Constructive Adaptive Visual Analogy

A Thesis
Presented to
The Academic Faculty

by

Jim Davies

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

College of Computing
Georgia Institute of Technology
June 2004

Constructive Adaptive Visual Analogy

Approved by:

Professor Ashok K. Goel, Committee Chair

Professor Hari Narayanan (Auburn University)

Professor Nancy J. Nersessian, Adviser

Professor Richard Catrambone

Professor Ronald W. Ferguson

Date Approved: July 30, 2004

ACKNOWLEDGEMENTS

All of the friends I've had since I've worked on this dissertation have given me great support. There are too many to name individually, but I thank you all. However some folks in particular have helped me enormously and in particularly tangible ways: my committee members Ron Ferguson and Hari Narayanan, my parents, James and Janet Davies, for their love and support, Jennifer Rivlin Roberts, for being a great statistician and friend, David Craig for the use of his data, Richard Catrambone, for help with the experiment, my psychology advisor Dorrit O. Billman, and Patrick Yaner. Most of all I would like to thank my advisors and mentors Ashok K. Goel and Nancy J. Nersessian. Through conversations over the years they have presented me with a great deal of their wisdom. I will be forever grateful for the little bit that has stuck.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Visual knowledge appears to be an important part of problem solving, but the role of visual knowledge in *analogical* problem solving is still somewhat mysterious. In this work I present the *Constructive Adaptive Visual Analogy* theory, which claims that visual knowledge is helpful for solving problems analogically and suggests a mechanism for how it might be accomplished.

Through evaluations using an implemented computer program, cognitive models of some of the visual aspects of experimental participants, and a psychological experiment, I support four claims:

First, visual knowledge alone is sufficient for transfer of some problem solving procedures. Second, visual knowledge facilitates transfer even when non-visual knowledge might be available. Third, the successful transfer of strongly-ordered procedures in which new objects are created requires the reasoner to generate intermediate knowledge states and mappings between the intermediate knowledge states of the source and target analogs. And finally, that visual knowledge alone is insufficient for evaluation of the results of transfer.

CHAPTER I

INTRODUCTION

One important way people solve problems is through analogical reasoning. Analogical problem solving is taking a solution from a source analog and applying some version of that solution to a target analog. Problem solving implies that some number of steps need to be carried out to reach a solution. Procedures with the following two properties I will call "strongly-ordered procedures:" 1) two or more steps are involved and 2) some steps cannot be executed before some other steps have already been executed. The first hypothesis of this work is that transfer of strongly-ordered procedures is computationally complex. This is true even when the reasoner already has the correct analogical mapping.

Some domains are rife with visual knowledge (e.g. libraries of computer-aided design files, photograph databases, diagrams, graphs). Scientists have found that analogical problem solving can occur with non-visual knowledge, but can analogical problem solving occur with visual knowledge as well? This work describes *Constructive Adaptive Visual Analogy*, the second hypothesis of which is that visual knowledge alone is sufficient for transfer of some problem solving procedures in some domains.

Some domains are inherently non-visual, but might be visually represented all the same. For example, effectively connecting a battery to some wires might be represented, among other ways, functionally (the battery needs to be physically touching the metal of the wire to conduct electricity) or visually (the image of the wire is adjacent to the image of the battery.) Even though other kinds of knowledge and representations might be used to reason about these domains, human beings appear to experience visual imagery when reasoning about them. Experimental evidence indicates that visual knowledge often plays an important role in human problem solving [68, 21, 7, 53]. There is also documentary evidence for visual reasoning in scientific problem solving (e.g. [55, 56]). Further, psychological evidence suggests that *analogical* problem solving is facilitated by animations [63], diagrams [4] as

well as visually evocative phrases in stimuli [33]. Why might this be? The third hypothesis of this work is that visual knowledge facilitates transfer even when non-visual knowledge might be available.

Following is a summary of my hypotheses:

1. **Transfer of strongly-ordered procedures is computationally complex, even given the correct mapping.**

2. **Visual knowledge alone is sufficient for transfer of problem solving procedures in some domains.**

3. **Visual knowledge facilitates transfer even when non-visual knowledge might be available.**

I use the term "visual knowledge" to mean any representation that encodes *visual information*. Visual *information* consists of the visual properties of something (i.e. the information that humans can extract by inspection from an image or the world by directing visual attention to it [9]), and visual *knowledge* is visual information encoded for use by a reasoner with some representation language. Specifically, this dissertation deals with the following kinds of visual information: shapes, their sizes, locations, motions, and spatial relationships between shapes (e.g. connections, overlaps).

The level of visual abstraction is a core issue in visual and diagrammatic reasoning. This work will use symbolic *descriptive* representations, which are structured descriptions of visual information. This is differentiated from *depictive* representations, or bitmaps. A depictive representation "specifies the locations and values of points in space" [47]. There is widespread agreement that visual reasoning, particularly in problem solving and analogy, is a symbolic process. Not surprisingly, all previous computational visual analogy programs also use symbols to represent visual information. The distinction is not simply between depictive and descriptive, however. There is a spectrum of complexity from dots to complex aggregations of shapes, on which every visual representation language must place itself. For this work I have chosen a level of abstraction higher than bits, but primitive enough such that the same primitives (both shapes and operations) could be re-used in multiple

examples. This acknowledges a trade-off: higher level symbols contain more information but lose similarity and transfer with other represented examples. A core assumption of this work is that this is a productive level of representation for transfer of problem-solving procedures. This level is similar to that used in other visual reasoning theories. [70, 52, 24, 11, 36]

In this chapter I will review the three main sources of supporting evidence for these hypotheses. First I will describe `Galatea`, the computer implementation that successfully transfers strongly-ordered procedures using only visual knowledge.

Next I will discuss four models, created with Galatea, of visual aspects of transfer as displayed by participants in a psychological experiment run by Dr. David Craig.

Finally I will describe a psychological experiment.

1.1 Computer Implementation: Galatea

Galatea is a computer implementation that successfully transfers strongly-ordered procedures using only visual knowledge. It provides an existence proof for the hypothesis that visual knowledge alone is sufficient for transfer of problem solving procedures in some domains, and that visual knowledge alone facilitates transfer even when non-visual knowledge might be available.

Galatea succesfully works with four problems. 1) four versions of the lab/weed-trimmer problem (described in the next section) 2) the fortress/tumor problem (described in this section), 3) the cake/pizza problem and 4) the Maxwell example (described in later chapters). I will describe the architecture of Galatea now, along with the fortress/tumor example. In later chapters I will describe the other models.

I will use Gick and Holyoak's fortress/tumor problem [33, 15] as a running example throughout this section. In this example experimental participants read a story about a general who must overthrow a dictator in a fortress. His army is poised to attack along one of many roads leading to the fortress when the general finds that the roads are mined such that large groups passing will set them off. To solve the problem, the general breaks the army into smaller groups, which take different roads simultaneously, arriving together at the fortress. Participants are then given a tumor problem, in which a tumor must be

destroyed with a ray of radiation, but the ray will destroy healthy tissue on the way in, killing the patient. The analogous solution is to have several weaker rays simultaneously converging on the tumor [33, 15].

A procedure for solving a problem can be represented as a series of knowledge states and `transformations` between them. A knowledge state characterizes the steps in the procedure by specifying information about the elements in the state and relationships between them. A transformation takes in a knowledge state, changes its configuration in some way, and produces the next knowledge state in the sequence. Two successive knowledge states are connected by a single transformation. The first knowledge state represents the initial description of the problem. The final knowledge state represents the state in which the problem is solved.

In the first knowledge state of the fortress/tumor problem, the large army takes a single road to the fortress.Starting from the first knowledge state in the fortress story, the first transformation is to break the army up into smaller groups. This leads to the second knowledge state containing those smaller armies. The second transformation is to move the armies to distinct roads, and so on.

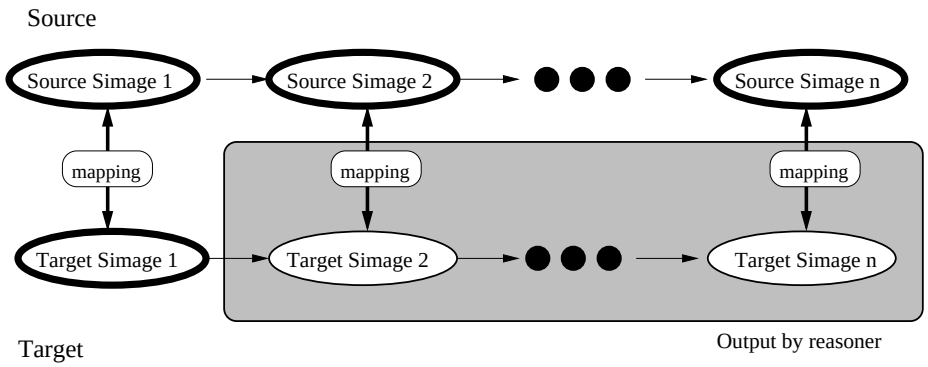

Figure 1: This Figure illustrates Galatea's input and output in the abstract. The knowledge states (s-images) in the source case are depicted as ovals along the top of the Figure. The knowledge states are visually represented as `s-images`. Transformations between the states in the Figure are depicted as arrows. The target problem is depicted as the leftmost bottom oval. All things in the gray box are output by Galatea.

Since the knowledge states for this model contain only visual knowledge represented symbolically, I call the states symbolic images or `s-images`. Figure 1 illustrates Galatea's

Table 1: Covlan's transformations.

Transformations	
Transformation name	**arguments**
add-element	object-type, location (optional)
add-connections	connection/connection-set
decompose	object, number-of-resultants, type
move-to-location	object, new-location
move-to-set	object, object2
put-between	object, object2, object3
replicate	object, number-of-resultants

input and output in the abstract. Galatea takes as input a source analog, an initial target problem `s-image`, and an analogical `mapping` between the initial `s-images` of the source and target. The source is a complete sequence of `s-images` and transformations representing the procedure that solves the source problem. Galatea transfers the visual `transformations` one at a time from the source to the target, creating new target `s-images` along the way, with new analogical `mappings` between the corresponding target and source `s-images`.

1.1.1 Knowledge and Representation

It is important that the analogs are represented with a consistent symbolic visual representation language. This fact is more important than the actual ontology of the language used. *Covlan* (Cognitive Visual Language)[12] provides an ontology of visual primitives. Table 1 shows Covlan's ontology of transformations.

`Add-element` adds a new primitive element in the next s-image. The first argument, `object-type`, must be one of the members of the `primitive elements` (e.g. `square` or `circle`, described below). It determines what kind of shape appears in the next s-image. The second argument is `location`, which must be one of Covlan's `locations`: `bottom`, `top`, `right`, `left`, or `center`. What this means is that the next `s-image` will have three relationships added. All of Galatea's memory is in propositional form. A relationship connects two ideas with a relation. See Figure 2. 1) The `s-image` connected with a

`has-component` relation to the name identifying the new component, 2) the new component's name with a `looks-like` relation to the `object-type`, and 3) the component's name with a `has-location` relation to the `location` input as an argument.

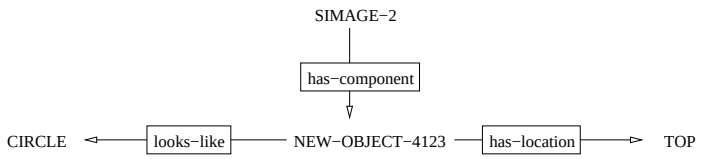

Figure 2: A graphical representation of the three relationships added by the `add-component` transformation. `Relations` are boxed. Objects at the beginning of arrows are the in the `ThingX` slot; the objects at the end of the arrows are in the `ThingY` slot.

`Add-connections` is a transformation that inserts a set of connections into the next `s-image`. The input is the name of the set of connections in the source. To determine the nature of the connections in the target, Galatea uses substitution for all the symbols to find the analogous names, so that analogous connections are placed in the next target `s-image`.

`Decompose` takes a primitive element and replaces it in the next `s-image` with some `n` number of elements. It also reduces the thickness for each of those elements.

`Move-to-location` changes the location of a primitive element from one location to another. This means that in the next `s-image`, the old `has-location` relation is removed and a new `has-location` relation is added, relating the element to the input `location`, which can be an absolute location or another element.

`Move-to-set` takes in two sets as input (we will call them *set-a* and *set-b*). The members of *set-a* are moved to the locations of the members of *set-b*. In the tumor example, the `decomposed` rays are placed on the locations of the distinct body-areas. If *set-a* and *set-b* have the same number of element instances, then each element of *set-a* is placed on a distinct element in *set-b*. The element instance matching is arbitrary.

If *set-a* has more elements, then multiple members of *set-a* are placed at the locations of each member of *set-b*. The number of element instances in these groups is determined by the number of elements in *set-b* divided by the number of elements in *set- a*.

If *set-b* has more elements, then elements of *set-a* are distributed evenly across the

Table 2: Covlan's primitive elements.

Primitive Element name	**attributes**
connection	subject, object, angle, distance
rectangle	location, size, height, width, orientation
circle	location, size, height
line	location, length, end-point1, end-point2, thickness
set	location, orientation, front, middle

locations of the members of *set-b*.

`Put-between` takes two objects that are assumed to be touching, and places some third object in between them. In the new `s-image` 1) the two objects are no longer touching and 2) the third is touching both of them.

`Replicate` takes in an element or set of elements and generates `n` new instances of that element or elements in the next `s-image`. Its behavior is similar to `decompose`, except that it does not change the size or thickness of elements, and can work on sets as well as single element instances.

Covlan's ontology of `primitive visual elements` (Table 5) contains: `rectangle, circle, line,` and `set`. The elements are frame-like structures with slots that can hold values. For example, a `rectangle` has a `location, size, height, width,` and `orientation`. All elements have a `location`, which holds a value representing an absolute location on an `s-image` (e.g. `top`, `right`).

See Figure 3 for an example of how instances of these elements can be arranged in an `s-image`.

The `set` is a special `element`. A `set` can contain any number of `instances` of `elements`. `sets` also have an `orientation,` the value of which is one of the primitive `directions.` An `element instance` in the `set` is specified in the representation as the `front`, and another as the `middle`. The orientation is defined as an imaginary line from the `middle` to the `front` in the `direction` specified in the `orientation`.

Sometimes a *part* of an `element instance` must be referenced. For example, if a line touches the middle of another line, there must be some way to describe the *end* of the first

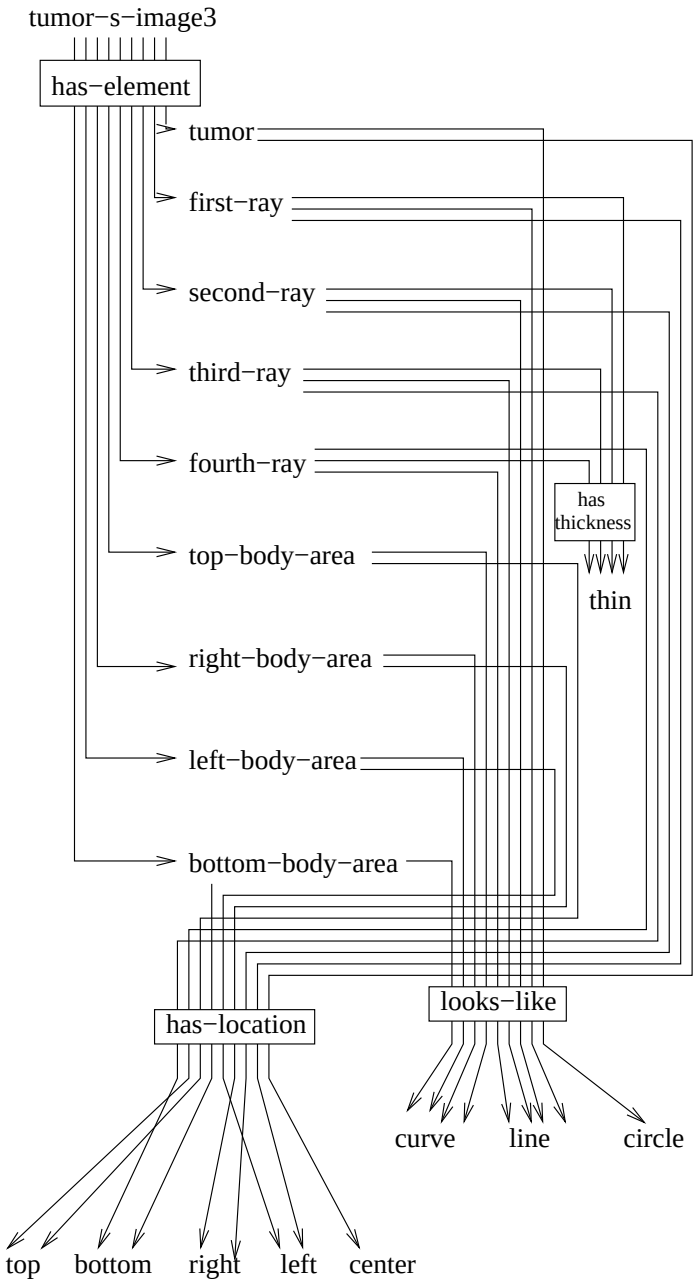

Figure 3: This Figure shows part of the third generated `s-image` in the tumor procedure. Each `relationship` is represented as an arrow. The start and ends of the arrows are the ideas connected by the relation in the proposition. At the beginning of the arrow is the `ThingX` of the relationship, and at the end of the arrow is the `ThingY`. The boxed text in the middle of the arrow is the `Relation.` Each string of unboxed text is a `concept`.

