application* aspects to future studies. The power of the program and its success on a wide variety of examples (over 40 as of this writing) provides additional evidence that the decomposition is a good one.

This paper examines the architecture of the Structure-Mapping Engine and how it has been used for machine learning and cognitive simulation. First, we review Gentner's Structure-mapping theory and some of the psychological evidence for it. Next we discuss the organization of SME, including

knowledge representation conventions and the algorithm. After a complexity analysis, we then illustrate SME's operation on several examples drawn from machine learning and cognitive simulation studies. Related work in both AI and psychology is reviewed next, followed by a discussion of future work.

2 Structure-mapping Theory

The theoretical framework for this research is Gentner's Structure-mapping theory of analogy [27,28,29,30,31,32]. Structure-mapping describes the set of implicit constraints by which people interpret analogy and similarity. The central idea is that an analogy is a mapping of knowledge from one domain (the base) into another (the target) which conveys that a system of relations known to hold in the base also holds in the target. The target objects do not have to resemble their corresponding base objects. Objects are placed in correspondence by virtue of corresponding roles in the common relational structure.

This structural view of analogy is based on the intuition that analogies are about relations, rather than simple features. No matter what kind of knowledge (causal models, plans, stories, etc.), it is the structural properties (i.e., the interrelationships between the facts) that determine the content of an analogy. For example, consider the water flow and heat flow situations shown in Figure 1. These situations are thought to be analogous because they share the complex relationship known as "flow". In each, we have a rough picture of something flowing downhill, from a source to a destination. We prefer to ignore the appearances and even specific defining properties of the objects, such as the fact that water and coffee are both liquids. Indeed, focusing on these attributes tends to confuse our picture of the analogy.

2.1 Subprocesses in analogy

Structure-mapping decomposes analogical processing into three stages ([33,26,30], see also [9,10,39,48]):

1. *Access*: Given a current *target* situation, retrieve from long-term memory another description, the *base*, which is analogous or similar to the target.

2. *Mapping and Inference*: Construct a mapping consisting of correspondences between the base and target. This mapping can include additional knowledge in the base that can be transferred to the target. These are the *candidate inferences* sanctioned by the analogy.

3. *Evaluation and Use*: Estimate the "quality" of the match. Three kinds of criteria are involved [30,31]. The *structural* criteria include the number of similarities and differences, the degree of structural similarity involved, and the amount and type of new knowledge the analogy provides via the candidate inferences. The second criteria concerns the *validity* of the match and the inferences it sanctions. The inferences must be checked against current world knowledge to ensure that the analogy at least makes sense, and may require additional inferential work to refine the results. The third criteria is *relevance*, i.e., whether or not the analogy is useful to the reasoner's current purposes. Structure-mapping focuses on structural criteria only, since they define and distinguish analogy from other kinds of inference.

The Structure-Mapping Engine emulates the mapping stage of analogy and provides a structural, domain-independent evaluation of the match. While we believe it can be used to model

access, and provides useful results for accounts of evaluation and use (see [16,17]), we will ignore these issues for most of this paper.

2.2 Constraints on Analogy

Structure-mapping defines similarity in terms of matches between the internal structures of the descriptions being compared. Consequently, we need some terminology for describing such structures. Section 3.1 will introduce several formal descriptions. Here we provide some motivating intuitions. Consider a propositional statement, like

```
CAUSE[GREATER-THAN(x,y), BREAK(x)]
```

The chief relation involved in this statement is `CAUSE`, and its arguments are `GREATER-THAN(x,y)` and `BREAK(x)`. We can view this statement in the usual way as a tree, i.e., the root of the tree is a node whose label is the predicate `CAUSE` and the root's children are nodes representing the relation's arguments (Figure 2 provides an example of this view). This view is useful in understanding Structure-mapping because it provides a spatial metaphor for collections of statements. For instance, we can say that the arguments are "below" the `CAUSE` statement in the internal structure of the description, and describe a collection of statements with logical constraints between them (explicitly represented by statements involving logical connectives and/or relationships) as a "connected" system of relations.

One formal definition is needed before proceeding. We define the *order* of an item in a representation as follows: Objects and constants are order 0. The order of a predicate is one plus the maximum of the order of its arguments. Thus `GREATER-THAN(x,y)` is first-order if `x` and `y` are objects, and `CAUSE[GREATER-THAN(x,y), BREAK(x)]` is second-order. Examples of higher-order relations include `CAUSE` and `IMPLIES`. This definition of order should *not* be confused with the standard definition of the order of a logic.[1] Using the tree view of statements, this definition of order indicates how deep the structure is below an item. Notice that intricate explanations with

[1]Under the standard definition, a logic is first-order if variables only range over objects and second-order when it permits variables to range over predicates as well.

many layers of justifications can give rise to representation structures of high order, since there will be a high degree of nesting.

Let $\{B_i\}$, $\{T_i\}$ denote the items in the base and target representations, respectively. Let the subsets $\{b_i\}$,$\{t_i\}$ denote the objects in the base and target, respectively. The tacit constraints on the analogical mapping M can be characterized as follows:

1. Objects in the base are placed in correspondence with objects in the target:

   M: $b_i \rightarrow t_i$

2. Isolated object descriptions are discarded unless they are involved in a larger relational structure.

   e.g. $\texttt{RED}(b_i) \not\rightarrow \texttt{RED}(t_i)$

3. Relations between objects in the base tend to be mapped across:

   e.g. $\texttt{COLLIDE}(b_i,b_j) \longrightarrow \texttt{COLLIDE}(t_i,t_j)$

4. The particular relations mapped are determined by *systematicity*, as defined by the existence of higher-order constraining relations which can themselves be mapped:

   e.g. $\texttt{CAUSE}[\texttt{PUSH}(b_i,b_j),\texttt{COLLIDE}(b_j,b_k)] \Longrightarrow$
   $\texttt{CAUSE}[\texttt{PUSH}(t_i,t_j),\texttt{COLLIDE}(t_j,t_k)]$

We require M to be *one-to-one*: that is, no base item maps to two target items and no target item maps to two base items. Furthermore, we require M to be *structurally consistent*. This means that, in addition to being 1:1, if M maps B_i onto T_j, then it must also map the arguments of B_i onto the corresponding arguments of T_j.

Consider for example a simple analogy between heat-flow and water-flow. Figure 2 shows a simplified version of what a learner might know about the situations pictured in Figure 1. In order to comprehend the analogy "Heat is like water" a learner must do the following (although not necessarily in this order):

1. Set up the object correspondences between the two domains:

   water $\rightarrow$ heat, pipe $\rightarrow$ bar, beaker $\rightarrow$ coffee, vial $\rightarrow$ ice-cube

2. Discard object attributes, such as LIQUID(water).

3. Map base relations such as

   GREATER-THAN[PRESSURE(beaker), PRESSURE(vial)]

   to the corresponding relations in the target domain.

Figure 2: Simplified water flow and heat flow descriptions.

4. Observe systematicity: i.e., keep relations belonging to a systematic relational structure in preference to isolated relationships. In this example,

```
CAUSE(GREATER-THAN[PRESSURE(beaker), PRESSURE(vial)],
      FLOW(beaker, vial, water, pipe))
```

   is mapped into

```
CAUSE(GREATER-THAN[TEMPERATURE(coffee), TEMPERATURE(ice-cube)],
      FLOW(coffee, ice-cube, heat, bar))
```

   while isolated relations, such as

```
GREATER-THAN[DIAMETER(beaker), DIAMETER(vial)]
```

   are discarded.

The *systematicity* principle is central to analogy. Analogy conveys a system of connected knowledge, not a mere assortment of independent facts. Preferring systems of predicates that contain higher-order relations with inferential import is a structural expression of this tacit preference for coherence and deductive power in analogy. Thus, it is the amount of common higher-order relational structure that determines which of several possible matches is preferred. For example, suppose in the previous example we were concerned with objects differing in specific heat, such as a metal ball-bearing and a marble of equal mass, rather than temperatures. Then DIAMETER would enter the mapping instead of (or in addition to) PRESSURE, since DIAMETER affects the capacity of a container, the analogue to specific heat.

2.3 Other types of similarity

In addition to analogy, the distinctions introduced by Structure-mapping theory provide definitions for several other kinds of similarity. In all cases, we require one-to-one, structurally consistent mappings. As we have seen, in *analogy* only relational structures are mapped. Aspects of object descriptions which play no role in the relational structure are ignored. By contrast, in *literal similarity* both relational predicates and object-descriptions are mapped.[2] Literal similarity typically occurs in within-domain comparisons, in which the objects involved look alike as well as act alike. An example of a literal similarity is the comparison "Kool-Aid is like juice." In *mere-appearance* matches, it is primarily the object-descriptions which are mapped, as in the metaphor

"The road is like a silver ribbon"

A fourth kind of mapping is the *abstraction mapping*. Here, the entities in the base domain are variables, rather than objects. Few, if any, attributes exist that do not contribute to the base's relational structure. Applying an abstraction match is very close to the instantiation of a rule. The difference is that only entities may be variables, whereas in many pattern-directed rule systems predicates may be used in substitutions as well.

2.4 Empirical evidence

Although the focus of this paper is on computational modeling, two sets of psychological findings are particularly relevant. First, empirical psychological studies have borne out the prediction that systematicity is a key element of people's implicit rules for analogical mapping. Adults focus on shared systematic relational structure in interpreting analogy. They tend to include relations and omit attributes in their interpretations of analogy, and they judge analogies as more sound and more apt if base and target share systematic relational structure [27,33,34]. In developmental work, it has been found that eight-year olds (but not five-year olds) are better at performing difficult mappings when the base structure is systematic [35]. Second, there is also empirical evidence that the different types of similarity comparisons defined by Structure-mapping have different psychological properties [29,30,31].

3 The Structure-Mapping Engine

A simulation of Gentner's theory has been implemented in the Structure-Mapping Engine (SME). Given descriptions of a base and target, SME constructs all structurally consistent mappings between them. The mappings consist of pairwise matches between statements and entities in the base and target, plus the set of analogical inferences sanctioned by the mapping. SME also provides a structural evaluation score for each mapping according to the constraints of systematicity and structural consistency. For example, given the descriptions of water flow and heat flow shown in Figure 2, SME would offer several alternative interpretations. In one interpretation, the central inference is that water flowing from the beaker to the vial corresponds to heat flowing from the coffee to the ice cube. Alternatively, one could map water to coffee, since they are both liquids.

[2] Notice that our structural characterization of literal similarity differs from some other psychological approaches (e.g., [63]).

The first interpretation has a higher structural evaluation score than the second, since a larger relational structure can be mapped.

Importantly, SME is not a single matcher, but a simulator for a class of matchers. The Structure-mapping notion of structural consistency is built into the system. However, what local elements can match and how these combinations are scored can be changed by implementing new *match rules* that govern what pairwise matches between predicates are allowable and provide local measures of evidence. Thus, for example, SME can be used to simulate all the similarity comparisons sanctioned by Structure-mapping theory, not just analogy. Since the match rules can include arbitrary lisp code, it is possible to implement many other kinds of matchers as well.

This section describes the SME algorithm in sufficient detail to allow replication. We start by specifying some simple conventions for knowledge representation which are essential to understanding the algorithm.

3.1 Representation conventions

We make as few representational assumptions as possible so that SME remains domain-independent. We use a typed (higher-order, in the standard sense) predicate calculus to represent facts. The constructs of this language are:

Entities: Individuals and constants.

Predicates: There are three types: *functions*, *attributes*, and *relations*. Each is described below.

Dgroup: A *description group* is a collection of entities and facts about them, considered as a unit.

We examine each construct in turn.

3.1.1 Entities

Entities are logical individuals, i.e., the objects and constants of a domain. Typical entities include physical objects, their temperature, and the substance they are made of. Primitive entities are the tokens or constants of a description and are declared with the `defEntity` form:

```
(defEntity ⟨name⟩
   [:type ⟨EntityType⟩]
   [:constant?  {t | nil}] )
```

Entities can also be specified in the usual way by compound terms, i.e. the term `(Pressure Well32)` refers to a quantity.

The `:type` option establishes a hierarchy of entity types. For example, we state that our sun is a particular instance of a star with

```
(defEntity sun :type Star)
```

Constants are declared by using the `:constant?` option, as in

```
(defEntity zero :type number :constant? t)
```

3.1.2 Predicates

Classically, "predicate" refers to any functor in a predicate calculus statement. We divide predicates into three categories:

Functions Functions map one or more entities into another entity or constant. For example, `(PRESSURE piston)` maps the physical object `piston` into the quantity which describes its pressure. We treat functions whose range are truth values as relations (see below), rather than functions. Consequently, Structure-mapping treats functions differently from other types of predicates. It allows substitution of functions to acknowledge their role as an indirect way of referring to entities.

Attributes An attribute describes some property of an entity. Examples of attributes include `RED` and `CIRCLE`. We restrict attributes to take only one argument – if there are multiple arguments we classify the predicate as a relation. It is well-known that a combination of a function and a constant is logically equivalent to an attribute. For example,

```
(RED BlockA)
```
and
```
(= (COLOR BlockA) RED)
```

are logically equivalent. However, these two forms do not behave identically under Structure-mapping. We assume that a reasoner has a particular piece of information represented in one form or another, but not both, at any particular time (we return to this issue in Section 6.1).

Relations Like attributes, relations range over truth values. Relations always have multiple arguments, and the arguments can be other predicates as well as entities. (However, we classify logical connectives, regardless of the number of arguments, as relations.) Examples of relations include `CAUSE`, `GREATER-THAN`, and `IMPLIES`.

Predicates are declared with the `defPredicate` form. It has several options:

`(defPredicate` ⟨*Name*⟩ ⟨*ArgumentDeclarations*⟩ ⟨*PredicateType*⟩
`:expression-type` ⟨*DefinedType*⟩
`[:commutative? {t | nil}]`
`[:n-ary? {t | nil}] )`

⟨*PredicateType*⟩ is either `function`, `attribute`, or `relation`, according to what kind of predicate ⟨*Name*⟩ is. The ⟨*ArgumentDeclarations*⟩ specifies the predicate's arity and allows the arguments to be named and typed. For example, the declaration:

```
(defPredicate CAUSE ((antecedent sevent) (consequent sevent)) relation)
```

states that `CAUSE` is a two-place relational predicate. Its arguments are called `antecedent` and `consequent`, both of type `sevent`. (We use `sevent` to mean the union of states and events.) The names and types of arguments are for the convenience of the representation builder, and are not

currently used by SME. However, the predicate type is very important to the algorithms, as we will see below.

The optional declarations :commutative? and :n-ary? provide SME with important syntactic information. :commutative? indicates that the predicate is commutative, and thus the order of arguments is unimportant when matching. :n-ary? indicates that the predicate can take any number of arguments. Declaring n-ary predicates reduces the need for applying associativity to binary predicates [62]. Examples of commutative n-ary predicates include AND, OR, and SUM. Making these distinctions allows SME to

3.1.3 Expressions and Dgroups

For simplicity, predicate instances and compound terms are called expressions. A *Description Group*, or *dgroup*, is a collection of primitive entities and expressions concerning them. Dgroups are defined with the defDescription form:

(defDescription ⟨*DescriptionName*⟩
  entities (⟨$Entity_1$⟩, ⟨$Entity_2$⟩,...,⟨$Entity_i$⟩)
  expressions (⟨*ExpressionDeclarations*⟩))

where ⟨*ExpressionDeclarations*⟩ take the form

⟨*expression*⟩ or
(⟨*expression*⟩ :name ⟨*ExpressionName*⟩)

The :name option is provided for convenience; ⟨*expression*⟩ will be substituted for every occurrence of ⟨*ExpressionName*⟩ in the dgroup's expressions when the dgroup is created. For example, the description of water flow depicted in Figure 2 was given to SME as

```
(defDescription simple-water-flow
      entities (water beaker vial pipe)
      expressions (((flow beaker vial water pipe) :name wflow)
                   ((pressure beaker) :name pressure-beaker)
                   ((pressure vial) :name pressure-vial)
                   ((greater pressure-beaker pressure-vial) :name >pressure)
                   ((greater (diameter beaker) (diameter vial)) :name >diameter)
                   ((cause >pressure wflow) :name cause-flow)
                   (flat-top water)
                   (liquid water)))
```

The description of heat flow depicted in Figure 2 was given to SME as

```
(defDescription simple-heat-flow
      entities (coffee ice-cube bar heat)
      expressions (((flow coffee ice-cube heat bar) :name hflow)
                   ((temperature coffee) :name temp-coffee)
                   ((temperature ice-cube) :name temp-ice-cube)
```

```
((greater temp-coffee temp-ice-cube) :name >temperature)
(flat-top coffee)
(liquid coffee)))
```

Notice that each expression does not need to be declared explicitly; for example, SME will automatically create and name expressions corresponding to `(diameter beaker)` and `(diameter vial)` in the water flow description.

