The SME User's Manual
(SME Version 2E)

by

Brian Falkenhainer

December 1988
This paper documents the Structure-Mapping Engine (SME), a general-purpose program for studying analogical processing. It provides a comprehensive description of the program and instructions for using it, including techniques for integrating it into larger systems. One section demonstrates methods for configuring SME to a variety of mapping preferences and suggests the range of theoretical variations available.
The SME User's Manual
(SME Version 2E)

Brian Falkenhainer
Qualitative Reasoning Group
Department of Computer Science
University of Illinois at Urbana-Champaign

December, 1988

Abstract

This paper documents the Structure-Mapping Engine (SME), a general-purpose program for studying analogical processing. It provides a comprehensive description of the program and instructions for using it, including techniques for integrating it into larger systems. One section demonstrates methods for configuring SME to a variety of mapping preferences and suggests the range of theoretical variations available.

This research is supported by the Office of Naval Research, Personnel and Training Research Programs, Contract No. N00014-85-K-0559.

Approved for public release; distribution unlimited.
CONTENTS

Contents

1 Introduction
   1.1 Conventions .................................................. 1
   1.2 File Organization ............................................ 1

2 System Review
   2.1 Algorithm Review ............................................... 3
      2.1.1 Step 1: Local match construction (create-match-hypotheses) . 4
      2.1.2 Step 2: Global Match Construction ........................ 8
      2.1.3 Step 3: Compute Candidate Inferences (gather-inferences) ... 9
      2.1.4 Step 4: Compute Structural Evaluation Scores (run-rules) ... 9
   2.2 Adding Theoretical Constraints .................................. 10

3 Declarations
   3.1 Declaring Predicates ........................................... 10
   3.2 Declaring Entities .............................................. 11
   3.3 Declaring Description Groups .................................. 11
   3.4 Adding new expressions .......................................... 12
   3.5 Typed Logic .................................................... 12

4 Using the rule system
   4.1 Rule file syntax ................................................. 13
      4.1.1 File declarations ........................................ 13
      4.1.2 Match Constructor rules .................................. 13
      4.1.3 Match Evidence rules ...................................... 14
   4.2 Making SME simulate structure-mapping theory .................. 15
   4.3 Making SME perform as SPROUTER ............................... 16
   4.4 Relaxing the identical predicates constraint .................. 16
   4.5 Pure isomorphisms ............................................... 16
   4.6 Imposing externally established pairings ....................... 17

5 Representation Issues .................................................. 19

6 Using SME
   6.1 Installing SME .................................................. 20
   6.2 Running SME .................................................... 20
   6.3 Batch mode ...................................................... 22
   6.4 Generalization mechanism ....................................... 23
   6.5 Inspecting MH and Gmap evidence ................................ 24
   6.6 Windows ......................................................... 25
   6.7 System parameters ............................................... 25
   6.8 System utilities ................................................ 26

7 User Hooks
   7.1 Applications control over display ............................... 26
   7.2 Useful miscellaneous functions .................................. 27
      7.2.1 Entities, predicates, and expressions ..................... 27
      7.2.2 Dgroups .................................................... 29
7.2.3 Creating and inspecting global matches

8 Algorithm Internals

8.1 The Match Function

8.2 Match Hypotheses

8.3 Global Mappings

8.4 Candidate Inference Generation

8.5 Rule System

9 Summary

10 Acknowledgements

References

Index
1 Introduction

The Structure-Mapping Engine (SME) is a general tool for performing various types of analogical mappings. SME was originally developed to simulate Gentner's Structure-Mapping theory of analogy \cite{12,13,14}. It was hoped that the developed system would also be able to model the other types of similarity comparisons sanctioned by Gentner's theory, such as literal similarity and mere appearance. What ended up being developed was an extremely flexible and efficient system. Most theoretical assumptions are left out of the program and are supplied through match rules. Thus, while SME was originally designed to simulate the comparisons of structure-mapping theory, it may simulate many others as well. Given a set of theoretical restrictions on what constitutes a reasonable analogical mapping, one may implement these restrictions in the form of rules and use SME to interactively test their consequences. This report is intended to make that task easier.

This paper is designed for those interested in using SME for studying analogical processing, testing alternate theories, or as the mapping component in a larger system. It describes the options and user support provided in SME, how to use it for testing theories, and how to integrate it with other programs. For a discussion of the theory behind SME, the general algorithm, and descriptions of the program in operation, one should consult \cite{8,9} prior to reading this manual. Descriptions of the use of SME in various research projects may be found in \cite{4,5,6,7,13,15,23}, while descriptions of Gentner's Structure-Mapping theory appear in \cite{11,12,13,14,10,9}.

1.1 Conventions

Throughout this guide, a few conventions will be used which should be explained at this time.

1. **CommonLisp Packages.** The SME system resides in its own package, SME, which is defined to use CommonLisp. As a result, any reference to an SME function or variable must specify the SME package, as in the function sme:define-predicate. To simplify the discussion, we will omit the package prefix when describing SME functions, macros, and variables. In addition, while the SME routines reside in the SME package, the structures it manipulates reside in the general USER package.

2. **The declarative interface.** In general, the routines used to present data items to SME, such as predicate definitions and concept descriptions, appear in two, functionally-equivalent forms. These two types have a naming convention associated with each. The most common type is the declarative or macro interface. The declarative routines do not evaluate their arguments and match the syntactic form defroutine-name. For example, to define the entity sun declaratively, one writes (defEntity sun). The second type is the functional interface, which is present to support declarations by external programs. These routines evaluate their arguments and match the syntactic form define-routine-name. For example, to define the entity sun functionally, one writes (define-entity 'sun).

1.2 File Organization

SME is contained within the following twelve files:

- **config.lisp** The declarations for site specific parameters.
- **defs.lisp** The basic structure definitions and macros used throughout SME.
- **bits.lisp** Routines for creating and manipulating bit vectors.
bms.lisp The belief-maintenance system (BMS) - a probabilistic TMS.

bms-tre.lisp The rule system and problem-solver front end for the BMS.

sme.lisp The SME top-level routines, such as initialization, defining facts about a concept, and fetching and storing facts and concept descriptions.

match.lisp The SME mapping algorithm.

match-rules-support.lisp A few functions useful for writing SME match rules.

display.lisp Machine independent output routines.

windowing.lisp Symbolics dependent interface routines.

batch.lisp Routines to enable execution in batch mode with a final report generated.

generalize.lisp Inductive generalization support.

2 System Review

The Structure-Mapping Engine can simulate a class of structural approaches to analogical mapping. In these approaches, there is a distinct stage of matching and carryover of predicates from one domain (the base) into another (the target) within the larger analogy process. Furthermore, although there are a number of differences, there is widespread agreement among these techniques on one fundamental restriction [1,2,16,19,20,21,22,25]:

1. **Structural consistency.** If a final analogical mapping includes a predicate in the base paired with a predicate in the target, then it must also include corresponding pairings between each of their arguments. This criterion simply asserts that an analogical mapping must not produce syntactically meaningless predicate calculus forms.

In SME, this restriction was enforced by the requirement of simulating Structure-Mapping theory; its impact on the algorithm is described in [9]. However, this restriction is only part of that theory and alone does not uniquely define a matching algorithm. Additional theoretical restrictions must be supplied through match rules. This enables SME to be used in exploring the space of theories consistent with this single criterion. An additional restriction is enforced by default:

- **One-to-one mapping:** No base item (predicate or object) may be paired with multiple target items. Likewise, no target item may be paired with multiple base items.

Enforcement of the one-to-one restriction is a global parameter which may be disabled. Support is provided to implement variations of one-to-one within the match rules.

Match rules specify what pairwise matches are possible and provide local measures of evidence used in computing the evaluation score. These rules are the key to SME's flexibility. To build a new matcher one simply loads a new set of match rules. This has several important advantages. First, we can simulate all of the types of comparisons sanctioned by Structure-Mapping theory with one program. Second, the rules could in theory be "tuned" if needed to simulate particular kinds of human performance. Third, a variety of other analogical mapping systems may be simulated for comparison and theoretical investigation. The breadth of the space of these structural approaches is suggested by the examples in Section 4.

In this section, the SME matching algorithm is briefly reviewed, followed by a short discussion of how theoretical guidelines may be added to the general mechanism. It is a summary of the algorithm description appearing in [9], annotated with the SME functions that carry out each step.
2.1 Algorithm Review

Given descriptions of a base and a target (called Dgroups), SME builds all structurally consistent interpretations of the comparison between them. Each interpretation of the match is called a global mapping, or Gmap. 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.

For example, given the descriptions of water flow and heat flow shown in Figure 1, SME might, depending on the current theoretical configuration, offer several alternative interpretations for this potential analogy. 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.

The SME algorithm (see Figure 2) is logically 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 and uses this evidence to compute structural evaluation scores for each Gmap.
- Run MHC rules to construct match hypotheses (create-match-hypotheses).
- Calculate the Conflicting set for each match hypothesis (calculate-nogoods).
- Calculate the EMaps and NoGood sets for each match hypothesis by upward propagation from entity mappings (generate-justifications and propagate-descendants).
- During the propagation, delete any match hypotheses that have justification holes (propagate-death).
- Merge match hypotheses into Gmaps (generate-gmaps).
  1. Interconnected and consistent (generate-structure-groups).
  2. Consistent members of same base structure (merge-base).
  3. Any further consistent combinations (full-gmap-merge).
- Calculate the candidate inferences for each GMap (gather-inferences).
- Score the matches (run-rules).
  1. Local match scores.
  2. Global structural evaluation scores.

Figure 2: Summary of SME algorithm.

Each computation will now be reviewed, using a simple example to illustrate their operation. In this example, the rules of structure-mapping theory are in use. It is important to distinguish the general SME system from its behavior when using the rules of a particular theory. Hence, when using the rules of structure-mapping theory, it will be called SME\textsubscript{SMT}.

2.1.1 Step 1: Local match construction (create-match-hypotheses)

SME begins by finding for each entity and predicate in the base the set of entities or predicates in the target that could plausibly match that item (see Figure 3). Plausibility is determined by match constructor rules, which are of the form:

\[
\text{(MHC-rule ((Trigger) (BaseVariable) (TargetVariable) /:test (TestForm)) (Body))}
\]

The body of these rules is run on each pair of items (one from the base and one from the target) that satisfy the condition and installs a match hypothesis which represents the possibility of them matching. For example, to state that an expression in the base may match an expression in the target whose functor is identical, we write:

\[
\]
The likelihood of each match hypothesis is found by running match evidence rules and combining their results. The evidence rules provide support for a match hypothesis by examining the structural and syntactic properties of the items matched. For example, the rule

\[
(MHRule ((:intern (MH ?b ?t)) :test (and (expression? ?b) (expression? ?t))
  (eq (expression-functor ?b) (expression-functor ?t))))
\]


\[
(assert! (implies same-functor (MH ?b ?t) (0.5 . 0.0)))
\]

states "If the two items are expressions and their functors are the same, then supply 0.5 evidence in favor of the match hypothesis." The rules may also examine match hypotheses associated with the arguments of these items to provide support based on systematicity. This causes evidence for a match hypothesis to increase with the amount of higher-order structure supporting it.

The state of the match between the water flow and heat flow descriptions of Figure 1 after running these first two sets of rules is shown in Figure 4. There are several important things to notice in this figure. First, there can be more than one match hypothesis involving any particular base or target item. Second, our rules required predicates to match identically while they allowed entities to match on the basis of their roles in the predicate structure. Thus while TEMPERATURE can match either PRESSURE or DIAMETER, IMPLIES cannot match anything but IMPLIES. Third, not every possible correspondence is created. Local matches between entities are only created when justified by some other identity. This significantly constrains the number of possible matches in the typical case.
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.
Figure 5: Water Flow - Heat Flow analogy after computation of Conflicting relationships. Simple lines show the tree-like graph that the grounding criteria 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.

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.
2.1.2 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 criteria:

1. One-to-one: No two match hypotheses 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 the match hypotheses which pair up all of the arguments of MH's base and target items.

The preservation criteria enforces strict one to one mappings. The grounding criteria preserves connected predicate structure. A collection is maximal if adding any additional match hypothesis would render the collection structurally inconsistent.

The formation of global matches is composed of two primary stages:

1. Compute consistency relationships (calculate-nogoods): Here we generate for each match hypothesis the sets of entity mappings it entails, what match hypotheses it locally conflicts with, and which match hypotheses it is structurally inconsistent with. This information simplifies the detection of contradictory sets of match hypotheses, a critical operation in the rest of the algorithm. The result of this stage of processing appears in Figure 5.

2. Merge match hypotheses (generate-gmaps): Compute Gmaps by successively combining match hypotheses as follows:

   (a) Form initial combinations (generate-structure-groups): Combine interconnected and consistent match hypotheses into an initial set of Gmaps (Figure 6a).