Table 3: Classifications of Miscellaneous Slot Values

angles	perpendicular-angle, right-angle-cw, 45-angle-cw, 45-angle-ccw, right-angle-ccw
locations	bottom, top, right, center, off-bottom off-top, off-right, off-left
sizes	small, medium, large
thicknesses	thin, thick, very-thick
speeds	slow, medium, fast
directions	left, right, up, down
lengths	short, medium, long

line and the *middle* of the next. In Covlan different primitive elements have different kinds of `areas`.

`Lines` have `start` and `end points`, as well as `right` and `left-side mid-points`. The element instance's names are related to the symbols naming these areas (e.g. `line1-end-point` with `area-relations`: `has-end-point`, `has-start-point`, `has-rightsidemiddle`, and `has-leftsidemiddle`.

`Circles`, `squares`, and `rectangles` have `sides`, which are related to element instances with the following relations: `has-side1` (the top), `has-side2` (the right side), `has-side3` (the bottom), and `has-side4` (the left side).

These are symbols that can give a value to element attributes or transformation arguments. They can be broken down into the following types: `angles`, `locations`[1], `sizes`, `thicknesses, numbers`, `speeds`, `directions`, and `lengths.`

The class of `primitive visual relations` (shown in Table 4) describe how certain visual elements relate to each other and `miscellaneous slot values. Motion relations` (see Table 4) describe how element instances are moving in an `s-image`. `Rotation` has the arguments `speed` and `direction`.

[1]Relative locations, as opposed to absolute locations, are classified under `primitive visual relations.`

Table 4: Primitive visual relations.

Visual Relations	touching, above-below, right-of-left-of, in-front-of-behind, off-s-image
Motion Relations	rotating, not-rotating

Many spatial relationships between primitive elements are represented with `connections`. A `connection` is a primitive element with a `name`. `Connections` are frames with two four slots: `subject`, `object`, `angle` and `distance`, represented with `is-subject-for-connection`, `is-object-for-connection`, `has-angle` and `has-distance`. These relations connect the `connection` name to `distances` and `angles`, which are qualitative `miscellaneous slot values`. See Figure 4. The `object` of the `connection` is `distance` away from the `subject` in the direction of `angle.`

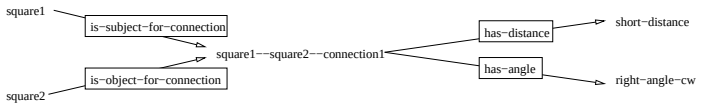

Figure 4: A representation of the relationships involved with a connection. `Square2` is a short distance to the right of `square1`. `Right-angle-cw` means that the angle is a right angle in the clock-wise direction.

The `distances` are `touching-distance`, `short-distance` and `long-distance`. The `angles` are `perpendicular-angle` (straight ahead), `right-angle-cw` (a right angle in the clockwise direction, or to the right), `45-angle-cw` (a forty-five degree angle to the right), `45-angle-ccw` (a forty-five degree angle in the counter-clockwise direction, or to the left), and `right-angle-ccw` (a right angle to the left). Figure 5 shows the different kinds of `connections` Covlan can represent. `Areas` of element instances, as well as element instances themselves, can be connected.

`S-images` can have `analogies` between them. Each analogy can have any number of analogical `mappings` associated with it (determining which `mapping` is the best is known as the *mapping problem*.) Each alignment between two `element instances` or `areas` in a given `mapping` is called a `map`.[2]

[2] A `map` is called a `match hypothesis` in the SME literature.[19]

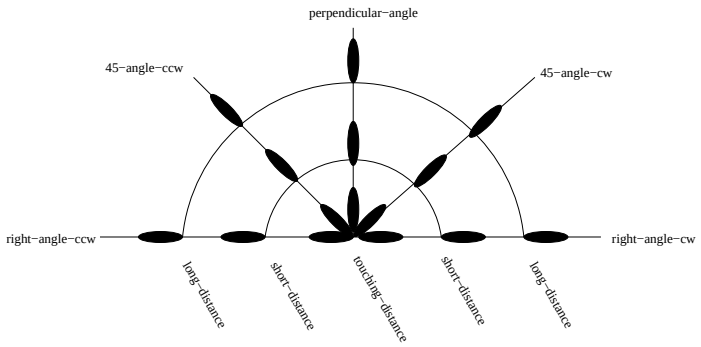

Figure 5: Each of the fifteen black dots in the Figure represents a qualitative `connection` area, with an `angle` and `direction`.

Similarly `s-images` next to each other in sequences have `transform-connections`. These are necessary so the reasoner can track how visual elements in a previous `s-image` change in the next. A difference between `analogies` and `transform-connections` are that there can be multiple analogical mappings for an `analogy`, but only one `mapping` for a `transform-connection`.

`Transformations` are attached, in fact, to a `map` between two elements in sequential `s-images`. So if a `rectangle` changes into a `circle`, the agent knows which `rectangle` in the previous `s-image` turns into which `circle` in the next `s-image`.

1.1.2 Inference and Processing

Analogy consists of several steps: *retrieval* is identifying a candidate source analog in memory; *mapping* is finding the best set of correspondences between components of the analogs; *transfer* is the application of knowledge from the source analog to the target analog, which might use various forms of *adaptation*; *evaluation* is determining if the target problem has been solved appropriately; *storage* is storing the target analog in memory for potential reuse. Galatea focuses on the transfer and adaptation stage of analogy. In particular, it adapts and transfers each `transformation` in the source problem to the target. The `transformations` are transferred literally and the arguments of those `transformations` can be adapted.

For example, the `transformation decompose` is used to turn a `primitive element` instance into some arbitrary number of resultants, taken as an argument. An argument of

a `transformation` can be an instance of one of three cases. First, the argument can be a literal, like the number `four` or the location `bottom`. Literals are transferred unchanged to the target.

Second, the argument could be a `element` instance member of the source `s-image`. In this case, the transfer procedure operates on the analogous `element` in the target `s-image`. For example, in the first `transformation` in the fortress story, the decomposed source `soldier path` gets adapted to the `ray` in the target tumor problem.

In the third case, the argument can be a function. Since this case does not occur in the fortress/tumor problem, we will use another example to describe it. Let us suppose that a reasoner needs to feed six people with one Sicilian slice sheet pizza. An analog in memory of cutting a sheet cake for four people is used to generate a solution. Transfer is still difficult because somehow the `4` in the cake analog must be adapted to the number `6` in the source analog. Knowing how many pieces into which to cut the cake or pizza depends on the number of people each problem. Some notion of count is needed. The use of `functions` as arguments to `transformations` addresses this problem. The cake analog is represented with a `function` that counts the number of people as its argument for the `decompose transformation`. This function has an argument of its own, namely the set of cake eaters, which during adaptation adapts into the set of pizza eaters. When the `transformation` is applied to the pizza, it counts the members of the set of people in the pizza problem (which results in six). `Decompose` produces six pieces of pizza in the next `s-image`.

1.1.3 Algorithm

I will describe Galatea's main algorithm informally in this subsection. In a later chapter I will describe it more formally. Throughout this subsection the reader should refer to Figure 6.

1. **Identify the first s-images of the target and source cases.** These are the current source and target s-images.

2. **Identify the `transformations` and their associated arguments in the current `s-image` of the source case.** This step finds out how the source case gets from its

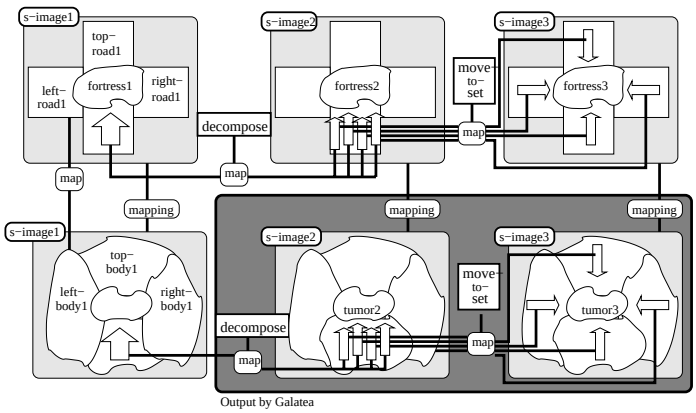

Figure 6: This Figure shows Galatea's input and output for the fortress/tumor problem. The top series of `s-images` in the Figure shows the visual representation of the solved fortress problem. The bottom series shows the target tumor problem. The bottom left `s-image` is the initial state of the tumor problem. The shaded box shows the output of the system.

current `s-image` to the next `s-image`. In the fortress/tumor example, the transformation is `decompose`, with `four` as the `number-of-resultants` argument (not shown).

3. **Identify the objects of the transformations.** The object of the transformation is what object, if any, the transformation acts upon. For the `decompose` transformation, the object is the `soldier-path1` (the thick arrow in the top left `s-image` in Figure 6.)

4. **Identify the corresponding objects in the target problem.** `Ray1` (the thick arrow in the bottom left `s-image`) is the corresponding component of the source case's `soldier-path1`, as specified by the `mapping` between the current source and target `s-images` (not shown). A single object can be mapped to any number of other objects. If the object in question is mapped to more than one other object in the target, then the same transformation is applied to all of them in the next step.

5. **Apply the transformation with the arguments to the target problem component.** A new `s-image` is generated for the target problem (bottom middle) to record the effects of the `transformation`. The `decompose transformation` is applied to the `ray1`, with the argument `four`. The result can be seen in the bottom

middle `s-image` in Figure 6. The new rays are created for this `s-image`. Adaptation of the arguments can happen in three ways, as described above: If the argument is an element of the source s-image, then its analog is found. If the argument is a function, then the function is run (note that the function itself may have arguments which follow the same adaptation rules as transformation arguments). Otherwise the arguments are transferred literally.

6. **Map the original objects in the target to the new objects in the target.** A `transform-connection` and `mapping` are created between the target problem `s-image` and the new `s-image` (not shown). `Maps` are created between the corresponding objects. In this example it would mean a `map` between `ray1` in the left bottom `s-image` and the four rays in the second bottom `s-image`. A `map` is also created between the `ray1` to the set of thinner rays. Galatea does not solve the `mapping` problem, but a `mapping` from the correspondences of the first `s-image` enables Galatea to automatically generate the `mappings` for the subsequent `s-images`.

7. **Map the new objects of the target case to the corresponding objects in the source case.** Here the rays of the second target `s-image` are mapped to soldier paths in the second source `s-image`. This step is necessary for the later iterations (i.e. going on to another`transformation` and `s-image`). Otherwise the reasoner would have no way of knowing on which parts of the target `s-image` the later transformations would operate.

8. **Check to see if there are any more source s-images.** If there are not, exit, and the solution is transferred. If there are further `s-images` in the source case, set the current `s-image` equal to the next `s-image` and go to step 1.

1.1.4 The Fortress/Tumor Problem

I chose the fortress/tumor example because some experimental participants have used visual inferences in solving it [43].

Table 5 shows some of the visual elements and their attribute values for the first fortress

Table 5: Primitive elements from fortress problem `s-image` 1.

Visual Object	**attributes**	**value**
Fortress	looks-like: location:	curve center
Bottom-road	looks-like:	line
Right-road	looks-like:	line
Left-road	looks-like:	line
Top-road	looks-like:	line
Soldier-path	looks-like: location: thickness:	line bottom-road thick

problem `s-image`.

I represented the fortress story with three `s-images` (see Figure 6.) The first was a representation of the original fortress problem. It had four roads, represented as thick lines, radiating out from the fortress, which was a `curve` in the `center` (`curves` are used to represent irregular shapes). I represented the original soldier path as a thick line on the bottom road. This `s-image` was connected to the second with a `decompose transformation`, where the arguments were `soldier-path1` for the `object` and `four` for the `number-of-resultants`. The second `s-image` shows the `soldier-path1` decomposed into four thin `lines`, all still on the bottom road. The `lines` are thinner to represent smaller groups.

I represented the start state of the tumor problem as a single `s-image`. The tumor itself is represented as a `curve`. The `ray` of radiation is a thick `line` that passes through the bottom body part.

In the fortress/tumor example, after the `decompose` transformation generates a number of smaller armies (by transforming a thick arrow into thinner arrows), those armies must be dispersed to the various roads, in various locations in the image. In a previous version of this model [12, 13] each army arrow was `moved-to-location` individually to each road line. This solution was brittle because the number of roads to which the armies moved needed

to match exactly the number of body areas the weaker rays moved to in the target.

The model now uses `sets` to address this problem. By grouping the armies, roads, rays, and body parts into their own `sets`, Galatea adapts the solution in the source analog to accommodate differing numbers of any of these elements. Rather than using the `move-to-locationtransformation` on each army, it uses `move-to-set` to the change the location of the `set` of armies. The argument to this function is a `set` of roads. The `move-to-set` function takes one set and distributes its members around the locations of another set.

I have described in some detail the how the fortress/tumor example was implemented in Galatea. This example shows the system's robustness with respect to transfer when different set sizes come into play. I will go into detail about the cake/pizza example (described in the description of functional arguments above), as well as a model of James Clerk Maxwell's analogical reasoning associated with his work on the electromagnetic field theory in a later chapter. In the next section I will describe the cognitive modelling of some psychological data.

I will re-iterate the hypotheses of this work and describe how Galatea relates to them. Hypotheses two and three are that visual knowledge alone is sufficient for transfer of some problem solving procedures, and that visual knowledge facilitates transfer even when non-visual knowledge might be available.

Galatea, implemented with four examples, shows that non-trivial problem-solving procedures can be represented visually and transferred successfully across domains. The cake/pizza example shows transfer for an inherently visual domain, and the fortress/tumor example shows cross-domain analogy where non-visual knowledge might be available to a human reasoner.

The first hypothesis is that transfer of strongly-ordered procedures is computationally complex, even given the correct mapping. I discovered that the successful transfer of strongly-ordered procedures *in which new objects are created* requires the reasoner to generate intermediate knowledge states and mappings between the intermediate knowledge

states of the source and target analogs. Galatea shows why, in detail, this is so. Components of the problem are *created* by the operations, and these components are acted on by later operations. In the tumor problem, for example, the strong ray must be turned into weaker rays before they can be moved. When the reasoner transfers the second operation of moving the soldier paths, how does it know that the corresponding objects in the target are the weaker rays? It must have some mapping to make this inference. And since the weaker rays do not exist in the start state of the tumor problem, this `mapping` cannot be given as input with the initial mapping. The new knowledge state with the weaker rays must be generated, and then a `mapping` must be made on the fly between it and the second knowledge state of the source.