We will refer to the expressions and entities in a dgroup collectively as *items*. To describe the SME algorithm we need some terminology to express the structural relations between items. These relationships form directed acyclic graphs, so we adopt some standard graph-theory terminology. Each item corresponds to a vertex in a graph. When item I_i has I_j as an argument, there will be a directed arc from the node corresponding to I_i to the node corresponding to I_j. The *offspring* of an expression are its arguments. By definition, primitive entities (i.e., those denoted by constants) have no offspring. Expressions which name entities by compound terms are treated like any other item. An item I_1 which is in the transitive closure (arguments of arguments, etc.) of another item I_2 is said to be a *descendant* of I_2, while I_2 is said to be an *ancestor* of I_1. An item with no ancestors is called a *root*. The term *Reachable(I)* refers to the transitive closure of the subgraph starting at I. We define the *depth* of an item with respect to *Reachable(I)* by the minimum number of arcs it takes to reach the item starting from I.

3.2 The SME Algorithm: Overview

Given descriptions of a base and a target, represented as dgroups, SME builds all structurally consistent interpretations of the comparison between them. Each interpretation of the match is called a *global mapping*, or *gmap*.[3] Gmaps consist of three parts:

1. *Correspondences:* A set of pairwise matches between the expressions and entities of the two dgroups.
2. *Candidate Inferences:* A set of new expressions which the comparison suggests holds in the target dgroup.
3. *Structural Evaluation Score:* (Called *SES* for brevity) A numerical estimate of match quality based on the gmap's structural properties.

Following the Structure-mapping theory, we use only purely structural criteria to construct and evaluate the mappings. SME has no other knowledge of either base or target domain. Neither rules of inference nor even logical connectives themselves are built into the algorithm. Each candidate inference must be interpreted as a surmise, rather than a logically valid conclusion. The SES reflects the aesthetics of the particular type of comparison, not validity or potential usefulness. Testing the validity of candidate inferences and determining the utility of a match are left to other modules, as described in Section 2.

Match rules specify what pairwise matches are possible and provide measures of quality used in computing the SES. These rules are the key to SME's flexibility. To build a new matcher one simply

[3]The definition of gmap is inspired in part by de Kleer's work on assumption-based truth maintenance, although we do not use an ATMS in the actual code. The idea of combining local solutions by constructing maximally consistent sets is analogous to the process of *interpretation construction* in an ATMS. We also find bit-vectors a useful implementation technique for the set operations needed to maintain structural consistency.

loads a new set of match rules. This has several important advantages. First, we can simulate all of the similarity comparisons sanctioned by Structure-mapping theory with one program. Second, we could in theory "tune" the rules if needed to simulate particular kinds of human performance (although, importantly, this flexibility has not been needed so far!). Third, we can also simulate a number of other analogy systems (including [40,73], as described below) for comparison purposes.

Conceptually, the SME algorithm is divided into four stages:

1. *Local match construction:* Finds all pairs of (⟨*BaseItem*⟩, ⟨*TargetItem*⟩) that potentially can match. A *Match Hypothesis* is created for each such pair to represent the possibility that this local match is part of a global match.

2. *Gmap construction:* Combines the local matches into maximal consistent collections of correspondences.

3. *Candidate inference construction:* Derives the inferences suggested by each gmap.

4. *Match Evaluation:* Attaches evidence to each local match hypothesis and uses this evidence to compute structural evaluation scores for each gmap.

We now describe each computation in detail, using a simple example to illustrate their operation.

3.2.1 Step 1: Local match construction

Given two dgroups, SME begins by finding potential matches between items in the base and target (see Figure 3). Allowable matches are specified by *match constructor* rules, which take the form:

```
(MHCrule (⟨Trigger⟩ ⟨BaseVariable⟩ ⟨TargetVariable⟩
                    [:test ⟨TestForm⟩])
    ⟨Body⟩)
```

In all match constructor rules, ⟨*Body*⟩ will be executed in an environment in which ⟨*BaseVariable*⟩ and ⟨*TargetVariable*⟩ are bound to items from the base and target dgroups, respectively. If ⟨*TestForm*⟩ is present, the bindings must satisfy the test (i.e., the form when evaluated must return non-NIL). There are two possible values for ⟨*TestForm*⟩. A `:filter` trigger indicates that the rule is applied to each pair of items from the base and target. These rules create an initial set of match hypotheses between individual base and target expressions. For example, the following rule hypothesizes a match between any two expressions that have the same functor:

```
(MHCrule (:filter ?b ?t :test (equal (expression-functor ?b)
                                     (expression-functor ?t)))
      (install-MH ?b ?t))
```

An `:intern` trigger indicates that the rule should be run on each newly created match hypothesis, binding the variables to its base and target items. These rules create additional matches suggested by the given match hypothesis. For example, hypothesizing matches between every pair of entities would lead to combinatorial explosions. Instead, we can use an `:intern` rule to create match hypotheses between entities in corresponding argument positions of other match hypotheses, since these correspondences will be required for structural consistency.

Figure 3: Local Match Construction. The graphs corresponding to the water flow and heat flow descriptions of Figure 2 are depicted on the left and right panels, respectively. The squares and triangles in the middle represent the match hypotheses created by the literal similarity rules for these dgroups. The dashed arrows indicate which base and target items are conjectured as matching by each match hypothesis. The squares represent match hypotheses involving expressions, while the triangles represent match hypotheses involving entities. Notice how sparse the match is. Expression matches are only created when relations are identical, and matches between functions and entities are only created to support expression matches. This "middle out" local match computation provides SME with much of its power.

Appendix A lists the rule sets used to implement each similarity comparison of Structure-Mapping (*analogy*, *literal similarity*, and *mere appearance*). Notice that each rule set is small and simple (we describe the evidence rules below). The literal similarity rule set uses only three match constructor rules. One rule is the filter rule shown above. The other two are intern rules. The content of the first is, roughly,

> "If the match hypothesis concerns two facts, then create match hypotheses between any corresponding arguments that are both functions or entities."

The second is a specialization of this which runs only on commutative predicates (i.e., the "*corresponding* arguments" condition is removed). The analogy rule set differs in that matches are created between attributes only when they are part of some higher-order structure. The mere appearance rule set differs by completely ignoring higher-order structure.

The result of running the match constructor rules is a collection of match hypotheses. We denote the hypothesis that b_i and t_j match by $MH(b_i, t_j)$. When no ambiguity will result, we will simply say MH. We will use the same terminology to refer to the structural properties of

graphs of match hypotheses (offspring, descendants, ancestors, root) as we use for describing items in dgroups. To wit, the collection of match hypotheses can be viewed as a directed acyclic graph, with at least one (and possibly many) roots.

Example: Simple analogy between heat and water In this example we will use the *literal similarity* rule set, rather than *analogy*, in order to better illustrate the algorithm. The result of running these rules on the water flow and heat flow dgroups of Figure 2 is shown in Figure 3 (see also Figure 4). Each match hypothesis locally pairs an item from the base dgroup with an item from the target dgroup.

There are several points to notice in Figure 4. First, there can be more than one match hypothesis involving any particular base or target item. Here, TEMPERATURE can match with both PRESSURE and DIAMETER, since there are corresponding matches between the GREATER-THAN expressions in both dgroups (MH-1 and MH-6). Second, note that with the exception of functions, predicates must match identically. Entities, on the other hand, are matched on the basis of their roles in the predicate structure. Thus while TEMPERATURE can match either PRESSURE or DIAMETER, GREATER cannot match anything but GREATER. This distinction reflects the fact that functions are often used to refer to objects, which are fair game for substitution under analogy. Third, not every possible correspondence is created. We do not, for example, attempt to match TEMPERATURE with water or heat with beaker. Functions only match with other functions; and local matches between entities are only created when justified by some other match. In general, this significantly constrains the number of possible matches.

3.2.2 Step 2: Global Match Construction

The second step in the SME algorithm combines local match hypotheses into collections of global matches (gmaps). Intuitively, each global match is the largest possible set of match hypotheses that depend on the same one to one object correspondences.

More formally, gmaps consist of *maximal, structurally consistent* collections of match hypotheses. A collection of match hypotheses is *structurally consistent* if it satisfies two constraints:

1. *One-to-one:* The match hypotheses in the collection do not assign the same base item to multiple target items or any target item to multiple base items.

2. *Support:* If a match hypothesis MH is in the collection, then so are match hypotheses which pair up all of the arguments of MH's base and target items.

The one-to-one constraint allows straightforward substitutions in candidate inferences. The support constraint preserves connected predicate structure. A collection is maximal if adding any additional match hypothesis would render the collection structurally inconsistent.

Requiring structural consistency both reduces the number of possible global collections and helps preserve the soundness and plausibility of the candidate inferences. Without it, every collection of local matches would need to be considered, and effort would be wasted on degenerate many-to-one mappings without any possible inferential value. The maximality condition also serves to reduce the number of gmaps, since otherwise every subset of a gmap could itself be a gmap.

Global matches are built in two steps:

Figure 4: Water Flow / Heat Flow Analogy After Local Match Construction. Here we show the graph of match hypotheses depicted schematically in Figure 3, augmented by links indicating expression-to-arguments relationships. Match hypotheses which are not descended from others are called *roots* (e.g., the matches between the GREATER predicates, MH-1 and MH-6, and the match for the predicate FLOW, MH-9). Match hypotheses between entities are called *Emaps* (e.g., the match between beaker and coffee, MH-4). Emaps play an important role in algorithms based on structural consistency.

1. *Compute consistency relationships:* For each match hypothesis, generate (a) the set of entity mappings it entails, (b) what match hypotheses it locally conflicts with, and (c) what match hypotheses it is structurally inconsistent with.

2. *Merge match hypotheses:* Compute gmaps by successively combining match hypotheses as follows:

   (a) *Form initial combinations:* Form an initial set of gmaps from each maximal, structurally consistent, connected subgraph of match hypotheses.

   (b) *Combine dependent gmaps:* Merge initial gmaps that have overlapping base structure, subject to structural consistency.

   (c) *Combine independent collections:* Form maximal, complete gmaps by merging the partial gmaps from the previous step, again subject to structural consistency.

Importantly, the process of gmap *construction* is completely independent of gmap *evaluation.* Which gmaps are constructed depends solely on structural consistency. Numerical evidence, described below, is used only to compare their relative merits.

We now describe the algorithm in detail.

Computing consistency relationships Consistency checking is the crux of gmap construction. Consequently, we compute for each match hypothesis (a) the entity mappings it entails and (b) the set of match hypotheses it is inconsistent with.

Consider a particular match hypothesis $MH(b_i, t_j)$ involving base item b_i and target item t_j. If b_i, t_j are expressions, then by the support constraint the match hypotheses linking their arguments must also be in any collection that $MH(b_i, t_j)$ is in. Appying this constraint recursively, all descendents of $MH(b_i, t_j)$ must be in the same collections if it is structurally consistent (see Figure 5). Since the chain of descendants ends with match hypotheses involving entities, each match hypothesis implies a specific set of entity correspondences:

Definition 1. An *emap* is a match hypothesis between entities. *Emaps(*$MH(b_i, t_j)$*)* represents the set of emaps implied by a match hypothesis $MH(b_i, t_j)$. *Emaps(*$MH(b_i, t_j)$*)* is simply the union of the Emaps supported by $MH(b_i, t_j)$'s descendants. We also include match hypotheses involving functions in *Emaps(*$MH(b_i, t_j)$*)*.

To enforce one-to-one mappings we must associate with each $MH(b_i, t_j)$ the set of match hypotheses that provide alternate mappings for for b_i and t_j. Clearly, no member of this set can be in the same gmap with $MH(b_i, t_j)$.

Definition 2. Given a match hypothesis $MH(b_i, t_j)$, the set *Conflicting(*$MH(b_i, t_j)$*)* consists of the set of match hypotheses that represent the alternate mappings for b_i and t_j:

$$Conflicting(MH(b_i,\ t_j)) \equiv [\cup_{b_k \in base}\{MH(b_k,\ t_j) \mid b_k \neq b_i\}] \cup [\cup_{t_k \in target}\{MH(b_i,\ t_k) \mid t_k \neq t_j\}]$$

Figure 5: Water Flow - Heat Flow analogy after computation of *Conflicting* relationships. Simple lines show the tree-like graph that the support constraint imposes upon match hypotheses. Lines with circular endpoints indicate the *Conflicting* relationships between matches. Some of the original lines from MH construction have been left in to show the source of a few *Conflicting* relations.

The set *Conflicting*$(MH(b_i, t_j))$ only notes local inconsistencies (see Figure 5). However, we can use it and *Emaps*$(MH(b_i, t_j))$ to recursively define the set of all match hypotheses that can never be in the same gmap as $MH(b_i, t_j)$.

Definition 3. The set *NoGood*(MH_i) is the set of all match hypotheses which can never appear in the same gmap as MH_i. This set is recursively defined as follows: if MH_i is an emap, then $NoGood(MH_i) = Conflicting(MH_i)$. Otherwise, $NoGood(MH_i)$ is the union of MH_i's *Conflicting* set with the *NoGood* sets for all of its descendents, i.e.,

$$NoGood(MH_i) = Conflicting(MH_i) \bigcup \cup_{MH_j \in Args(MH_i)} NoGood(MH_j)$$

We compute *Conflicting*, *Emaps*, and *NoGood* sets as follows. First, *Conflicting* is computed for each match hypothesis, since it requires only local information. Second, *Emaps* and *NoGood* are computed for each emap. Third, *Emaps* and *NoGood* sets are computed for all other match hypotheses by propagating the results from Emaps upwards to their ancestors.

We make two observations about this computation. First, these operations can be efficiently implemented via bit vectors. For example, `SME` assigns a unique bit position to each match hypothesis, and carries out union and intersection operations by using `OR` and `AND` bit operations. Second, it is important to look for *justification holes* in the match hypothesis graph — match hypotheses whose arguments fail to match. Such match hypotheses will always violate the support constraint, and

hence should be removed. For example, if one of the PRESSURE - TEMPERATURE match hypotheses had not been formed (see Figure 4), then the match between their governing GREATER predicates would be removed. Notice that removing justification holes eliminates many blatantly incorrect matches, such as trying to place an eighth-order IMPLIES in correspondence with a second-order IMPLIES.

The next step in gmap construction is to identify those match hypotheses which are internally inconsistent, and thus cannot be part of any gmap. This can happen when the descendents of a match hypothesis imply mutually incompatible bindings.

Definition 4. A match hypothesis is *inconsistent* if the emaps of one subgraph of its descendants conflicts with the emaps entailed by another subgraph of its descendants:

$$Inconsistent(MH_i) \Leftrightarrow Emaps(MH_i) \cap NoGood(MH_i) \neq \emptyset$$

Clearly, every ancestor of an inconsistent match hypothesis is also inconsistent. By caching the *NoGood* sets, inconsistent match hypotheses can be identified easily.

Global match construction proceeds by collecting sets of consistent match hypotheses. Since gmaps are defined to be maximal, we begin from roots and work downward rather than starting bottom-up. If a root is consistent, then the entire structure under it must be consistent, and thus forms an initial gmap. If the graph of match hypotheses had only a single consistent root, this step would suffice. However, typically there are several roots, and hence several initial gmaps. To obtain true gmaps, that is, *maximal* collections of match hypotheses, these initial gmaps must then be merged into larger, structurally consistent collections.