   (b) Combine dependent Gmaps (merge-base): Since base and target dgroups are rarely isomorphic, some Gmaps in the initial set will overlap in ways that allow them to be merged. The advantage in merging them is that the new combination may provide structural support for candidate inferences (Figure 6b).

   (c) Combine independent collections (full-gmap-merge). The results of the previous step are next combined to form maximal consistent collections (Figure 6c).

A parameter option allows the support criterion to be weakened so that it does not cross the boundaries of a relational group [7]. A relational group is distinguished as an unordered collection of relational structures that may be collectively referred to as a unit. They correspond to the abstract notion of a “set” and are associated to predicates taking any number of arguments. For example, a set of relations joined by the predicate AND defines a relational group. Other examples include the axioms of a theory, a decomposable compound object, or the relations holding over an interval of time. Intuitively, we would like to say that two groups correspond without requiring that their contents are exhaustively mapped.

If base and target propositions each contain a group as an argument, the propositions should not be prevented from matching if the groups’ members cannot be exhaustively paired. For example, the set of relations

\[\text{These two merge steps (b and c) are called by merge-gmaps, which is in turn called by generate-gmaps.}\]
B: \text{Implies}[\text{And}(P_1, P_2, P_3), P_4] \quad (1)
T: \text{Implies}[\text{And}(P'_1, P'_2), P'_4]

should match better than the set of relations

B: \text{Implies}[\text{And}(P_1, P_2, P_3), P_4] \quad (2)
T': P'_1, P'_2, P'_4

The original model of structural consistency would score (1) and (2) equally, since the Implies relations of (1) would not be allowed to match. This is a particularly important consideration when matching sequential, state-based descriptions (e.g., the behavior of a system through time). The set of relations describing a pair of states often do not exhaustively match or are of different cardinality. Yet, higher-order relations over states, such as temporal orderings, are vital and must appear in the mapping.

2.1.3 Step 3: Compute Candidate Inferences (gather-inferences)

Associated with each Gmap is a (possibly empty) set of candidate inferences. Candidate inferences are base predicates that would fill in structure which is not in the Gmap (and hence not already in the target). If a candidate inference contains a base entity that has no corresponding target entity (i.e., the base entity is not part of any match hypothesis for that gmap), SME introduces a new, hypothetical entity into the target. Such entities are represented as a skolem function of the original base entity (i.e., (:skolem base-entity)).

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. If the FLOW predicate was not present in the target, then the candidate inferences for a Gmap corresponding to the pressure inequality would be both CAUSE and FLOW. Note that GREATER-THAN(DIAMETER(coffee), DIAMETER(ice cube)) is not a valid candidate inference for the first Gmap because it does not intersect the existing Gmap structure.

2.1.4 Step 4: Compute Structural Evaluation Scores (run-rules)

Typically a particular pair of base and target will give rise to several Gmaps, each representing a different interpretation of the match. Often it is desired to select only a single Gmap, for example to represent the best interpretation of an analogy. Many of these evaluation criteria (including validity, usefulness, and so forth) lie outside the province of Structure-Mapping, and rely heavily on the domain and application. However, one important component of evaluation is structural — for example, one Gmap may be considered a better analogy than another if it embodies a more systematic match. 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 in selecting the “best” analogy, or as a factor in a more complex (but external) evaluation procedure. In SME, the structural evaluation score is currently computed by simply adding the belief of each local match hypothesis to the belief of the Gmaps it is a member of.

Returning to Figure 7, note that the “strongest” interpretation (i.e., the one which has the highest structural evaluation score) is the one we would intuitively expect. In other words, 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.
2.2 Adding Theoretical Constraints

Given the general program, we may then add theoretical constraints in the form of rules. For instance, the example just presented used the literal similarity rules of structure-mapping theory. These rules augment SME's one-to-one mapping and structural consistency criteria with two additional restrictions. First, evidence is computed according to systematicity, that is, highly interconnected systems of relations are preferred over independent facts. Second, only identical relations are allowed to match (i.e., CAUSE is not allowed to match GREATER-TAIl). Had another set of rules been used, the results might have been substantially different. For example, the mere appearance rules of structure-mapping theory would have determined that the water to coffee mapping was the best, due to their superficial similarity.

3 Declarations

The descriptions given to SME are constructed from a user-defined vocabulary of entities and predicates. This section discusses the conventions for defining languages for SME's use.

3.1 Declaring Predicates

defPredicate name argument-declarations predicate-class [Macro]
  &key :expression-type logical-type
     :commutative? {t | nil}
     :n-ary? {t | nil}
     :documentation descriptive-string
     :eval procedural-attachment 

define-predicate name argument-declarations predicate-class ... [Function]
**predicate-class** is either function, attribute, or relation, according to what kind of predicate name is. The *argument-declarations* allows the arguments to be named and typed. For example, the declaration:

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

states that CAUSE is a two-place relational predicate. Its arguments are called antecedent and consequent, both of type event. The names and types of arguments are for the convenience of the representation builder and any external routines (including the match rules), and are not currently used by SME internally. Likewise, the predicate class may be very important to the theoretical constraints imposed in the rules, but is ignored by SME internally.

The optional declaration :expression-type indicates the logical type of an expression headed by the given predicate. For example, the predicate throw may represent a kind of action, while the predicate mass may represent an extensive-quantity.

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. Examples of commutative nary predicates include AND, SUM, and SET.

The :documentation option allows one to attach a descriptive string to a predicate. This documentation may then be accessed through the lisp machine supplied interface (C-Shift-D) or some externally written routine. If no documentation is supplied, the list of argument names is used. Another option provided strictly for potential user routines is the optional :eval parameter. This allows one to declare a procedural attachment for a predicate.

### 3.2 Declaring Entities

**defEntity** name &key type constant?

**define-Entity** name &key type constant?  

**Macro**

**define** Entity name &key type constant?

**Function**

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 declared with the defEntity form (a non-primitive entity would be (temperature sun), which is a functional form representing a particular numeric temperature entity). Primitive entities declared in this way represent global entity types, that is, they represent a class of entities rather than an actual instance of an entity. When an entity type is actually used in a domain description, a unique entity instance is created for that type (e.g., Mary is translated to Mary43).

Since the language is typed, each entity type can be declared as a subtype of an existing type using the :type option. For example, we might have

```
(defEntity star :type inanimate)
(defEntity Sun :type star)
```
to say that stars are inanimate objects, and our Sun is a particular star. Constants are declared by using the :constant? option, as in

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

### 3.3 Declaring Description Groups

**defDescription** description-name  

**Macro**

entities (entity1, entity2,...,entityi)

expressions (expression-declarations)
define-description description-name entities expressions  

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, where expression-declarations take the form

expression or  
(expression :name expression-name)

For example, the description of water flow depicted in Figure 1 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)))

All entities must have been previously defined and every entity referred to in the Dgroup's expressions must appear in the entities list of the defDescription.

3.4 Adding new expressions

expression form dgroup-name $key expression-expression-name update-structure?  

Expressions are normally defined as a side effect of creating a description group (Dgroup). However, the facility is provided for dynamically adding new expressions to a Dgroup. The syntax is essentially the same as for expressions declared within a defDescription. The expression's form may refer to the names of existing Dgroup expressions and the form may be given a name. When expression is used to add expressions to an existing Dgroup, the update-structure? keyword must be invoked with a non-nil (e.g., T) value. This keyword indicates that the Dgroup's structure must be reexamined, since the known structural roots will change as a result of this new expression.

3.5 Typed Logic

A mechanism exists for attaching types to predicates and their arguments (see defPredicate). This facility is designed to constrain the operation of SME, particularly candidate inference generation. However, it has not been extensively used to date. The ability to attach types may be useful for consistency checking by external systems.

4 Using the rule system

The rule system is the heart of SME's flexibility. It allows one to specify what types of things might match and how strongly these matches should be believed. This section describes the required syntax for a rule set and different strategies for rule specification.
4.1 Rule file syntax

A rule set, or rule file, consists of a declaration, a set of match constructor rules, and a set of match evidence rules. In order to describe each, we will examine the syntax and functionality of each part of the smt-analogy rule file.

4.1.1 File declarations

sme-rules-file identification-string

Each rule file must begin with the initialization command sme-rules-file. This function clears the rule system in preparation for a new set of rules (rules are cached for efficiency) and stores the name of the rule set for output identification purposes. For example, our sample rules file begins with:

(sme-rules-file "smt-analogy.rules")

The rule file must then end with the tre-save-rules command:

tre-save-rules

4.1.2 Match Constructor rules

MHC-rule (trigger ?base-variable ?target-variable

body

install-MH base-item target-item

SME begins by finding for each entity and predicate in the base the set of entities or predicates in the target that could plausibly match that item. Plausibility is determined by match constructor rules, which are responsible for installing all match hypotheses processed by SME. There are two types of constructor rules, each indicated by a different value for trigger. The first type of rule is indicated by a :filter trigger. These rules are applied to each pair of base and target expressions, executing the code in body. If the :test option is used, test-form must return true for the body to be run. For example, the following rule states that an expression in the base may match an expression in the target whose functor is identical, unless they are attributes (a structure-mapping analogy criterion):

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

(not (attribute? (expression-functor ?b)))))

(install-MH ?b ?t))

The second type of MHC rule is indicated by a trigger of :intern. These rules are run on each match hypothesis as it is created. Typically they create match hypotheses between any functions or entities that are the arguments of the expressions joined by the match hypothesis that triggered the rule. The following is one of two that appear in smt-analogy.rules:

Notice the file extension *.rules. While rule files are not required to end in *.rules, all user interface facilities for simplifying the loading of rule files depend upon this extension. Another useful point is that rule files are typically defined to be in the SME package to avoid having to use sme: throughout the rule set.

---

2 Notice the file extension *.rules. While rule files are not required to end in *.rules, all user interface facilities for simplifying the loading of rule files depend upon this extension. Another useful point is that rule files are typically defined to be in the SME package to avoid having to use sme: throughout the rule set.
(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-ME (first bchildren) (first tchildren)))
          ((and (function? (expression-functor (first bchildren)))
              (function? (expression-functor (first tchildren)))
          (install-ME (first bchildren) (first tchildren)))
          ((and (attribute? (expression-functor (first bchildren)))
              (eq (expression-functor (first bchildren))
                  (expression-functor (first tchildren))))
          (install-ME (first bchildren) (first tchildren)))))

Notice that the third test allows identical attributes to match, whereas the previous MHC rule did not allow such matches. This design does not allow isolated attributes to match, but recognizes that attributes appearing in a larger overall structure should be matched.

4.1.3 Match Evidence rules

rule nested-triggers body  [Macro]
rassert! expression &optional (belief+ 1.0) (belief- 0.0)  [Macro]
assert! expression &optional (belief+ 1.0) (belief- 0.0)  [Function]
initial-expression assertion-form  [Macro]

The structural evaluation score is computed in two phases. First, each match hypothesis is assigned some local degree of evidence, independently of what Gmaps it belongs to. Second, the score for each Gmap is computed based on the evidence for its match hypotheses. The management of evidence rules is performed by the Belief Maintenance System (BMS) [3]. A BMS is a form of Truth-Maintenance system, extended to handle numerical weights for evidence and degree of belief (see [3] for a description of what the weights mean). Pattern-directed rules are provided that trigger on certain events in the knowledge base.

The following is a simple rule for giving evidence to match hypotheses between expressions that have the same predicate:

(initial-expression (assert! 'same-functor))

(rule (:intern (ME ?b ?t) :test (and (expression? ?b) (expression? ?t))
        (eq (expression-functor ?b)
            (expression-functor ?t))))

(if (function? (expression-functor ?b))
   (rassert! (implies same-functor (ME ?b ?t) (0.2 . 0.0)))
   (rassert! (implies same-functor (ME ?b ?t) (0.5 . 0.0))))

There are two things to notice here in addition to the evidence rule. First, the proposition same-functor was asserted to be true (a belief of 1.0) and then used as the antecedent for the implication of evidence. In this way, the source of this particular piece of evidence is identified and is available for inspection. Second, the assertion of same-functor was placed inside the initial-assertion form. Since SME caches the current rule file, it must be told if there are any functions embedded in the rule file that must be invoked each time SME is initialized.
Nested triggers within an evidence rule may be used to locate interdependencies between different match hypotheses. For example, structure-mapping's systematicity principle is implemented in a local fashion by propagating evidence from a match hypothesis to its children:

```
(rule ((:intern (NH ?b1 ?t1) :test (and (expression? ?b1) (expression? ?t1)
      (not (commutative? (expression-functor ?b1))))))
  (rassert! (implies (NH ?b1 ?t1) (NH ?b2 ?t2) (0.8 . 0.0))))
```

Evidence for a Gmap is given by:

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

The BMS allows a set of nodes to be declared special and will treat evidence to these nodes differently. An additive-nodes function is provided which takes a set of BMS nodes and modifies them so that their evidence is added rather than normalized using Dempster's rule. SME automatically invokes additive-nodes on the derived set of Gmaps once they are created. Thus, when the above Gmap rule is executed and the implies statement is used to supply evidence from each match hypothesis to the Gmap, that evidence is simply automatically added to the total Gmap evidence rather than propagated using Dempster's probabilistic sum.