1.2 Cognitive Modelling

For the next source of evidence I modelled some of the visual aspects of four experimental participants' drawings.

Dr. David Craig ran 34 participants in an analogical transfer experiment. Participants were shown a problem-solving solution with a laboratory, presented with text and a diagram. They were asked to solve an analogous problem with a weed-trimmer, presented with text only. Of these, 17 participants (in three conditions) correctly described the analogous solution. All participants were asked to draw a diagram to illustrate their suggested solutions.

A laboratory cleanroom strategy is transferred to adding redundant doors to a weed-trimmer arm so that it can pass through street signs (See Figure 28.) The analogous solution is to design an arm with two latching doors, so that while one is open to let the sign pass, the other stays closed to support the arm and trimmer. Participants produced diagrams describing their solutions to the problems. I modelled four of these experimental participants in Galatea: L14, L15, L16, and L22. In this section I will describe L14 and L22 in detail.

1.2.1 The Galatea Model of L14

L14 received Condition 1 of the lab problem (see Figure 7). Figure 9 shows what L14 wrote on his or her data sheet during the experiment.

I represented the source analog as a series of `s-images` connected with `transformations`. See the top of Figure 10 for an abstract diagram of this analog, and see Figure 8 for a diagram of some of the propositions in its first `s-image`.

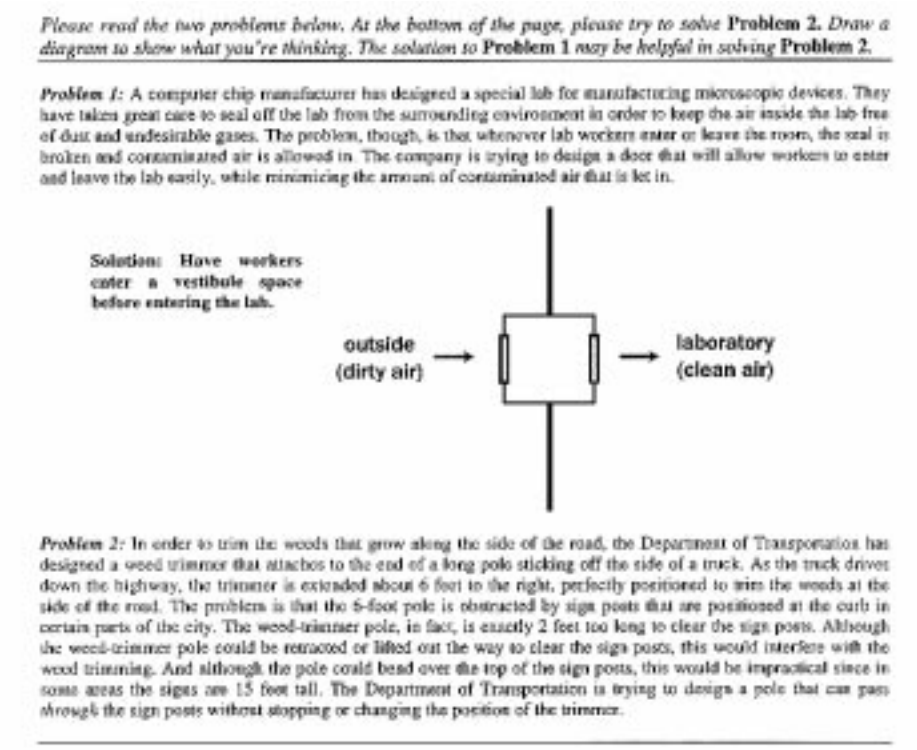

Figure 7: Condition 1: Plan view of lab, with the vestibule centered.

The model of L14 involves five `transformations` (See Figure 10). The first `transformation` is `replicate`. It takes in the `door-set-l14s1` as an argument, generating `door-set1-l14s2` and `door-set2-l14s2` in the next `s-image`.

The second `transformation` is `add-connections` which places the door sets in the correct position in relation to the top and bottom walls.

The third and fourth `transformations` are `add-component`, which add the top and bottom containment walls.

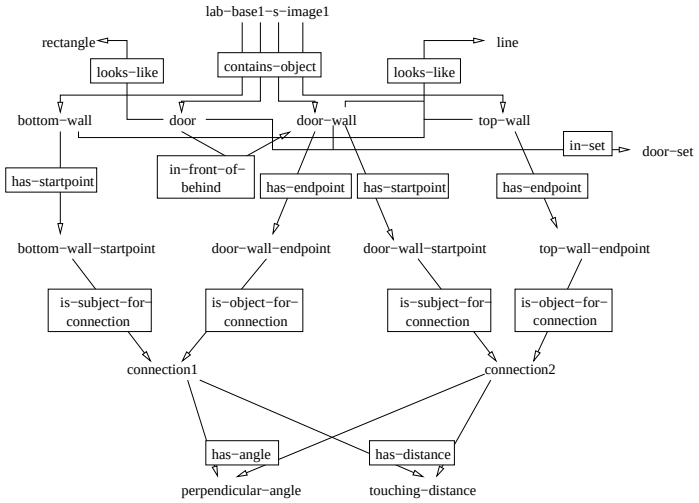

Figure 8: This Figure shows part of the first `s-image` in L14's source `s-image` series. Each `relationship` is represented as an arrow. At the beginning of the arrow is the `ThingX` of the relationship, and at the end of the arrow is the `ThingY`. The boxed text in the middle of the arrow is the `Relation.` Each string of unboxed text is a `concept`.

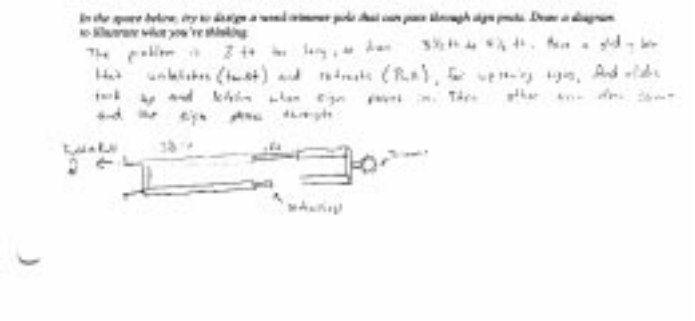

Figure 9: The source data for L14. The drawing above and handwritten text are what participant L14 produced on the experiment sheet.

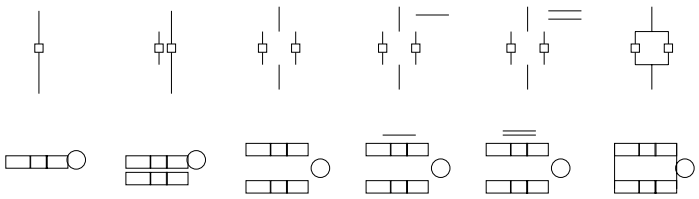

Figure 10: The implementation of L14. The top series of `s-images` represents the source analog (the lab problem) and the bottom series the target. There are six `s-images` for the five `transformations`.

The fifth `transformation`, another `add-connections`, places these containment walls in the correct positions in relation to the door sets and the top and bottom walls.

I will describe the first two `transformations` in detail. The first `transformation` in the lab-base1 source is a `replicate`, which takes two arguments: some `object` and some `number-of-resultants`. In this case the `object` is `door-set-b1s1` (represented as `door-set` in Figure 8. `b1s1` means "base one, `s-image` one.") and the `number-of-arguments` is `two`. The `replicate` is applied to the first L14 `s-image`, with the appropriate adaptation to the arguments: The `mapping` between the first source and target `s-images` indicates that the `door-set-b1s1` maps to the `door-set-l14s1`, so the former is used for the target's `object` argument. The number `two` is a literal, so it is transferred directly.

Using a function that takes in the name of an element instance or set (in this case `door-set-l14s1`) and recursively returns all set names and element instances, Galatea retrieves (from memory of the source `s-image` with the `replications` in it) all propositions with any of those set names and element instances in the `thingX` or `thingY` slots. These propositions are put through a function that creates the same number of new propositions with the same `relations` and `literals`, but with new names for the element instances. These new propositions are stored in memory. The effect of this is a replication of the intended structure. This occurs once for each replication.

Galatea chooses an arbitrary name for the superset of door-sets (in this case `door-sets-set-l14s2`) and connects `door-set1-l14s2` and `door-set2-l14s2` to it with `in-set` relations. It makes a map between L14's `s-image1` and `s-image2`, connecting `door-set-l14s1` to `door-sets-set-l14s2`. It also creates maps from `door-set-l14s1` to `door-set1-l14s2` and another to `door-set2-l14s2`.

The other propositions from L14's `s-image1` are put through a function that finds analagous propositions: `literals` and `relations` are kept the same, and element instance names are replaced with new names for the new `s-image`. For example, the `top-door-l14s1` becomes `top-door-l14s2`.

Maps between the element instances in the target `s-image1` and the target `s-image2` are stored in memory as well.

Galatea automatically generates the `mapping` between `lab-base1-simage2` and `l14-simage2`. Element instances that are results of source `transformations` are mapped to newly-generated instances in the target. All other maps are carried over to the new `s-images` with their new names.

The second `transformation` is `add-connections`. The effect of this transformation is to place the replicated door-sets in the correct spatial relationships with the other element instances. It takes `connection-sets-set-b1s3` as the `connection/connection-set` argument. This is a `set` containing four `connections`. Galatea uses a function to recursively retrieve all `connection` and set proposition members of this `set`. These propositions are put through a function which creates new propositions for the target. Each proposition's `relation` and `literals` are kept the same. The element instance names are changed to newly generated analogous names. For example, `door1-endpoint-b1s3` turns into `door1-endpoint-l14s3`.

Then, similarly to the `replicate` function, horizontal target maps are generated, and the other propositions from the previous `s-image` are instantiated in the new `s-image`.

We can now examine what made L14 (Figure 9) differ from the stimulus drawing: L14 features a longer vestibule in the drawing than the vestibule pictured in the stimulus. In fact, there is no trimmer arm (analogous to the wall in the lab problem) in the drawing at all that is distinct from the vestibule, save a very small section, apparently to keep the spinning trimmer blade from hitting the vestibule. The entire drawing is rotated ninety degrees from the source. The single lines in the source are changed to double lines in the target. The doors also slide in and out of the vestibule walls. What's interesting about this modification is that it does not appear that this kind of door opening is possible with the diagram given of the lab in the source: Since the door is a rectangle that is thicker than the lines representing the walls, the door could not fit into the walls. In contrast L14 explicitly makes the doors and walls thick (with two lines) and makes the doors somewhat thinner. L14 adds objects to the target not found in the source: a blade and a twisting mechanism to describe how the doors can work. L14 also included numerical parameters to describe the design of the trimmer, to describe length. Finally, L14 includes some mechanistic description of how the

trimmer would work.

In summary, these behaviors are:

1. long vestibule
2. rotation
3. line to double line
4. sliding doors
5. added objects
6. numeric dimensions added
7. mechanisms added

Of these seven differences, Galatea successfully models four of them. The *rotation* of the source is modelled by a rotation in the target start `s-image`. In this `s-image`, all spatial relationships are defined only relative to other element instances in the `s-image`. Each instance is a part of a single set which has an orientation and direction. In the case of `s-image` 1 of the target, it is facing right. Since all locations are relative, there is no problem with transfer and each `s-image` in the model of L14 is rotated to the right.

The *line to double line* difference is accounted for by representing the vestibule walls with rectangles rather than with lines, as it is in the source. Because the `mapping` between the source and target correctly maps the `side1` of the rectangle to the `startpoint` of its analogous line, the rectangle/line difference does not adversely affect processing and transfer works smoothly.

The *long vestibule* difference is accounted for by specifying that the heights of the vestibule wall rectangles are `long`. In the source the vestibule wall lines are of length `medium`, but this does not interfere with transfer.

The blade *added object* is accounted for by adding a circle to the first `s-image` in the target.

Unaccounted for are the two bent lines emerging from the vestibule on the left side, the numeric dimensions and words describing the mechanism. Also, L14 shows one of the doors retracting, and the model does not. The model also fails to capture the double line used to connect the door sections, because the single line is transferred without adaptation from the source. This could be fixed, perhaps, by representing the argument to the `add-component` as a function referring to whatever element is used to represent another wall, rather than as a `line`.

1.2.2 The Galatea Model of L22

L22 received Condition 2 (see Figure 11.) Figure 13 shows what L22 wrote on his or her data sheet during the experiment.

I represented the source analog as a series of `s-images` connected with `transformations`. See the top of Figure 12 for an abstract diagram of this analog.

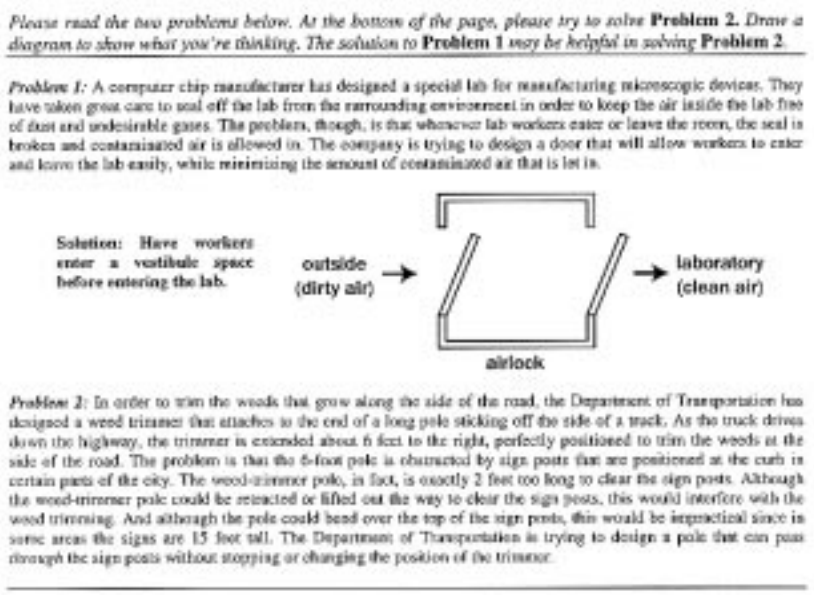

Figure 11: Condition 2: Plan view of lab, with no walls.

The model of L22 involves five `transformations` (See Figure 33). The first `transformation` is `replicate`. It takes in the `door-set-l22s1` as an argument, generating `door-set1-l22s2` and `door-set2-l22s2` in the next `s-image`. Note that the door set replicated here is different from the door set replicated for L14. In this case, there are three connected rectangles,

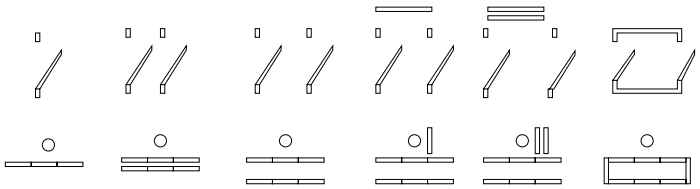

Figure 12: The implementation of L22. The top series of `s-images` represents the source analog (the lab problem) and the bottom series the target. There are six `s-images` for the five transformations.

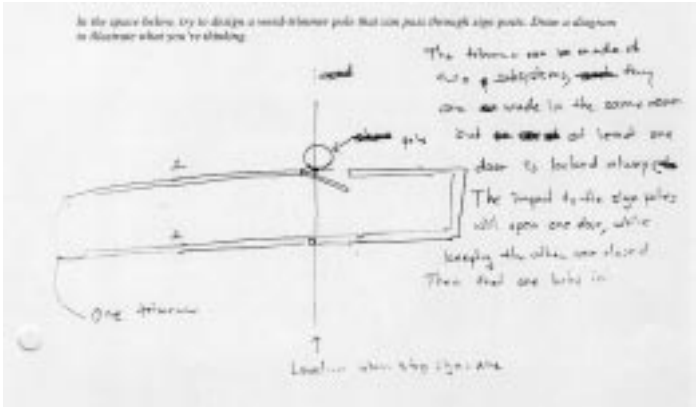

Figure 13: The source data for L22. The drawing above and handwritten text are what participant L22 inscribed on the experimental sheet.

corresponding to the top wall, door, and bottom wall. In the case of L14, the door set is made of a single long rectangle (representing the wall) with another rectangle (representing the door) in front of it. But because `replicate` can work on any set of element instances, Galatea can accomodate the kind of doorway L22 had in mind.