Merge Step 1: Form initial combinations. The first step is to combine interconnected and consistent structures (Figure 6a). Each consistent root, and its descendants, forms an initial gmap. If a root is inconsistent, then the same procedure is applied recursively to each descendant (i.e., each immediate descendant is now considered as a root). The resulting set will be called $Gmaps_1$. The procedure is:

1. Let $Gmaps_1 = \emptyset$.
2. For every root $MH(b_i, t_j)$
   (a) If $\neg Inconsistent(MH(b_i, t_j))$, then create a gmap GM such that $Elements(GM) = Reachable(MH(b_i, t_j))$.
   (b) If $Inconsistent(MH(b_i, t_j))$, then recurse on $Offspring(MH(b_i, t_j))$.
3. For every $GM \in Gmaps_1$,
   (a) $NoGood(GM) = \bigcup_{MH(b_i,t_j) \in Roots(GM)} NoGood(MH(b_i, t_j))$
   (b) $Emaps(GM) = \bigcup_{MH(b_i,t_j) \in Roots(GM)} Emaps(MH(b_i, t_j))$

In this step inconsistent match hypotheses have been completely eliminated. However, we do not have true gmaps, since the sets of correspondences are not maximal. To obtain maximality, elements of $Gmaps_1$that are consistent with one another must be merged. Consistency between two gmaps can be defined as follows:

$$Consistent(GMap_i, GMap_j) \text{ iff } \begin{array}{l} Elements(GMap_i) \cap NoGood(GMap_j) = \emptyset \\ \wedge \; NoGood(GMap_i) \cap Elements(GMap_j) = \emptyset \end{array}$$

Figure 6: Gmap Construction. (a) Merge step 1: Interconnected and consistent. (b) Merge step 2: Consistent members of the same base structure. (c) Merge step 3: Any further consistent combinations.

Merge Step 2: Combine connected gmaps. Consider two elements of $Gmaps_1$ which share base structure, i.e., whose roots in the base structure are identical. Since we are assuming distinct elements, either (a) their correspondences are structurally inconsistent or (b) there is some structure in the base which connects them that does not appear in the target (if it did, match hypotheses would have been created which would bring the two elements under a common match hypothesis root, hence they would not be distinct). Combining such elements, when consistent, leads to potential support for candidate inferences. We call the partial gmaps resulting from this merge $Gmaps_2$(Figure 6b).

Merge Step 3: Combine independent collections. Consider two elements of $Gmaps_2$ which have no overlap between their relational correspondences. Clearly, any such pair could be merged without inconsistency, if they sanction consistent sets of emaps. This final step generates all consistent combinations of gmaps from $Gmaps_2$ by successive unions, keeping only those combinations that are maximal (Figure 6c).

Example: Simple analogy between heat and water Figure 6 shows how the gmaps are formed from the collection of match hypotheses for the simple water-flow/heat-flow example. After merge step 1, only isolated collections stemming from common roots exist. Merge step 2 combines the `PRESSURE` to `TEMPERATURE` mapping with the `FLOW` mapping, since they have common base structure (i.e., the base structure root is the `CAUSE` predication). Finally, merge step 3 combines the isolated water and coffee attributes (see Figure 7). Notice that the `FLOW` mapping is structurally

consistent with the DIAMETER to TEMPERATURE mapping. However, because merge step 2 placed the FLOW mapping into the same gmap as the PRESSURE to TEMPERATURE mapping, merge step 3 was unable to combine the FLOW mapping with the DIAMETER to TEMPERATURE gmap.

3.2.3 Step 3: Compute Candidate Inferences

Each gmap represents a set of correspondences that can serve as an interpretation of the match. For new knowledge to be generated about the target, there must be information from the base which can be carried over into the target. Not just any information can be carried over — it must be consistent with the substitutions imposed by the gmap, and it must be *structurally grounded* in the gmap. By structural grounding, we mean that its subexpressions must at some point intersect the base information belonging to the gmap. Such structures form the *candidate inferences* of a gmap.

To compute the candidate inferences for a gmap GM, SME begins by examining each root $B_{\mathcal{R}}$ in the base dgroup to see if it is an ancestor of any match hypothesis roots in the gmap. If it is, then any elements in $Descendants(B_{\mathcal{R}})$ which are not in $BaseItems(GM)$ are included in the set of candidate inferences.

The candidate inferences often include entities. Whenever possible, SME replaces all occurrences of base entities with their corresponding target entities. Sometimes, however, there will be base entities that have no corresponding target entity; i.e., the base entity is not part of any match hypothesis for that gmap. What SME does depends on the type of entity. If the base entity is a constant, such as zero, it can be brought directly into the target unchanged (a flag is provided to turn on this behavior). Otherwise, SME introduces a new, hypothetical entity into the target which is represented as a skolem function of the original base entity. Such entities are represented as `(:skolem base-entity)`.

Recall that Structure-mapping does not guarantee that any candidate inference is valid. Each candidate inference is only a surmise, which must be tested by other means. By theoretical assumption, general testing for validity and relevance is the province of other modules which use SME's output.[4] However, SME does provide a weak consistency check based on purely structural considerations. In particular, it discards a candidate inference when (a) the predicate is non-commutatitive and (b) its arguments are simply a permuted version of the arguments to another expression involving that predicate in the target domain. For example, if `(GREATER (MASS sun) (MASS planet))` existed in the target, `(GREATER (MASS planet) (MASS sun))` would be discarded as a candidate inference.

Example: Simple analogy between heat and water In Figure 7, gmap #1 has the top level CAUSE predicate as its sole candidate inference. In other words, this gmap suggests that the cause of the flow in the heat dgroup is the difference in temperatures.

Suppose the FLOW predicate was missing in the target dgroup. Then the candidate inferences for a gmap corresponding to the pressure inequality would include expressions involving both CAUSE and FLOW, as well as conjectured target entities corresponding to water (heat) and pipe (bar). The two skolemized entities would be required because the FLOW match provides the match from water and pipe to heat and bar, respectively. Note also that GREATER-THAN[DIAMETER(coffee),

[4] One such module is described in [16,17].

```
Rule File: literal-similarity.rules     Number of Match Hypotheses: 14

Match Hypotheses:

  (0.6500  0.0000)  (>PRESSURE >TEMP)
  (0.7120  0.0000)  (PRESS-BEAKER TEMP-COFFEE)
  (0.7120  0.0000)  (PRESS-VIAL TEMP-ICE-CUBE)
  (0.9318  0.0000)  (BEAKER-6 COFFEE-1)
  (0.6320  0.0000)  (PIPE-8 BAR-3)
       o        o             o
       o        o             o

Global Mappings:

  Gmap #1:  (>PRESSURE  >TEMPERATURE)  (PRESSURE-BEAKER  TEMP-COFFEE)
            (PRESSURE-VIAL  TEMP-ICE-CUBE)  (WFLOW  HFLOW)
      Emaps:   (beaker coffee)  (vial ice-cube)  (water heat)  (pipe bar)
      Weight:  5.99
      Candidate Inferences: (CAUSE >TEMPERATURE HFLOW)

  Gmap #2:  (>DIAMETER  >TEMPERATURE)  (DIAMETER-1  TEMP-COFFEE)
            (DIAMETER-2  TEMP-ICE-CUBE)
      Emaps:   (beaker coffee)  (vial ice-cube)
      Weight:  3.94
      Candidate Inferences:

  Gmap #3:  (LIQUID-3  LIQUID-5)  (FLAT-TOP-4  FLAT-TOP-6)
      Emaps:   (water coffee)
      Weight:  2.44
      Candidate Inferences:
```

Figure 7: Complete SME interpretation of Water Flow - Heat Flow Analogy.

DIAMETER(ice cube)] is not a valid candidate inference for the first gmap because it does not intersect the existing gmap structure.

3.2.4 Step 4: Compute Structural Evaluation Scores

Typically a particular base and target pair will give rise to several gmaps, each representing a different interpretation. Selecting the "best" interpretation of an analogy, as mentioned previously, can involve non-structural critera. However, as the psychological results indicated, evaluation includes an important structural component. SME provides a programmable mechanism for computing a *structural evaluation score* (SES) for each gmap. This score can be used to rank-order the gmaps or as a factor in some external evaluation procedure.

The structural evaluation score is computed in two phases, each using *match evidence rules* to assign and manage numerical scores. The first phase assigns weights to individual match hypotheses,

and the second phase computes a score for each gmap by combining the evidence for the match hypotheses comprising its correspondences. After a brief introduction to the evidence processing mechanism, we describe each phase in turn.

The management of numerical evidence is performed by a *Belief Maintenance System* (BMS) [15]. The BMS is much like a standard TMS, using horn clauses as justifications. However, the justifications are annotated with evidential weights, so that "degrees of belief" may be propagated. A modified version of Dempster-Shafer formalism is used for expressing and combining evidence. Belief in a proposition is expressed by the pair `(s(A), s(¬ A))`, where `s(A)` represents the current amount of support for `A` and `s(¬ A)` is the current support against `A`. A simplified form of Dempster's rule of combination [60,53,37,15] allows combining evidence from multiple justifications. For example, given that `Belief(A)=(0.4, 0)` and `Belief(B)=(0.6, 0)`, together with `(IMPLIES A C)`$_{(0.8,0)}$ and `(IMPLIES B C)`$_{(1,0)}$, Dempster's rule provides a belief in `C` equal to (0.728, 0.0). In addition to providing evidence combination, these justifications provide useful explanations about the structural evaluation (see [15]).

Two caveats about the role of numerical evidence in SME: (1) While we have found Dempster-Shafer useful, our algorithms are independent of its details, and should work with any reasonable formalism for combining evidence. (2) We use numerical evidence to provide a simple way to combine local information. These weights have nothing to do with any probabilistic or evidential information about the base or target per se.

Assigning local evidence Each match hypothesis and gmap has an associated BMS node to record evidential information. The match evidence rules can add evidence directly to a match hypothesis based on its local properties or indirectly by installing relationships between them. Syntactically, these rules are similar to the match constructor rules. For example,

```
(assert! 'same-functor)
(rule ((:intern (MH ?b ?t) :test (and (expression? ?b) (expression? ?t)
                                      (eq (expression-functor ?b)
                                          (expression-functor ?t)))))
    (assert! (implies same-functor (MH ?b ?t) (0.5 . 0.0))))
```

states that "if the base item and target item of a match hypothesis are expressions with the same functors, then supply 0.5 evidence in favor of the match hypothesis." (The assertion of `same-functor` provides a global record for explanatory purposes that this factor was considered in the structural evaluation.) The complete set of evidence rules used in this paper are listed in Appendix A.

The ability to install relationships between match hypotheses provides a simple, local implementation of the systematicity constraint. Recall that the systematicity constraint calls for prefering expressions involving higher-order relationships belonging to a systematic structure over isolated relationships. We implement this preference by passing evidence from a match involving a relationship to the matches involving its arguments. The following rule accomplishes this, propagating 80% of a match hypothesis' belief to its offspring:

```
(rule ((:intern (MH ?b1 ?t1))
       (:intern (MH ?b2 ?t2) :test (children-of? ?b2 ?t2 ?b1 ?t1)))
    (assert! (implies (MH ?b1 ?t1) (MH ?b2 ?t2) (0.8 . 0.0))))
```

The more matched structure that exists above a given match hypothesis, the more that hypothesis will be believed. The effect cascades, so that entity mappings involved in a large systematic structure receive much higher scores than those which are not. Thus this "trickle down" effect provides a local encoding of the systematicity principle.

Computing the Structural Evaluation Score The structural evaluation score for a gmap is simply the sum of the evidence for its match hypotheses. While simplistic, summation has sufficed for most of the examples encountered so far. There are a number of other factors that are potentially relevant as well, which we discuss in Section 6.3.1. Consequently, to provide maximum flexibility, evidence rules are used to compute the evidence of gmaps as well.

Originally we combined evidence for gmaps according to Dempster's rule, so that the sum of beliefs for all the gmaps equaled 1 [20]. We discovered two problems with this scheme. First, Dempster's rule is susceptible to roundoff, which caused stability problems when a large number of match hypotheses supported a gmap. Second, normalizing gmap evidence prevents us from comparing matches using different base domains (as one would want to do for access experiments), since the score would be a function of the other gmaps for a particular base and target pair.

Example: Simple analogy between heat and water Returning to Figure 7, note that the best interpretation (i.e., the one which has the highest structural evaluation score) is the one we would intuitively expect. In this interpretation, `beaker` maps to `coffee`, `vial` maps to `ice-cube`, `water` maps to `heat`, `pipe` maps to `bar`, and `PRESSURE` maps to `TEMPERATURE`. Furthermore, we have the candidate inference that the temperature difference is what causes the flow of heat.

3.3 Complexity analysis

Here we analyze the complexity of the `SME` algorithm. Because it depends critically on both the input descriptions and the match rules, strict bounds are hard to determine. However, we give both best and worst case analyses for each step, and provide estimates of typical performance based on our experience. The decomposition used in the analysis is show in Figure 8. We use the following notation in the analysis:

$\mathcal{E}_b \equiv$ Number of entities in the base dgroup.

$\mathcal{E}_t \equiv$ Number of entities in the target dgroup.

$\mathcal{F}_b \equiv$ Number of expressions in the base dgroup.

$\mathcal{F}_t \equiv$ Number of expressions in the target dgroup.

$\mathcal{M} \equiv$ Number of match hypotheses.

$\mathcal{G} \equiv$ Number of gmaps

$N_b \equiv \mathcal{E}_b + \mathcal{F}_b$

$N_t \equiv \mathcal{E}_t + \mathcal{F}_t$

$N \equiv \frac{N_b + N_t}{2}$

1. Run MHC rules to construct match hypotheses.
2. Calculate the *Conflicting* set for each match hypothesis.
3. Calculate the *EMaps* and *NoGood* sets for each match hypothesis by upward propagation from entity mappings.
4. Merge match hypotheses into gmaps.
   (a) Interconnected and consistent.
   (b) Consistent members of same base structure.
   (c) Any further consistent combinations.
5. Calculate the candidate inferences for each gmap.
6. Score the matches
   (a) Local match scores.
   (b) Global structural evaluation scores.

Figure 8: Summary of SME algorithm.

3.3.1 Analysis of Step # 1: local match construction

SME does not restrict either the number of match rules or their complexity. There is nothing to prevent one from writing a rule that examines extensive information from external sources (e.g., a knowledge-base, plans, goals, etc.). However, the rule sets which implement the comparisons of Structure-mapping theory consist of only a few simple rules each. This reduction of computational complexity is one of the advantages of the Structure-mapping account, since it restricts the tests performed in rules to local properties of the representation. Consequently, we assume rule execution takes unit time, and focus on the total number of rules executed. The `:filter` rules are run for each pair of base and target predicates. Consequently, they will always require $\mathcal{O}(N_b * N_t)$. Each `:intern` rule is run once on every match hypothesis. In the worst case, $\mathcal{M} = N_b * N_t$, or roughly N^2. But in practice, the actual number of match hypotheses is substantially less, usually on the order of cN, where c is less than 5 and N is the average of N_b and N_t. Thus, in practice, `:intern` rules have a run time of approximately $\mathcal{O}(N)$.

3.3.2 Analysis of Step # 2: Calculating *Conflicting*

Recall that SME assigns a *Conflicting* set to each match hypothesis, $MH(b_i, t_j)$ which represents the alternate mappings for b_i and t_j. The conflicting sets are calculated by examining each base and target item to gather the match hypotheses which mention them. Let $\mathcal{C}$ be the average number of alternative matches each item in the base and target appears in. SME loops through the $\mathcal{C}$ match hypotheses twice: once to form the bitwise union of these match hypotheses and once to update each hypotheses' *Conflicting* set. Thus, the entire number of bit vector operations is

$$(\mathcal{F}_b * 2\mathcal{C}) + (\mathcal{E}_b * 2\mathcal{C}) + (\mathcal{F}_t * 2\mathcal{C}) + (\mathcal{E}_t * 2\mathcal{C})$$

The worst case is when a match hypothesis is created between every base and target item. If we also assume $N_b = N_t$, then $\mathcal{C} = N_t$ in that case. The number of operations becomes $4{N_t}^2$ or approximately $\mathcal{O}(N^2)$. Conversely, the best case performance occurs when $\mathcal{C}$ is 1, producing $\mathcal{O}(max(N_b, N_t))$ operations. In our experiments so far, we find that $\mathcal{C}$ is typically quite small, and so far has always been less than 10. Consequently, the typical performance lies between $\mathcal{O}(N)$ and $\mathcal{O}(N^2)$.

3.3.3 Analysis of Step # 3: *Emaps* and *NoGood* calculation

Recall that once the $Conflicting$ sets are calculated, the *Emaps* and $NoGood$ sets are propagated upwards from the entity mappings through the match hypotheses. By caching which $MH(b_i, t_j)$'s correspond to emaps and using a queue, we only operate on each node once. Hence the worst and best case performance of this operation is $\mathcal{O}(\mathcal{M})$, which in the worst case is $\mathcal{O}(N^2)$.