The following destructive rule is often used instead of the previous one to give a significant speed up:

```
(rule ((:intern (OMAP ?gm) :var ?the-group))
  (setf (node-belief+ (Wi-bms-node ?gm)) 0)
  (dolist (mh (gm-elements ?gm))
    (incl (node-belief+ (gm-bas-node ?gm))
      (node-belief+ (ah-bins-node mh))))
```

This rule bypasses the BMS entirely, thus increasing speed by not creating justification links. It also renders the additive-nodes distinction irrelevant. However, such rules must be used with extreme caution. For example, the source of a Gmap's evidence cannot be inspected when using this type of operation (see Section 6.5).

### 4.2 Making SME simulate structure-mapping theory

The previous section examined the general structure of an SME rule file. In the process, the basic elements of the structure-mapping-theory analogy rule set were presented. The literal similarity and mere-appearance rules are essentially the same as the analogy rules. They differ in the first match constructor rule. The analogy rule set has the test (not (attribute? (expression-functor ?b))) which is absent from the corresponding literal similarity rule. Conversely, the corresponding mere appearance rule forces the opposite condition (attribute? (expression-functor ?b)).

One should consult Appendix A of [9] for listings of all three structure-mapping rule sets.

---

*A number of functions (e.g., children-of?) are provided to simplify the writing of rules. These appear in the file match-rules-support.lisp.*
4.3 Making SME perform as SPROUTER

The SPROUTER program [17] was developed as an approach to the problem of inductively forming characteristic concept descriptions. That is, given a sequence of events (e.g., a list of pictures), produce a single, conjunctive description which represents a generalized, characteristic description of the sequence. SPROUTER generalized a sequence of \( N \) descriptions by finding the commonalities between the first two descriptions, generalizing these common elements (i.e., variabilize the literals), and then repeating the process using this generalized description and the next, unprocessed description. These steps would be repeated until the generalization had propagated through the entire list of input descriptions.

SME may be used to implement SPROUTER's interference matching technique by giving it a set of match constructor rules which require all matching predicates to have the same name (i.e., the literal similarity rules without the condition that allows functions with different names to match). The SPROUTER generalization mechanism may then be implemented with the following algorithm:

Procedure SPROUTER (event-list)
begin
    generalization := pop(event-list)
    while event-list
        pairwise-match := match(pop(event-list), generalization)
        generalization := generalize(pairwise-match)
    return generalization
end

4.4 Relaxing the identical predicates constraint

The current structure-mapping theory rules are sensitive to representation by requiring that relational predicates match only if they are identical. This is an important restriction that ensures the structures being compared are semantically similar. However, it can also be overly restrictive. We are currently exploring different methods to relax the identicality requirement while still maintaining a strong sense of semantic similarity. One approach, called the minimal ascension principle, allows relations to match if they share a common ancestor in a multi-root is-a hierarchy of expression types [7] (i.e., the identicality test in the match constructor rule is replaced by a call to predicate-type-intersection?). The local evidence score for their match is inversely proportional (exponentially) to the relations' distance in the hierarchy. This enables SME to match non-identical relations if such a match is supported by the surrounding structure, while still maintaining a strong preference for matching semantically close relations. This is similar to approaches used in [1,16,24].

Problems with an unconstrained minimal ascension match technique are discussed in [7]. A mapping approach which considers the current context when determining pairwise similarity is also discussed.

4.5 Pure isomorphisms

While it is important to assure that the structures being compared are semantically similar, one can in principle remove all semantic comparisons. This would allow match creation to be guided strictly by SME's structural consistency and 1-1 mapping criteria and match selection to be based strictly on systematicity.

Consider the isomorphic mapping between the formal definitions of numeric addition and set
union shown in Figure 8. These formal descriptions may be given to SME in the standard manner, as in (plus N3 (plus N4 N5)) for the left side of the associativity rule (the representation (plus N4 N5 result45) also works, although it results in a slightly longer run time due to the flattening of structure). When presented as formal definitions, the concepts of addition and union are structurally isomorphic, independent of the meaning of the predicates. Thus, while it could be argued that the predicates plus and union share a certain degree of semantic overlap, this example demonstrates that it is possible to make SME ignore predicates entirely and simply look for isomorphic mappings. The rule set for isomorphic mappings is shown in Figure 9. (This is called the ACME rule set, as it configures SME to emulate the ACME program on this example [18]). The only constraint this rule set enforces is that each predicate has the same number of arguments. While it includes the Structure-Mapping notion of systematicity to prefer systems of relations, it does not enforce identicality of predicates. Using this rule set, SME produces the unique best mapping that we would expect between the formal definitions of addition and union.

Since SME enforces the “same number of arguments” restriction by defeating any match hypotheses that are not structurally sound, we could in principle effectively remove the rule file entirely. This could be done with one match constructor rule to match everything with everything and one evidence rule to measure systematicity. When this free-for-all rules file was given to SME, the same single best Gmap was produced, but at the expense of increasing the run time from 13 seconds to 3.25 minutes.

4.6 Imposing externally established pairings

In certain situations, a number of entity and predicate mappings may already be known prior to invoking SME. These mappings may have been provided as an analogical hint from an instructor or derived by the application program during earlier processing. For example, PHINEAS [4,5] uses SME to analogically relate observed physical phenomena to known theories of the world. PHINEAS uses two analogical mappings to learn about a new physical process. First, behavioral correspondences are established (i.e., what entities and quantities are behaving in the same manner). Second, the relevant base theories are analogically mapped into the new domain, guided by the behavioral correspondences. The two-stage mapping process solves the problem of using analogy in cases where one does not have a pre-existing theory, as occurs with truly novel learning. The assumption made in PHINEAS is that similar behaviors will have similar theoretical explanations. The first mapping provides the correspondences between entities and functions required to guide the importation of an old theory to explain a new domain in the second mapping.

SME includes facilities to simplify writing PHINEAS-like programs, by enabling the results of earlier processing to constrain subsequent mapping tasks. These routines are divided into two

---

4 This example is taken from an advance copy of a paper by Holyoak and Thagard [18]. I include it here simply to demonstrate the range of matching preferences available in SME.
(MHC-rule (:filter ?b ?t :test (= (numargs (expression-functor ?b)) (numargs (expression-functor ?t))))
(install-MH ?b ?t))

;;; Intern rule to match entities (non-commutative predicates)
(MHC-rule (:intern ?b ?t :test (and (expression? ?b) (expression? ?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-ME (first bchildren) (first tchildren))))

;;; Give a uniform initial priming to each MH
(initial-assertion (assert! 'initial-priming))

(rule ((:intern (MH ?b ?t)))
   (rassert! (implies initial-priming (MH ?b ?t) (0.2 . 0.0))))

;;; Propagate interconnections - systematicity
(rule ((:intern (MH ?b1 ?t1) :test (and (expression? ?b1) (expression? ?t1)))
(rassert! (implies (MH ?b1 ?t1) (MH ?b2 ?t2) (0.8 . 0.0))))

;;; Support from its MH's
(rule ((:intern (GNAP ?gm) :var ?the-group))
   (dolist (mh (gm-elements ?gm))
      (assert! '(implies , (mh-form mh) , ?the-group))))

Figure 9: Rule set for forming general isomorphic mappings.

categories, declaration and test. The declaration routines tell SME what predicate and entity correspondences are known a-priori. The test routines enable the match constructor rules to adhere to these imposed constraints. Known mappings are declared through the following functions:

defGiven-Mappings
  entities  ((base-entity1 target-entity1)...
       (base-entityi target-entityi) ...)
  predicates ((base-predicate1 target-predicate1)...
       (base-predicatej target-predicatej) ...)
declare-given-mappings entities predicates

clear-given-mappings

Both defGiven-Mappings and declare-given-mappings have identical functionality. The first does not evaluate its arguments while the second one does. Disjunctive constraints may be imposed by including all of the possible pairings (e.g., defining both (base-entity1 target-entity1) and (base-entity2 target-entity2)).

Once a set of given mappings has been declared, the following test routines may be used within the match constructor rules to enforce these mappings:
sanctioned-pairing? base-item target-item
paired-item? &key base-item target-item

sanctioned-pairing? tests if the given pair is one of the a-prior pairings. paired-item? takes either a base item or a target item and returns true if the mapping for that item has been externally determined.

These functions help in writing rules which respect established mappings. For example, the following two rules are used in the PHINEAS system to allow observed behavioral correspondences to constrain the mapping of the relevant theory:

(MHC-rule (:filter ?b ?t :test (and (eq (expression-functor ?b) (expression-functor ?t))
(not (paired-item? :base-item (expression-functor ?b)))
(not (paired-item? :target-item (expression-functor ?t))))
(install-MH ?b ?t))

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

When an analogy is being made between two behaviors, clear-given-mappings is used to make SME perform in normal analogy mode. The discovered entity and function correspondences are then given to declare-given-mappings prior to using SME to map the relevant theory.

5 Representation Issues

The proper representation becomes an issue in SME due to its significant impact on speed performance. Hierarchical representations provide an important source of constraint on generating potential matches. They tend to make the semantic interrelations explicit in the structure of the syntax. For example, Section 4.5 described a comparison between the laws of addition and union. There it was noted that part of the additive associativity rule may be represented as (plus N3 (plus N4 N5)) or as the pair (plus N4 N5 result45) and (plus N3 result45 result3-45). The latter "flat" representation takes more time for SME to process, sometimes a significant difference for large domain descriptions. This is because the functional representation makes the associativity rule structurally explicit, while the flat representation buries it among the tokens appearing as arguments to plus. However, it is important to note that SME is able to process domain descriptions in any predicate-based format. It is simply speed considerations that render standard, flat forms of representation undesirable.

Due to SME's ability to accept commutative, n-ary predicates, it is able to match arbitrary sets (which must be of equal size at this time). This has two consequences. First, the explicit use of sets becomes a viable form of representation. Thus, a theory might be represented concisely as

(Theory T1 (SET axiom-8 axiom-14 ...))
rather than as (Axiom-of T1 axiom-8), etc. Second, sets may be used to add structure to descriptions. For example, the set representation for theories results in greatly reduced run times compared to the non-set representation.\footnote{The difference in speed is due to the operation of merge step 2, which combines matches sharing a common base structure. The set notation for theory T1 enables merge step 2 to know that matches for axiom-8 and axiom-14 should be placed in the same gmap, thus reducing the number of possibilities in merge step 3.} I am currently investigating the use of a similar representation for temporal states, as in:
(Situation S1 (SET (Increasing (Amount-of water1))
  (Increasing (Pressure water1))
  o o o ))

A PHINEAS problem which took SME 53 minutes (using (Increasing (Amount-of (at water1 S1))))) was reduced to 34 seconds using this more structured representation.

6 Using SME

This section describes how to install SME on your machine, load it, and operate it.

6.1 Installing SME

*A smuggle-language-file*
*A smuggle-default-rules*
*A smuggle-rules-pathname*
*A smuggle-dgroup-pathname*

To configure SME to a particular site, a handful of variables storing system directory information must be edited and set to the appropriate values. These variables appear in config.lisp, a separate file for this purpose. Of primary importance are *smuggle-language-file* and *smuggle-default-rules*. These are used by smuggle-init to initialize the language and rule systems. The two variables storing the rules and dgroup pathnames are used by the user interface routines.

*the-lisp-package*
*the-user-package*

In most Common Lisp implementations, one package exists for general user definitions and another exists for the lisp implementation. It is important to notify SME what these are for the Common Lisp in use. For example, on a Symbolics (version 6.2), the lisp package is called common-lisp and the user package is called cl-user. These are the default settings.

*smuggle-system-pathname*
*smuggle-files*

These variables are used to automate compiling and loading. If the system is being loaded on something other than a Symbolics or TI Explorer, the file windowing should not be included in the list *smuggle-files*. Otherwise, it should be left in the list of SME files, which is the default.

The SME routines assume a set of naming conventions on domain description and rule files. The names of files containing domain descriptions (defDescription) should end with a *.dgroup extension. The name of a file containing a rule set should end with the *.rules extension.

6.2 Running SME

*smuggle-init* &optional (initialize-language? T) (initialize-rules? T)

This section gives a brief overview of the process of using SME for matching, generalization, and inspection tasks.
I
(sme:smetntt)
Initializing SHE...
Loading default language file: prof:>falken>sme>language
Loading default rules file: prof:>falken>sme>literal-similarity.bin
Complete.