The second `transformation` is `add-connections` which places the door sets in the correct position in relation to each other. Unlike for L14, there are no top and bottom walls.

The third and fourth `transformations` are `add-component`, which add the top and bottom containment walls.

The fifth transformation, another `add-connections`, places these containment walls in the correct positions in relation to the door sets.

The processing and adaptation of these `transformations` resembles the processing done with L14.

We can now examine what made L22 (Figure 13) differ from the stimulus drawing: The entire drawing is rotated ninety degrees from the source. An object is added to the target that has no analog in the source: the trimmer. L22 features a proportionately longer vestibule than in the source, and has some explicit simulation diagrammed. Of these differences, all but the last were modelled by changing the nature of the start `s-image` for L22.

L22 shows that Galatea's models of these participants works with different source as well as target analogs. The modelling of L15 and L16 were modelled similarly. For *all* of these models, no core processing code was changed. Only `transformations` were added to code. All differences I was able to accomodate I did by changing the input representation, rather than the code itself.

1.2.3 What the Implementation Shows

Above I described models of some of the visual aspects of four experimental participants. Specifically, I have modelled the visual input and output for this participant data–a good

start to a full cognitive model. Though people likely use non-visual as well as visual knowledge in analogical problem solving, this work shows how visual knowledge alone *could* be used. The implementation speaks to my third hypothesis, that visual knowledge facilitates transfer even when non-visual knowledge might be available.

L14, L15, L16, and L22 are representative of some of the more difficult experimental participants that I could have modelled. They were given a source analog diagram and participants produced drawings describing their solutions. My models of them show how the analogical transfer could be done using only visual knowledge. The drawings produced by the participants differed from the stimulus diagrams in many ways, and in all four cases my models accounted for most of these differences, supporting the hypothesis that visual knowledge enables transfer.

The previous section showed how the implementation of Galatea addressed the three hypotheses of this work. Modelling these four participants shows that hypotheses are supported in the same way for the modelling of human cognition as well.

1.3 Psychologial Experimentation

As Constructive Adaptive Visual Analogy is a cognitive theory, I tested the theory with a psychological experiment. In the previous chapters I described Galatea and the models created with it. The focus of Galatea is on the transfer subtask of analogy. Implicit in this formulation is the idea that there is difficulty in analogical problem solving above and beyond the difficulty associated with mapping. I tested this idea in the experiment as well as the third of my main hypotheses: that visual knowledge facilitates transfer even when non-visual knowledge might be available.

In this experiment participants are given Gick and Holyoak's classic tumor problem to solve, using the fortress problem as an analogy. Experimental participants read a story about a general who must overthrow a dictator in a fortress. His army is poised to attack along one of many roads leading to the fortress when the general finds that the roads are mined such that large groups passing will set them off. To solve the problem, the general breaks the army into smaller groups, and they take different roads simultaneously, arriving

together at the fortress. Participants are then given a tumor problem, in which a tumor must be destroyed with a ray of radiation, but the ray will destroy healthy tissue on the way in, killing the patient. The analogous solution (which in this document I will call the "correct" solution) is to have several weaker rays simultaneously converging on the tumor [33, 15].

Much of the analogical problem solving research with the fortress/tumor problem assumes that the difficult parts of analogy are retrieval and mapping. Studies of this sort manipulate retrieval hints, manipulate changes in the fortress story, use completely different source stories, manipulate the timing of the source story [33], force participants to make comparisons, or change instructions. Analogy involves many tasks; these experiments sometimes distinguish between the retrieval stage and later ones, but not between, for example, mapping and transfer. Novick and Holyoak [60] however found that for math word problems only around 40% of participants (50% in one experiment, 32% in the next) were able to find the analogous solution even when the mapping was given as a part of the stimuli. This suggests that the mapping stage is not the only difficult analogical subtask.

This work hypothesizes that transfer of strongly-ordered procedures is computationally complex, even given the correct mapping. To get an idea of how difficult analogical problem solving is above and beyond and mapping, this experiment manipulated whether or not the participants were *given* the mapping between the source and target. If mapping is the only/major source of difficulty in analogical reasoning, then experimental participants given the correct mapping in a cross-domain analogical problem-solving task should have little difficulty successfully transferring the solution. The experiment investigates whether this is the case for cross-domain analogical problem solving.

Diagrams have been shown to help in analogical problem solving in general (e.g. [4]), but not specifically with analogical transfer. The main hypothesis of this experiment is that visual knowledge facilitates transfer even when non-visual knowledge might be available.

1.3.1 Method

1.3.1.1 Participants.

Eighty undergraduate students received extra class credit in exchange for taking part in the experiment. They were randomly assigned to one of the six experimental groups.

1.3.1.2 Design.

Each participant read a description of the fortress problem and how it was solved: "A small country fell under the iron rule of a dictator. The dictator ruled the country from a strong fortress. The fortress was situated in the middle of the country, surrounded by farms and villages. Many roads radiated outward from the fortress like spokes on a wheel. A great general arose who raised a large army at the border and vowed to capture the fortress. His troops were poised at the head of one of the roads leading to the fortress, ready to attack. However, a spy brought the general a disturbing report. The ruthless dictator had planted mines on each of the roads. The mines were set so that small bodies of men could pass over them safely, since the dictator would then destroy many villages in retaliation. A full-scale direct attack on the fortress therefore appeared impossible."

Participants in `diagram` conditions (groups A and D) were given a diagram (see Figure 14) with the following text: "Here is an abstract diagram that describes the problem the general faced, and what he did to solve it. The arrows represent the groups of soldiers marching on roads to the fortress in the center."

Participants in the `draw` condition (group C) were asked to "Please draw a diagram or diagrams that describes the problem the general faced (NOT the solution–we will ask for a drawing of that later.) Please make it abstract. So please don't draw realistic drawings of the fortress, for example."

Then all participants read the solution to the fortress problem: "The general, however, was undaunted. He divided his army up into small groups and dispatched each group to the head of a different road. When all was ready he gave the signal, and each group charged down a different road. All of the small groups passed safely over the mines, and the army then attacked the fortress in full strength. In this way, the general was able to capture the

fortress and overthrow the dictator."

This text is from Gick and Holyoak [33].

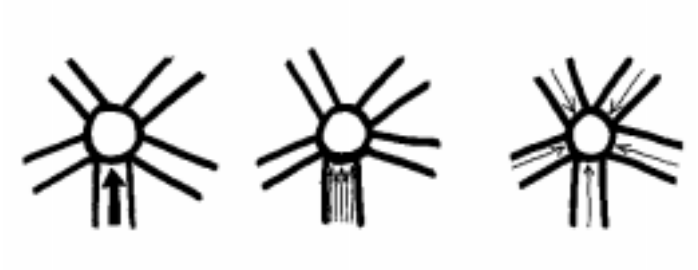

Figure 14: The experimental fortress story diagram used in the diagram conditions (groups A and D.)

Participants in the `draw` condition (group C) were then asked to "Please draw an abstract diagram or diagrams that describes the general's solution to this problem."

All participants looked at the tumor problem: "Suppose you are a doctor faced with a patient who has a malignant tumor in his stomach. It is impossible to operate on the patient, but unless the tumor is destroyed the patient will die. There is a kind of ray that can be used to destroy the tumor. If the rays reach the tumor all at once at a sufficiently high intensity, the tumor will be destroyed. At lower intensities the rays are harmless to healthy tissue, but they will not affect the tumor either. What type of procedure might be used to destroy the tumor with the rays, and at the same time avoid destroying the healthy tissue?"

Participants in the `draw` condition (group C) were then asked to "Please draw a diagram that describes the above problem (NOT the solution–we will ask for a drawing of that later.) Again, please make it abstract. So please don't draw realistic drawings of a tumor, for example."

Participants in the `mapping` conditions (groups A, B, C, and E) read "These problems are analogous. In these stories, the tumor is like the fortress, and the ray of radiation is like the big army that wants to march. The expolding mines are like the patient's body getting hurt by radiation."

Participants in the `diagram` conditions (groups A and D) were then shown a diagram

of the tumor problem, shown in Figure 15.

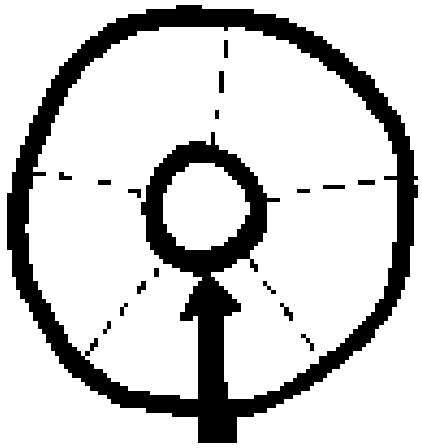

Figure 15: The experimental diagram of the tumor problem used in the diagram condition (groups A and D.)

Participants in all groups read "How would you solve the tumor problem? What type of procedures might be used to destroy the tumor with the rays, and at the same time avoid destroying the healthy tissue? *Use the fortress story as an analogy to help you solve the tumor problem.* Give as many possible solutions as you can think of. This is a difficult problem that requires creativity to solve–you may need to work at it."

Participants in the `draw solution` conditions (groups A, B, C, and D) were then asked to "Please draw diagrams to accompany your written solutions."

Table 6 shows each group (A through F) in this design. The table further shows the number of participants in each group, whether that group gave the participants the mapping, whether diagrams were given, whether they asked to draw diagrams, and whether they were asked to draw solutions, as described above.

The specific hypotheses for this experiment are: First, there will be no large effect of mapping. Second, there will be a positive effect of being in the `diagram` condition. Third, there will be a positive effect of being in the `draw-solution` condition. The `draw` condition

Table 6: Experimental results by group.

Group ID	N	Mapping	Diagram	Draw	Draw-Solution	Correct	%
A	16	x	x		x	15/16	94%
B	14	x			x	14/14	100%
C	15	x		x	x	12/15	80%
D	12		x		x	12/12	100%
E	10	x				7/10	70%
F	11					10/11	91%

does not have a hypothesis associated with it because participants in it tended to draw the solution rather than the problem. This condition was discontinued halfway through the experimentation process.

1.3.1.3 Procedure.

Participants signed a consent form, and were given a sheet of paper with the stimuli (described in the previous section) printed on it. They were asked to take their time and to follow the instructions on the sheet. No participant took more than 30 minutes to complete the experiment. After they finished, they were asked if they had ever heard of the fortress/tumor problems before. They were then debriefed and shown out.

1.3.1.4 Analysis and Scoring.

A given participant was classified as getting the correct answer if any of his or her descriptions of the tumor solution (drawn and written) described 1) multiple rays, 2) weaker rays, and 3) coming in from multiple directions. Those missing any one of these three criteria were classified as having gotten an incorrect answer.

1.3.2 Results

The results are shown in tables 6 and 7. Two participants were excluded from the analysis because they reported having encountered the fortress/tumor problem before.

It is difficult to see the pure effects the conditions by looking at the results tables because it is not a between-subjects design. That is, most participants participated in

Table 7: Experimental results by condition. The only significant difference found was for those with and without the `draw solution` manipulation.

Condition	**with**	**without**
Mapping	87% (48/55)	96% (22/23)
Diagram	96% (27/28)	86% (43/50)
Draw Solution	93% (53/57)	81% (17/21)

multiple conditions. The statistical results reported are from methods that control for co-variation, allowing for *statistical* control such as an ANCOVA, or Analysis of Covariance, and regression. These methods use statistical control of conditions when experimental control is impossible. So, for example, when calculating the correlation between `mapping` and `correct`, for example, it is a *partial* correlation that is meaningful; it is measured controlling for the variables associated with the other factors.

The first goal of this experiment is to investigate the effect of mapping for a cross-domain analogical problem-solving task. This experiment showed no effect of mapping. Controlling for the diagram and draw-solution conditions, the partial correlation between mapping and correctness is negative: -.171 The probability that there was an effect of mapping is insignificant (p=0.144). Even if this result were significant, it is in the wrong direction. That is, those given the mapping fared (insignificantly) worse than those without. The 95% confidence interval for the effect of mapping on correctness is -.296 to .044.[3] Because the interval crosses zero, it is statistically indistinguishable from zero. A regression of mapping on correctness is also shown to be insignificant: r squared (.010) $F(1,61)$=.625, p=.432.

The mapping groups had 87% correct; the non-mapping groups had 96% correct. Because I am relying on a null result, it is important to have enough power to detect a true difference if there is one.

This experiment has to power to detect a medium-sized effect (.31). Thus the positive effect of mapping cannot be *more* than .31. Because 50% or greater is considered a large

[3]This means that if you performed this experiment 100 times, the true population mean would fall between these 95% of the time.

effect, we are 95% confident that there is no large effect, casting doubt on the overwhelming importance placed on mapping.

Another hypothesis is that the `diagram` condition will help. Groups with diagrams (A and D) have 96% correct (n=28) while those without diagrams have 86% correct. On the face of it it looks like it should be significant. But the result is confounded with draw solution (all subjects in the `diagram` condition also have the `draw solution` condition). Controlling for `draw solution` and `mapping` leaves the partial correlation between `diagram` and correct at .101, and not distinguishable from zero (p=.390). A regression of `diagram` on correct is insignificant when it is the only variable in the equation $F(1, 76)$=2.124, p=.149 and remains an insignificant contributor to the model after `mapping` is added (p=.219) and remains insignificant after `draw solution` is added to the equation (p=.876). Though the difference is insignificant, the results are in the predicted direction: Those shown diagrams fared (insignificantly) better than those not shown diagrams.

The second hypothesis is that drawing the solution helps participants get the correct solution. Controlling for `mapping` and `diagram`, the partial correlation between `draw solution` and correctness is significant (.180, p=.024); and the 95% confidence interval is .034 to .479. 93% of the people in the `draw-solution` conditions got it correct. For those not asked to draw the solution the percent correct is 81%. Even controlling for `mapping` and `diagram`, this difference is significant. Not only does it appear that the `draw solution` condition improves performance, but because the confidence intervals do not overlap, the effect of `draw solution` is significantly greater than the effect of `mapping`.

1.3.3 Experimental Conclusions

In conclusion, this experiment has two results: the participants given the mapping did not perform better than those who were not given it, and those asked to draw their solution to the tumor problem outperformed those were not asked to draw it, supporting the claims that there is difficulty in analogical problem solving above and beyond the difficulty associated with mapping and that visual knowledge facilitates transfer even when non-visual knowledge might be available.

Researchers have found other manipulations to this task that have facilitated the participants' finding the analogical soultion. Catrambone and Holyoak [8] facilitated transfer by 1) specifically asking participants to compare the analogs and 2) manipulating the wording in the stimuli such that the solution-relevant information was more salient.

The hypotheses come directly from this work's main three hypotheses. I found that though groups given diagrams did not benefit, those asked to draw their solutions did, partially supporting the notion that visual knowledge facilitates transfer even when non-visual knowledge might be available.

In terms of visual stimuli, animations have been found to be helpful [63]. Gick and Holyoak [34] used diagrams similar to the ones I used to facilitate tranfer, but did not find an effect. A follow up study by Beveridge and Parkins [4] found an effect using diagrams with translucent ray representations where the cumulative effect can be perceptually identified. The similarity of my stimulus to those of Gick and Holyoak could account for why my study did not find an effect of `diagram`. It may also be that perhaps it is the act of *creation* of the visual representation that helps more than a given diagram because the act of creation is more likely to be associated with the correct things in memory. Further investigation is needed to fully understand this discrepancy.