3.3.4 Analysis of Step # 4: Gmap construction

Global matches are formed in three steps. The first step collects all of the consistent connected components of match hypotheses by starting at the match hypothesis roots, walking downwards to find consistent structures. Each graph walk takes at most $\mathcal{O}(N_i)$, where N_i is the number of nodes *Reachable* from the current match hypothesis root. If there are N_R roots, then the first merge step (Step 4(a)) takes $\mathcal{O}(N_R * N_i)$. Assuming that most of the match hypotheses will appear in only one or two subgraphs (some roots may share substructure), we can approximate this by saying that the first merge step is $\mathcal{O}(\mathcal{M})$. Call the number of partial gmaps formed at this stage $\mathcal{G}_{P1}$.

Perhaps surprisingly, the complexity of the previous steps has been uniformly low. Sophisticated matching computations usually have much worse performance, and SME cannot completely escape this. In particular, the worst case for steps 4(b) and 4(c) is $\mathcal{O}(N!)$ (although worst-case for one implies best-case for the other).

Step 4(b) combines partial gmaps from Step 4(a) that intersect the same base structure. This requires looping through each base description root to find which partial gmaps intersect it, and then generating every consistent, maximal combination of them. In the worst case, every gmap could intersect the same base structure. This would mean generating all possible consistent, maximal sets of gmaps, which is equivalent to Step 4(c), so we defer this part of the analysis until then. In the other extreme, none of the gmaps share a common base structure, and so step 4(b) requires $\mathcal{O}({\mathcal{G}_{P1}}^2)$ operations, although this is not the best-case performance (see below). Typically, the second merge step is very quick and displays near best-case performance.

Step 4(c) completes gmap construction by generating all consistent combinations of the partial gmaps, discarding those which are not maximal. The complexity of this final merge step is directly related to the degree of structure in the base and target domains and how many different predicates are in use. Worst-case performance occurs when the description language is flat (i.e., no higher-order structure) and the same predicate occurs many times in both the base and the target. Consider a language with a single, unary predicate, and base and target dgroups each consisting of N distinct expressions. In this case every base expression can match with every target expression, and each such match will suggest matching in turn the entities that serve as their arguments. This reduces to the problem of finding all isomorphic mappings between two equal size sets, which is $\mathcal{O}(N!)$.

Now let us consider the best case. If the base and target dgroups give rise to a match hypothesis graph that has but one root, and that root is consistent, then there is only one gmap! The second

and third merge steps in this case are now independent of N, i.e., constant-time.

Of course, the typical case is somewhere between these two extremes. Typically the vocabulary of predicates is large, and the relationships between entities diverse. Structure provides a strong restriction on the number of possible interpretations for an analogy. By the time SME gets to Step 4, many of the match hypotheses have been filtered out as being structurally impossible. Steps 4(a) and 4(b) have already merged many partial gmaps, reducing the number of elements which may be combined. The identicality constraint of Structure-Mapping (encoded in the match rules) also reduces typical-case complexity, since match hypotheses are only created between relations when functors are identical. Thus, SME will perform badly on large descriptions with no structure and extensive predicate repetition, but SME will perform well on large descriptions with deep networks of diverse higher-order relationships. Semantically, the former case roughly corresponds to a jumble of unconnected expressions, and the latter case to a complex argument or theory. The better organized and justified the knowledge, the better SME will perform.

While the potential complexity of Step 4(b) is $\mathcal{O}(N!)$, our experience is that this step is very quick and displays near best-case performance in practice. We suspect the worst-case behavior is very unlikely to occur, since it requires that all members of $Gmaps_1$ intersect the same base-structure and so must be merged in all possible ways. However, partial gmaps intersecting the same base structure are almost always consistent with one another, meaning that step 2 would usually merge $Gmaps_1$ into one gmap in $\mathcal{O}(\mathcal{G}_{P1})$ time. On the other hand, it is easy to hand-generate examples which illustrate the worst-case performance for Step 4(c), and this step in practice can take signficant work.

3.3.5 Analysis of Step # 5: Finding candidate inferences

The candidate inferences are gathered by looping through the base description roots for each gmap, collecting missing base expressions whenever their structure intersects a match hypothesis in the gmap. Each expression is tested to ensure that (1) it is not already matched with part of the target description, and (2) whether it represents a contradiction of an existing target expression. The size of the typical candidate inference is inversely related to the percentage of base structure roots: more roots implies less structure to infer, and vice versa. Thus in the worst case we have $\mathcal{O}(\mathcal{G} * \mathcal{F}_b * \mathcal{F}_t)$, or roughly $\mathcal{O}(N^4)$. However, this is an extreme worst-case. First, the $\mathcal{F}_t$ term implies that we check every target expression on each iteration. The algorithm actually only checks the pertinent target expressions (i.e., those with the same functor), giving a tighter bound of $\mathcal{O}(N^3)$. In the best case, there will only be one gmap and no candidate inferences, producing constant time behavior.

3.3.6 Analysis of Step # 6: SES computation

The complexity of the BMS is difficult to ascertain. Fortunately, it is irrelevant to our analysis since the BMS can be eliminated if detailed justifications of evidential results are not required. For example, the first version of SME [20] used specialized evidence rules which had most of the flexibility of the BMS-based rules yet ran in $\mathcal{O}(\mathcal{M})$ time.

Although the flexibility of the BMS can be valuable, in fact the majority of SME's processing time takes place within it – typically 70 to 80%. So far this has not been a serious performance limtation, since on the examples in this paper (and most of the examples we have examined), SME runs in a matter of a few seconds on a Symbolics machine.

4 Examples

The Structure-Mapping Engine has been applied to over 40 analogies, drawn from a variety of domains and tasks. It is being used in psychological studies, comparing human responses with those of SME for both short stories and metaphors. It is also serving as a module in a machine learning program called PHINEAS, which uses analogy to discover and refine qualitative models of physical processes such as water flow and heat flow. Here we discuss a few examples to demonstrate SME's flexibility and generality.

4.1 Methodological constraints

Flexibility is a two-edged sword. The danger in using a program like SME is that one could imagine tailoring the match construction and evidence rules for each new example. Little would be learned by using the program in this way — we would have at best a series of "wind-up toys", a collection of ad-hoc programs which shed little theoretical light. Here we describe our techniques for reducing tailorability.

First, all the cognitive simulation experiments were run with a fixed collection of rule sets, listed in Appendix A. Each rule set represented a particular type of comparison sanctioned by the Structure-mapping theory (i.e., analogy, literal similarity, and mere appearance). The mere appearance rules (MA) match only low-order items: attributes and first-order relations. The analogy rules (AN) match systems of higher-order relations, while the literal similarity rules (LS) match both low-order and higher-order structure. The first two examples in this section use the AN rules, while the last uses both AN and MA rules, as indicated.

While the choice of match construction rules is dictated by Structure-mapping, the particular values of evidence weights are not. Although we have not performed a sensitivity analysis, in our preliminary explorations it appears that the gmap rankings are not overly sensitive to the particular values of evidence weights. (Recall that which gmaps are constructed is *independent* of the weights, and is determined only by the construction rules and structural consistency.)

Second, we have accumulated a standard description vocabulary which is used in all experiments. This is particularly important when encoding natural language stories, where the translation into a formal representation is underconstrained. By accumulating representation choices across stories, we attempt to free ourselves from biasing the descriptions for particular examples.

Third, we have tested SME with descriptions generated automatically by other AI programs. A representation developed to perform useful inferences has fewer arbitrary choices than a representation developed specifically for learning research. So far, we have used descriptions generated by two different qualitative simulation programs with encouraging results. For example, SME actually performs better on a water-flow / heat-flow comparison using more complex descriptions generated by GIZMO [23] than on many hand-generated descriptions. We are working on other, similar systems, as described in Section 6.3.1.

4.2 Solar System - Rutherford Atom Analogy

The Rutherford model of the hydrogen atom was a classic use of analogy in science. The hydrogen atom was explained in terms of the better understood behavior of the solar system. We illustrate SME's operation on this example with a simplified representation, shown in Figure 9.

Figure 9: Solar System - Rutherford Atom Analogy.

SME constructed three possible interpretations. The highest-ranked mapping (SES = 6.03) pairs up the nucleus with the sun and the planet with the electron. This mapping is based on the mass inequality in the solar system playing the same role as the mass inequality in the atom. It sanctions the inference that the differences in masses, together with the mutual attraction of the nucleus and the electron, causes the electron to revolve around the nucleus. This is the standard interpretation of this analogy.

The other major gmap (SES = 4.04) has the same entity correspondences, but maps the temperature difference between the sun and the planets onto the mass difference between the nucleus and the electron. The SES for this gmap is low for two reasons. First, temperature and mass are different functions, and hence they receive less local evidence. The second, and more important, reason is that there is no mappable systematic structure associated with temperature in the base dgroup. Thus other relations, such as the match for ATTRACTS, do not enter into this gmap. We could in theory know alot more about the thermal properties of the solar system than its dynamics, yet unless there is some relational ground in the target description there will not be a set of *mappable* systematic relations. (If we instead were explaining a home heating system in terms of the solar system the situation would be the reverse.)

The third gmap is a spurious collection of match hypotheses which imply that the mass of the sun should correspond to the mass of the electron, and the mass of the planet should correspond to the mass of the nucleus. There is even less structural support for this interpretation (SES = 1.87).

This example demonstrates an important aspect of the Structure-mapping account of analogy. The interpretation preferred on structural grounds is also the one with the most inferential import. This is not an accident; the systematicity principle captures the structural features of well-supported arguments. Using the Structure-mapping analogy rules (AN), SME prefers interpretations based on a deep theory (i.e., a subset of a dgroup containing a system of higher-order relations) to those based on shallow associations (i.e., a subset of a dgroup containing an assortment of miscellaneous facts).

4.3 Discovering heat flow

Figure 10: Two examples of water-flow and heat-flow.

The PHINEAS program [16,17,19] learns by observation. When presented with a new behavior, it attempts to explain it in terms of its theories of the world. These theories are expressed as qualitative models of physical processes using Forbus' *Qualitative Process Theory* [21,22]. When it is given a behavior that it cannot explain, an analogical learning module is invoked which attempts to generate a new or revised model that can account for the new observation. This module uses SME in two ways.[5] First SME is used to form a match between a previous experience which has been explained and the current behavior. These correspondences then provide the foundation for constructing a model that can explain the new observation based on the model for the previous behavior.

For example, suppose that the program was presented with measurements of the heat-flow situation depicted in Figure 10 and described in Figure 11. If the domain model does not include a theory of heat flow, PHINEAS will be unable to interpret the new observation.[6] Using SME, PHINEAS constructs an analogy with the previously encountered water-flow experience also shown in Figures 10 and 11. This match establishes that certain properties from the two situations behave in the same way. As shown in Figure 11, the roles of the beaker and the vial in the water flow history are found to correspond to the roles of the horse shoe and water in the heat flow history, respectively. PHINEAS stores the correspondences that provide a mapping between entities or between their quantities (e.g., `Pressure` and `Temperature`) for later reference.

When it is satisfied that the chosen water-flow history is sufficiently analogous to the current situation, PHINEAS begins a deeper analysis of the analogy. It fetches the domain used to generate its prior understanding of the base (water-flow) experience. Its description of water-flow, shown in Figure 12, is a straightforward qualitative model similar to that used in other projects [23,26]. This model states that if we have an aligned fluid path between the beaker and the vial (i.e., the path either has no valves or if it does, they are all open), and the pressure in the beaker is greater than the pressure in the vial, then a liquid-flow process will be active. This process has a flow rate which is proportional to the difference between the two pressures. The flow rate has a positive influence on the amount of water in the vial and a negative influence on the amount of water in the beaker.

[5] In this example PHINEAS is using the Structure-mapping analogy rules. In normal operation, it uses a rule set that examines an IS-A hiearchy to relax the identicality constraint and a relevance-influenced match evaluation criteria that is sensitive to the system's current reasoning goals [19].

[6] PHINEAS uses the ATMI theory of measurement interpretation to explain observations. See [24] for details.

Water-Flow History

```
(Situation S0)
(Decreasing (Pressure (At beaker S0)))
(Increasing (Pressure (At vial S0)))
(Decreasing (Amount-of (At beaker S0)))
(Increasing (Amount-of (At vial S0)))
(Greater (Pressure (At beaker S0))
         (Pressure (At vial S0)))

(Situation S1)
(Meets S0 S1)
(Constant (Pressure (At beaker S1)))
(Constant (Pressure (At vial S1)))
(Constant (Amount-of (At beaker S1)))
(Constant (Amount-of (At vial S1)))
(Equal-To (Pressure (At beaker S1))
          (Pressure (At vial S1)))

(Function-Of (Pressure ?x)
             (Amount-of ?x))
```

Heat-Flow History

```
(Situation S0)
(Decreasing (Temp (At horse-shoe S0)))
(Increasing (Temp (At water S0)))
(Greater (Temp (At horse-shoe S0))
         (Temp (At water S0)))

(Situation S1)
(Meets S0 S1)
(Constant (Temp (At horse-shoe S1)))
(Constant (Temp (At water S1)))
(Equal-To (Temp (At horse-shoe S1))
          (Temp (At water S1)))

(Function-Of (Temp ?x)
             (Heat ?x))
```

Behavioral Correspondences

```
Pressure  ↔  Temperature
Amount-of ↔  Heat
       S0 ↔  S0
       S1 ↔  S1
   beaker ↔  horse-shoe
    vial  ↔  water
```

Figure 11: Analogical match between water-flow history and heat-flow history.

Figure 12: Qualitative Process Theory model of liquid flow.

Using SME a second time, this theory is matched to the current heat-flow situation using the correspondences established with the behavioral analogy. The output is shown in Figure 13. The entity and function correspondences provided by the behavioral analogy provide signficant constraint for carrying over the explanation. SME's rule-based architecture is critical to this operation: PHINEAS imposes these constraints by using a set of match constructor rules that only allow hypotheses consistent with the specific entity and function correspondences previously established. Entities and functions left without a match after the accessing stage are still allowed to match other unmatched entities and functions. For example, the rule

```
(MHC-rule (:filter ?b ?t :test (sanctioned-pairing? (expression-functor ?b)
                                                    (expression-functor ?t)))
      (install-MH ?b ?t))
```

forces a match between those quantities which were found to be analogous in the behavioral analogy (e.g., PRESSURE and TEMPERATURE) and prevents any alternate matches for these quantities (e.g., AMOUNT-OF and TEMPERATURE).

This example demonstrates several points. First, the second analogy which imports the theoretical explanation of the new phenomena is composed almost entirely of candidate inferences, since the system had no prior model of heat flow. Hence, the model was *constructed* by analogy

```
Gmap #1: { (AMOUNT-OF-35  HEAT-WATER)  (AMOUNT-OF-33  HEAT-HSHOE)
           (PRESSURE-BEAKER  TEMP-HSHOE) (PRESSURE-VIAL  TEMP-WATER) }
    Emaps:  { (beaker  horse-shoe)  (vial  water) }
    Weight:  2.675
    Candidate Inferences: (IMPLIES
                           (AND (ALIGNED (:skolem pipe))
                                (GREATER-THAN (A TEMP-HSHOE) (A TEMP-WATER)))
                           (AND (Q= (FLOW-RATE pi) (- TEMP-HSHOE TEMP-WATER))
                                (GREATER-THAN (A (FLOW-RATE pi)) zero)
                                (I+ HEAT-WATER (A (FLOW-RATE pi)))
                                (I- HEAT-HSHOE (A (FLOW-RATE pi)))))
```

Figure 13: An Analogically Inferred Model of Heat Flow.

rather than augmented by analogy. This shows the power of SME's candidate inference mechanism. Second, the example illustrates how SME's rule-based architecture can support tasks in which the entity correspondences are given prior to the match, rather than derived as a result of the match. Finally, it shows the utility of introducing skolemized entities into the candidate inferences. The results produced by SME (Figure 13) contain the entity `(:skolem pipe)`. This indicates that, at the moment, the heat path is a conjectured entity. At this time, the system inspects its knowledge of paths to infer that immersion or physical contact is a likely heat path. However, we note that much knowledge gathering and refinement may still take place while leaving the heat path as a conjectured entity. For example, in the history of science *ether* was postulated to provide a medium for the flow of light waves because other kinds of waves required a medium.

4.4 Modeling Human Analogical Processing

SME is being used in several cognitive simulation studies. Our goal is to compare human responses with those of SME's for a variety of tasks and problems. For example, two psychological studies [33,56] have explored the variables that determine the *accessibility* of a similarity match and the *inferential soundness* of a match. Structure-mapping predicts that the degree of systematic relational overlap will determine soundness [29]. In contrast, Gentner [30,31] has suggested that the accessibility of potential matches in long-term memory is heavily influenced by surface similarity. Psychological studies have supported both hypotheses [33,56,58]. In order to verify the computational assumptions we then ran SME on the same examples. Here we briefly summarize the simulation methodology and the results; for details see [65].