> (load "prof:>falken>sme>simple-water-flow.dgroup")
Loading PROF:>falken>sme>simple-water-flow.dgroup into package USER
NIL
> (load "prof:>falken>sme>simple-heat-flow.dgroup")
Loading PROF:>falken>sme>simple-heat-flow.dgroup into package USER
NIL
> (sme:match 'swater-flow 'sheat-flow T)
SME Version 2E

Analogical Match from SWATER-FLOW to SHEAT-FLOW.

Rule File: literal-similarity.rules

<table>
<thead>
<tr>
<th>Entities</th>
<th>Expr.</th>
<th>Maximum order</th>
<th>Average order</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Statistics</td>
<td>4</td>
<td>11</td>
<td>3</td>
</tr>
<tr>
<td>Target Statistics</td>
<td>4</td>
<td>6</td>
<td>2</td>
</tr>
</tbody>
</table>

# HH's | # Gaps | Merge Step 3 | CI Generation | Show Best Only |
14 | 3 | ACTIVE | ACTIVE | OFF |

Total Run Time: 0 Minutes, 0.821 Seconds
BMS Run Time: 0 Minutes, 0.630 Seconds
Best Gaps: 3

Match Hypotheses:
(0.6820 0.0000) (PIPE4 BAR7)
(0.7900 0.0000) (FLAT-WATER FLAT-COFFEE) ;a number of match hypotheses appeared here
(0.8648 0.0000) (WATER1 COFFEE5)
(0.7900 0.0000) (LIQUID-WATER LIQUID-COFFEE)

Gmap #1: (BEAKER2 COFFEES) (DIAM-BEAKE TEMP-COFFEE) (VIALS ICE-CUBES)
(DIAN-VIAL TEMP-ICE-CUBE) (>DIAMETER >TEMP)
Gmap #2: (BEAKER2 COFFEES) (VIALS ICE-CUBES)

Emaps: (BEAKER2 COFFEES) (VIALS ICE-CUBES)

Weight: 3.937660
Candidate Inferences:

;Gmap #2 appeared here...

Gmap #3: (>PRESSURE >TEMP) (PRESS-VIAL TEMP-ICE-CUBE) (PRESS-BEAKE TEMP-COFFEE)
(BEAKER2 COFFEES) (VIALS ICE-CUBES) (WATER1 HEATS)
(PIPE4 BAR7) (WFLOW HLOW)

Emaps: (BEAKER2 COFFEES) (VIALS ICE-CUBES) (WATER1 HEATS) (PIPE4 BAR7)

Weight: 5.991660
Candidate Inferences: (CAUSE >TEMP HFLOW)

---

Figure 10: Initializing and running SME.
1. **Loading the files.** To load or compile SME, the file `config` should be loaded and then `(load-sme)` or `(compile-sme)` called. A defSystem definition is provided in `system.lisp` for Symbolics machines.

2. **System startup.** This stage is only appropriate for full, lisp machine startup. The SME window environment may be created with `Select-S` on a Symbolics or `System-S` on a TI Explorer.

3. **Initialization.** The function `sme-init` should be called to initialize the database. If `initialize-language` is non-nil, the default language file (predicate definitions) will be loaded. If `initialize-rules` is non-nil, the default rules file will be loaded. Prior to operating SME, the language and rule systems must be established.

4. **Loading Dgroups.** Any description groups that are to be matched must be declared. These declarations are typically stored in files, with the extension *.dgroup. If the windowing system is active, the command Load Dgroup will offer a menu of all *.dgroup files in `*sme-dgroup-pathname* to select what to load.

5. **Analogical mapping.** The function `match` may be called to form a mapping between two given Dgroups. This is discussed in Section 7.2.3. If the windowing system is active, the command Match will offer a menu to select base and target Dgroups. It is prior to this step that one might want to think about whether to modify any system parameters (e.g., print the match hypotheses, print only the best Gmaps, generate candidate inferences, etc.).

6. **Describing Dgroups.** Once Dgroups are defined, the `describe-dgroup` facility will provide a description of any particular Dgroup.

7. **Graphically displaying Dgroups.** If the windowing system is active, Dgroups may also be displayed graphically, through the Display Dgroup utility.

8. **Generalizing.** Once a mapping is formed, it may be generalized using the `generalize` function or the Generalize command in the system menu.

9. **Saving the results of a session.** If the windowing system is active, the results of commands like Match and Generalize are sent to the scroll window by default. These results may be written to a file using the dump-scroll system utility.

10. **Comparing two apparently identical Gmaps.** When two Gmaps are formed that appear to be identical, their differences can be identified using the compare-gmaps system utility.

A trace of SME performing the basic mapping task is given in Figure 10. Each of the other options are described in greater detail in the following sections.

### 6.3 Batch mode

- **run-batch-file** `pathname &key (gmap-display :all) (gmap-statistics :none)` `[Function]`
- **language-file** `pathname` `[Macro]`
- **dgroup-directory** `pathname` `[Macro]`
- **dgroup-file** `file-name` `[Macro]`
- **rule-directory** `pathname` `[Macro]`
- **rule-file** `file-name` `[Macro]`
- **rule-sets** `&rest rule-file-names` `[Macro]`
SME User's Manual

Figure 11: Sample SME batch file.

```
same:report-comments string
send-report-to pathname &key (text-driver :LPR) (style :STANDARD)
run-matcher-on base-name target-name
defPostMatcher function
```

SME is normally used as an interactive utility or as a module to some larger program. However, when performing statistical analyses across a broad space of matching preferences (i.e., rule sets) and domain descriptions, an interactive format soon becomes inconvenient. Utilities are provided so that a file of SME instructions may be defined and then executed using run-batch-file (e.g., Figure 11). This would instruct SME to perform a series of matches, potentially over a variety of rule sets and domain descriptions, and generate a detailed report of the execution and a summary of the results. When a single rule set is specified using rule-file, all subsequent matches (invoked by run-matcher-on) will use this rule file until another one is specified. Using rule-sets, one may instead specify a series of rule files to be used, so that a single run-matcher-on command will cause SME to run once for each rule file in the list. If a user-defined function name is given to defPostMatcher, this function will be called after each match is performed, in case special post-match routines are desired or extra information is to be added to the report being generated. A variety of text drivers are supported for report generation (send-report-to), such as :lpr (line printer), :latex, and :troff.

6.4 Generalization mechanism

generalize gmap

The generalize function takes a global mapping structure and returns three alternate generalizations (using the Common-Lisp values protocol), each one successively larger than the previous:

1. Literally common aspects only. This generalization locates those sub-structures which are identical in both base and target Dgroups. This is a type of generalization typically found in
Generalizations for Match from SWATER-FLOW to SHEAT-FLOW:

Generalization #1 (Literally Common Aspects Only):
(FLOW ENTITY6 ENTITY8 ENTITY13 ENTITY14)

Generalization #2 (All Common Aspects Only):
(FLOW ENTITY6 ENTITY8 ENTITY13 ENTITY14)
(GREATER (FUNCTIONO ENTITY6) (FUNCTIONO ENTITY8))

Generalization #3 (Maximal Generalization):
(CAUSE (GREATER (FUNCTIONO ENTITY6) (FUNCTIONO ENTITY8))
(FLOW ENTITY6 ENTITY8 ENTITY13 ENTITY14))

Figure 12: SME generalizations for the simple water flow – heat flow analogy.

inductive generalization programs.

2. All common aspects only. In addition to common, identical substructures, this generalization includes cases where functions of a different name were allowed to match. Where this occurs in the common structure, a skolemized function predicate is created.

3. Maximal generalization. The largest generalization (in terms of amount) includes all candidate inferences sanctioned by the Gmap, as well as the common substructure of generalization mode 2. This represents the entire shared structure between the two Dgroups under the assumption that the candidate inferences are valid.

For example, given the best Gmap from the simple water flow – heat flow analogy described in Section 2.1 and shown in Figure 7, SME will produce the set of generalizations shown in Figure 12. The first generalization indicates that the only thing in common between the two situations is the existence of flow. The second generalization loosens the meaning of “in common” to include the fact that a quantity associated with the source of flow was greater than the same quantity measured for the destination. The final generalization assumes that this inequality, which was the cause of flow in the water flow domain, is actually the cause of flow for both situations.

6.5 Inspecting MH and Gmap evidence

match-evidence-inspector

When developing a theory about what types of rules should be used and how much evidence for a particular match they should provide, it is often useful to explicitly see what the different sources of evidence were for a particular match item. The system utility match-evidence-inspector may be used to display a trace of the entire evidence facility or just the evidence for a particular match hypothesis or Gmap. For example, the following information was printed out about the pressure to temperature match hypothesis in the water flow – heat flow analogy:

(MH F#PRESS-BEAKER F#TEMP-COFFEE) has evidence (0.7120, 0.0000) due to
IMPLICATION((MH F#PRESSURE F#TEMP)) (0.5200, 0.0000)
IMPLICATION(CHILDEREN-POTENTIAL) (0.4000, 0.0000)
While the following information appears for the best Gmap in this analogy:

(GMAP #GM3) has evidence (5.9917, 0.0000) due to
IMPLICATION((MF MF#FLOW F#HFLOW)) (0.7900, 0.0000)
IMPLICATION((MF MF#PIPE20 MF#BAR23)) (0.6320, 0.0000)
IMPLICATION((MF MF#WATER17 MF#HEAT24)) (0.6320, 0.0000)
IMPLICATION((MF MF#VIAL19 MF#ICE-CUBE22)) (0.9318, 0.0000)
IMPLICATION((MF MF#BEAKER18 MF#COFFEE21)) (0.9318, 0.0000)
IMPLICATION((MF MF#PRESS-BEAKER MF#TEMP-COFFEE)) (0.7120, 0.0000)
IMPLICATION((MF MF#PRESS-VIAL MF#TEMP-ICE-CUBE)) (0.7120, 0.0000)
IMPLICATION((MF MF#PRESS-PRESSURE MF#TEMP)) (0.6500, 0.0000)

The inspection facility will not work for Gmaps if their scores were produced by an external (to the BMS), destructive operation. One such destructive rule appeared at the end of Section 4.1.3.

6.6 Windows

dump-scroll-menu
dump-scroll output-patname
clear-scroll
select-windowing-configuration
select-scroll
select-double-scroll
select-graphics
select-large-graphics
select-split
*sme-frame*
*graphics-pane*
*scroll-pane*
*sparse-scroll-pane*
*lisp-pane*

The windowing system is lisp machine dependent and appears in the file windowing.lisp. The loading of this file is optional. When the windowing system is used, a number of window configurations are possible, such as having a single scroll window, two side by side, a single graphics window, or a scroll and graphics window side by side. These configurations may be selected through their individual functions (e.g., select-scroll), or through the central configuration facility select-scroll-graphics. By default, when the windowing system is active, all SME output is sent to the primary scroll pane. When both scroll windows are in use, the configuration facility allows one to specify which scroll window is currently active. The two scroll dumping routines write the contents of the primary scroll window to a specified file. Output sent to the secondary (scratchpad) scroll pane is for observation only and cannot be written to a file.

6.7 System parameters

*sme-parameters*
*parameter-menu-options*
defSME-Parameter  variable-name string-description type &optional type-choices

The defSME-Parameter form adds a new variable to the list of known SME parameters. This list is used by the windowing interface routines to query the user about possible parameter settings. It is provided primarily for application programs wanting to use the standard SME parameter
setting facility. The arguments to defSME-Parameter correspond to the appropriate definitions for the choose-variable-values function of your particular machine. For example, the following declaration appears in match.lisp:

```
(defSME-Parameter *display-all-MH* "Display all the Match Hypotheses" :boolean)
```

The windowing system provides a menu facility for viewing and changing the current values of system parameters. This menu is shown in Figure 13.

6.8 System utilities

```
*system-utilities-menu*
defSME-Utility string-name lisp-form
menu-utilities
get-dgroup
get-rules
```

The SME system utilities are the options that appear when the Utilities command is evoked, the right mouse button is pressed, or the function menu-utilities is called. These utilities include changing the system parameters, choosing to load a Dgroup or rule file from those in the defined directories, and clearing or writing to file the contents of the scroll window. These routines are lisp machine dependent. The following declaration appears in match.lisp:

```
(defSME-Utility "Inspect Evidence" (match-evidence-inspector))
```

7 User Hooks

This section describes the global variables and routines that are available to the user and application programs for the creation and inspection of analogical mappings.

7.1 Applications control over display

```
*sme-output-stream*  [Variable]
*windowing?*          [Variable]
*sme-graphics-output* [Variable]
```
All SME textual display routines send their output to *sme-output-stream*. By default, the value of this variable is T, which causes output to be sent to *terminal-io*, Common Lisp's default pointer to the user's console. When *sme-output-stream* is a scroll window (determined by the presence of an :append-item handler), the appropriate scroll window routines for sending display items are invoked. Otherwise, text is sent to the current output stream using format. Text routines are machine independent.

In a similar manner, all SME graphics output is sent to the current *sme-graphics-output* window. Graphics output is lisp machine dependent and relies on the ZGRAPH graphics system.