1.4 Conclusion

In this chapter I have briefly described the three sources of evaluation for my hypotheses:

1. Transfer of strongly-ordered procedures is computationally complex, even given the correct mapping.

2. Visual knowledge alone is sufficient for transfer of problem solving procedures in some domains.

3. Visual knowledge facilitates transfer even when non-visual knowledge might be available.

In light of the evidence found, I can now state the claims of this work.

Claim One: Visual knowledge alone is sufficient for transfer of problem solving procedures in some domains.

The Galatea implementation shows that problem-solving procedures for inherently visual domains like the cake/pizza problem can be represented visually, and solutions can be transferred successfully.

Claim Two: Visual knowledge facilitates transfer even when non-visual knowledge might be available.

The fortress/tumor example is an example of a domain which need not be visually represented. Galatea shows that visual knowledge of it can be used to transfer a non-trivial procedure across domains.

The implemented models of L14, L15, L16, and L22 show how Galatea's model of visual processing can account for human participant data as well, and provides details of how visual problem-solving transfer might work. The pen-and-paper models of the rest of the participants in Dr. Craig's experiment show how Galatea might model even more, using only visual knowledge, as well as describing the limits of visual knowledge.

The experiment partially supported the claim in that those who were asked to draw the solutions were more likely to get the analogous answer.

The third hypothesis of this work that visual knowledge facilitates transfer of strongly-ordered procedures. It turns out that the computational details involved in transfer of strongly-ordered procedures appear to bear no relationship with visual knowledge. However, in the course of building Galatea and the models in it, I discovered something about analogical transfer in general:

Claim Three: The successful transfer of strongly-ordered procedures in which new objects are created requires the reasoner to generate intermediate knowledge states and mappings between the intermediate knowledge states of the source and target analogs.

Galatea shows why, in detail, this is hypothesis might be right. A characteristic of strongly-ordered procedures is that components of the problem are *created* by the operations, and these components are acted on by later operations.

The psychological modelling further supports this for human cognition: The doorway is replicated, then moved, then sealed with containing walls. For the transfer of multi-step, strongly-ordered procedures it was necessary for Galatea to generate intermediate knowledge states and mappings.

Claim Four: Evaluation appears to require non-visual knowledge.

Though Galatea transfers problem-solving procedures, it still has no way of knowing if the transferred solution was *adequate* for the new problem. In the tumor problem, in order for the agent to determine if the tumor was destroyed and the patient was still alive, it needed some causal knowledge. By causal we mean knowledge of how things in a system change as they interact. Pre- and post-conditions are a straightforward way to represent this, but it is difficult to imagine what "visual" pre- and post-conditions might look like. Visual representations alone cannot enable evaluation of the solution. Other visual reasoning work that does evaluation, such as Funt [30], must use causal knowledge about things such as the force of gravity to make its evaluative simulations.

The rest of this document is organized as follows: Chapter 2 introduces Galatea and how it works. It goes into the computational details of the algorithms, representation language, and the fortress/tumor example. Chapter 3 describes the cognitive models implemented in Galatea. Chapter 4 describes how the Constructive Adaptive Visual Analogy Theory can account for the other participants in David Craig's data set. Chapter 5 describes the psychological experiment. Chapter 6 discusses in depth the re-representation theory of which Constructive Adaptive Visual Analogy is a part. Chapter 7 describes related work. Chapter 8 concludes. The Appendix has the input and output propositions for all of the examples not already described in the chapters.

CHAPTER II

COMPUTER IMPLEMENTATION: GALATEA

Galatea is a computer implementation of the Constructive Adaptive Visual Analogy model that successfully transfers problem-solving procedures using only visual knowledge. It provides an existence proof for the hypothesis that representation of visual knowledge is sufficient for transfer of problem solving procedures in some domains.

Galatea succesfully works with four problems. 1) four versions of the lab/weed-trimmer problem, 2) the fortress/tumor problem, 3) the cake/pizza problem and 4) the Maxwell example. In later chapters I will describe the lab/weed-trimmer models, and in this chapter I will describe the others.

I will use Gick and Holyoak's fortress/tumor problem [33, 15], described in the introduction, as a running example to describe Galatea.

A procedure for solving a problem can be represented as a series of knowledge states and transformations between them. A knowledge state characterizes the steps in the procedure by specifying information about the elements in the state and relationships between them. A transformation takes in a knowledge state, changes its configuration in some way, and results in the next knowledge state in the sequence. Two successive states are connected by a single transformation. The first knowledge state represents the initial description of the problem. The final knowledge state represents the state in which the problem is solved.

In the first knowledge state of the fortress/tumor problem, the large army takes a single road to the fortress. Starting from the first knowledge state in the fortress story, the first transformation is to break the army up into the smaller armies. This leads to the second knowledge state containing the smaller armies. The second transformation is to move the armies to different roads. The final knowledge state shows all of the armies approaching on different roads.

Since the knowledge states for this model contain only visual knowledge represented

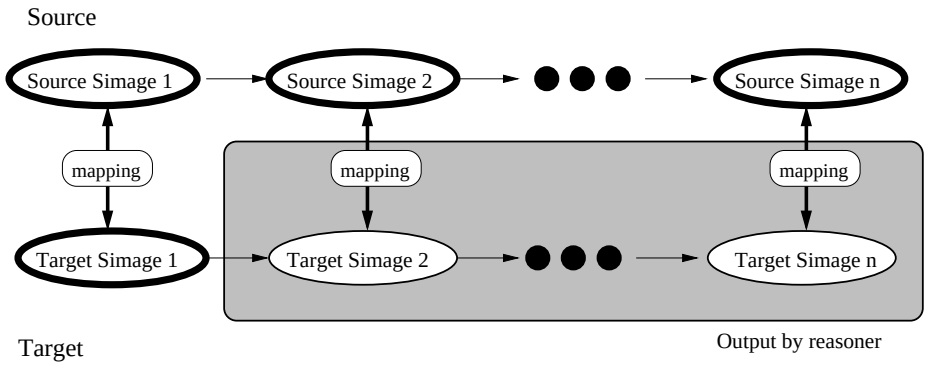

Figure 16: This Figure illustrates Galatea's input and output in the abstract. The knowledge states in the source case are depicted as ovals along the top of the Figure. The knowledge states are visually represented as `s-images`. Transformations between the states in the Figure are depicted as arrows. The target problem is depicted as the leftmost bottom oval. All things in the gray box are output by Galatea.

symbolically, I call the states symbolic images or `s-images`. Figure 16 illustrates Galatea's input and output in the abstract. Galatea takes as input a source analog, an initial target problem `s-image`, and an analogical `mapping` between the initial `s-images` of the source and target. The source is a complete sequence of `s-images` and transformations representing the procedure that solves the source problem. The model transfers the visual transformations one at a time from the source to the target, creating new target `s-images` along the way, with new analogical `mappings` between the corresponding target and source `s-images`.

2.1 Knowledge and Representation

At the high level, the knowledge is partitioned into the following groups: transformations, primitive visual elements, miscellaneous slot values, primitive visual relations, and analogy representations. First I will describe the low-level architecture, then I will describe each category in turn.

2.1.1 Knowledge Architecture

All knowledge in Galatea is represented with two types of propositions (called `chunks`): concepts and relationships. Relationships are frames with four slots: `English`, `Name`, `Type`, `ThingX`, `Relation`, and `ThingY`. Following is an example of a relationship:

```
((ENGLISH ("The door wall in the first s-image of L14 looks like a rectangle."))
(NAME [DOOR-WALL-L14S1_LOOKS-LIKE-RELATION_RECTANGLE])
(TYPE RELATIONSHIP)
(THINGX DOOR-WALL-L14S1)
(RELATION LOOKS-LIKE-RELATION)
(THINGY RECTANGLE))
```

The English slot takes a string of text as a value. The value for the English slot of any chunk is hand-written by the modeller. It is there only to make easy-to-read output and is not used in Galatea's internal functioning. The Name of the relationship is a symbol used as an identifier for the chunk. The Type takes one of two values, specifying whether the chunk is a relationship or a concept. In the case of [DOOR-WALL-L14S1_LOOKS-LIKE-RELATION_RECTANGLE], the relationship between the DOOR-WALL-L14S1 and RECTANGLE is that the DOOR-WALL-L14S1 looks like the RECTANGLE. This relationship is expressed with the ThingX, Relation, and ThingY slots. All relationships express a relation between two things. Those things can only be chunk names, though the chunks they are naming can be either concepts or other relationships.

The Name slot uses a canonical form and can be automatically generated by the system. ThingX, Relation, and ThingY values are seperated with an underscore, and chunk names are enclosed in brackets. So, for example, a more complicated chunk name with a relationship in the ThingX slot looks like:

```
[[DOOR-SET-B1S1_MAPS-TO_DOOR-SET-SET-B1S2]_USES-TRANSFORMATION-RELATION_REPLICATE-TRANSFORMATION]
```

Concepts are simpler, with only three slots: English, Name, and Type.

```
((ENGLISH ("Door-L14s1 is a concept")) (NAME DOOR-L14S1) (TYPE CONCEPT))
```

For the remainder of this document, I will use a shorthand for these chunks for readability:

```
((ENGLISH ("Door-L14s1 is a concept")) (NAME DOOR-L14S1) (TYPE CONCEPT))
```

will read

```
(DOOR-L14S1)
```

Table 8: Covlan's transformations.

Transformations	
Transformation name	**arguments**
add-element	object-type, location (optional)
add-connections	connection/connection-set
decompose	object, number-of-resultants, type
move-to-location	object, new-location
move-to-set	object, object2
put-between	object, object2, object3
replicate	object, number-of-resultants

and

```
((ENGLISH ("The door wall in the first s-image of L14 looks like a rectangle."))
(NAME [DOOR-WALL-L14S1_LOOKS-LIKE-RELATION_RECTANGLE])
(TYPE RELATIONSHIP)
(THINGX DOOR-WALL-L14S1)
(RELATION LOOKS-LIKE-RELATION)
(THINGY RECTANGLE))
```

will read

`(DOOR-WALL-L14S1 LOOKS-LIKE-RELATION RECTANGLE).`

2.1.2 Transformations

It is important that the analogs are represented with a consistent symbolic visual representation language. This fact is more important than the actual ontology of the language used. *Covlan* (Cognitive Visual Language) provides an ontology of visual primitives [12]. Table 8 shows Covlan's ontology of transformations.

2.1.2.1 Add-element

`Add-element` adds a new primitive element in the next `s-image.` The ontology of primitive visual elements are described in the next subsection. The first argument, `object-type`, must be one of the members of the `primitive elements` (e.g. `square` or `circle`). It

determines what kind of shape appears in the next `s-image`. The second argument is `location`, which must be one of Covlan's `locations`: `bottom`, `top`, `right`, `left`, and `center`. What this means is that the next `s-image` will have three relationships added: 1) The `s-image` connected with a `has-component` relation to the name identifying the new component, 2) the new component's name with a `looks-like` relation to the `object-type`, and 3) the component's name with a `has-location` relation to the `location` input as an argument. See Figure 17. `Add-element` is used in the Maxwell example, and will be described in more detail in a later section.

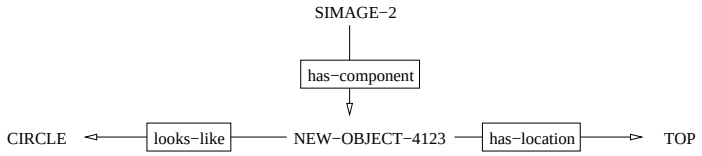

Figure 17: A graphical representation of the three relationships added by the `add-element` transformation. `Relations` are boxed. Objects at the beginning of arrows are in the `ThingX` slot; the objects at the end of the arrows are in the `ThingY` slot.

2.1.2.2 Add-connections

`Add-connections` (See Algorithm 1) is a transformation that inserts a set of connections into the next `s-image`. Input is the name of the set of connections in the source. To determine the nature of the connections in the target, Galatea uses substitution for all the symbols to find the analogous names, so that analogous connections are placed in the next target `s-image`. `Add-connections` is used in the cognitive modelling, and will be described with respect to those examples in a later chapter.

2.1.2.3 Decompose

`Decompose` (See Algorithm 2) takes a primitive element and replaces it in the next `s-image` with some `n` number of elements. It also reduces thickness for each of those elements.

In the fortress/tumor example, `decompose` takes as input 1) The object to be replicated, `RAY`, 2) The number of resultants, `FOUR`, 3) The horizontal mapping between the current and next `s-images`, `TUMOR-SIMAGE2-SOLUTION-MAPPING1`, and 4) the system's memory.

Algorithm 1 add-connections

Inputs:

connections − set a group of connection names;
vmapping The mapping between the current source and target analogs;
current − target − s − image The current target `s-image`;
next − target − s − image The next target `s-image`;
hmapping The mapping between the current and next target analogs;
memory The system memory;

Outputs:

For each added connection, a concept representing that connection;
For each added connection, a proposition describing the `is-subject-for-connection` relationship between a component and the connection concept;
For each added connection, a proposition describing the `is-object-for-connection` relationship between a component and the connection concept;
For each added connection, a proposition describing the distance of the connection;
For each added connection, a proposition describing the angle of the connection;
Horizontal maps between the old elements of the current s-image and new elements of the next s-image;

Procedure:

for all connections-set **do**
memory ← memory + analogous-relationship(connection, v-mapping)
create horizontal maps between changed components

Algorithm 2 decompose

Input:

1. the object to be decomposed: object
2. how many things to decompose the object into: number-of-resultants
3.The mapping between the current and next target analogs
4. The system memory

Output:

1. For each added component, a proposition representing the component.
2. For each added component, a concept describing that that component has a `thickness` of `thin.`
3. For each added component, a proposition describing the `looks-like` relationship to a primitive element.
4. Horizontal maps between the old elements of the current `s-image` and new elements of the next `s-image.`

Procedure:

type ← get-object-type(object)
new-objects ← create-new-objects(number-of-resultants)
memory ← memory + create-relationship(new-objects, has-thickness-relation, thin)
memory ← memory + create-relationship(new-objects, looks-like, type)
create horizontal maps between changed components

`Decompose` input for the fortress/tumor example:

`RAY`

`FOUR`

`TUMOR-PROBLEM`

`TUMOR-SOLUTION`

`TUMOR-SIMAGE2-SOLUTION-MAPPING1`

`MEMORY`

In the fortress/tumor example, `FOUR` new symbols are created. These are the names of the new objects: `SRAY1, LEFT-SRAY1, RIGHT-SRAY1`, and `TOP-SRAY1`. A new set name is also created: `SET7`. The new symbols are connected in relationships to `SET7` with `IN-SET` relations. The type of the original object (the `RAY`), is retrieved from memory. There is a fact in memory: `(RAY LOOKS-LIKE-RELATION LINE)` that Galatea retrieves to know that the resultant objects will also have relationships that connect the new symbols to `LINE` using the `LOOKS-LIKE-RELATION` relation. Also created are those relationships that connect the symbols to the notion of `THIN`, connected by `HAS-THICKNESS.` The horizontal maps are created as well, the process of which I will describe later. Horizontal maps are those maps between elements in subsequent `s-images` in the same series.