The hypotheses were tested psychologically as follows. Pairs of short stories were constructed which were similar in different ways: in particular, some pairs embodied mere appearance and some analogy.[7] Subjects first read a large set of stories. Then, in a second session, subjects saw similar stories and tried to retrieve the original stories (the access measure). After that, the subjects were then asked to judge the inferential soundness of each of the story pairs. For the cognitive

[7]Other kinds of matches, including literal similarity, were also used. Here we discuss only analogy and mere appearance

Base Story

Karla, an old hawk, lived at the top of a tall oak tree. One afternoon, she saw a hunter on the ground with a bow and some crude arrows that had no feathers. The hunter took aim and shot at the hawk but missed. Karla knew that hunter wanted her feathers so she glided down to the hunter and offered to give him a few. The hunter was so grateful that he pledged never to shoot at a hawk again. He went off and shot deer instead.

Target Story - Analogy

Once there was a small country called Zerdia that learned to make the world's smartest computer.

One day Zerdia was attacked by its warlike neighbor, Gagrach. But the missiles were badly aimed and the attack failed. The Zerdian government realized that Gagrach wanted Zerdian computers so it offered to sell some of its computers to the country. The government of Gagrach was very pleased. It promised never to attack Zerdia again.

Target Story - Mere-Appearance

Once there was an eagle named Zerdia who donated a few of her tailfeathers to a sportsman so he would promise never to attack eagles.

One day Zerdia was nesting high on a rocky cliff when she saw the sportsman coming with a crossbow. Zerdia flew down to meet the man, but he attacked and felled her with a single bolt. As she fluttered to the ground Zerdia realized that the bolt had her own tailfeathers on it.

Figure 14: Story Set Number 5.

simulation study, five triads of stories — a base, a mere-appearance match, and an analogy match were encoded (15 in all). Then pairs of stories were presented to SME, using different rule sets corresponding to analogy (the AN rules) and mere appearance (the MA rules). The results from the AN rules were used to estimate soundness, while the results from the MA rules were used to estimate accessibility. One of these story groups will be discussed in detail, showing how SME was used to simulate a test subject.

In the story set shown in Figure 14, the original story concerned a hawk named Karla who survives an attack by a hunter. Two target stories were used as potential analogies for the Karla narration. One was designed to be truly analogous (TA5) and describes a small country named Zerdia that survives an attack by another country. The other story (MA5) was designed to be only superficially similar and describes an eagle named Zerdia who is killed by a sportsman. The representation of the Karla story given to SME was:

```
(CAUSE (EQUALS (HAPPINESS HUNTER) HIGH)
       (PROMISE HUNTER KARLA (NOT (ATTACK HUNTER KARLA))))
(CAUSE (OBTAIN HUNTER FEATHERS) (EQUALS (HAPPINESS HUNTER) HIGH))
(CAUSE (OFFER KARLA FEATHERS HUNTER) (OBTAIN HUNTER FEATHERS))
(CAUSE (REALIZE KARLA (DESIRE HUNTER FEATHERS))  (OFFER KARLA FEATHERS HUNTER))
(FOLLOW (EQUALS (SUCCESS (ATTACK HUNTER KARLA)) FAILED)
        (REALIZE KARLA (DESIRE HUNTER FEATHERS)))
(CAUSE (NOT (USED-FOR FEATHERS CROSS-BOW))  (EQUALS (SUCCESS (ATTACK HUNTER KARLA)) FAILED))
(FOLLOW (SEE KARLA HUNTER) (ATTACK HUNTER KARLA))
(WEAPON CROSS-BOW)
(KARLAS-ASSET FEATHERS)
(WARLIKE HUNTER)
(PERSON HUNTER)
(BIRD KARLA)
```

The results from human subjects showed that (1) in the soundness evaluation task, as predicted

Analogical Match from *Karla* **to** *Zerdia the country (TA5).*

```
Rule File: analogy.rules     Number of Match Hypotheses: 54     Number of GMaps: 1

Gmap #1:
   (CAUSE-PROMISE CAUSE-PROMISE) (SUCCESS-ATTACK SUCCESS-ATTACK) (HAPPY-HUNTER HAPPY-GAGRACH)
   (HAPPINESS-HUNTER HAPPINESS-GAGRACH) (REALIZE-DESIRE REALIZE-DESIRE) (CAUSE-TAKE CAUSE-BUY)
   (ATTACK-HUNTER ATTACK-GAGRACH) (DESIRE-FEATHERS DESIRE-SUPERCOMPUTER) (FAILED-ATTACK FAILED-ATTACK)
   (TAKE-FEATHERS BUY-SUPERCOMPUTER) (CAUSE-FAILED-ATTACK CAUSE-FAILED-ATTACK)
   (CAUSE-OFFER CAUSE-OFFER) (FOLLOW-REALIZE FOLLOW-REALIZE) (HAS-FEATHERS USE-SUPERCOMPUTER)
   (CAUSE-HAPPY CAUSE-HAPPY) (NOT-ATTACK NOT-ATTACK) (PROMISE-HUNTER PROMISE)
   (NOT-HAS-FEATHERS NOT-USE-SUPERCOMPUTER) (OFFER-FEATHERS OFFER-SUPERCOMPUTER)
 Emaps:  (HIGH23 HIGH17) (FEATHERS20 SUPERCOMPUTER14) (CROSS-BOW21 MISSILES15)
         (HUNTER19 GAGRACH13) (KARLA18 ZERDIA12) (FAILED22 FAILED16)
 Weight: 22.362718
```

Analogical Match from *Karla* **to** *Zerdia the eagle (MA5).*

```
Rule File: analogy.rules     Number of Match Hypotheses: 47     Number of GMaps: 1

Gmap #1:
   (PROMISE-HUNTER PROMISE) (DESIRE-FEATHERS DESIRE-FEATHERS) (TAKE-FEATHERS TAKE-FEATHERS)
   (CAUSE-OFFER CAUSE-OFFER) (OFFER-FEATHERS OFFER-FEATHERS) (HAS-FEATHERS HAS-FEATHERS)
   (REALIZE-DESIRE REALIZE-DESIRE) (ATTACK-HUNTER ATTACK-SPORTSMAN) (NOT-ATTACK NOT-ATTACK)
   (SUCCESS-ATTACK SUCCESS-ATTACK) (FOLLOW-SEE-ATTACK FOLLOW-SEE) (SEE-KARLA SEE-ZERDIA)
   (FAILED-ATTACK SUCCESSFUL-ATTACK) (CAUSE-TAKE CAUSE-TAKE)
 Emaps:  (FAILED22 TRUE11) (KARLA18 ZERDIA7) (HUNTER19 SPORTSMAN8)
         (FEATHERS20 FEATHERS9) (CROSS-BOW21 CROSS-BOW10)
 Weight: 16.816530
```

Figure 15: SME's Analysis of Story Set 5, Using the TA Rules.

by Gentner's systematicity principle, people judged analogies as more sound than mere appearance matches; and (2) in the memory access task, people were far more likely to retrieve surface similarity matches than analogical matches.

To test SME as a cognitive simulation of how people determine the soundness of an analogy, SME was run using its analogy (AN) match rules on each base-target pair of stories – that is, base/mere-appearance story and base/analogical story. Figure 15 shows the output of SME for the AN task. For example, "Zerdia the country" (the analogy) was found to be a better analogical match (SES = 22.4) to the original Karla story than "Zerdia the eagle" (SES = 16.8). Overall, SME as an analogical mapping engine agrees quite well with the soundness rating of human subjects.

We also used SME to test the claim that the human access patterns resulting from a dependence on surface similarity matches (objects and object-attribute overlap). To test this, SME was run on each of the pairs using its mere-appearance (MA) match rules. This measured their degree of superficial overlap. Again, over the five stories SME's rankings match those of human subjects. For example, the output of SME for the MA task is given in Figure 16, which shows that the eagle story (SES = 7.7) has a higher MA rating than the country story (SES = 6.4).

Analogical Match from *Karla* **to** *Zerdia the country (TA5).*

```
Rule File: appearance-match.rules   Number of Match Hypotheses: 12   Number of GMaps: 1

Gmap #1:
   (HAPPINESS-HUNTER HAPPINESS-GAGRACH) (ATTACK-HUNTER ATTACK-GAGRACH) (TAKE-FEATHERS BUY-SUPERCOMPUTER)
   (WARLIKE-HUNTER WARLIKE-GAGRACH) (DESIRE-FEATHERS DESIRE-SUPERCOMPUTER)
   (HAS-FEATHERS USE-SUPERCOMPUTER) (OFFER-FEATHERS OFFER-SUPERCOMPUTER) (WEAPON-BOW WEAPON-BOW)
 Emaps:  (KARLA1 ZERDIA12) (FEATHERS3 SUPERCOMPUTER14) (CROSS-BOW4 MISSILES15) (HUNTER2 GAGRACH13)
 Weight: 6.411572
```

Analogical Match from *Karla* **to** *Zerdia the eagle (MA5).*

```
Rule File: appearance-match.rules   Number of Match Hypotheses: 14   Number of GMaps: 1

Gmap #1:
   (OFFER-FEATHERS OFFER-FEATHERS) (TAKE-FEATHERS TAKE-FEATHERS) (ATTACK-HUNTER ATTACK-SPORTSMAN)
   (SEE-KARLA SEE-ZERDIA) (HAS-FEATHERS HAS-FEATHERS) (BIRD-KARLA BIRD-ZERDIA) (WEAPON-BOW WEAPON-BOW)
   (DESIRE-FEATHERS DESIRE-FEATHERS) (WARLIKE-HUNTER WARLIKE-SPORTSMAN) (PERSON-HUNTER PERSON-SPORTSMAN)
 Emaps:  (FEATHERS3 FEATHERS9) (CROSS-BOW4 CROSS-BOW10) (HUNTER2 SPORTSMAN8) (KARLA1 ZERDIA7)
 Weight: 7.703568
```

Figure 16: SME's Analysis of Story Set 5, Using the MA Rules.

It should be noted that the access mimicking task is not a true simulation. To do this would require finding and selecting the prior story from a large set of potential matches. Rather, SME is acting as a bookkeeper to count the variable (here, degree of surface overlap) being claimed as causally related to the variable being measured (accessibility of matches). The results demonstrate that surface similarity, as strictly defined and used in SME's match rules, match well with people's retrieval patterns in an access task.

This study illustrates the viability of SME as a cognitive simulation of human processing of analogy. We make two additional observations. First, the results demonstrate the considerable leverage for cognitive modeling that SME's architecture provides. We know of no other general-purpose matcher which successfully models two *distinct* kinds of human similarity comparisons. Second, the short story analogies show that SME is capable of matching large structures as well as the smaller, simpler structures shown previously.

4.5 Removing all external constraints

What example should this be - an isomorphic type example, or one of the ones already given (WF-HF or SS-RA) to show comparison?

4.6 Performance Evaluation

SME is written in Common Lisp. The examples in this paper were run on a Symbolics 3640 with 8 megabytes of RAM. Table 1 shows SME's performance for each example in this paper. All run

Table 1: SME performance on described examples.

Example	Number base expressions/entities	Number target expressions/entities	# MH's	# Gmaps	Total BMS run time	Total match run time
Simple Water-Heat	11/4	6/4	14	3	0.70	0.23
Solar System-Atom	12/2	9/2	16	3	0.91	0.28
PHINEAS behavioral	40/8	27/6	69	6	9.68	1.92
PHINEAS theory	19/11	13/6	10	1	0.17	0.66
Base5-TA5 (AN)	26/6	24/6	54	1	5.34	0.87
Base5-MA5 (AN)	26/6	24/5	47	1	4.55	0.98
Base5-TA5 (MA)	26/6	24/6	12	1	0.38	0.36
Base5-MA5 (MA)	26/6	24/5	14	1	0.73	0.46

NOTE: All times are given in seconds. Total match time is total SME run time minus BMS run time.

times are in seconds. We have separated the BMS run time from the total run time to give a more accurate account of SME's speed, since the computational cost of the BMS can be removed if necessary. This data indicated that SME is extremely fast at producing unevaluated gmaps. In fact, it would seem to be close to linear in the number of match hypotheses and in the number of base and target expressions. The majority of the run time is spent within the BMS, producing structural evaluation scores. However, the total run times are sufficiently short that we have opted to continue using the BMS for now, since it has proven to be a valuable analysis tool.

The longest runtime occurred for the behavioral match between the water-flow and heat-flow observations (PHINEAS behavioral). While the descriptions for this example were the largest, the primary source of slowdown was the flat representations used to describe the situations.

5 Comparison With Other Work

The Structure-mapping theory has received a great deal of convergent theoretical support in artificial intelligence and psychology. Although there are differences in emphasis, there is now widespread agreement on the basic elements of one-to-one mappings of objects with carryover of predicates ([5,6,38,41,46,47,57,59,73,66]). Moreover, several of these researchers have adopted special cases of the systematicity principle. For example, Carbonell focuses on plans and goals as the high-order relations that give constraint to a system, while Winston [74] focuses on causality. Structure-mapping theory subsumes these treatments in three ways. First, it defines mapping rules which are independent of particular domains or primitives. Second, the Structure-mapping characterization applies across a range of applications of analogy, including problem solving, understanding explanations, etc. Third, the Structure-mapping account treats analogy as one of a family of similarity comparisons, each with particular psychological privileges, and thus explains more phenomena.

Some models have combined an explicit Structure-mapping component to generate potential interpretations of a given analogy with a pragmatic component to select the relevant interpretation (e.g., [5,47]). Given our experience with PHINEAS, we believe SME will prove to be a useful tool for such systems.

SME computes a structural match first, and then uses this structural match to derive candidate inferences. The implementations of Winston [73] and Burstein [5] are similar to SME in this respect. An alternate strategy is used by Winston [74], Kedar-Cabelli [47], Carbonell [6,7], and Greiner

[38]. These programs do not perform a match *per se*, but instead attempt to carry over "relevant" structure first and modify it until it applies to the target domain. The match arises as an implicit result of the structure modification. We know of no complexity results available for this technique, but we suspect it is much worse than SME. It appears that there is great potential for extensive search in the modification method. Furthermore, the modification method effectively requires that the access mechanism is able to provide only salient structures (e.g., *purpose-directed* [47]), since the focusing mechanism of a partial match is not present. This means these systems are unlikely to ever derive a surprising result from an analogy.

A very different approach is taken by Holyoak [44]. In this account, there is no separate stage of structural matching. Instead, analogy is completely driven by the goals of the current problem-solving context. Retrieval of the base domain is driven by an abstract scheme of current problem-solving goals. Creating the mapping is interleaved with other problem-solving activities. This "pragmatic" account, while appealing in some ways, has several crucial limitations. First, the pragmatic model has no account of soundness in terms of systematicity. Without structural consistency, the search space for matching explodes (see below). Second, the pragmatic account can only be defined in problem-solving contexts. Yet analogy is used for purposes other than problem solving, including many contexts in which relevance does not apply. Analogy can be used to explain a new concept and to focus attention on a particular aspect of a situation. Analogy can result in noticing commonalities and conclusions that are totally irrelevant to the purpose at hand. Thus an analogy interpretation algorithm that requires relevance cannot be a general solution [30,31]. Third, psychological data indicates that access is driven by surface similarity, not relevance, as described previously.

We believe the modularity imposed by the Structure-mapping account has several desirable features over the pragmatic account. In the Structure-mapping account, the same match procedure is used for all applications of analogy. For example, in a problem-solving environment, current plans and goals influence what is accessed. Once base and target are both present, the analogy mapping is performed, independently of the particular context. Its results can then be examined and tested as part of the problem-solving process (see [30,31]).

SME demonstrates that an independent, structural matcher can be built which is useful in several tasks and for a variety of examples (over 40 at this writing). By contrast, no clear algorithms have been presented based on the pragmatic account, and published accounts so far [43] describe only two running examples. Another issue is that of potential complexity. The "typical case" bounds we have been able to derive so far are not very precise, and a more complete complexity analysis would certainly be desirable. However, the analysis so far indicates reasonable typical case performance (roughly, $\mathcal{O}(N^2)$), and the empirical results bear this out. Our excellent performance arises from the fact that SME focuses on *local* properties of the representation. On the other hand, the pragmatic account appears to involve arbitrary inference, and arbitrary amounts of knowledge, in the mapping process. Thus we would expect that the average-case computational complexity of a pragmatically oriented matcher will be dramatically worse than SME.