When the SME windowing system is in operation, *sme-output-stream* is set to the primary SME scroll pane (*scroll-pane*), *sme-graphics-output* is set to the SME graphics pane (*graphics-pane*), and windowing? is set to T.

`sme-format format-string &rest format-args` [Macro]
`sme-print string` [Function]
`sme-terpri &optional (N i)` [Function]

These routines provide a general interface for sending textual output to the current SME output stream. sme-format is equivalent to CommonLisp's format routine, except that the printed output is always followed by a newline. The sme-print routine is provided for simple situations where only a string is printed or for situations requiring the standard use of format, as in building up a line of text through multiple invocations. The printed output of sme-print is followed by a newline. When the routine is used for multiple calls of format, it should be used in conjunction with CommonLisp's with-output-to-string, as in:

```lisp
(sme-print
 (with-output-to-string (stream)
   (format stream "Beginning of a line...")
   (format stream "middle of a line...")
   (format stream "end of a line.")))
```

When a whole set of operations are carried out within the context of a single sme-print, one must be careful not nest calls to sme-print (e.g., calling a function in the context of an sme-print which itself invokes sme-print). Such nesting will cause output to appear backwards from what was intended and may cause the output stream to close improperly.

### 7.2 Useful miscellaneous functions

Data exists within SME in three forms: (1) local items such as entities, predicates, and expressions, (2) description groups (Dgroups), and (3) analogical mapping information. The routines to create and query these items are described in the following sections.

#### 7.2.1 Entities, predicates, and expressions

/entity? item` [Function]
/entity-type? item` [Function]
/entity-name? symbol` [Macro]
/fetch-entity-definition symbol` [Macro]
/entity-domain symbol` [Macro]
/constant-entity? symbol` [Macro]
Entities declared through defEntity represent global entity types, that is, they represent a class of entities rather than an actual instance of an entity. When an entity type is used in the definition of a description group, a unique entity instance is created for that type (e.g., beaker is translated to beaker73). Thus, a given entity token will represent either a type or an instance. The structure predicate entity? returns true if the given item is an entity-instance structure, while entity-type? returns true if the item is an entity-type structure. The macro entity-name? returns true if the given symbol represents either an entity type or instance. fetch-entity-definition will return the entity-type structure for an entity type token or the entity-instance structure for an entity instance token. The routines entity-domain and constant-entity? refer to the type and constant? keyword values given to defEntity when the corresponding entity type was created.

*sme-predicates* [Variable]
fetch-predicate-definition predicate-symbol [Macro]
predicate? symbol [Macro]
predicate-type predicate-symbol [Macro]
relation? predicate-symbol [Macro]
attribute? predicate-symbol [Macro]
function? predicate-symbol [Macro]
commutative? predicate-symbol [Macro]
n-ary? predicate-symbol [Macro]
arg-list predicate-symbol [Macro]
numargs predicate-symbol [Macro]
expression-type predicate-symbol [Macro]
eval-form predicate-symbol [Macro]

These routines provide the facility to access the various predicate properties defined with the defPredicate form. fetch-predicate-definition returns the actual sme-predicate structure containing all the information about a given predicate.

fetch-expression expression-name dgroup &optional (error-if-absent? T) [Function]
expression-functor expression-structure [Function]
fully-expand-expression expression-structure dgroup [Function]
fully-expand-expression-form expression-form dgroup [Function]

An “expression” represents an actual predicate instance within a Dgroup. Notice that this includes terms corresponding to function applications as expressions. Each use of a predicate gets its own expression with its own name, so that a higher-order relation gets translated into several expressions, with some having expressions as arguments. These routines allow one to retrieve and inspect expressions in the database. fetch-expression returns the expression structure with the given name.

The routines fully-expand-expression and fully-expand-expression-form are useful for examining the form of an expression. Typically, the expression (greater-than (diameter beaker) 5) is stored as the expression greater-than23, which has the form (greater-than diameter24 5). These routines return a fully expanded expression form, where all expression names are replaced by their corresponding forms.
7.2.2 Dgroups

 fetch-dgroup  
dgroup-name &optional create?
  [Function]

 return-dgroup  
dgroup-or-dgroup-name
  [Function]

Description groups (Dgroups) are stored in a simple data base managed primarily by routines in
sme.lisp. The general procedures for Dgroup and expression creation were described in sections 3.3
and 3.4. A Dgroup may be retrieved by name using fetch-dgroup, or created if create? is non-nil
and no Dgroup with the given name currently exists. return-dgroup is designed for routines that
may take either an actual Dgroup or simply a Dgroup name (e.g., fetch-expression). It will
cause an error if the Dgroup does not previously exist.

describe-dgroup  
dgroup
  [Function]

 menu-display-dgroup
  [Function]

 menu-display-pairs
  [Function]

A Dgroup may be textually described using describe-Dgroup, which writes to the SME output
stream. Graphical display is provided in the windowing system by menu-display-dgroup for a
single Dgroup or menu-display-pairs for a display of two Dgroups side by side.

7.2.3 Creating and inspecting global matches

 match  
 base-name target-name &optional display?
  [Function]

 best-gmaps &optional (gmaps *gmaps*) (percentage-range 0.02)
  [Function]

display-match  
 base target &optional (total-run-time 0) (bms-run-time 0)
  [Function]

The match function is the central SME procedure. Given the names of two Dgroups, it forms the
complete set of global mappings between them. If display? is non-nil, a description of the results
will be sent to the current SME output stream. The function itself returns two values, the total run
time of the match process in seconds and the subset of that time spent running the BMS evidence
rules. The analogical mapping results are stored in the following global variables, which are then
accessible by the user or application program.

*base*  
[Variable]

*target*  
[Variable]

*match-hypotheses*  
[Variable]

*gmaps*  
[Variable]

The Gmap(s) with the highest evaluation score are retrieved by best-gmaps, which returns all
Gmaps having a score within a given percentage (default is 2%) of the highest score. best-gmaps
returns two values: the list of best Gmaps and the actual real-valued highest score.

 compare-gmaps  
gmap1 gmap2 &optional display?
  [Function]

 Occasionally, SME will produce two or more “best” Gmaps that appear to be identical yet have
been classified as distinct. When these Gmaps are large, the “here’s the set of match hypotheses”
output format can make it frustrating to find what the slight differences are between a pair of
Gmaps. When given two Gmap structures, compare-gmaps will return (using values) the list of
the match hypotheses that are uniquely part of the first Gmap and a list of match hypotheses that
are uniquely part of the second Gmap (i.e., (gmap1 - gmap2) and (gmap2 - gmap1)). When the
windowing system is active, this option is available through the system utilities menu.
8 Algorithm Internals

This section quickly describes a few internal points of the program in case one has specialized needs for interfacing to the code. It assumes knowledge of CommonLisp and the realization that for many questions, the only feasible answer must be to examine the SME program.

8.1 The Match Function

The function match is primarily a sequence of calls to these functions. create-match-hypotheses runs the match constructor rules to form the individual match hypotheses. The BMS evidence rules are then run on these match hypotheses (run-rules) and their dependence and inconsistency relationships are determined (calculate-nogoods). The function generate-gmaps executes the three merge steps, resulting in the set of complete global mappings being placed in the variable *gmaps*. The candidate inferences each Gmap sanctions is then calculated (gather-inferences) and additive BMS nodes for each Gmap are formed (intern-gmaps), allowing the evidence rules to run on each Gmap.

8.2 Match Hypotheses

Most programs using SME will need to interact with the match hypothesis structures. Slots to this structure type use the mh- prefix. There are several slots that might be important. The MH form, which is a list of (MH <base-item> <target-item>), is found using mh-form. This is the form used for triggering the MH evidence rules, and is asserted in the BMS for each match hypothesis. The base and target items are expression or entity structures. The base item or target item may be obtained directly using mh-base-item and mh-target-item, respectively. The BMS node for each match hypothesis is found by mh-bms-node. In turn, the weight for that node may be obtained using node-belief+. Finally, each match hypothesis structure has a property list slot (mh-plist) which may be useful for various purposes.

8.3 Global Mappings
Each Gmap is stored as a *global-mapping* structure. Slots to this structure type use the gm-prefix. Each Gmap is assigned a unique integer identifier, found through gm-id. The Gmap form used by the BMS is not explicitly available, but is asserted as (Gmap <gmap-structure>). The match hypotheses associated with a Gmap are stored in gm-elements, while the subset of these that are entity mappings is stored in gm-emaps. The candidate inferences sanctioned by the Gmap appear in gm-inferences and are stored as a simple list data type, using the syntax defined in Section 3.3 for description group expressions.

### 8.4 Candidate Inference Generation

The original candidate inference generator, as described in [8], will take any base structure "intersecting the Gmap structure". The newer (V. 2) edition only takes base structure "intersecting a Gmap root". Thus, the newer edition is more cautious and far more efficient than the older edition. Both versions of the code are available (in match.lisp), with the default being the newer, more cautious version. There are theoretical arguments for and against each approach. For example, one might want to use only the inferences from the newer, more cautious approach at first since they are supported by more target knowledge and thus more likely valid. If an analogy proves fruitful, one may want to relax these constraints, and use the older version to find out what additional inferences might be made.

### 8.5 Rule System

When a new rules file is loaded, the sme-rules-file command at the top of the file initializes the BMS rule system prior to loading the new set of rules. At the bottom of the rules file, the fresh, just-loaded set of rules are saved in the global variable *tre-rules-saver* by the command tre-save-rules. This variable saves the status of the rule counters and the list of initial rules. A similar variable, *initial-assertions*, is used to store all assertions appearing in the rules file. When the BMS is run, new rules may be created and added to the known set of rules. As a result, each time match is invoked, the BMS is reinitialized to its status just after loading the rule file, that is, it is restored to the status indicated in *tre-rules-saver*.

This facility may be used by application programs to save different rule sets in memory and swap them as needed, without having to load rule files each time. For example, suppose SME is
invoked, which will cause it to run the current rule set for Gmap scoring. If a second scoring
criterion is then desired, a second set of rules may be invoked using code of the form:

```lisp
(let ((save-rules sme:*tre-rules-saver*) ; save the previous rule set
      (save-assertions sme:*initial-assertions*))
  (setq sme:*tre-rules-saver* *my-other-rules-set-rules*)
  (setq sme:*initial-assertions* *my-other-rules-set-assertions*)
  (sme:tre-init) ; initialize with new rules
  (sme:run-rules) ; run the new rule set
  (setq sme:*tre-rules-saver* save-rules) ; restore the original rules
  (setq sme:*initial-assertions* save-assertions))
```

This saves SME's normal rule set, runs a different one, and then restores the rule set to its
previous value. In this example, tre-init was used, which fully initializes the BMS. If the desire
is simply to supply additional rules without destroying the current BMS state, restore-rules
should be used in place of tre-init. The variables corresponding to "my-other-rules-set" may be
initialized by a similar program which saves the current rule set, loads the desired "other rule set"
file, sets the "my-other-rules-set" variables from *tre-rules-saver* and *initial-assertions,
and then restores the original rule set.

9 Summary

The SME program has been described from the perspective of how to actually use it. A number of
methods have been presented about how to configure SME to perform a variety of different types
of matches. It is hoped that SME may serve as a general mapping tool for research on analogical
mapping, allowing researchers to focus on the more substantive issue of general theories of analogical
mapping, as opposed to worrying about implementation details. The latter has the unfortunate
effect of producing the repeated scenario in which analogy researcher A goes to analogy researcher
B and says "My program can do X, which yours cannot", followed by researcher B returning a
month later with this simple modification added. By testing different theories within the same
program, we may now compare the more critical "This is a logical consequence of my theory". A
program does not a theory make. It can, however, function as a useful analytical tool.

Of course, not all of the problems of implementing analogical mapping have been solved. Most
critical is redesigning the potentially combinatoric merge step 3, perhaps using either a heuristic
search or connectionist relaxation network as suggested in [9]. Of theoretical relevance is the
appropriateness of the abstract structural approach which SME embodies.

10 Acknowledgements

The development of SME has been a collaborative effort with Ken Forbus and Dedre Gentner, with
significant influence provided by Janice Skorstad. This work has also benefited from discussions
with Steve Chien, John Collins, and Ray Mooney.

This research is supported in part by an IBM graduate fellowship and in part by the Office of

References

[1] Burstein, M., Concept formation by incremental analogical reasoning and debugging, Pro-
ceedings of the Second International Workshop on Machine Learning, University of Illinois,
REFERENCES


[18] Holyoak, K., & Thagard, P. Analogical mapping by constraint satisfaction, June, 1988, (*submitted for publication*).