`Decompose` output propositions:

`(SRAY1)`

`(LEFT-SRAY1)`

`(RIGHT-SRAY1)`

`(TOP-SRAY1)`

`(SET7)`

`(SET7 LOOKS-LIKE-RELATION SET)`

`(SRAY1 IN-SET SET7)`

`(LEFT-SRAY1 IN-SET SET7) (RIGHT-SRAY1 IN-SET SET7)`

`(TOP-SRAY1 IN-SET SET7)`

`(SRAY1 HAS-THICKNESS THIN)`

`(LEFT-SRAY1 HAS-THICKNESS THIN)`

```
(RIGHT-SRAY1 HAS-THICKNESS THIN)
(TOP-SRAY1 HAS-THICKNESS THIN)
(SRAY1 LOOKS-LIKE-RELATION LINE)
(LEFT-SRAY1 LOOKS-LIKE-RELATION LINE)
(RIGHT-SRAY1 LOOKS-LIKE-RELATION LINE)
(TOP-SRAY1 LOOKS-LIKE-RELATION LINE)
(SET7 MAPS-TO RAY)
(TUMOR-SOLUTION-PROBLEM-MAPPING1 HAS-MAP [RAY_MAPS-TO_SRAY1])
(RAY MAPS-TO SRAY1)
(TUMOR-SOLUTION-PROBLEM-MAPPING1 HAS-MAP [RAY_MAPS-TO_LEFT-SRAY1])
(RAY MAPS-TO LEFT-SRAY1)
(TUMOR-SOLUTION-PROBLEM-MAPPING1 HAS-MAP [RAY_MAPS-TO_RIGHT-SRAY1])
(RAY MAPS-TO RIGHT-SRAY1)
(TUMOR-SOLUTION-PROBLEM-MAPPING1 HAS-MAP [RAY_MAPS-TO_TOP-SRAY1])
(RAY MAPS-TO TOP-SRAY1)
(TUMOR-SOLUTION-PROBLEM-MAPPING1 HAS-MAP [SET7_MAPS-TO_RAY])
```

2.1.2.4 Move-to-location

`Move-to-location` (see Algorithm 3) changes the location of a primitive element from one location to another. This means that in the next `s-image`, the old `has-location` relation is removed and a new `has-location` relation is added, relating the element to the input `location`, which can be an absolute location or another element. `Move-to-location` worked with an older, less robust implementation of the fortress/tumor problem, and is not running with any examples in the current implementation.

2.1.2.5 Move-to-set

`Move-to-set` (see Algorithm 4) takes in two sets as input (we will call them *set-a* and *set-b*). The members of *set-a* are moved to the locations of the members of *set-b*. In the tumor example, the `decomposed` rays are placed on the locations of the distinct body-areas.

Algorithm 3 move-to-location

Input:
1. An object to be moved: object
2. A new location: new-location
3. new-S-image
4. The system memory

Output:
1. A proposition describing that the object is in the new location.
2. Horizontal maps between the old elements of the current s-image and new elements of the next s-image.

Procedure:
memory ← memory + create-relationship(object, new-location, new-s-image)
create horizontal maps between changed components

If *set-a* and *set-b* have the same number of element instances, then each element of *set-a* is placed at the location of a distinct element in *set-b*. The element instance matching is arbitrary.

If *set-a* has more elements, then multiple members of *set-a* are placed at the locations of each member of *set-b*. The number of element instances in these groups is determined by the number of elements in *set-b* divided by the number of elements in *set-a*.

If *set-b* has more elements, then elements of *set-a* are distributed evenly across the locations of the members of *set-b*.

Move-to-set input:

`SET7`

`SET9`

`TUMOR-SIMAGE2-SOLUTION-MAPPING1`

`MEMORY`

The `move-to-set` transformation, in the fortress/tumor example, takes in 1) the set of things to be moved, `SET7`, in this case the set of thin lines representing weak rays, 2) the set of things, the locations of which to move them to, `SET9`, in this case the set of qualitative body-areas, 3) the horizontal mapping `TUMOR-SIMAGE2-SOLUTION-MAPPING1`, and 4) the system memory `MEMORY`.

In the case in which `SET7` is the same size as `SET9`, Galatea retrieves from `MEMORY` the

Algorithm 4 move-to-set

Input:

1. A set of things to be moved:seta
2. A set of things to move the elements of seta to:setb
3. Horizontal mapping
4. The system memory

Output:

1. For each in seta, a proposition describing that that component is at the same location of some element of setb.
2. Horizontal maps between the old elements of the current s-image and new elements of the next s-image.

Procedure:

if size-of(seta) = size-of(setb) **then**
  for all seta and setb **do**
    memory ← memory + make-relationship(elementa, "has-location", location-of(elementb)
else if size-of(seta) > size-of(setb) **then**
  for all setb **do**
    memory ← memory + place (size-of(seta) / size-of(setb)) elements-from-a into each element from b
else if size-of(seta) < size-of(setb) **then**
  for all seta **do**
    memory ← memory + place one elementa in every (size-of(setb) / size-of(seta)) elementb
create horizontal maps between changed components

locations of the members of SET9. This information is in the set of has-location relations. Then relationships are created connecting the members of SET7 (SRAY, LEFT-SRAY, RIGHT-SRAY, TOP-SRAY) to those retrieved locations: (TOP, BOTTOM, LEFT, RIGHT.)

The Move-to-set output propositions follow. The maps here are horizontal. That is, they relate objects in one s-image to objects in the next s-image in the series.

(SRAY)

(SRAY IS-LOCATED-RELATION BOTTOM)

(SRAY MAPS-TO SRAY1)

(TUMOR-SIMAGE2-SOLUTION-MAPPING1 HAS-MAP [SRAY_MAPS-TO_SRAY1])

(LEFT-SRAY)

(LEFT-SRAY IS-LOCATED-RELATION LEFT)

(LEFT-SRAY MAPS-TO LEFT-SRAY1)

(TUMOR-SIMAGE2-SOLUTION-MAPPING1 HAS-MAP [LEFT-SRAY_MAPS-TO_LEFT-SRAY1])

(RIGHT-SRAY)

(RIGHT-SRAY IS-LOCATED-RELATION RIGHT)

(RIGHT-SRAY MAPS-TO RIGHT-SRAY1)

(TUMOR-SIMAGE2-SOLUTION-MAPPING1 HAS-MAP [RIGHT-SRAY_MAPS-TO_RIGHT-SRAY1])

(TOP-SRAY) (TOP-SRAY IS-LOCATED-RELATION TOP)

(TOP-SRAY MAPS-TO TOP-SRAY1)

(TUMOR-SIMAGE2-SOLUTION-MAPPING1 HAS-MAP [TOP-SRAY_MAPS-TO_TOP-SRAY1])

2.1.2.6 Put-between

Put-between (see Algorithm 5) takes two objects that are assumed to be touching, and places some third object in between them. In the new s-image 1) the two objects are no longer touching and 2) the third is touching both of them. Put-between is used in the Maxwell example, and will be described for that example in a later section.

2.1.2.7 Replicate

Replicate (see Algorithm 6) takes in an element or set of elements and generates n new

Algorithm 5 put-between

Input:
1. Some object getting put between others: object
2. Another object that the object is getting put between: object2
3. A third object on the other side of object from object2: object3
4. The horizontal mapping: hmapping
5. The system memory

Output:
1. A proposition describing that object is connected to object2
2. A proposition describing that object is connected to object3
3. A proposition describing that object2 is not connected to object 3.
4. horizontal maps between the old elements of the current s-image and new elements of the next s-image.

Procedure:
memory ← memory + make-relationship(object "is-connected-to" object2)
memory ← memory + make-relationship(object "is-connected-to" object3)
memory ← memory + make-relationship(object2 "is-not-connected-to" object3)
create horizontal maps between changed components

Table 9: Covlan's primitive elements.

Primitive Element name	**attributes**
connection	subject, object, angle, distance
rectangle	location, size, height, width, orientation
circle	location, size, height
line	location, length, end-point1, end-point2, thickness
set	location, orientation, front, middle
curve	location, start-point, mid-point, end-point, thickness

instances of that element or elements in the next `s-image`. Its behavior is similar to `decompose`, except that it does not change the size or thicknes of elements, and can work on sets as well as single element instances. `Replicate` is used in the cognitive modelling, and will be described for those examples in a later chapter.

2.1.3 Primitive Visual Elements

Covlan's ontology of `primitive visual elements` (Table 9) contains: `rectangle, circle, line,` and `set`. Symbols are connected to an `element` type with a relation called `looks-like-relation`. These symbols are `instances` of that `element`. The elements are frame-like structures with

Algorithm 6 replicate

Input:
1. some object to be replicated: object
2. the number of resulting objects: number-of-resultants
3. the horizontal mapping: hmapping
4. the system memory

Output:
1. For each new component, a proposition representing the new component.
2. For each new component, a proposition describing what that component looks like.
3. Horizontal maps between the old elements of the current s-image and new elements of the next s-image.

Procedure:
type ← get-object-type(object)
new-objects ← create-new-objects(number-of-resultants)
memory ← memory + create-relationships(new-objects, looks-like, type)
create horizontal maps between changed components

slots that can hold values. For example, a `rectangle` has a `location, size, height, width,` and `orientation`. All elements can have a `location`, which holds a value representing an absolute location on an `s-image` (e.g. `top`, `right`).

See Figure 18 for an example of how instances of these elements can be arranged in an `s-image`.

The `set` is a special `element`. A `set` can contain any number of `instances` of `elements`. These `instances` are connected with relationships to the set with the `in-set` relation. Sets also have an `orientation,` the value of which is one of the primitive `directions.` An `element instance` in the set is specified in the representation as the `front`, and another as the `middle`. The orientation is defined as an imaginary line from the `middle` to the `front` in the `direction` specified in the `orientation`.

Sometimes a *part* of an `element instance` must be referenced. For example, if a line touches the middle of another line, there must be some way to describe the *end* of the first line and the *middle* of the next. In Covlan different primitive elements have different kinds of `areas`.

`Lines` have `start` and `end points`, as well as `right` and `left-side mid-points`. The element instance's names are related to the symbols naming these areas (e.g. `line1-end-point` with `area-relations`: `has-end-point`, `has-start-point`, `has-rightsidemiddle`, and

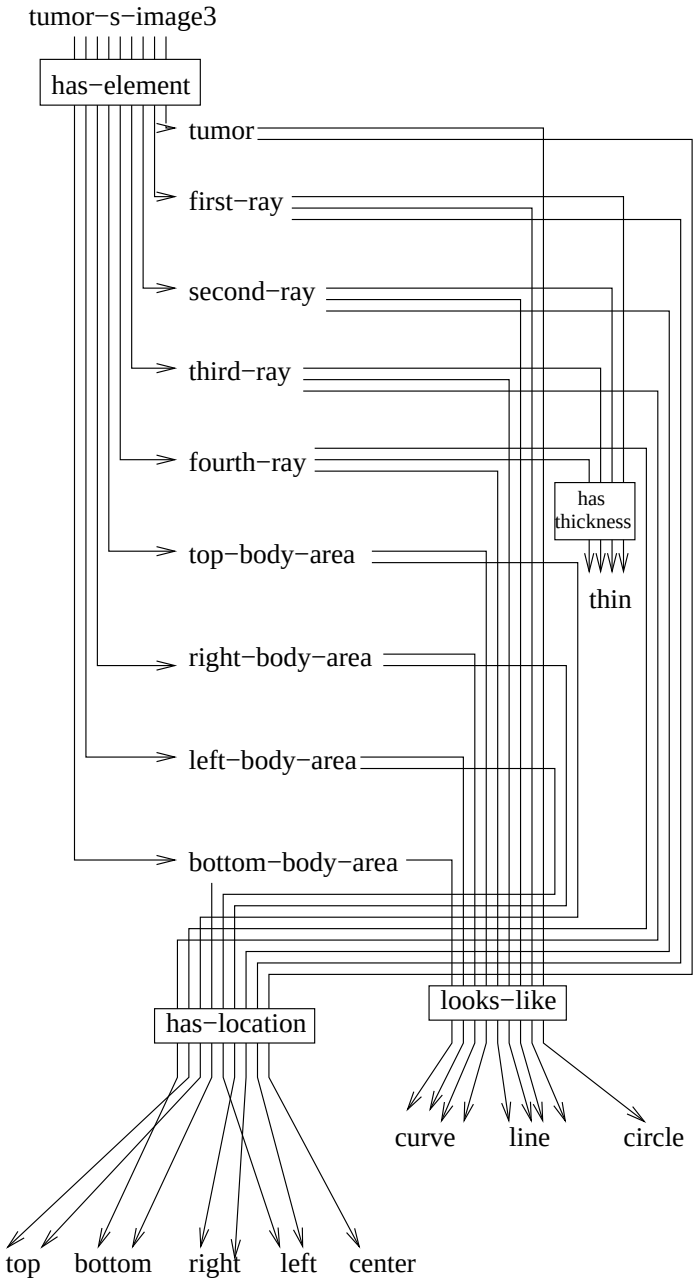

Figure 18: This Figure shows part of the third generated `s-image` in the tumor procedure. Each `relationship` is represented as an arrow. At the beginning of the arrow is the `ThingX` of the relationship, and at the end of the arrow is the `ThingY`. The boxed text in the middle of the arrow is the `Relation.` Each string of unboxed text is a `concept`.

has-leftsidemiddle.

Circles, squares, and rectangles have sides, which are related to element instances with the following relations: has-side1 (the top), has-side2 (the right side), has-side3 (the bottom), and has-side4 (the left side).

Many spatial relationships between primitive elements are represented with connections. A connection is a primitive element with a name. Connections are frames with two four slots: subject, object, angle and distance, represented with is-subject-for-connection, is-object-for-connection, has-angle and has-distance. These relations connect the connection name to distances and angles, which are qualitative miscellaneous slot values. See Figure 19. The object of the connection is distance away from the subject in the direction of angle.

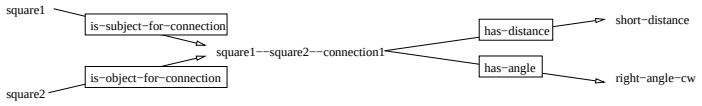

Figure 19: A representation of the relationships involved with a connection. Square2 is a short distance to the right of square1. Right-angle-cw means that the angle is a right angle in the clock-wise direction.

The distances are touching-distance, short-distance and long-distance. The angles are perpendicular-angle (straight ahead), right-angle-cw (a right angle in the clockwise direction, or to the right), 45-angle-cw (a forty-five degree angle to the right), 45-angle-ccw (a forty-five degree angle in the counter-clockwise direction, or to the left), and right-angle-ccw (a right angle to the left). Figure 20 shows the different kinds of connections Covlan can represent. Areas of element instances, as well as element instances themselves, can be connected.

2.1.4 Miscellaneous Slot Values

Miscellaneous slot values are symbols that can give a value to element attributes or transformation arguments. See Table 10. They can be broken down into the following types:

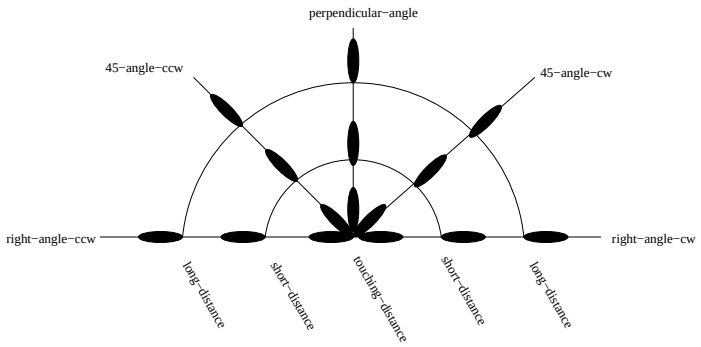

Figure 20: Each of the fifteen black dots in the Figure represents a qualitative `connection` area, with an `angle` and `direction`.

Table 10: Classifications of Miscellaneous Slot Values.

angles	perpendicular-angle, right-angle-cw, 45-angle-cw, 45-angle-ccw, right-angle-ccw
locations	bottom, top, right, center, off-bottom off-top, off-right, off-left
sizes	small, medium, large
thicknesses	thin, thick, very-thick
speeds	slow, medium, fast
directions	left, right, up, down
lengths	short, medium, long

`angles`, `locations`[1], `sizes`, `thicknesses`, `numbers`, `speeds`, `directions`, and `lengths`.

2.1.5 Primitive Visual Relations

The class of `primitive visual relations` (shown in Table 11) describe how certain visual `elements` relate to each other and `miscellaneous slot values`. `Motion relations` describe how element instances are moving in an `s-image`. `Rotating` has the arguments speed and direction.

[1]Relative locations, as opposed to absolute locations, are classified under `primitive visual relations`.

Table 11: Visual and Motion Relations.