5.1 Matching Algorithms

To our knowledge, SME is unique in that it generates all structurally consistent analogical mappings without search. Previous matchers have utilized heuristic search through the space of possible matches, typically returning a single, best match (e.g., [14,40,49,51,68,69,72,73,74]). Some

researchers on analogy have suggested that generating all possible interpretations is computationally intractable [40,73,49]. Our analysis and empirical results indicate that this conclusion must be substantially modified. Only when structural constraints do not exist, or are ignored, does the computation become intractable. For instance, in [49] the knowledge base was uniform and had no higher-order structure. In such cases exponential explosions are unavoidable.

Winston's original matcher [73] heuristically searched for a single best match. It begins by enumerating all entity pairings and works upward to match relations, thus generating all $N_{Eb}!/(N_{Eb}-N_{Et})!$ possible entity pairings. Because SME only introduces entity pairings when suggested by potential shared relational structure, it typically generates many fewer entity pairings. Some limited amount of pruning due to domain-specific category information was also available on demand, such as requiring that males match with males. By contrast, SME ignores attributes when in analogy mode, unless they play a role in a larger systematic structure. Winston's scoring scheme would attribute one point for each shared relation (e.g., LOVE, CAUSE), property (e.g., STRONG, BEAUTIFUL), and class classification (e.g., A-KIND-OF(?x, woman)). Unlike SME's analogy rules, this scheme makes no distinction between a single, systematic relational chain and a large collection of independent facts.

Winston's later system [74] used *importance-dominated matching*, where certain relationships (such as causal or other constraining relationships) were placed in correspondence first, and helped guide the rest of the match. This is similar in spirit to SME's :intern match constructor rules, which generate hypotheses necessary for structural consistency based on a local hypothesis. However, instead of assembling global solutions from local matches as SME does, Winston's matcher constructed correspondences by heuristic search, guided in part by functions which determine the similarity of parts. The notion of structural consistency was never formalized and exploited as a constraint. However, Winston's system was the first to be tested on a wide variety of examples from several domains, thus setting an important methodological example. It still stands today as the most complete analogical reasoning and learning system, incorporating a model of access, reasoning via precedents, and learning new rules from examples.

Kline's RELAX system [49] focused on matching relations rather than entities. RELAX did not attempt to maintain structural consistency, allowing many-to-one mappings between entities or predicate instances. In conjunction with a semantic network, RELAX was able to match items having quite different syntax (e.g., (Segment A1 A2) matching (Angle A1 X A2)). However, there was no guarantee that the best match would be found due to local pruning during search.

Programs for forming inductive generalizations have also addressed the partial matching problem. These systems use a heuristically pruned search to build up sets of correspondences between terms which are then variablized to form generalized concept descriptions. Since these systems were not designed for analogy, they resemble the operation of SME programmed as a literal graph matcher (e.g., they could not match Pressure to Temperature). Hayes-Roth & McDermott's SPROUTER [40] and Dietrich & Michalski's INDUCE 1.2 [14] possess our restriction of one-to-one consistency in matching. Vere's THOTH system [68,69] uses less stringent match criteria. Once the initial sets of matched terms are built, previously unmatched terms may be added to the match if their constants are in related positions. In the process, THOTH may allow many–to–one mappings between terms.

The usefulness of many–to–one mappings in matches has been discussed in the literature [40,49]. Hayes-Roth & McDermott [40] advocate the need for many-to-one mappings among entities. Kline [49] calls for many-to-one mappings between propositions as well. For example, Kline points out that in trying to match a description of National League baseball to American

League baseball, the statement `(male NLpitcher)` should match both `(male ALpitcher)` and `(male ALdesignatedhitter)`.

Allowing many–to–one mappings undercuts structural consistency, which in our view is central to analogy. Many–to–one mappings appear to be permitted in artistic metaphor, but are not viewed as acceptable by subjects in explanatory, predictive analogies [28,36]. However, we agree that multiple mappings are sometimes useful [11]. We propose that many–to–one mappings should be viewed as multiple analogies between the same base and target. Since SME produces all of the interpretations of an analogy, a postprocessor could keep more than one of them to achieve the advantages of many–to–one mappings, without sacrificing consistency and structural clarity. Thus, in the baseball example, SME would produce an *offense* interpretation and a *defense* interpretation.

5.2 Other Pattern-Matching systems

Clearly Structure-mapping is a form of pattern-matching, but it is different than previous pattern-matchers. For example, it should be clear that Structure-mapping neither subsumes unification nor is subsumed by it. Consider the pair of statements

```
(CAUSE (FLY PERSON1) (FALL PERSON1))
(CAUSE (FLY PERSON2) (FALL PERSON2))
```

These could be part of a legitimate analogy, with `PERSON1` being mapped to `PERSON2`, but these two statements do not unify since `PERSON1` and `PERSON2` are distinct constants. Conversely,

```
(CAUSE (?X PERSON1) (FALL PERSON1))
(CAUSE (FLY ?Y) (FALL ?Z))
```

will unify, assuming `?` indicates variables, with the substitutions:

```
?X ↔ FLY
?Y ↔ PERSON1
?Z ↔ PERSON1
```

However, since Structure-mapping treats variables as constants, these statments fail to be analogous in two ways. First, `FLY` and `?X` are treated as distinct relations, and thus cannot match. Second, `?Y` and `?Z` are considered to be distinct entities, and thus are forbidden to map to the same target item (i.e., `PERSON1`).

Most importantly, the goals of Structure-mapping and unification are completely different. Unification seeks a set of substitutions which makes two statements identical. Structure-mapping seeks a set of correspondences between two descriptions which can suggest additional inferences. Unlike unification, partial matches are perfectly acceptable.

Several of the implementation techniques used in SME are however similar in spirit to those used in *axiomatized unifiers* [4,52,54], which use equational theories (such as associativity and commutativity) to extend equality beyond identicality.

6 Discussion

We have described the Structure-Mapping Engine, a tool-kit for building matchers consistent with Gentner's Structure-mapping theory of analogy and similarity. We have described SME's algorithm in sufficient detail to allow replication by other researchers.[8] SME is both efficient and flexible. A particular matching algorithm is specified by a set of *constructor rules* and *evidence rules*. It produces all structurally consistent interpretations of a match, without backtracking. The interpretations include the candidate inferences suggested by the match and a structural evaluation score, which gives a rough measure of quality. SME has been used both in cognitive simulation studies and a machine learning project. In the cognitive simulation studies, the results so far indicate that SME, when guided with analogy rules, replicates human performance. In the machine learning project (PHINEAS), SME's flexibility provides the means for constructing new qualitative theories to explain observations.

While our complexity analysis indicates that SME's worst-case performance is factorial, the empirical experience is that the typical behavior is much better than that. Importantly, the characteristic which determines efficiency is not size, but the degree of structure of the knowledge. Unlike many AI systems, SME performs better with more systematic, relational descriptions.

In this section we discuss some broader implications of the project, and sketch some of our plans for future work.

6.1 Representational issues

The SME algorithm is of necessity sensitive to the detailed form of the representation, since we are forbidding domain-specific inference in the matching process. Existing AI systems rarely have more than one or two distinct ways to describe any particular situation or theory. But as our programs grow more complex (or as we consider modeling the range and depth of human knowledge) the number of structurally distinct representations for the same situation is likely to increase. For example, a story might be represented at the highest level by a simple classification (i.e., GREEK-TRAGEDY), at an intermediate level by relationships involving the major characters (i.e., (CAUSE (MELTING WAX) FALL23)), and at the lowest level by something like conceptual dependencies. An engineer's knowledge of a calculator might include its functional description, the algorithms it uses, and the axioms of arithmetic expressed in set theory. Unless there is some window of overlap between the levels of description for base and target, no analogy will be found. When our representations reach this complexity, how could SME cope?

There are several possible approaches to this problem. Consider the set of possible representations for a description. Assume these representations can be ordered (at least partially) in terms of degree of abstraction. If two descriptions are too abstract, there will either be no predicate overlap (e.g., GREEK-TRAGEDY versus SHAKESPEARE-DRAMA) or identity (e.g., TRAGEDY versus TRAGEDY). On the other hand, if two descriptions are greatly detailed, there can be too many spurious, inconsequential matches (e.g., describing the actions of characters every microsecond). The problem is to find levels of description which provide useful analogies. We believe one solution is to invoke SME repeatedly, using knowledge of the definitions of predicates to "slide" the base or target descriptions up or down in the space of possible representations appropriately.

[8] SME is publically available for interested researchers. There is a manual available [18] which provides extensive implementation-level details and interface information.

An orthogonal consideration is the degree of systematicity. Worst-case behavior tends to occur when representations are large and relatively flat. Changes in representation can make large differences. For example, a PHINEAS problem which took SME 53 minutes was reduced to 34 seconds by imposing more systematic structure. We are currently exploring these trade-offs to formulate more precise constraints on useful representations for analogical reasoning and learning.

6.2 Addressing the Combinatorics

As we have shown, SME is $\mathcal{O}(N^2)$ except for the last critical merge step, which has $\mathcal{O}(N!)$ worst-case performance. Our experience with both small (11 expressions) and large (71 expressions) domain descriptions indicates that performance is more a function of representation and repetitiveness rather than a function of size. We have found that even moderately structural domain descriptions produce excellent performance. However, in practice it is not always convenient to avoid traditional, flat domain representations. For example, SME is unable to duplicate Kline's baseball analogy [49] within a reasonable amount of time (i.e., hours). This is due to his flat description of the domain (e.g., `(MALE catcher)`, `(BATS left-fielder)`, `(BATS center-fielder)`, etc.). Thus for some cases, generating all possible interpretations of an analogy may be prohibitive. Previous analogy programs used matching algorithms that are specifically designed around heuristic search mechanisms. SME offers a clean line between generating all possibilities and imposing heuristic limitations. If we stop after the first merge step, SME provides an $\mathcal{O}(N^2)$ algorithm for generating the complete set of initial gmaps! The subsequent merge steps could then be heuristically driven through a limited search procedure (e.g., beam-search, best-first, etc.) to produce the best or N best maximal interpretations. Alternatively, we could retain the current SME design (recall that the second merge step is required to support candidate inference generation and is almost always $\mathcal{O}(N^2)$ or better) and simply drop the troublesome third merge step. This is an (unused) option that the current implementation provides. We have not yet explored the ramifications of dropping merge step 3, although work with PHINEAS has indicated the need for the maximality criterion in practice.

In the next sections, we discuss the potential for parallel versions of the SME algorithm. In particular, we argue that (1) there are many opportunities for parallel speedup, and (2) the expensive merge steps can be eliminated in principle.

6.2.1 Medium-grained Parallel Architectures

We begin by examining each stage of the algorithm to see how it might be decomposed into parallel operations, and what kinds of speedups might result. First we assume a software architecture that allows tasks to be spawned for parallel execution (such as [1]), and we ignore communications and setup costs.

Constructing Match Hypotheses All `:filter` rules can be run independently, giving rise to $\mathcal{O}(N^2)$ tasks. With enough processors this could be done in constant time, assuming the Structure-Mapping match constructor rules. Each `:intern` rule can be run on every match hypothesis as it gets created. Since these rules can in turn create new match hypotheses, but only involving an expression's arguments, the best speed-up would be roughly the log of the input.

Computing *Conflicting*, *Emaps*, and *NoGood* sets The *Conflicting* set computation is completely local. It could either be organized around each base or target item, or around pairs of match hypotheses. Finding the *Emaps* and *NoGood* sets require propagation of results upwards, and hence again will take log time.

Merge Step 1: Form initial combinations Recall that this step starts from the roots of the match hypothesis graph, adding the subgraph to the list of gmaps if the hypothesis is not inconsistent and recursing on its offspring otherwise. The results from each root are independent, and so may be done as separate tasks. If each recursive step spawns a new process to handle each offspring, then the minimum time is proportional again to the order of the highest root in the graph.

Merge Step 2: Combine dependent but unconnected gmaps Recall that this step combines initial gmaps which share common base structure and are not inconsistent when taken together. This procedure can be carried out bottom-up, merging pairs which share base structure and are consistent together and then recursing on the results. The computation time will be logarithmic in the number of gmaps.

Merge Step 3: Combine independent collections This can be performed like the previous step, but skipping pairs of gmaps that have common structure (since they would have been merged previously and hence must be inconsistent). Again, with enough processors the time is bounded by the log of the number of gmaps. However, since the number of gmaps is in the worst case factorial, the number of tasks required could become rather large.

This cursory analysis no doubt glosses over several problems lurking in creating a highly parallel version of the SME algorithm. However, we believe such algorithms could be very promising.

SME's simplicity also raises another interesting experimental possibility. Given that currently many medium-grain parallel computers are being built with reasonable amounts of RAM and a lisp environment on each machine, one can imagine simply loading a copy of SME into each processor. Access experiments, for example, would be greatly sped up by allowing a pool of SME's to work over the knowledge base in a distributed fashion.

6.2.2 Connectionist Architectures

Another interesting approach would be to only generate a single, best gmap while still maintaining SME's "no search" policy. The problem of choosing among all possible interpretations in analogy processing is very much like choosing among possible interpretations of the sentence "John shot two bucks" in natural language processing. A "no search" solution to this natural language problem was provided by the connectionist work of Waltz and Pollack [71]. Rather than explicitly constructing all possible sentence interpretations and then choosing the best one, Waltz and Pollack used their networks to implicitly represent all of the possible choices. Given a particular network, spreading activation and lateral inhibition were used to find the single best interpretation. This work in fact inspired the use of the BMS for representing evidential relationships and helped motivate the decomposition of the processing into the local/global steps.

Consider the network produced by SME prior to the gmap merge steps (shown in Figure 5). Some match hypotheses support each other (grounding criterion) while others inhibit each other (*Conflicting* relations). Viewing this as a spreading activation, lateral inhibition network, it appears

that standard connectionist relaxation techniques could be used to produce a "best" interpretation without explicitly generating all gmaps. Furthermore, it may be possible to generate the second-best, third-best, etc. interpretations on demand by inhibiting the nodes of the best interpretation, forcing the second best to rise. Thus SME would be able to establish a global interpretation simply as an indirect consequence of the establishment of local structural consistency and systematicity. This would eliminate the single most expensive computation of the SME algorithm. By eliminating explicit generation of all gmaps, the complexity of the algorithm could drop to the $\mathcal{O}(N^2)$ required to generate the connectionist network.

6.3 Future Work

6.3.1 Cognitive Simulation

We are conducting additional cognitive simulation studies of analogical reasoning, memory, and learning involving SME. One line of experiments concerns the development of analogical reasoning. Psychological research shows a marked developmental shift in analogical processing. Young children rely on surface information in analogical mapping; at older ages, systematic mappings are preferred [34,35,45,70]. Further, there is some evidence that a similar shift from surface to systematic mappings occurs in the novice-expert transition in adults [8,50,57,58].

In both cases there are two very different interpretations for the analogical shift: (1) acquisition of knowledge; or (2) a change in the analogy algorithm. The knowledge-based interpretation is that children and novices lack the necessary relational structures to guide their analogizing. The second explanation is that the algorithm for analogical mapping changes, either due to maturation or learning. In human learning it is difficult to decide this issue, since exposure to domain knowledge and practice in analogy and reasoning tend to occur simultaneously. SME gives us a unique opportunity to vary independently the analogy algorithm and the amount and kind of domain knowledge. For example, we can compare identical evaluation algorithms operating on novice versus expert representations, or we can compare different analogy evaluation rules operating on the same representation.

There are two problems with our current structural evaluation score computation. First, there are several other structural properties which should enter into the SES, such as the number and size of connected components, the existence and structure of the candidate inferences. Second, it is not normalized with respect to the sizes of the base and target domains. The current SES can be used to compare matches of different bases to the same target, or different targets to the same base. But it cannot be used to compare two completely different analogies (i.e., different bases and different targets). Janice Skorstad is building a programmable *structural evaluator* module that will let us experiment with these factors and different normalization schemes [64]. We suspect that being able to tune the structural evaluation criteria might allow us to model individual differences in analogical processing. For example, a conservative strategy might favor taking gmaps with some candidate inferences but not too many, in order to maximize the probability of being correct.