Index

algorithm, 4-9, 9
   summary, 3
analogical hint, 16
arg-list macro, 27
assert! function, 13
attribute? macro, 27

*base* variable, 28
batch mode, 22
best-gmaps function, 28
calculate-nogoods function, 29
candidate inference, 8
   alternate algorithms, 30
gather-inferences, 29
change-parms function, 25
children-of? function, 14
clear-given-mappings function, 17
clear-scroll function, 24
commutative? macro, 27
compare-gmaps function, 28
constant-entity? macro, 27
conventions, 1
   file names, 19
   function names, 1
create-match-hypotheses function, 29
declare-given-mappings function, 17
defDescription macro, 11
defEntity macro, 10
define-description macro, 11
define-Entity function, 10
define-predicate function, 10
defPostMatcher macro, 22
defPredicate macro, 10
defSME-Parameter macro, 25
defSME-Utility macro, 25
describe-dgroup function, 28
description group, 11
   graphical display, 28
   retrieval, 28
   textual description, 28
Dgroup, 11
dgroup-directory macro, 22
dgroup-file macro, 22
display-match function, 28
dump-scroll function, 24
dump-scroll-menu function, 24

dump-scroll function, 24
dump-scroll-menu function, 24
dump-scroll-menu function, 24

dump-scroll function, 24
dump-scroll-menu function, 24

dump-scroll function, 24
dump-scroll-menu function, 24

dump-scroll function, 24
dump-scroll-menu function, 24

dump-scroll function, 24
dump-scroll-menu function, 24

dump-scroll function, 24
dump-scroll-menu function, 24

dump-scroll function, 24
dump-scroll-menu function, 24

dump-scroll function, 24
dump-scroll-menu function, 24

dump-scroll function, 24
dump-scroll-menu function, 24

dump-scroll function, 24
dump-scroll-menu function, 24

entity, 10
declaring, 10
entity? function, 26
   inspection functions, 27
entity-domain macro, 27
entity-name? macro, 27
entity-type? function, 26
eval-type? macro, 27
execution, 20
expression, 11
   adding new expressions, 11
   expression function, 11
   inspection functions, 27
   expression-functor function, 27
   expression-type macro, 27
fetch-dgroup function, 28
fetch-entity-definition macro, 27
fetch-expression function, 27
fetch-predicate-definition macro, 27
file organization, 1
fully-expand-expression function, 27
fully-expand-expression-form function, 27
function? macro, 27
gather-inferences function, 29
generalize, 15
generalize function, 23
generate-gmaps function, 29
get-dgroup function, 25
get-rules function, 25
global mapping, 5, 7-9, 9
   comparing apparently identical gmaps, 28
   creation, 28
defstruct, 30
   *gmaps*, 28
   scoring, 8, 14
   selecting the best, 28
textual display, 28
   *gmaps* variable, 28, 30
gm-base subst, 30
gm-bms-node subst, 30
gm-elements subst, 30
INDEX

gm-emaps subst, 30
gm-id subst, 30
gm-inferences subst, 30
gm-plist subst, 30
gm-target subst, 30
*graphics-pane* variable, 24

initial-assertion macro, 13
*initial-assertions* variable, 30
installation, 19
install-MH function, 12
intern-gmaps function, 29

language-file macro, 22
lisp machine interface, 24	predicate documentation, 10
*lisp-pane* variable, 24

match constructor rules, 12
match evidence rules, 13
match function, 28
match hypothesis, 4-5, 5
defstruct, 29
inspecting evidence justifications, 23
installing, 12
*match-hypotheses*, 28
scoring, 13-14, 14
match-evidence-inspector function, 23
*match-hypotheses* variable, 28-29, 29
menu-display-dgroup function, 28
menu-display-pairs function, 28
menu-utilities function, 25
mh-base-item subst, 29
mh-bms-node subst, 29
MHC-rule, 12
mh-form subst, 29
mh-plist subst, 29
mh-target-item subst, 29

n-ary? macro, 27
node-belief+ subst, 29
numargs macro, 27

packages, 1
	site specific, 19
paired-item? function, 18
*parameter-menu-options* variable, 25
parameters, 25
	site specific, 1, 19

predicate, 10
declaring, 10
inspection functions, 27
predicate? macro, 27
predicate-type macro, 27
rassert! macro, 13
relation? macro, 27
report generation, 22
report-comments macro, 22
representation issues, 18
restore-rules function, 30
return-dgroup function, 28
rule macro, 13

rule system, 12-18, 18
	analogical hints, 16	dynamically swapping rule sets, 30
giving it access to gmaps, 29
match constructor rules, 12
match evidence rules, 13
pure isomorphisms, 16
relaxing identical predicates, 15
rule file syntax, 12
run-rules, 29
simulating SPROUTER, 15
simulating structure-mapping theory, 15
rule-directory macro, 22
rule-file macro, 22
rule-sets macro, 22
run-batch-file function, 22
run-matcher-on macro, 22
running SME, 20
run-rules function, 29-30, 30
sanctioned-pairing? function, 18
scroll windows, 24
	saving contents, 24
	writing to, 24, 26
*scroll-pane* variable, 24
select-double-scroll function, 24
select-graphics function, 24
select-large-graphics function, 24
select-scroll function, 24
select-split function, 24
select-windowing-configuration function, 24
send-report-to macro, 22
site specific information, 19
packages, 19
INDEX

pathnames, 19
*sme-default-rules* variable, 19
*sme-dgroup-pathname* variable, 19
*sme-files* variable, 19
sme-format macro, 26
*sme-frame* variable, 24
*sme-graphics-output* variable, 26
sme-init function, 20
*sme-language-file* variable, 19
*sme-output-stream* variable, 26
*sme-parameters* variable, 25
*sme-predicates* variable, 27
sme-print function, 26
sme-rules-file function, 12
*sme-rules-pathname* variable, 19
*sme-system-pathname* variable, 19
sme-terpri function, 26
*sparse-scroll-pane* variable, 24
system utilities, 25
*system-utilities-menu* variable, 25
*target* variable, 28
*the-lisp-package* variable, 19
*the-user-package* variable, 19
tre-init function, 30
tre-rules-file function, 30
*tre-rules-saver* variable, 30
tre-save-rules function, 12, 30
*windowing?* variable, 26
Distribution List [Illinois/Gentner] NR 667-551

Dr. Phillip L. Ackerman  
University of Minnesota  
Department of Psychology  
75 East River Road  
N218 Elliott Hall  
Minneapolis, MN 55455

Dr. Beth Adelson  
Department of Computer Science  
Tufts University  
Medford, MA 02155

AFOSR,  
Life Sciences Directorate  
Bolling Air Force Base  
Washington, DC 20332

Dr. Robert Ahlers  
Code 7N11  
Human Factors Laboratory  
Naval Training Systems Center  
Orlando, FL 32813

Dr. John R. Anderson  
Department of Psychology  
Carnegie-Mellon University  
Schenley Park  
Pittsburgh, PA 15213

Dr. Stephen J. Andriole, Chairman  
Department of Information Systems and Systems Engineering  
George Mason University  
4400 University Drive  
Fairfax, VA 22030

Technical Director, ARI  
5001 Eisenhower Avenue  
Alexandria, VA 22333

Dr. Patricia Baggett  
School of Education  
610 E. University, Rm 1302D  
University of Michigan  
Ann Arbor, MI 48109-1259

Dr. Eva L. Baker  
UCLA Center for the Study of Evaluation  
145 Moore Hall  
University of California  
Los Angeles, CA 90024

Dr. Meryl S. Baker  
Navy Personnel R&D Center  
San Diego, CA 92152-5600

prof. dott. Bruno G. Bara  
Unita di ricerca di intelligenza artificiale  
Università di Milano  
20122 Milano – via F. Sforza 23  
ITALY

Dr. William M. Bart  
University of Minnesota  
Dept. of Educ. Psychology  
330 Burton Hall  
178 Pillsbury Dr., S.E.  
Minneapolis, MN 55455

Leo Beltracchi  
United States Nuclear Regulatory Commission  
Washington DC 20555

Dr. Gautam Biswas  
Department of Computer Science  
Box 1688, Station B  
Vanderbilt University  
Nashville, TN 37235

Dr. John Black  
Teachers College, Box 8  
Columbia University  
525 West 120th Street  
New York, NY 10027

Dr. John Burns  
Army Research Institute  
ATTN: PERI-SF  
5001 Eisenhower Avenue  
Alexandria, VA 22333-5600

Dr. Sue Bogner  
Army Research Institute  
ATTN: PERI-SF  
5001 Eisenhower Avenue  
Alexandria, VA 22333-5600

Dr. Jeff Bonar  
Learning R&D Center  
University of Pittsburgh  
Pittsburgh, PA 15260

Dr. Gordon H. Bower  
Department of Psychology  
Stanford University  
Stanford, CA 94306

Dr. Robert Breaux  
Code 7B  
Naval Training Systems Center  
Orlando, FL 32813-7100

Dr. Ann Brown  
Center for the Study of Reading  
University of Illinois  
51 Gerty Drive  
Champaign, IL 61820

Dr. John B. Brown  
Univereita di Milano  
University of Illinois  
Department of Computer Science  
20122 Milano – via F. Sforza 23  
ITALY

Dr. Beth Adelson  
Univerita di Milano  
University of Illinois  
Department of Computer Science  
20122 Milano – via F. Sforza 23  
ITALY

Dr. John T. Bruer  
James S. McDonell Foundation  
Suite 1510  
1034 South Brentwood Blvd.  
St. Louis, MO 63117

Dr. Bruce Buchanan  
Computer Science Department  
Stanford University  
Stanford, CA 94305

LT COL Hugh Burns  
AFHRL/IDI  
Brooks AFB, TX 78235

Dr. Joseph C. Campione  
Center for the Study of Reading  
University of Illinois  
51 Gerty Drive  
Champaign, IL 61820

Dr. Joel C. Cappel, Director  
Center for Research into Practice  
1718 Connecticut Ave., N.W.  
Washington, DC 20009

Dr. James G. Carbonell  
Computer Science Department  
Carnegie-Mellon University  
Schenley Park  
Pittsburgh, PA 15213

Dr. Susan Carey  
Department of Cognitive and Neural Science  
MIT  
Cambridge, MA 02139
Dr. Pat Carpenter  
Carnegie-Mellon University  
Department of Psychology  
Pittsburgh, PA 15213

Assistant Chief of Staff  
for Research, Development,  
Test, and Evaluation  
Naval Education and  
Training Command (N-5)  
NAS Pensacola, FL 32508

CDR Robert Carter  
Office of the Chief  
of Naval Operations  
OP-933D4  
Washington, DC 20350-2000

Dr. Allan M. Collins  
Bolt Beranek & Newman, Inc.  
10 Moulton Street  
Cambridge, MA 02238

Chair, Dept. of Psychology  
College of Arts and Sciences  
Catholic Univ. of America  
Washington, DC 20064

Dr. Stanley Collyer  
Office of Naval Technology  
Code 222  
800 N. Quincy Street  
Arlington, VA 22217-5000

Dr. Fred Chang  
Pacific Bell  
2600 Camino Ramon  
Room 35-450  
San Ramon, CA 94583

Dr. Davida Charney  
English Department  
Penn State University  
University Park, PA 16802

Georgy Delacote  
Directeur de L'informatique  
Scientifique et Technique  
CNRS  
15, Quai Anatole France  
75700 Paris, FRANCE

Dr. Michele Chi  
Learning R & D Center  
University of Pittsburgh  
3333 O'Hara Street  
Pittsburgh, PA 15230

Dr. Denise Dellarosa  
Psychology Department  
Box 11A, Yale Station  
Yale University  
New Haven, CT 06520-7447

Dr. William Clancey  
Institute for Research  
on Learning  
3333 Coyote Hill Road  
Palo Alto, CA 94304

Dr. Andre di Sessa  
University of California  
School of Education  
Tolman Hall  
Berkeley, CA 94720

Dr. Charles Clifton  
Tobin Hall  
Department of Psychology  
University of  
Massachusetts  
Amherst, MA 01003

Dr. R. K. Dismukes  
Associate Director for Life Sciences  
AFOSR  
Boeing AFB  
Washington, DC 20332

CDR Robert Carter  
Office of the Chief  
of Naval Operations  
OP-933D4  
Washington, DC 20350-2000

Dr. Allan M. Collins  
Bolt Beranek & Newman, Inc.  
10 Moulton Street  
Cambridge, MA 02238

Chair, Dept. of Psychology  
College of Arts and Sciences  
Catholic Univ. of America  
Washington, DC 20064

Dr. Stanley Collyer  
Office of Naval Technology  
Code 222  
800 N. Quincy Street  
Arlington, VA 22217-5000

Dr. Fred Chang  
Pacific Bell  
2600 Camino Ramon  
Room 35-450  
San Ramon, CA 94583

Dr. Davida Charney  
English Department  
Penn State University  
University Park, PA 16802

Georgy Delacote  
Directeur de L'informatique  
Scientifique et Technique  
CNRS  
15, Quai Anatole France  
75700 Paris, FRANCE

Dr. Michele Chi  
Learning R & D Center  
University of Pittsburgh  
3333 O'Hara Street  
Pittsburgh, PA 15230