Visual Relations	touching, above-below, right-of-left-of, in-front-of-behind, off-s-image
Motion Relations	rotationing, not-rotating

2.1.6 Analogy Representations

Covlan has representations for reasoning about analogies, similar to the analogy ontology of Forbus, Mostek and Ferguson [29]. `S-images` can have `analogies` between them. Each analogy can have any number of analogical `mappings` associated with it (determining which mapping is the best is the *mapping problem.*) Each alignment between two `element` instances or `areas` in a given mapping is called a `map`.[2]

Similarly `s-images` next to each other in sequences have `transform-connections.` These are necessary so the agent can track how `visual elements` in a previous `s-image` change in the next. A difference between `analogies` and `transform-connections` are that there can be multiple analogical `mappings` for an `analogy`, but only one `mapping` for a `transform-connection`. `Mappings` between sequential `s-images` are called `horizontal mappings` (based on the way I have made my diagrams). Analogical `mappings`, between source and target `s-images` are `vertial mappings.`

`Transformations` are attached, in fact, to a `map` between two `elements` in sequential `s-images`. So if a `rectangle` changes into a `circle`, the agent knows which `rectangle` in the previous `s-image` turns into which `circle` in the next `s-image`.

2.2 Inference and Processing

Analogy consists of several steps: *retrieval* is identifying a candidate source analog in memory; *mapping* is finding the best set of correspondences between components of the analogs; *transfer* is the application of knowledge from the source analog to the target analog, which might use various forms of *adaptation*; *evaluation* is determining if the target problem has been solved appropriately; *storage* is storing the target analog in memory for potential reuse. Galatea focuses on the transfer and adaptation stage of analogy. In particular, it adapts and

[2] A map is called a `match hypothesis` in the SME literature.[19]

transfers each `transformation` in the source problem to the target. The `transformations` are transferred literally and the arguments of those `transformations` can be adapted.

For example, the `transformation decompose` is used to turn a `primitive element` instance into some arbitrary number of resultants, taken as an argument. An argument of a `transformation` can be an instance of one of three cases. First, the argument can be a literal, like the number `4` or the location `bottom`. Literals are transferred unchanged to the target.

Second, the argument could be a `element` instance member of the source `s-image`. In this case, the transfer procedure operates on the analogous `element` in the target `s-image`. For example, in the first `transformation` in the fortress story, the decomposed source `soldier path` gets adapted to the `ray` in the target tumor problem.

In the third case, the argument can be a function. Since this case does not occur in the fortress/tumor problem, we will use another example to describe it. Let us suppose that a reasoner needs to feed six people with one Sicilian slice sheet pizza. An analog in memory of cutting a sheet cake for four people is used to generate a solution. Transfer is still difficult because somehow the `four` in the cake analog must be adapted to the number `six` in the source analog. Knowing how many pieces into which to cut the cake or pizza depends on the number of people in each problem. Some notion of count is needed. The use of `functions` as arguments to `transformations` addresses this problem. The cake analog is represented with a `function` that counts the number of people as its argument for the `decompose transformation`. This function has an argument of its own, namely the set of cake eaters, which during adaptation adapts into the set of pizza eaters. When the `transformation` is applied to the pizza, it counts the members of the set of people in the pizza problem (which results in six). `Decompose` produces six pieces of pizza in the next `s-image`.

2.3 Algorithm

Following is an informal description of Galatea's main algorithm, using the fortress/tumor problem as a running example.

Algorithm 7 main-algorithm

Input:

1. Source
2. Target problem
3. Vertical mapping between source and target

Output:

1. A set of new target knowledge states
2. Vertical mappings between corresponding source and target states
3. Horizontal mappings between successive target states
4. Transformations connecting successive target states

Procedure

while more-source-states(goal-conditions, memory) **do**

current-s-image ← get-next-target-s-image(target problem, current-s-image)
current-source-s-image ← get-next-source-s-image(source, current-source-s-image)
current-transformation ← get-transformation(current-s-image)
current-arguments ← get-arguments(current-source-s-image)
source-objects-of-transformation ← get-target-object-of-trans(current-source-s-image)
current-vertical-mapping ← get-mapping(current-target-s-image, current-source-s-image)
target-object-of-transformation ← get-source-object-of-transformation(current-vertical-mapping, source-objects-of-transformation)
target-arguments ← adapt-arguments(get-arguments(current-source-s-image, current-source-s-image))
memory ← memory + apply-transformation(current-transformation, target-object-of-transformation, target-arguments)
memory ← memory + create-horizontal-mapping(current-target-s-image, get-next-target-s-image)
current-target-s-image ← get-next-target-s-image
current-source-s-image ← get-next-source-s-image
memory ← memory + carry-over-unchanged-relationships(applied-transformation)
memory ← memory + create-vertical-mapping(current-target-s-image, current-source-s-image)

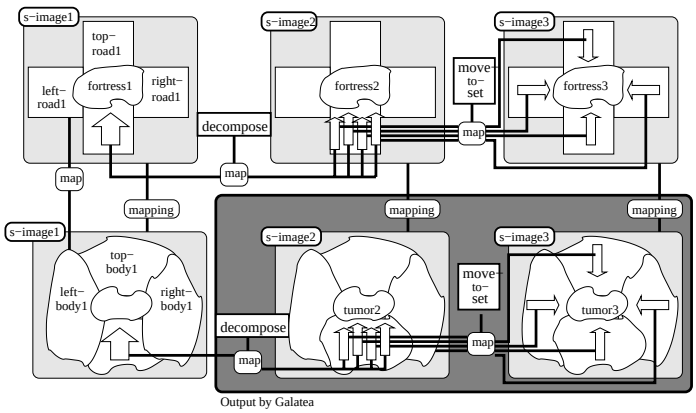

Figure 21: This Figure shows Galatea's input and output for the fortress/tumor problem. The top series of `s-images` in the Figure shows the visual representation of the solved fortress problem. The bottom series shows the target tumor problem. The bottom left `s-image` is the initial state of the tumor problem. The shaded box shows the output of the system.

In this subsection the reader should refer to Figure 21. A more formal representation of this algorithm can be found at Algorithm 7.

1. **Identify the first s-images of the target and source cases.** These are the current source and target s-images.

2. **Identify the `transformations` and their associated arguments in the current `s-image` of the source case.** This step finds out how the source case gets from its current `s-image` to the next `s-image`. In the fortress/tumor example, the transformation is `decompose`, with `four` as the `number-of-resultants` argument (not shown).

3. **Identify the objects of the transformations.** The object of the transformation is what object, if any, the transformation acts upon. For the `decompose` transformation, the object is the `soldier-path1` (the thick arrow in the top left `s-image` in Figure 21.)

4. **Identify the corresponding objects in the target problem.** `Ray1` (the thick arrow in the bottom left `s-image`) is the corresponding component of the source case's `soldier-path1`, as specified by the `mapping` between the current source and target `s-images` (not shown). A single object can be mapped to any number of other objects.

If the object in question is mapped to more than one other object in the target, then the same transformation is applied to all of them in the next step.

5. **Apply the transformation with the arguments to the target problem component.** A new `s-image` is generated for the target problem (bottom middle) to record the effects of the `transformation`. The `decomposetransformation` is applied to the `ray1`, with the argument `four`. The result can be seen in the bottom middle `s-image` in Figure 21. The new rays are created for this `s-image`. Adaptation of the arguments can happen in three ways, as described above: If the argument is an element of the source s-image, then its analog is found. If the argument is a function, then the function is run (note that the function itself may have arguments which follow the same adaptation rules as transformation arguments). Otherwise the arguments are transferred literally.

6. **Map the original objects in the target to the new objects in the target.** A `transform-connection` and `mapping` are created between the target problem `s-image` and the new `s-image` (not shown). `Maps` are created between the corresponding objects. In this example it would mean a `map` between `ray1` in the left bottom `s-image` and the four rays in the second bottom `s-image`. A `map` is also created between the `ray1` to the set of thinner rays. Galatea does not solve the `mapping` problem, but a `mapping` from the correspondences of the first `s-image` enables Galatea to automatically generate the `mappings` for the subsequent `s-images`.

7. **Map the new objects of the target case to the corresponding objects in the source case.** Here the rays of the second target `s-image` are mapped to soldier paths in the second source `s-image`. This step is necessary for the later iterations (i.e. going on to another`transformation` and `s-image`). Otherwise the reasoner would have no way of knowing on which parts of the target `s-image` the later transformations would operate.

8. **Check to see if there are any more source s-images.** If there are not, exit, and the solution is transferred. If there are further `s-images` in the source case, set the

current `s-image` equal to the next `s-image` and go to step 1.

In the fortress/tumor example, the input to the main algorithm is

`tumor-problem`

`fortress-problem memory`.

Now I will describe the output produced by Galatea.

First the new objects are created: `SRAY1, LEFT-SRAY1, RIGHT-SRAY1`, `TOP-SRAY1`, and `set7`. As far as Galatea is concerned, the symbol names could be anything; they have meaningful names only so they make sense to humans reading the input and output. "Sray" is short for "solution ray," to distinguish it from the ray in the problem state. The "1" is my notation, in this example, that it is not the ray in the final state, but in the intermediate state.

```
(SRAY1) (LEFT-SRAY1) (RIGHT-SRAY1) (TOP-SRAY1) (SET7)
```

The following chunks place the different objects in the set:

```
(SET7 LOOKS-LIKE-RELATION SET)
(SRAY1 IN-SET SET7)
(LEFT-SRAY1 IN-SET SET7)
(RIGHT-SRAY1 IN-SET SET7)
(TOP-SRAY1 IN-SET SET7)
```

The objects are now thin, not thick, and are specified as being `lines`:

```
(SRAY1 HAS-THICKNESS THIN)
(LEFT-SRAY1 HAS-THICKNESS THIN)
(RIGHT-SRAY1 HAS-THICKNESS THIN)
(TOP-SRAY1 HAS-THICKNESS THIN)
(SRAY1 LOOKS-LIKE-RELATION LINE)
(LEFT-SRAY1 LOOKS-LIKE-RELATION LINE)
(RIGHT-SRAY1 LOOKS-LIKE-RELATION LINE)
(TOP-SRAY1 LOOKS-LIKE-RELATION LINE)
```

The s-images are connected with a transform-connection, which has a mapping.

```
(TUMOR-PROBLEM TRANSFORM-CONNECTION TUMOR-SIMAGE2)
([TUMOR-PROBLEM_TRANSFORM-CONNECTION_TUMOR-SIMAGE2]
HAS-MAPPING TUMOR-PROBLEM-SIMAGE2-MAPPING1)
```

In the following propositions Galatea connects the new objects created to the object it came from in the previous s-image (horizontal maps). All the new objects map to ray. These maps are also connected to the horizontal mapping between the problem and the second s-image.

```
   (TUMOR-PROBLEM-SIMAGE2-MAPPING1
HAS-MAP [RAY_MAPS-TO_SRAY1])
(RAY MAPS-TO SRAY1)
(TUMOR-PROBLEM-SIMAGE2-MAPPING1
HAS-MAP [RAY_MAPS-TO_LEFT-SRAY1])
(RAY MAPS-TO LEFT-SRAY1)
(TUMOR-PROBLEM-SIMAGE2-MAPPING1
HAS-MAP [RAY_MAPS-TO_RIGHT-SRAY1])
(RAY MAPS-TO RIGHT-SRAY1)
(TUMOR-PROBLEM-SIMAGE2-MAPPING1
HAS-MAP [RAY_MAPS-TO_TOP-SRAY1])
(RAY MAPS-TO TOP-SRAY1)
(TUMOR-PROBLEM-SIMAGE2-MAPPING1
HAS-MAP [SET7_MAPS-TO_RAY])
(SET7 MAPS-TO RAY)
```

The *unchanged* objects in the previous s-image are carried over to the new s-image, with new symbol names.[3] Also, relations that describe them are also carried over. The new

[3]The new symbol names come from a function that returns an appropriate new symbol name. These

s-image, tumor-simage2, must contain the new objects.

```
   (TUMOR-SIMAGE2)
(TUMOR-SIMAGE2 CONTAINS-OBJECT TOP-BODY-S2)
(TOP-BODY-S2)
(TOP-BODY-S2 LOOKS-LIKE-RELATION CIRCLE)
(TOP-BODY-S2 HAS-START-POINT-RELATION TOP)
(TUMOR-SIMAGE2 CONTAINS-OBJECT RIGHT-BODY-S2) (RIGHT-BODY-S2)
(RIGHT-BODY-S2 LOOKS-LIKE-RELATION CIRCLE)
(RIGHT-BODY-S2 HAS-START-POINT-RELATION RIGHT)
(TUMOR-SIMAGE2 CONTAINS-OBJECT LEFT-BODY-S2) (LEFT-BODY-S2)
(LEFT-BODY-S2 LOOKS-LIKE-RELATION CIRCLE)
(LEFT-BODY-S2 HAS-START-POINT-RELATION LEFT)
(TUMOR-SIMAGE2 CONTAINS-OBJECT BODY-S2) (BODY-S2)
(BODY-S2 LOOKS-LIKE-RELATION CIRCLE)
(BODY-S2 HAS-START-POINT-RELATION BOTTOM)
(TUMOR-SIMAGE2 CONTAINS-OBJECT SET9)
(SET9)
(BODY-S2 IN-SET SET9)
(LEFT-BODY-S2 IN-SET SET9)
(RIGHT-BODY-S2 IN-SET SET9)
(TOP-BODY-S2 IN-SET SET9)
(SET9 LOOKS-LIKE-RELATION SET)
(TUMOR-SIMAGE2 CONTAINS-OBJECT TUMOR-S2)
(TOP-SRAY1 HAS-START-POINT-RELATION BOTTOM)
(TUMOR-S2)
(TOP-SRAY1 HAS-END-POINT-RELATION TUMOR-S2)
(BODY-S2 HAS-END-POINT-RELATION TUMOR-S2)
(LEFT-BODY-S2 HAS-END-POINT-RELATION TUMOR-S2)
```

names are hand-coded for readability.

(RIGHT-BODY-S2 HAS-END-POINT-RELATION TUMOR-S2)

(TOP-BODY-S2 HAS-END-POINT-RELATION TUMOR-S2)

(TUMOR-S2 LOOKS-LIKE-RELATION CIRCLE)

(TUMOR-S2 HAS-LOCATION-RELATION CENTER)

(TUMOR-S2 HAS-SIZE-RELATION SMALL)

The new `s-image` contains the new objects.

(TUMOR-SIMAGE2 CONTAINS-OBJECT TOP-SRAY1)

(TUMOR-SIMAGE2 CONTAINS-OBJECT SRAY1)

(TUMOR-SIMAGE2 CONTAINS-OBJECT RIGHT-SRAY1)

(TUMOR-SIMAGE2 CONTAINS-OBJECT LEFT-SRAY1)

Galatea also puts in the vertical maps between the second source `s-image` and the second target `s-image`.