We are also exploring ways to reduce the potential for tailorability in the process of translating descriptions provided as experimental stimuli for human subjects into formal representations for SME input. For example, Janice Skorstad is creating a graphical editor for producing graphical figures for experimental stimuli. One output of the editor is a picture which can be used as a stimulus for psychological experiments. The other output is a set of symbolic assertions with numerical parameters, which is expanded into SME input by a simple inference engine that calculates spatial

relationships, such as `INSIDE` or `LEFT-OF`. Inspired by Winston's use of a pidgin-English parser for input [74], we are also seeking a parser that, perhaps in conjunction with a simple inference engine, can produce useful descriptions of stories.

6.3.2 Machine Learning Studies

Falkenhainer's `PHINEAS` program is part of the *Automated Physicist Project* at the University of Illinois. This project, led by Forbus and Gerald DeJong, is building a collection of programs that use qualitative and quantitative techniques for reasoning and learning about the physical world. DeJong and his students have built several programs that use Explanation-Based Learning [12,13] to acquire knowledge of the physical world [61,55]. Forbus' group has developed a number of useful qualitative reasoning programs [24,25,42] which can be used in learning projects (as `PHINEAS` demonstrates). By combining these results, we hope to build systems that can reason about a wide range of physical phenomena and learn both from observation and by being taught.

7 Acknowledgements

The authors wish to thank Janice Skorstad, Danny Bobrow, and Steve Chien for helpful comments on prior drafts of this paper. Janice Skorstad provided invaluable assistance in encoding domain models. Alan Frisch provided pointers into the unification literature.

This research is supported by the Office of Naval Research, Contract No. N00014-85-K-0559. Additional support has been provided by IBM, both in the form of a Graduate Fellowship for Falkenhainer and a Faculty Development award for Forbus. The equipment used in this research was provided by an equipment grant from the Information Sciences Division of the Office of Naval Research, and from a gift from Texas Instruments.

References

[1] Allen, D., Steinberg, S. and Stabile, L. Recent developments in Butterfly Lisp. *Proceedings of AAAI-87*, Seattle, 1987.

[2] Anderson, J., *The Architecture of Cognition*, Harvard University Press, Cambridge, Mass, 1983.

[3] Buckley, S., *Sun up to sun down*, McGraw-Hill Company, New York, 1979.

[4] Bundy, A., *The computer modelling of mathematical reasoning*, Academic Press, 1983.

[5] Burstein, M., Concept formation by incremental analogical reasoning and debugging, in: *Proceedings of the Second International Workshop on Machine Learning*, University of Illinois, Monticello, Illinois, June, 1983. A revised version appears in *Machine Learning: An Artificial Intelligence Approach Vol. II*, R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (Eds.), Morgan Kaufman, 1986.

[6] Carbonell, J.G., Learning by Analogy: Formulating and generalizing plans from past experience, in: *Machine Learning: An Artificial Intelligence Approach*, R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (Eds.), Morgan Kaufman, 1983.

[7] Carbonell, J.G., Derivational analogy in problem solving and knowledge acquisition, in: *Proceedings of the Second International Machine Learning Workshop*, University of Illinois, Monticello, Illinois, June, 1983. A revised version appears in *Machine Learning: An Artificial Approach Vol. II*, R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (Eds.), Morgan Kaufman, 1986.

[8] Chi, M.T.H., R. Glaser, E. Reese, Expertise in problem solving. In R. Sternberg (Ed.), *Advances in the psychology of human intelligence* (Vol. 1). Hillsdale, N.J., Erlbaum, 1982.

[9] Clement, J. Analogy generation in scientific problem solving. *Proceedings of the third annual meeting of the Cognitive Science Society*, 1981.

[10] Clement, J. Analogical reasoning patterns in expert problem solving. *Proceedings of the fourth annual meeting of the Cognitive Science Society*, 1982.

[11] Collins, A.M., & Gentner, D. How people construct mental models. In D. Holland and N. Quinn (Eds.) *Cultural models in language and thought.* Cambridge, England: Cambridge University, 1987.

[12] DeJong, G. Generalizations based on explanations. *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, August, 1981

[13] DeJong, G., and Mooney, R. Explanation-based Learning: An alternative view. *Machine Learning*, Volume 1, No. 2, 1986

[14] Diettrich, T., & Michalski, R.S., Inductive learning of structural descriptions: evaluation criteria and comparative review of selected methods, *Artificial Intelligence* **16**, 257-294, 1981.

[15] Falkenhainer, B., Towards a general-purpose belief maintenance system, in: J.F. Lemmer (Ed.), *Uncertainty in Artificial Intelligence, Volume II*, 1987. Also Technical Report, UIUCDCS-R-87-1717, Department of Computer Science, University of Illinois, 1987.

[16] Falkenhainer, B., An examination of the third stage in the analogy process: Verification-Based Analogical Learning, Technical Report UIUCDCS-R-86-1302, Department of Computer Science, University of Illinois, October, 1986. A summary appears in *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, August, 1987.

[17] Falkenhainer, B., Scientific theory formation through analogical inference, *Proceedings of the Fourth International Machine Learning Workshop*, Irvine, CA, June, 1987.

[18] Falkenhainer, B., The SME user's manual, Technical Report UIUCDCS-R-88-1421, Department of Computer Science, University of Illinois, 1988.

[19] Falkenhainer, B., Learning from Physical Analogies: An adaptive approach to understanding physical observations, Ph.D. Thesis, University of Illinois, (in preparation).

[20] Falkenhainer, B., K.D. Forbus, D. Gentner, The Structure-Mapping Engine, *Proceedings of the Fifth National Conference on Artificial Intelligence*, August, 1986.

[21] Forbus, K.D., "Qualitative Reasoning about Physical Processes", *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, August, 1981.

[22] Forbus, K.D., Qualitative Process Theory, *Artificial Intelligence* **24**, 1984.

[23] Forbus, K.D., Qualitative Process Theory, Technical Report No. 789, MIT Artificial Intelligence Laboratory, July, 1984.

[24] Forbus, K. Interpreting measurements of physical systems, in: *Proceedings of the Fifth National Conference on Artificial Intelligence*, August, 1986.

[25] Forbus, K. The Qualitative Process Engine, Technical Report UIUCDCS-R-86-1288, Department of Computer Science, University of Illinois, December, 1986.

[26] Forbus, K.D. and D. Gentner, Learning Physical Domains: Towards a theoretical framework, In *Proceedings of the Second International Machine Learning Workshop*, University of Illinois, Monticello, Illinois, June, 1983. A revised version appears in *Machine Learning: An Artificial Approach Vol. II*, R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (Eds.), Morgan Kaufmann, 1986.

[27] Gentner, D., The structure of analogical models in science, BBN Tech. Report No. 4451, Cambridge, MA., Bolt Beranek and Newman Inc., 1980.

[28] Gentner, D., Are scientific analogies metaphors?, in: Miall, D., *Metaphor: Problems and Perspectives*, Harvester Press, Ltd., Brighton, England, 1982.

[29] Gentner, D., Structure-mapping: A theoretical framework for analogy, *Cognitive Science* **7**(2), 1983.

[30] Gentner, D., Mechanisms of analogy. To appear in S. Vosniadou and A. Ortony, (Eds.), *Similarity and analogical reasoning.* Presented in June, 1986.

[31] Gentner, D., Analogical inference and analogical access, in A. Preiditis (Ed.), *Analogica: Proceedings of the First Workshop on Analogical Reasoning*, London, Pitman Publishing Co., 1988 Presented in December, 1986.

[32] Gentner, D., & D.R. Gentner, Flowing waters or teeming crowds: Mental models of electricity, In D. Gentner & A.L. Stevens, (Eds.), *Mental Models*, Erlbaum Associates, Hillsdale, N.J., 1983.

[33] Gentner, D., & R. Landers, Analogical reminding: A good match is hard to find. In *Proceedings of the International Conference on Systems, Man and Cybernetics.* Tucson, Arizona, 1985.

[34] Gentner, D. Metaphor as structure-mapping: The relational shift. *Child Development*, **59**, 47-59, 1988.

[35] Gentner, D., & C. Toupin, Systematicity and Surface Similarity in the Development of Analogy, *Cognitive Science*, 1986.

[36] Gentner, D., Falkenhainer, B., & Skorstad, J. Metaphor: The good, the bad and the ugly. *Proceedings of the Third Conference on Theoretical Issues in Natural Language Processing*, Las Cruces, New Mexico, January, 1987.

[37] Ginsberg, M.L., Non-Monotonic reasoning using Dempster's rule, *Proceedings of the Fourth National Conference on Artificial Intelligence*, August, 1984.

[38] Greiner, R., Learning by understanding analogies, *Artificial Intelligence* **35** (1), 81-125, 1988.

[39] Hall, R. Computational approaches to analogical reasoning: A comparative analysis. To appear in *Artificial Intelligence.*

[40] Hayes-Roth, F., McDermott, J. An interference matching technique for inducing abstractions, *Communications of the ACM*, **21**(5), May, 1978.

[41] Hofstadter, D.R., The Copycat project: An experiment in nondeterministic and creative analogies. M.I.T. A.I. Laboratory memo 755. Cambridge, Mass: M.I.T., 1984.

[42] Hogge, J. Compiling plan operators from domains expressed in qualitative process theory, *Proceedings of the Sixth National Conference on Artificial Intelligence*, Seattle, WA, July, 1987.

[43] Holland, J.H., Holyoak, K.J., Nisbett, R.E., & Thagard, P., *Induction: Processes of inference, learning, and discovery*, 1987.

[44] Holyoak, K.J. The pragmatics of analogical transfer. In G.H. Bower (Ed.), *The psychology of learning and motivation. Vol. I.* New York: Academic Press, 1984.

[45] Holyoak, K.J., E.N. Juin, D.O. Billman (in press). Development of analogical problem-solving skill. *Child Development.*

[46] Indurkhya, B., "Constrained Semantic Transference: A formal theory of metaphors," Technical Report 85/008, Boston University, Department of Computer Science, October, 1985.

[47] Kedar-Cabelli, S., Purpose-Directed Analogy. *Proceedings of the Seventh Annual Conference of the Cognitive Science Society*, Irvine, CA, 1985.

[48] Kedar-Cabelli, S. T. (in press). Analogy: From a unified perspective. To appear in D. H. Helman (Ed.), *Analogical reasoning: Perspectives of artificial intelligence, cognitive science, and philosophy.* Dordrecht, Nolland: D. Reidel Publishing Company.

[49] Kline, P.J., "Computing the similarity of structured objects by means of a heuristic search for correspondences", Ph.D. Thesis, Department of Psychology, University of Michigan, 1983.

[50] Larkin, J.H. Problem representations in physics. In D. Gentner & A.L. Stevens (Eds.) *Mental Models.* Hillsdale, N.J., Lawrence Erlbaum Associates, 1983.

[51] Michalski, R.S., "Pattern recognition as rule-guided inductive inference" *IEEE Transactions on Pattern Analysis and Machine Intelligence* **2**(4), pp. 349-361, 1980.

[52] Plotkin, G.D., Building in equational theories, in: *Machine Intelligence 7*, Meltzer, B. & Michie, D. (Eds.), John Wiley & Sons, 1972.

[53] Prade, H., "A synthetic view of approximate reasoning techniques," *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, 1983.

[54] Raulefs, P., Siekmann J., Szabo, P., & Unvericht, E., A short survey on the state of the art in matching and unification problems, *ACM SIGSAM Bulletin* **13**(2), 14-20, May, 1979.

[55] Rajamoney, S., DeJong, G., and Faltings, B. Towards a model of conceptual knowledge acquisition through directed experimentation. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, CA, August, 1985.

[56] Rattermann, M.J., and Gentner, D. Analogy and Similarity: Determinants of accessibility and inferential soundness, *Proceedings of the Cognitive Science Society*, July, 1987.

[57] Reed, S.K., A Structure-mapping model for word problems. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, **13**(1), 124–139, 1987.

[58] Ross, B.H., Remindings and their effects in learning a cognitive skill, *Cognitive Psychology*, **16**, 371-416, 1984.

[59] Rumelhart, D.E., & Norman, D.A., Analogical processes in learning. In J.R. Anderson (Ed.), *Cognitive skills and their acquisition*, Hillsdale, N.J., Erlbaum, 1981.

[60] Shafer, G., *A mathematical theory of evidence*, Princeton University Press, Princeton, New Jersey, 1976.

[61] Shavlik, J.W. Learning about momentum conservation. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, CA, August, 1985

[62] Stickel, M. A complete unification algorithm for associative-commutative functions, in: *Proceedings of IJCAI-75*, Tbilisi, Georgia, USSR, 71-76, 1975.

[63] Tversky, A. Representation of structure in similarity data: Problems and prospects. *Psychometrika* **39**, 373-421, 1974.

[64] Skorstad, J., A structural approach to abstraction processes during concept learning, Master's Thesis, 1988.

[65] Skorstad, J., Falkenhainer, B., Gentner, D., Analogical Processing: A simulation and empirical corroboration, in: *Proceedings of the Sixth National Conference on Artificial Intelligence*, Seattle, WA, August, 1987.

[66] Van Lehn, K. & J.S. Brown, Planning nets: A representation for formalizing analogies and semantic models of procedural skills. In R.E. Snow, P.A. Federico & W.E. Montague (Eds.), *Aptitude, learning and instruction: Cognitive process analyses.* Hillsdale, N.J. Erlbaum, 1980.

[67] Van Lehn, K., "Felicity conditions for human skill acquisition: Validating an AI-based theory," Xerox Palo Alto Research Center Technical Report CIS-21, 1983.

[68] Vere, S., "Induction of concepts in the predicate calculus", *Proceedings of the Fourth International Joint Conference on Artificial Intelligence*, 1975.

[69] Vere, S., "Inductive learning of relational productions", In Waterman & Hayes-Roth (Eds.), *Pattern-Directed Inference Systems*, 1978.

[70] Vosniadou, S., On the development of metaphoric competence. University of Illinois: Manuscript submitted for publication, 1985.

[71] Waltz, D.L. & Pollack, J.B., Massively Parallel Parsing: A strongly interactive model of natural language interpretation, *Cognitive Science* **9**, 51-74, 1985.

[72] Winston, P.H., Learning structural descriptions from examples, Ph.D. thesis, Report AI-TR-231, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, 1970.

[73] Winston, P.H., Learning and Reasoning by Analogy, *Communications of the ACM*, **23**(12), 1980.

[74] Winston, P.H., Learning new principles from precedents and exercises, *Artificial Intelligence*, **19**, 321-350, 1982.

A SME Match Rules

The construction of a match is guided by a set of *match rules* that specify which expressions and entities in the base and target might match and estimate the believability of each possible component of a match. In our experiments using SME, we currently use three types of rule sets, *literal similarity*, *analogy*, and *mere appearance*.