Dr. Denise Dellarosa  
Psychology Department  
Box 11A, Yale Station  
Yale University  
New Haven, CT 06520-7447

Dr. William Clancey  
Institute for Research  
on Learning  
3333 Coyote Hill Road  
Palo Alto, CA 94304

Dr. Andre di Sessa  
University of California  
School of Education  
Tolman Hall  
Berkeley, CA 94720

Dr. Charles Clifton  
Tobin Hall  
Department of Psychology  
University of  
Massachusetts  
Amherst, MA 01003

Dr. R. K. Dismukes  
Associate Director for Life Sciences  
AFOSR  
Boeing AFB  
Washington, DC 20332
<table>
<thead>
<tr>
<th>Name</th>
<th>Institution/Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mr. Wallace Feurteig</td>
<td>Educational Technology</td>
</tr>
<tr>
<td></td>
<td>Boit Beranek &amp; Newman</td>
</tr>
<tr>
<td></td>
<td>10 Moulton St.</td>
</tr>
<tr>
<td></td>
<td>Cambridge, MA 02238</td>
</tr>
<tr>
<td>Dr. Gerhard Fischer</td>
<td>University of Colorado</td>
</tr>
<tr>
<td></td>
<td>Department of Computer Science</td>
</tr>
<tr>
<td></td>
<td>Boulder, CO 80309</td>
</tr>
<tr>
<td>Dr. J. D. Fletcher</td>
<td>Institute for Defense Analyses</td>
</tr>
<tr>
<td></td>
<td>1801 N. Beauregard St.</td>
</tr>
<tr>
<td></td>
<td>Alexandria, VA 22311</td>
</tr>
<tr>
<td>Dr. Linda Flower</td>
<td>Carnegie-Mellon University</td>
</tr>
<tr>
<td></td>
<td>Department of English</td>
</tr>
<tr>
<td></td>
<td>Pittsburgh, PA 15213</td>
</tr>
<tr>
<td>Dr. Kenneth D. Forbus</td>
<td>University of Illinois</td>
</tr>
<tr>
<td></td>
<td>Department of Computer Science</td>
</tr>
<tr>
<td></td>
<td>Urbana, IL 61801</td>
</tr>
<tr>
<td>Dr. Barbara A. Fox</td>
<td>University of Colorado</td>
</tr>
<tr>
<td></td>
<td>Department of Linguistics</td>
</tr>
<tr>
<td></td>
<td>Boulder, CO 80309</td>
</tr>
<tr>
<td>Dr. John R. Frederiksen</td>
<td>BBN Laboratories</td>
</tr>
<tr>
<td></td>
<td>10 Moulton Street</td>
</tr>
<tr>
<td></td>
<td>Cambridge, MA 02238</td>
</tr>
<tr>
<td>Dr. Norman Frederiksen</td>
<td>Educational Testing Service (AT&amp;T)</td>
</tr>
<tr>
<td></td>
<td>Princeton, NJ 08541</td>
</tr>
<tr>
<td>Julie A. Gaddams</td>
<td>Information Technology</td>
</tr>
<tr>
<td></td>
<td>Applications Division</td>
</tr>
<tr>
<td></td>
<td>Admiralty Research Establishment</td>
</tr>
<tr>
<td></td>
<td>Portadown, Portmouth PO8 4AA</td>
</tr>
<tr>
<td></td>
<td>UNITED KINGDOM</td>
</tr>
<tr>
<td>Dr. Dedra Gentner</td>
<td>University of Illinois</td>
</tr>
<tr>
<td></td>
<td>Department of Psychology</td>
</tr>
<tr>
<td></td>
<td>603 E. Daniel St.</td>
</tr>
<tr>
<td></td>
<td>Champaign, IL 61820</td>
</tr>
<tr>
<td>Chair, Department of Psychology</td>
<td>Georgetown University</td>
</tr>
<tr>
<td></td>
<td>Washington, DC 20057</td>
</tr>
<tr>
<td>Dr. Robert Glaser</td>
<td>Learning Research &amp; Development Center</td>
</tr>
<tr>
<td></td>
<td>University of Pittsburgh</td>
</tr>
<tr>
<td></td>
<td>3939 O'Hara Street</td>
</tr>
<tr>
<td></td>
<td>Pittsburgh, PA 15260</td>
</tr>
<tr>
<td>Dr. Arthur M. Glieberg</td>
<td>University of Wisconsin</td>
</tr>
<tr>
<td></td>
<td>W. J. Brogden Psychology Bldg.</td>
</tr>
<tr>
<td></td>
<td>1202 W. Johnson Street</td>
</tr>
<tr>
<td></td>
<td>Madison, WI 53706</td>
</tr>
<tr>
<td>Dr. Sam Glucksberg</td>
<td>Department of Psychology</td>
</tr>
<tr>
<td></td>
<td>Princeton University</td>
</tr>
<tr>
<td></td>
<td>Princeton, NJ 08540</td>
</tr>
<tr>
<td>Dr. Susan R. Goldman</td>
<td>Dept. of Education</td>
</tr>
<tr>
<td></td>
<td>University of California</td>
</tr>
<tr>
<td></td>
<td>Santa Barbara, CA 92108</td>
</tr>
<tr>
<td>Dr. Sherrie Gott</td>
<td>AFHRL/MOMJ</td>
</tr>
<tr>
<td></td>
<td>Brooks AFB, TX 78235-5601</td>
</tr>
<tr>
<td>Dr. T. Govindaraj</td>
<td>Georgia Institute of Technology</td>
</tr>
<tr>
<td></td>
<td>School of Industrial and Systems Engineering</td>
</tr>
<tr>
<td></td>
<td>Atlanta, GA 30332-0205</td>
</tr>
<tr>
<td>Dr. Wayne Gray</td>
<td>Army Research Institute</td>
</tr>
<tr>
<td></td>
<td>5001 Eisenhower Avenue</td>
</tr>
<tr>
<td></td>
<td>Alexandria, VA 22333</td>
</tr>
<tr>
<td>Dr. James G. Greeno</td>
<td>School of Education</td>
</tr>
<tr>
<td></td>
<td>Stanford University</td>
</tr>
<tr>
<td></td>
<td>Room 311</td>
</tr>
<tr>
<td></td>
<td>Stanford, CA 94305</td>
</tr>
<tr>
<td>Dr. Dick Gregory</td>
<td>Admiralty Research Establishment/AXB</td>
</tr>
<tr>
<td></td>
<td>Queens Road</td>
</tr>
<tr>
<td></td>
<td>Teddington</td>
</tr>
<tr>
<td></td>
<td>Middlesex, ENGLAND TW110LN</td>
</tr>
<tr>
<td>Dr. Gerhard Grossing</td>
<td>Atom institut</td>
</tr>
<tr>
<td></td>
<td>Schotttalstrasse 115</td>
</tr>
<tr>
<td></td>
<td>Vienna</td>
</tr>
<tr>
<td></td>
<td>AUSTRIA A-1020</td>
</tr>
<tr>
<td>Prof. Edward Haertel</td>
<td>School of Education</td>
</tr>
<tr>
<td></td>
<td>Stanford University</td>
</tr>
<tr>
<td></td>
<td>Stanford, CA 94305</td>
</tr>
<tr>
<td>Dr. Henry M. Halfa</td>
<td>Halff Resources, Inc.</td>
</tr>
<tr>
<td></td>
<td>4918 33rd Road, North</td>
</tr>
<tr>
<td></td>
<td>Arlington, VA 22207</td>
</tr>
<tr>
<td>Dr. Ronald K. Hamblyton</td>
<td>University of Massachusetts</td>
</tr>
<tr>
<td></td>
<td>Laboratory of Psychometric and Evaluative Research</td>
</tr>
<tr>
<td></td>
<td>Hills South, Room 152</td>
</tr>
<tr>
<td></td>
<td>Amherst, MA 01003</td>
</tr>
<tr>
<td>Dr. Bruce W. Hamill</td>
<td>Research Center</td>
</tr>
<tr>
<td></td>
<td>The Johns Hopkins University</td>
</tr>
<tr>
<td></td>
<td>Applied Physics Laboratory</td>
</tr>
<tr>
<td></td>
<td>Johns Hopkins Road</td>
</tr>
<tr>
<td></td>
<td>Laurel, MD 20707</td>
</tr>
<tr>
<td>Steve Hanard</td>
<td>Editor, The Behavioral and Brain Sciences</td>
</tr>
<tr>
<td></td>
<td>20 Nassau Street, Suite 240</td>
</tr>
<tr>
<td></td>
<td>Princeton, NJ 08542</td>
</tr>
<tr>
<td>Dr. Reid Hastie</td>
<td>Northwestern University</td>
</tr>
<tr>
<td></td>
<td>Department of Psychology</td>
</tr>
<tr>
<td></td>
<td>Evanston, IL 60206</td>
</tr>
<tr>
<td>Dr. John R. Hayes</td>
<td>Carnegie-Mellon University</td>
</tr>
<tr>
<td></td>
<td>Department of Psychology</td>
</tr>
<tr>
<td></td>
<td>Schenley Park</td>
</tr>
<tr>
<td></td>
<td>Pittsburgh, PA 15213</td>
</tr>
</tbody>
</table>
Distribution List [Illinois/Gentner] NR. 857-551

Dr. Barbara Hayes-Roth
Knowledge Systems Laboratory
Stanford University
701 Welch Road
Palo Alto, CA 94304

Dr. Janet Jackson
Rijksuniversiteit Groningen
Biologisch Centrum, Vleugel D
Kerklaan 30, 9751 NN Haren
The NETHERLANDS

Dr. Robert Jannarone
University of South Carolina
Columbia, SC 29208

Dr. Milton S. Katz
European Science Coordination
Office
U.S. Army Research Institute
Box 65
FPO New York 09510-1500

Dr. Frederick Hayes-Roth
Teknowledge
P.O. Box 10119
1850 Embarcadero Rd.
Palo Alto, CA 94303

Dr. Robert Jassarone
Biologisch Centrum, Vicugel
D Office
701 Welch Road
Kerlan 30, 9751 NN Haren
The NETHERLANDS

Dr. Frank Keil
University of California
Palo Alto, CA 94304

Dr. James D. Hollan
MCC
3500 W. Balcones Ctr. Dr.
Austin, TX 78759

Dr. Claude Janvier
Université du Québec a Montréal
P.O. Box 8888, succ. A
Montréal, Quebec H3C 3P8
CANADA

Dr. Dennis Kibler
University of California
Department of Information
and Computer Science
Irvine, CA 92717

Dr. Keith Holyoak
Department of Psychology
University of California
Los Angeles, CA 90024

Chair, Department of
Psychology
The Johns Hopkins University
Baltimore, MD 21218

Dr. David Kieras
Technical Communication Program
TIDAL Bldg., 2360 Bonisteel Blvd.
University of Michigan
Ann Arbor, MI 48109-2108

Dr. Ed Hutchins
Intelligent Systems Group
Institute for
Cognitive Science (C-015)
UCSD
La Jolla, CA 92030

Dr. J. Peter Kinsaid
Army Research Institute
Orlando Field Unit
c/o PM TRADE-E
Orlando, FL 32813

Dr. L. Julia S. Hough
Lawrence Erlbaum Associates
110 W. Harvey Street
Philadelphia, PA 19144

Dr. John Jost
Thatcher Jones Associates
P.O. Box 6466
10 Trafaigar Court
Lawrenceville, NJ 08648

Dr. Walter Kintsch
Department of Psychology
University of Colorado
Boulder, CO 80309-0245

Dr. Barbara Huts
Department of Psychology
University of Maryland
Collegeville, MD 21228

Dr. Robin Jeffries
Hewlett-Packard Laboratories, 3L
P.O. Box 10490
Palo Alto, CA 94303-0971

Dr. J. Peter Kinsaid
Army Research Institute
Orlando Field Unit
c/o PM TRADE-E
Orlando, FL 32813

Dr. Keith Holyoak
Department of Psychology
University of California
Los Angeles, CA 90024

Dr. Douglas H. Jones
Thatcher Jones Associates
P.O. Box 6466
10 Trafaigar Court
Lawrenceville, NJ 08648

Dr. Marcel Just
Carnegie-Mellon University
Department of Psychology
Schenley Park
Pittsburgh, PA 15213

Dr. David Klahr
Carnegie-Mellon University
Department of Psychology
Schenley Park
Pittsburgh, PA 15213

Dr. Alice M. Isen
Department of Psychology
University of Maryland
Collegeville, MD 21228

Dr. Ruth Kanfer
University of Minnesota
Department of Psychology
Elliott Hall
75 E. River Road
Minneapolis, MN 55455
Distribution List [Illinois/Gentner] NR 667-551

Dr. Kenneth Kotovsky
Community College of Allegheny County
808 Ridge Avenue
Pittsburgh, PA 15212