A.1 Literal Similarity (LS) Rules

The *literal similarity* rules look at both relations and object descriptions.

```
;;;; Define MH constructor rules

;; If predicates are the same, match them

(MHC-rule (:filter ?b ?t :test (eq (expression-functor ?b) (expression-functor ?t)))
    (install-MH ?b ?t))

;; Intern rule for non-commutative predicates - corresponding arguments only.
;; Match compatible arguments of already matched items

(MHC-rule (:intern ?b ?t :test (and (expression? ?b) (expression? ?t)
                                    (not (commutative? (expression-functor ?b)))
                                    (not (commutative? (expression-functor ?t)))))
    (do ((bchildren (expression-arguments ?b) (cdr bchildren))
         (tchildren (expression-arguments ?t) (cdr tchildren)))
        ((or (null bchildren) (null tchildren)))
      (cond ((and (entity? (first bchildren)) (entity? (first tchildren)))
             (install-MH (first bchildren) (first tchildren)))
            ((and (function? (expression-functor (first bchildren)))
                  (function? (expression-functor (first tchildren))))
             (install-MH (first bchildren) (first tchildren))))))

;; Intern rule for commutative predicates - any "compatible" arguments, regardless of order.
;; Match compatible arguments of already matched items

(MHC-rule (:intern ?b ?t :test (and (expression? ?b) (expression? ?t)
                                    (commutative? (expression-functor ?b))
                                    (commutative? (expression-functor ?t))))
    (dolist (bchild (expression-arguments ?b))
      (dolist (tchild (expression-arguments ?t))
        (cond ((and (entity? bchild) (entity? tchild))
               (install-MH bchild tchild))
              ((and (function? (expression-functor bchild)) (function? (expression-functor tchild)))
               (install-MH bchild tchild))))))

;;;; Define MH evidence rules

;; having the same functor is a good sign
;;

(assert! same-functor)

(rule ((:intern (MH ?b ?t) :test (and (expression? ?b) (expression? ?t)
                                      (eq (expression-functor ?b) (expression-functor ?t)))))
   (if (function? (expression-functor ?b))
       (assert! (implies same-functor (MH ?b ?t) (0.2 . 0.0)))
       (assert! (implies same-functor (MH ?b ?t) (0.5 . 0.0)))))
```

```
;;check children (arguments) match potential
;;

(initial-assertion (assert! 'arguments-potentially-match))

(rule ((:intern (MH ?b ?t) :test (and (expression? ?b) (expression? ?t))))
   (if (children-match-potential ?b ?t)
       (assert! (implies arguments-potentially-match (MH ?b ?t) (0.4 . 0.0)))
       (assert! (implies arguments-potentially-match (MH ?b ?t) (0.0 . 0.8)))))

;;if their order is similar, this is good. If the item is a function,
;; ignore since order comparisons give false support here.

(initial-assertion (assert! 'order-similarity))

(rule ((:intern (MH ?b ?t) :test (and (expression? ?b) (expression? ?t)
                                      (not (function? (expression-functor ?b)))
                                      (not (function? (expression-functor ?t))))))
   (cond ((= (expression-order ?b) (expression-order ?t))
          (assert! (implies order-similarity (MH ?b ?t) (0.3 . 0.0))))
         ((or (= (expression-order ?b) (1+ (expression-order ?t)))
              (= (expression-order ?b) (1- (expression-order ?t))))
          (assert! (implies order-similarity (MH ?b ?t) (0.2 . 0.05))))))

;;propagate evidence down - systematicity
;; support for the arg will be 0.8 of the current support for the parent

(rule ((:intern (MH ?b1 ?t1) :test (and (expression? ?b1) (expression? ?t1)
                                        (not (commutative? (expression-functor ?b1)))))
       (:intern (MH ?b2 ?t2) :test (children-of? ?b2 ?t2 ?b1 ?t1)))
    (sme:assert! (implies (MH ?b1 ?t1) (MH ?b2 ?t2) (0.8 . 0.0))))


(rule ((:intern (MH ?b1 ?t1) :test (and (expression? ?b1) (expression? ?t1)
                                        (commutative? (expression-functor ?b1))))
       (:intern (MH ?b2 ?t2) :test (and (member ?b2 (expression-arguments ?b1) :test #'eq)
                                        (member ?t2 (expression-arguments ?t1) :test #'eq))))
    (sme:assert! (implies (MH ?b1 ?t1) (MH ?b2 ?t2) (0.8 . 0.0))))

;;;; Gmap rules

;; Support from its MH's. At this time we ignore other expressionors such as number
;; of candidate inferences, etc.

(rule ((:intern (GMAP ?gm)))
   (dolist (mh (gm-elements ?gm))
      (assert! `(implies ,(mh-form mh) (GMAP ?gm)))))
```

A.2 Analogy (AN) Rules

The *analogy* rules prefer systems of relations and discriminate against object descriptions. The analogy evidence rules are identical to the literal similarity evidence rules and are not repeated here. The match constructor rules only differ in their check for attributes:

```
;;;; Define MH constructor rules
```

```
;; If predicates are the same, match them

(MHC-rule (:filter ?b ?t :test (and (eq (expression-functor ?b) (expression-functor ?t))
                                    (not (attribute? (expression-functor ?b)))))
    (install-MH ?b ?t))

;; Match compatible arguments of already matched items.
;; Notice attributes are allowed to match here, since they are part of some higher relation that matched.

;; Intern rule for non-commutative predicates - corresponding arguments only.

(MHC-rule (:intern ?b ?t :test (and (expression? ?b) (expression? ?t)
                                    (not (commutative? (expression-functor ?b)))
                                    (not (commutative? (expression-functor ?t)))))
    (do ((bchildren (expression-arguments ?b) (cdr bchildren))
         (tchildren (expression-arguments ?t) (cdr tchildren)))
        ((or (null bchildren) (null tchildren)))
      (cond ((and (entity? (first bchildren)) (entity? (first tchildren)))
             (install-MH (first bchildren) (first tchildren)))
            ((and (function? (expression-functor (first bchildren)))
                  (function? (expression-functor (first tchildren))))
             (install-MH (first bchildren) (first tchildren)))
            ((and (attribute? (expression-functor (first bchildren)))
                  (eq (expression-functor (first bchildren)) (expression-functor (first tchildren))))
             (install-MH (first bchildren) (first tchildren))))))

;; Intern rule for commutative predicates - any "compatible" arguments, not necessarily corresponding.

(MHC-rule (:intern ?b ?t :test (and (expression? ?b) (expression? ?t)
                                    (commutative? (expression-functor ?b))
                                    (commutative? (expression-functor ?t))))
    (dolist (bchild (expression-arguments ?b))
      (dolist (tchild (expression-arguments ?t))
        (cond ((and (entity? bchild) (entity? tchild))
               (install-MH bchild tchild))
              ((and (function? (expression-functor bchild))
                    (function? (expression-functor tchild)))
               (install-MH bchild tchild))
              ((and (attribute? (expression-functor bchild))
                    (eq (expression-functor bchild) (expression-functor tchild)))
               (install-MH bchild tchild))))))
```

A.3 Mere Appearance (MA) Rules

The *mere appearance* rules focus on object descriptions and prevent matches between functions or relations. As a result, the number of evidence rules is greatly reduced.

```
;;;; Define MH constructor rules

(MHC-rule (:filter ?b ?t :test (and (eq (expression-functor ?b) (expression-functor ?t))
                                    (<= (expression-order ?b) 1)
                                    (<= (expression-order ?t) 1)))
    (install-MH ?b ?t))

(MHC-rule (:intern ?b ?t :test (and (expression? ?b) (expression? ?t)
```

```
                                    (not (commutative? (expression-functor ?b)))
                                    (not (commutative? (expression-functor ?t)))))
    (do ((bchildren (expression-arguments ?b) (cdr bchildren))
         (tchildren (expression-arguments ?t) (cdr tchildren)))
        ((or (null bchildren) (null tchildren)))
      (if (and (entity? (first bchildren)) (entity? (first tchildren)))
          (install-MH (first bchildren) (first tchildren)))))

(MHC-rule (:intern ?b ?t :test (and (expression? ?b) (expression? ?t)
                                    (commutative? (expression-functor ?b))
                                    (commutative? (expression-functor ?t))))
    (dolist (bchild (expression-arguments ?b))
      (dolist (tchild (expression-arguments ?t))
        (if (and (entity? bchild) (entity? tchild))
           (install-MH bchild tchild)))))
```

;;;; Define MH evidence rules

;; having the same functor is a good sign

```
(initial-assertion (assert! 'same-functor))

(rule ((:intern (MH ?b ?t) :test (and (expression? ?b) (expression? ?t)
                                      (eq (expression-functor ?b) (expression-functor ?t)))))
       (cond ((attribute? (expression-functor ?b))
              (assert! (implies same-functor (MH ?b ?t) (0.5 . 0.0))))
             ((= 1 (max (expression-order ?b) (expression-order ?t)))
              (assert! (implies same-functor (MH ?b ?t) (0.4 . 0.0))))))
```

;;propagate evidence down - only for entity MH's caused by attribute pairings
;; support for the arg will be 0.9 of the current support for the parent

```
(rule ((:intern (MH ?b1 ?t1) :test (and (expression? ?b1) (expression? ?t1)
                                        (<=  (max (expression-order ?b1)(expression-order ?t1))  1)
                                        (not (commutative? (expression-functor ?b1)))))
       (:intern (MH ?b2 ?t2) :test (children-of? ?b2 ?t2 ?b1 ?t1)))
    (sme:assert! (implies (MH ?b1 ?t1) (MH ?b2 ?t2) (0.9 . 0.0))))


(rule ((:intern (MH ?b1 ?t1) :test (and (expression? ?b1) (expression? ?t1)
                                        (<=  (max (expression-order ?b1)(expression-order ?t1))  1)
                                        (commutative? (expression-functor ?b1))))
       (:intern (MH ?b2 ?t2) :test (and (member ?b2 (expression-arguments ?b1) :test #'eq)
                                        (member ?t2 (expression-arguments ?t1) :test #'eq))))
    (sme:assert! (implies (MH ?b1 ?t1) (MH ?b2 ?t2) (0.9 . 0.0))))
```

;;;; Gmap rules

;;; Support from its MH's. At this time we ignore other expressionors such as number of candidate inferences

```
(rule ((:intern (GMAP ?gm)))
   (dolist (mh (gm-elements ?gm))
      (assert! `(implies ,(mh-form mh) (GMAP ?gm)))))
```

B Sample Domain Descriptions

In this section we show the domain descriptions given to SME for the described examples.

B.1 Simple Water Flow - Heat Flow

Water Flow

```
(defEntity water :type inanimate)
(defEntity beaker :type inanimate)
(defEntity vial :type inanimate)
(defEntity pipe :type inanimate)


(defDescription simple-water-flow
    entities (water beaker vial pipe)
    expressions  (((flow beaker vial water pipe) :name wflow)
            ((pressure beaker) :name pressure-beaker)
            ((pressure vial)   :name pressure-vial)
            ((greater pressure-beaker pressure-vial) :name >pressure)
            ((greater (diameter beaker) (diameter vial)) :name >diameter)
            ((cause >pressure wflow) :name cause-flow)
            (flat-top water)
            (liquid water)))
```

Heat Flow

```
(defEntity coffee :type inanimate)
(defEntity ice-cube :type inanimate)
(defEntity bar :type inanimate)
(defEntity heat :type inanimate)

(defDescription simple-heat-flow
    entities (coffee ice-cube bar heat)
    expressions  (((flow coffee ice-cube heat bar) :name hflow)
            ((temperature coffee) :name temp-coffee)
            ((temperature ice-cube) :name temp-ice-cube)
            ((greater temp-coffee temp-ice-cube) :name >temperature)
            (flat-top coffee)
            (liquid coffee)))
```

B.2 Solar-System - Rutherford Atom

Solar System

```
(defEntity sun :type inanimate)
(defEntity planet :type inanimate)

(defDescription solar-system
    entities (sun planet)
    expressions  (((mass sun) :name mass-sun)
            ((mass planet) :name mass-planet)
            ((greater mass-sun mass-planet) :name >mass)
            ((attracts sun planet) :name attracts)
            ((revolve-around planet sun) :name revolve)
            ((and >mass attracts) :name and1)
            ((cause and1 revolve) :name cause-revolve)
            ((temperature sun) :name temp-sun)
            ((temperature planet) :name temp-planet)
```

```
            ((greater temp-sun temp-planet) :name >temp)
            ((gravity mass-sun mass-planet) :name force-gravity)
            ((cause force-gravity attracts) :name why-attracts)))
```

Rutherford Atom

```
(defEntity nucleus :type inanimate)
(defEntity electron :type inanimate)

(defDescription rutherford-atom
    entities (nucleus electron)
    expressions  (((mass nucleus) :name mass-n)
            ((mass electron) :name mass-e)
            ((greater mass-n mass-e) :name >mass)
            ((attracts nucleus electron) :name attracts)
            ((revolve-around electron nucleus) :name revolve)
            ((charge electron) :name q-electron)
            ((charge nucleus) :name q-nucleus)
            ((opposite-sign q-nucleus q-electron) :name >charge)
            ((cause >charge attracts) :name why-attracts)))
```

B.3 Karla Stories

Zerdia the eagle - base story

```
(defEntity Karla)
(defEntity hunter)
(defEntity feathers)
(defEntity cross-bow)
(defEntity Failed)
(defEntity high)

(defDescription base-5
    entities  (Karla hunter feathers cross-bow Failed high)
    expressions  (((bird Karla)    :name bird-Karla)
            ((person hunter) :name person-hunter)
            ((warlike hunter)        :name warlike-hunter)
            ((Karlas-asset feathers) :name feathers-asset)
            ((weapon cross-bow)      :name weapon-bow)
            ((used-for feathers cross-bow ) :name has-feathers)
            ((not has-feathers)      :name not-has-feathers)
            ((attack hunter Karla)   :name attack-hunter)
            ((not attack-hunter)     :name not-attack)
            ((see Karla hunter)      :name see-Karla)
            ((follow see-Karla attack-hunter) :name follow-see-attack)
            ((success attack-hunter) :name success-attack)
            ((equals success-attack Failed)       :name failed-attack)
            ((cause not-has-feathers failed-attack)  :name cause-failed-attack)
            ((desire hunter feathers)        :name desire-feathers)
            ((realize Karla desire-feathers) :name realize-desire)
            ((follow failed-attack realize-desire)    :name follow-realize)
            ((offer Karla feathers hunter)   :name offer-feathers)
            ((cause realize-desire offer-feathers)   :name cause-offer)
            ((obtain hunter feathers)  :name take-feathers)
            ((cause offer-feathers take-feathers)    :name cause-take)
```

```
            ((happiness hunter)      :name happiness-hunter)
            ((equals happiness-hunter high)  :name happy-hunter)
            ((cause take-feathers happy-hunter)      :name cause-happy)
            ((promise hunter Karla not-attack)       :name promise-hunter)
            ((cause happy-hunter promise-hunter)    :name cause-promise)))
```

Zerdia the country - TA5

```
(defEntity Zerdia)
(defEntity Gagrach)
(defEntity supercomputer)
(defEntity missiles)
(defEntity failed)
(defEntity high)

(defDescription ta-5
    entities   (Zerdia Gagrach supercomputer missiles failed high)
    expressions  (((country Zerdia)        :name country-Zerdia)
            ((country Gagrach)       :name country-Gagrach)
            ((warlike Gagrach)       :name warlike-Gagrach)
            ((Zerdias-asset supercomputer)   :name supercomputer-asset)
            ((weapon missiles)       :name weapon-bow)
            ((used-for supercomputer missiles ) :name use-supercomputer)
            ((not use-supercomputer) :name not-use-supercomputer)
            ((attack Gagrach Zerdia) :name attack-Gagrach)
            ((not attack-Gagrach)    :name not-attack)
            ((success attack-Gagrach)        :name success-attack)
            ((equals success-attack failed)       :name failed-attack)
            ((cause not-use-supercomputer failed-attack) :name cause-failed-attack)
            ((desire Gagrach supercomputer)  :name desire-supercomputer)
            ((realize Zerdia desire-supercomputer)   :name realize-desire)
            ((follow failed-attack realize-desire)    :name follow-realize)
            ((offer Zerdia supercomputer Gagrach)    :name offer-supercomputer)
            ((cause realize-desire offer-supercomputer) :name cause-offer)
            ((obtain Gagrach supercomputer)     :name buy-supercomputer)
            ((cause offer-supercomputer buy-supercomputer)  :name cause-buy)
            ((happiness Gagrach)     :name happiness-Gagrach)
            ((equals happiness-Gagrach high) :name happy-Gagrach)
            ((cause buy-supercomputer happy-Gagrach) :name cause-happy)
            ((promise Gagrach Zerdia not-attack)     :name promise)
            ((cause happy-Gagrach promise)   :name cause-promise)))
```

Zerdia the hawk - MA5

```
(defEntity Zerdia)
(defEntity sportsman)
(defEntity feathers)
(defEntity cross-bow)
(defEntity true)

(defDescription ma-5
    entities   (Zerdia sportsman feathers cross-bow true)
    expressions  (((bird Zerdia)   :name bird-Zerdia)
            ((person sportsman)       :name person-sportsman)
            ((warlike sportsman)      :name warlike-sportsman)
```

```
((Zerdias-asset feathers) :name feathers-asset)
((weapon cross-bow)       :name weapon-bow)
((used-for feathers cross-bow )  :name has-feathers)
((desire sportsman feathers)     :name desire-feathers)
((realize Zerdia desire-feathers)      :name realize-desire)
((offer Zerdia feathers sportsman)     :name offer-feathers)
((cause realize-desire offer-feathers) :name cause-offer)
((obtain sportsman feathers)           :name take-feathers)
((cause offer-feathers take-feathers)  :name cause-take)
((attack sportsman Zerdia)     :name attack-sportsman)
((not attack-sportsman)        :name not-attack)
((promise sportsman Zerdia not-attack)   :name promise)
((cause take-feathers promise) :name cause-promise)
((see Zerdia sportsman)        :name see-Zerdia)
((follow promise see-Zerdia)   :name follow-promise)
((follow see-Zerdia attack-sportsman)    :name follow-see)
((success attack-sportsman)    :name success-attack)
((equals success-attack true)  :name successful-attack)
((cause has-feathers successful-attack)  :name cause-success-attack)
((realize Zerdia has-feathers) :name realize-Zerdia)
((follow successful-attack realize-Zerdia) :name follow-succ-attack)))
```