Dr. Alan M. Lesgold
Learning R&D Center
University of Pittsburgh
Pittsburgh, PA 15260

Dr. William L. Maloy
Code 04
NETPMSA
Pensacola, FL 32509-5000

Dr. David H. Krantz
Department of Psychology
Columbia University
406 Schermershorn Hall
New York, NY 10027

Dr. Jim Levin
Department of Educational Psychology
210 Education Building
1310 South Sixth Street
Champaign, IL 61801

Dr. Elaine Marsh
Naval Center for Applied Research in Artificial Intelligence
Naval Research Laboratory
Code 5510
Washington, DC 20375-5000

Dr. Benjamin Kuipers
University of Pittsburgh
Learning R&D Center
Pittsburgh, PA 15260

Dr. Sandra P. Marshall
Dept. of Psychology
San Diego State University
San Diego, CA 92182

Dr. David R. Lambert
Naval Ocean Systems Center
Code 772
271 Catalina Boulevard
San Diego, CA 92152-5000

Dr. John Levine
Learning R&D Center
University of Pittsburgh
Pittsburgh, PA 15260

Dr. Manton M. Matthews
Department of Computer Science
University of South Carolina
Columbia, SC 29208

Dr. Pat Langley
University of California
Department of Information and Computer Science
Irvine, CA 92717

Dr. Michael Levine
Educational Psychology
210 Education Bldg.
University of Illinois
Champaign, IL 61821

Dr. Richard E. Mayer
Department of Psychology
University of California
Santa Barbara, CA 93105

Dr. Marcy Lansman
University of North Carolina
The L. L. Thurstone Lab.
Davis Hall CB 3370
Chapel Hill, NC 27514

Dr. Clayton Lewis
University of Colorado
Department of Computer Science
Campus Box 430
Boulder, CO 80309

Dr. Joseph C. McLachlan
Code 52
Navy Personnel R&D Center
San Diego, CA 92152-6800

Dr. Jill Larkin
Carnegie-Mellon University
Department of Psychology
Pittsburgh, PA 15213

Dr. James McMichael
Technical Director
Navy Personnel R&D Center
San Diego, CA 92152-6800

Dr. Matt Lewis
Department of Psychology
Carnegie-Mellon University
Pittsburgh, PA 15213

Library
Naval Training Systems Center
Orlando, FL 32813

Dr. Barbara Means
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025

Dr. Jean Lave
Institute for Research on Learning
3333 Coyote Hill Road
Palo Alto, CA 92304

Library
Naval War College
Newport, RI 02840

Dr. Douglas L. Medin
Department of Psychology
University of Illinois
603 E. Daniel Street
Champaign, IL 61820

Dr. Jill Larkin
Institute for Research on Learning
3333 Coyote Hill Road
Palo Alto, CA 92304

Library
Naval Training Systems Center
Orlando, FL 32813

Dr. Robert W. Lawler
Matthews 118
Purdue University
West Lafayette, IN 47907

Science and Technology Division
Library of Congress
Washington, DC 20540

Dr. George A. Miller
Dept. of Psychology
Green Hall
Princeton University
Princeton, NJ 08540

Dr. Jane Malin
Mail Code EF5
NASA Johnson Space Center
Houston, TX 77058
Distribution List [Illinois/Gentner NR 667-551]

Dr. William Montague  
NPRDC Code 13  
San Diego, CA 92152-6800

Director, Manpower and Personnel Laboratory  
NPRDC (Code 06)  
San Diego, CA 92152-6800

Office of Naval Research,  
Code 1142CS  
800 N. Quincy Street  
Arlington, VA 22217-5000  
(6 Copies)

Dr. Randy Mumaw  
Training Research Division  
HumRRO  
1100 S. Washington  
Alexandria, VA 22314

Director, Human Factors  
& Organizational Systems Lab,  
NPRDC (Code 07)  
San Diego, CA 92152-6800

Office of Naval Research,  
Code 1142PS  
800 N. Quincy Street  
Arlington, VA 22217-5000

Dr. Allen Mauro  
Behavioral Technology  
Laboratories – USC  
1845 S. Elena Ave., 4th Floor  
Redondo Beach, CA 90277

Library, NPRDC  
Code P201L  
San Diego, CA 92152-6800

Psychologist  
Office of Naval Research  
Branch Office, London  
Box 39  
FPO New York, NY 09510

Chair, Department of  
Computer Science  
U.S. Naval Academy  
Annapolis, MD 21402

Technical Director  
Navy Personnel R&D Center  
San Diego, CA 92152-6800

Special Assistant for Marine  
Corps Matters,  
ONR Code 00MC  
800 N. Quincy St.  
Arlington, VA 22217-5000

Dr. Allen Newell  
Department of Psychology  
Carnegie-Mellon University  
Schenley Park  
Pittsburgh, PA 15213

Commanding Officer,  
Navy Research Laboratory  
Code 2927  
Washington, DC 20390

Dr. Judith Orasanu  
Basic Research Office  
Army Research Institute  
5001 Eisenhower Avenue  
Alexandria, VA 22333

Dr. Richard E. Niabett  
University of Michigan  
Institute for Social Research  
Room 5201  
Ann Arbor, MI 48109

Dr. Harold F. O'Neil, Jr.  
School of Education – WPH 801  
Department of Educational  
Psychology & Technology  
University of Southern California  
Los Angeles, CA 90089-0031

Dr. James Paulson  
Department of Psychology  
Portland State University  
P.O. Box 751  
Portland, OR 97207

Dr. A. F. Norcio  
Code 5530  
Naval Research Laboratory  
Washington, DC 20375-5000

Military Assistant for Training and  
Personnel Technology,  
OUSD (R & E)  
Room 3D129, The Pentagon  
Washington, DC 20391-3080

Dr. Donald A. Norman  
Code 015  
Institute for Cognitive Science  
University of California  
La Jolla, CA 92033

Office of Naval Research  
Code 1133  
800 North Quincy Street  
Arlington, VA 22217-5000

Dr. David N. Perkins  
Project Zero  
Harvard Graduate School  
of Education  
7 Appian Way  
Cambridge, MA 02138

Deputy Technical Director  
NPRDC Code 01A  
San Diego, CA 92152-6800

Office of Naval Research,  
Code 1142B  
800 N. Quincy Street  
Arlington, VA 22217-5000

Dr. Nancy N. Perry  
Naval Education and Training  
Program Support Activity  
Code 047  
Building 2435  
Pensacola, FL 32509-5000

Director, Training Laboratory,  
NPRDC (Code 06)  
San Diego, CA 92152-6800
Distribution List [Illinois/Gentner] NR 667-661

Department of Computer Science, Naval Postgraduate School
Monterey, CA 93940

Dr. James A. Reggia
University of Maryland
School of Medicine
Department of Neurology
22 South Greene Street
Baltimore, MD 21201

Dr. Janet W. Schoenfeld
816 LRDC Building
University of Pittsburgh
3639 O'Hara Street
Pittsburgh, PA 15260

Dr. Judith W. Segal
OERI
555 New Jersey Ave., NW
Washington, DC 20208

Dr. Steven Pinker
Department of Psychology
E10–018
MIT
Cambridge, MA 02139

Dr. J. Wesley Reggia
AFHRL/IDI
Brooks AFB, TX 78235

Dr. Colleen M. Seifert
Institute for Cognitive Science
Mail Code C-015
University of California, San Diego
La Jolla, CA 92039

Dr. Tjeerd Plomp
Twente University of Technology
Department of Education
P.O. Box 217
7500 AE Enschede
The Netherlands

Dr. Fred Reif
Physics Department
University of California
Berkeley, CA 94720

Dr. Lee S. Shulman
School of Education
507 Ceres Hall
Stanford University
Stanford, CA 94305–3084

Dr. Steven E. Poltrock
MCC
3500 West Balcones Center Dr.
Austin, TX 78759–5800

Dr. Laurence Resnick
Learning R & D Center
University of Pittsburgh
3336 O'Hara Street
Pittsburgh, PA 15213

Dr. Ben Shneiderman
Dept. of Computer Science
University of Maryland
College Park, MD 20742

Dr. Harry E. Pople
University of Pittsburgh
Decision Systems Laboratory
1360 Scaife Hall
Pittsburgh, PA 15261

Dr. Gilbert Ricard
Mail Stop K02–14
Grumman Aircraft Systems
Bethpage, NY 11787

Dr. Robert S. Siegler
Carnegie-Mellon University
Department of Psychology
Schenley Park
Pittsburgh, PA 15213

Dr. Mary C. Potter
Department of Brain and Cognitive Sciences
MIT (E–10–039)
Cambridge, MA 02139

Dr. Linda G. Roberts
Science, Education, and Transportation Program
Office of Technology Assessment
Congress of the United States
Washington, DC 20510

Dr. Robert A. Sorensen
Carnegie-Mellon University
Department of Psychology
Schenley Park
Pittsburgh, PA 15213

Dr. Joseph Psotka
ATTN: PERI-IC
Army Research Institute
5001 Eisenhower Ave.
Alexandria, VA 22333–5600

Dr. William B. Rouse
Search Technology, Inc.
4725 Peachtree Corners Circle
Suite 200
Norcross, GA 30092

Dr. Derek Sleeman
Computing Science Department
King's College
Old Aberdeen AB9 2UB
Scotland
United Kingdom

Dr. Lynn Reder
Department of Psychology
Carnegie-Mellon University
Schenley Park
Pittsburgh, PA 15213

Dr. Roger Schank
Yale University
Computer Science Department
P.O. Box 2158
New Haven, CT 06520

Dr. Richard E. Snow
School of Education
Stanford University
Stanford, CA 94305

Dr. Steve Reder
Northwest Regional Educational Laboratory
400 Lindsay Bldg.
710 S.W. Second Ave.
Portland, OR 97204

Dr. Alan H. Schoenfeld
University of California
Department of Education
Berkeley, CA 94720

Dr. Elliot Soloway
Yale University
Computer Science Department
P.O. Box 2158
New Haven, CT 06520
Distribution List [Illinois/Gentner] NR 667-551

Dr. Richard C. Sorensen
Navy Personnel R&D Center
San Diego, CA 92152-5800

Dr. Kathryn T. Spoehr
Brown University
Department of Psychology
Providence, RI 02912

Dr. Robert J. Sternberg
Department of Psychology
Yale University
Box 11A, Yale Station
New Haven, CT 06520

Dr. Thomas Sticht
Applied Behavioral and
Cognitive Sciences, Inc.
P.O. Box 6640
San Diego, CA 92106

Dr. John Tangney
AFOSR/NL, Bldg. 410
Bolling AFB, DC 20332-4648

Dr. Kikumi Tatsuoka
CERL
252 Engineering Research
Laboratory
103 S. Mathews Avenue
Urbana, IL 61801

Dr. Perry W. Thorndyke
FMC Corporation
Central Engineering Labs
1205 Coleman Avenue, Box 580
Santa Clara, CA 95052

Dr. Martin A. Tolcott
3001 Vassey Terr., N.W.
Apt. 1617
Washington, DC 20008

Dr. Douglas Towne
Behavioral Technology Labs
University of Southern California
1845 S. Eliza Ave.
Redondo Beach, CA 90277

Chair, Department of
Computer Science
Towson State University
Towson, MD 21204

Headquarters, U. S. Marine Corps
Code MPI-20
Washington, DC 20380

Dr. Kurt Van Lehn
Department of Psychology
Carnegie-Mellon University
Schenley Park
Pittsburgh, PA 15213

Dr. Jerry Vogt
Navy Personnel R&D Center
Code 51
San Diego, CA 92152-5800

Dr. Beth Warren
BBN Laboratories, Inc.
10 Moulton Street
Cambridge, MA 02238

Dr. Keith T. Wescourt
FMC Corporation
Central Engineering Labs
1205 Coleman Ave., Box 580
Santa Clara, CA 95052

Dr. Douglas Wetsel
Navy Personnel R&D Center
San Diego, CA 92152-5800

Dr. Barbara White
BBN Laboratories
10 Moulton Street
Cambridge, MA 02238

Dr. Robert A. Wiser
U.S. Army Institute for the
Behavioral and Social Sciences
5001 Eisenhower Avenue
Alexandria, VA 22333-5600

Dr. Martin F. Wiskoff
Defense Manpower Data Center
550 Camino El Estero
Suite 200
Monterey, CA 93943-3231

Mr. Carl York
System Development Foundation
1 Maritime Plaza, #770
San Francisco, CA 94111

Dr. Joseph L. Young
National Science Foundation
Room 320
1800 G Street, N.W.
Washington, DC 20550

Mr. John H. Wolfe
Navy Personnel R&D Center
San Diego, CA 92152-5800

Dr. Wallace Wolfeck, III
Navy Personnel R&D Center
Code 51
San Diego, CA 92152-5800

Dr. Masoud Yasdani
Dept. of Computer Science
University of Exeter
Princes of Wales Road
Exeter EX44PT
ENGLAND