(FORTRESS-SIMAGE2-TUMOR-SIMAGE2-MAPPING1

HAS-MAP [LEFT-SRAY1_MAPS-TO_LEFT-SSOLDIER1-PATH])

(LEFT-SRAY1 MAPS-TO LEFT-SSOLDIER1-PATH)

(FORTRESS-SIMAGE2-TUMOR-SIMAGE2-MAPPING1

HAS-MAP [RIGHT-SRAY1_MAPS-TO_RIGHT-SSOLDIER1-PATH])

(RIGHT-SRAY1 MAPS-TO RIGHT-SSOLDIER1-PATH)

(FORTRESS-SIMAGE2-TUMOR-SIMAGE2-MAPPING1 HAS-MAP [SET7_MAPS-TO_SET3])

(SET7 MAPS-TO SET3)

(FORTRESS-SIMAGE2-TUMOR-SIMAGE2-MAPPING1

HAS-MAP [SRAY1_MAPS-TO_SSOLDIER1-PATH])

(SRAY1 MAPS-TO SSOLDIER1-PATH)

(FORTRESS-SIMAGE2-TUMOR-SIMAGE2-MAPPING1

HAS-MAP [TOP-SRAY1_MAPS-TO_TOP-SSOLDIER1-PATH])

(TOP-SRAY1 MAPS-TO TOP-SSOLDIER1-PATH)

(TUMOR-PROBLEM-SIMAGE2-MAPPING1 HAS-MAP [TUMOR_MAPS-TO_TUMOR-S2])

```
(TUMOR MAPS-TO TUMOR-S2)
(TUMOR-PROBLEM-SIMAGE2-MAPPING1 HAS-MAP [SET2_MAPS-TO_SET9])
(SET2 MAPS-TO SET9)
(TUMOR-PROBLEM-SIMAGE2-MAPPING1 HAS-MAP [BODY_MAPS-TO_BODY-S2])
(BODY MAPS-TO BODY-S2)
(TUMOR-PROBLEM-SIMAGE2-MAPPING1
HAS-MAP [LEFT-BODY_MAPS-TO_LEFT-BODY-S2])
(LEFT-BODY MAPS-TO LEFT-BODY-S2)
(TUMOR-PROBLEM-SIMAGE2-MAPPING1
HAS-MAP [RIGHT-BODY_MAPS-TO_RIGHT-BODY-S2])
(RIGHT-BODY MAPS-TO RIGHT-BODY-S2)
(TUMOR-PROBLEM-SIMAGE2-MAPPING1 HAS-MAP [TOP-BODY_MAPS-TO_TOP-BODY-S2])
(TOP-BODY MAPS-TO TOP-BODY-S2)
```

Having finished completing the second `s-image`, Galatea produces the third `s-image` in the series. It generates propositions representing the things to be moved, with their new locations.

```
   (SRAY)
(SRAY IS-LOCATED-RELATION BOTTOM)
(TOP-SRAY)
(TOP-SRAY IS-LOCATED-RELATION TOP)
(LEFT-SRAY)
(LEFT-SRAY IS-LOCATED-RELATION LEFT)
(RIGHT-SRAY)
(RIGHT-SRAY IS-LOCATED-RELATION RIGHT)
```

The horizontal maps between the previous and new `s-image` for the changed objects:

```
(SRAY MAPS-TO SRAY1)
(TUMOR-SIMAGE2-SOLUTION-MAPPING1 HAS-MAP [SRAY_MAPS-TO_SRAY1])
```

```
(LEFT-SRAY MAPS-TO LEFT-SRAY1)
(TUMOR-SIMAGE2-SOLUTION-MAPPING1
HAS-MAP [LEFT-SRAY_MAPS-TO_LEFT-SRAY1])
(TOP-SRAY MAPS-TO TOP-SRAY1)
(TUMOR-SIMAGE2-SOLUTION-MAPPING1 HAS-MAP [TOP-SRAY_MAPS-TO_TOP-SRAY1])
(RIGHT-SRAY MAPS-TO RIGHT-SRAY1)
(TUMOR-SIMAGE2-SOLUTION-MAPPING1
HAS-MAP [RIGHT-SRAY_MAPS-TO_RIGHT-SRAY1])
```

The horizontal maps for unchanged objects:

```
(TOP-BODY-S2 MAPS-TO TOP-BODY-S3)
(TUMOR-SIMAGE2-SOLUTION-MAPPING1
HAS-MAP [RIGHT-BODY-S2_MAPS-TO_RIGHT-BODY-S3])
(RIGHT-BODY-S2 MAPS-TO RIGHT-BODY-S3)
(TUMOR-SIMAGE2-SOLUTION-MAPPING1
HAS-MAP [LEFT-BODY-S2_MAPS-TO_LEFT-BODY-S3])
(LEFT-BODY-S2 MAPS-TO LEFT-BODY-S3)
(TUMOR-SIMAGE2-SOLUTION-MAPPING1 HAS-MAP [BODY-S2_MAPS-TO_BODY-S3])
(BODY-S2 MAPS-TO BODY-S3)
(TUMOR-SIMAGE2-SOLUTION-MAPPING1 HAS-MAP [SET9_MAPS-TO_SET10])
(SET9 MAPS-TO SET10)
(TUMOR-SIMAGE2-SOLUTION-MAPPING1 HAS-MAP [TOP-SRAY1_MAPS-TO_TOP-SRAY])
(TOP-SRAY1 MAPS-TO TOP-SRAY)
(TUMOR-SIMAGE2-SOLUTION-MAPPING1 HAS-MAP [TUMOR-S2_MAPS-TO_TUMOR-S3])
(TUMOR-S2 MAPS-TO TUMOR-S3))
```

As well as the horizontal analogy representations:

```
(TUMOR-SIMAGE2 TRANSFORM-CONNECTION TUMOR-SOLUTION)
([TUMOR-SIMAGE2_TRANSFORM-CONNECTION_TUMOR-SOLUTION] HAS-MAPPING
```

TUMOR-SIMAGE2-SOLUTION-MAPPING1)

The vertical maps between the source and target `s-images`:

(FORTRESS-SOLUTION-TUMOR-SOLUTION-MAPPING1

HAS-MAP [SET8_MAPS-TO_LEFT-SSOLDIER-PATH])

(SET8 MAPS-TO LEFT-SSOLDIER-PATH)

(TUMOR-SIMAGE2-SOLUTION-MAPPING1

HAS-MAP [TOP-BODY-S2_MAPS-TO_TOP-BODY-S3])

The unchaged objects and relations:

(TUMOR-SOLUTION CONTAINS-OBJECT TUMOR-S3)

(TUMOR-SOLUTION)

(TUMOR-S3)

(TUMOR-S3 LOOKS-LIKE-RELATION CIRCLE)

(TUMOR-S3 HAS-LOCATION-RELATION CENTER)

(TUMOR-S3 HAS-SIZE-RELATION SMALL)

(TUMOR-SOLUTION CONTAINS-OBJECT TOP-SRAY) (TOP-SRAY IN-SET SET8)

(SET8)

(TOP-SRAY HAS-THICKNESS THIN)

(TOP-SRAY LOOKS-LIKE-RELATION LINE)

(TOP-SRAY HAS-START-POINT-RELATION BOTTOM)

(TOP-SRAY HAS-END-POINT-RELATION TUMOR-S3)

(TUMOR-SOLUTION CONTAINS-OBJECT SET10)

(SET10)

(SET10 LOOKS-LIKE-RELATION SET)

(TUMOR-SOLUTION CONTAINS-OBJECT BODY-S3)

(BODY-S3)

(BODY-S3 LOOKS-LIKE-RELATION CIRCLE)

(BODY-S3 HAS-START-POINT-RELATION BOTTOM)

```
(BODY-S3 IN-SET SET10)
(BODY-S3 HAS-END-POINT-RELATION TUMOR-S3)
(TUMOR-SOLUTION CONTAINS-OBJECT LEFT-BODY-S3)
(LEFT-BODY-S3)
(LEFT-BODY-S3 LOOKS-LIKE-RELATION CIRCLE)
(LEFT-BODY-S3 HAS-START-POINT-RELATION LEFT)
(LEFT-BODY-S3 IN-SET SET10)
(LEFT-BODY-S3 HAS-END-POINT-RELATION TUMOR-S3)
(TUMOR-SOLUTION CONTAINS-OBJECT RIGHT-BODY-S3) (RIGHT-BODY-S3)
(RIGHT-BODY-S3 LOOKS-LIKE-RELATION CIRCLE)
(RIGHT-BODY-S3 HAS-START-POINT-RELATION RIGHT)
(RIGHT-BODY-S3 IN-SET SET10)
(RIGHT-BODY-S3 HAS-END-POINT-RELATION TUMOR-S3)
(TUMOR-SOLUTION CONTAINS-OBJECT TOP-BODY-S3)
(TOP-BODY-S3)
(TOP-BODY-S3 LOOKS-LIKE-RELATION CIRCLE)
(TOP-BODY-S3 HAS-START-POINT-RELATION TOP)
(TOP-BODY-S3 IN-SET SET10)
(TOP-BODY-S3 HAS-END-POINT-RELATION TUMOR-S3)
```

2.3.1 Adapt-arguments

When an argument needs to be adapted to the target analog, Galatea looks at the argument and determines whether it is a `literal`, a `function`, or a `component` of an `s-image`. Literals are returned verbatim. If the argument is a function (e.g. the number of people in a group) then Galatea applies the same function to the analogous group in the target and returns that value. If the argument is a component, then Galatea returns the analogous object in the target.

In the fortress/tumor problem, the `adapt-arguments` algorithm takes in the symbols

Algorithm 8 adapt-arguments

Input:
1. argument
2. mapping

Output:
1. an adapted argument.

Procedure:
if literal? argument **then**
  return argument
else if function? argument **then**
  return calculate-function(argument)
else if component? argument **then**
  return (get-analogous-component(argument, mapping))

`FOUR` and `FORTRESS-PROBLEM-TUMOR-PROBLEM-MAPPING1`. Since `FOUR` is in Galatea's list of literals, it executes the "literal" case and returns the symbol as is: `FOUR.`

2.3.2 Carry-over-unchanged-relationships

Following is a description of the `carry-over-unchanged-relationships` function. See Algorithm 9. The `get-analogous-chunks` sub-function constructs returns chunks that are identical to the input chunks, except that the symbols that have `maps` in the input `mapping` are replaced with those symbols they are associated with in those `maps.` The vertical `map` relationships are carried over as well, constituting the vertical maps for unchanged components.

Algorithm 9 carry-over-unchanged-relationships

Input:
1. The Memory: memory
2. The horizontal mapping: h-mapping
3. Transformation
4. Previous-s-image

Output:
1. Analogous chunks.

Procedure:
new-chunks ← get-chunks((run-transformation(transformation))
old-analogous-chunks ← get-analogous-chunks(new-chunks, h-mapping)
old-chunks ← get-all-chunks(previous-s-image)
chunks-to-transfer ← old-chunks − old-analogous-chunks
memory ← memory + create-analogous-chunks(chunks-to-transfer, h-mapping)

Algorithm 10 creation-of-horizontal-maps-between-changed-components

Input:
1. Transformation results
2. Target-objects-of-transformation

Output:
1. New horizontal maps between the current and next target s-image.

Procedure:
post-transform-components ← get-chunks((run-transformation(transformation))
memory ← memory + create-maps(post-transform-components, target-objects-of-transformation)

2.3.3 Creation-of-horizontal-maps-between-changed-components

The `creation-of-horizontal-maps-between-changed-components` (see Algorithm 10) is embedded in each of the `transformations`. The `transformation` results are obtained from running the `transformation`. The `target-objects-of-transformation` are known because they are the input to the `transformation`. The two lists are put in alphabetical order and `maps` are created between each *nth* list object.

The output is:

```
(SET7 MAPS-TO RAY)
(TUMOR-PROBLEM-SIMAGE2-MAPPING1 HAS-MAP [RAY_MAPS-TO_SRAY1])
(RAY MAPS-TO SRAY1)
(TUMOR-PROBLEM-SIMAGE2-MAPPING1 HAS-MAP [RAY_MAPS-TO_LEFT-SRAY1])
(RAY MAPS-TO LEFT-SRAY1)
(TUMOR-PROBLEM-SIMAGE2-MAPPING1 HAS-MAP [RAY_MAPS-TO_RIGHT-SRAY1])
(RAY MAPS-TO RIGHT-SRAY1)
(TUMOR-PROBLEM-SIMAGE2-MAPPING1 HAS-MAP [RAY_MAPS-TO_TOP-SRAY1])
(RAY MAPS-TO TOP-SRAY1)
(TUMOR-PROBLEM-SIMAGE2-MAPPING1 HAS-MAP [SET7_MAPS-TO_RAY])
```

Similarly, `creation-of-horizontal-maps-between-unchanged-components` (see Algorithm 11) makes `maps` between old objects (the objects in the `old-s-image` and new objects (from the `current-s-image`, minus the objects created by the `transformation`),

Algorithm 11 creation-of-horizontal-maps-between-unchanged-components

Input:
1. Transformation results
2. Old-s-image
3. Current-s-image
4. Post-transform-components
5. Old-components
6. Current-components

Output:
1. new horizontal maps between the current and next target s-image.

Procedure:
old-components ← get-all-components(old-s-image) − target-objects-of-transformation
current-components ← get-all-components(current-s-image) − post-transform-components
memory ← memory + create-maps(old-components, current-components)

alphabetizes them, and creates maps between the *nth* item in each list.

The output follows. `Set2` is the set of body areas in the first `s-image`, and `set9` is the set of body areas in the second `s-image`.

```
   (TUMOR-PROBLEM-SIMAGE2-MAPPING1

HAS-MAP [TUMOR_MAPS-TO_TUMOR-S2])

(TUMOR MAPS-TO TUMOR-S2)

(TUMOR-PROBLEM-SIMAGE2-MAPPING1

HAS-MAP [RAY_MAPS-TO_TOP-SRAY1])

(RAY MAPS-TO TOP-SRAY1)

(TUMOR-PROBLEM-SIMAGE2-MAPPING1

HAS-MAP [SET2_MAPS-TO_SET9])

(SET2 MAPS-TO SET9)

(TUMOR-PROBLEM-SIMAGE2-MAPPING1

HAS-MAP [BODY_MAPS-TO_BODY-S2])

(BODY MAPS-TO BODY-S2)

(TUMOR-PROBLEM-SIMAGE2-MAPPING1

HAS-MAP [LEFT-BODY_MAPS-TO_LEFT-BODY-S2])

(LEFT-BODY MAPS-TO LEFT-BODY-S2)
```

(TUMOR-PROBLEM-SIMAGE2-MAPPING1

HAS-MAP [RIGHT-BODY_MAPS-TO_RIGHT-BODY-S2])

(RIGHT-BODY MAPS-TO RIGHT-BODY-S2)

(TUMOR-PROBLEM-SIMAGE2-MAPPING1

HAS-MAP [TOP-BODY_MAPS-TO_TOP-BODY-S2])

(TOP-BODY MAPS-TO TOP-BODY-S2))

Algorithm 12 creation-of-vertical-maps-between-changed-components

---

Input:

1. Target transformation results
2. Source transformation results
3. New-target-components
4. New-source-components

Output:

1. new vertical maps between the current source and target `s-images`.

Procedure:

new-target-components ← target transformation results
new-source-components ← source transformation results
memory ← memory + create-maps(new-target-components, new-source-components)

2.3.4 Creation-of-vertical-maps-between-changed-components

The algorithm for creating vertical maps between changed components (see Algorithm 12) takes as input the transformation results in the source *and* target, alphabetizes them, and creates `maps` between the *nth* item in each list.

Output:

(FORTRESS-SIMAGE2-TUMOR-SIMAGE2-MAPPING1

HAS-MAP [LEFT-SRAY1_MAPS-TO_LEFT-SSOLDIER1-PATH])

(LEFT-SRAY1 MAPS-TO LEFT-SSOLDIER1-PATH)

(FORTRESS-SIMAGE2-TUMOR-SIMAGE2-MAPPING1

HAS-MAP [RIGHT-SRAY1_MAPS-TO_RIGHT-SSOLDIER1-PATH])

(RIGHT-SRAY1 MAPS-TO RIGHT-SSOLDIER1-PATH)

(FORTRESS-SIMAGE2-TUMOR-SIMAGE2-MAPPING1

Table 12: Primitive elements from fortress problem s-image 1.

Visual Object	attributes	value
Fortress	looks-like: location:	curve center
Bottom-road	looks-like:	line
Right-road	looks-like:	line
Left-road	looks-like:	line
Top-road	looks-like:	line
Soldier-path	looks-like: location: thickness:	line bottom-road thick

```
HAS-MAP [SET7_MAPS-TO_SET3])
(SET7 MAPS-TO SET3)
(FORTRESS-SIMAGE2-TUMOR-SIMAGE2-MAPPING1
HAS-MAP [SRAY1_MAPS-TO_SSOLDIER1-PATH])
(SRAY1 MAPS-TO SSOLDIER1-PATH)
(FORTRESS-SIMAGE2-TUMOR-SIMAGE2-MAPPING1
HAS-MAP [TOP-SRAY1_MAPS-TO_TOP-SSOLDIER1-PATH])
(TOP-SRAY1 MAPS-TO TOP-SSOLDIER1-PATH)
```

2.4 The Fortress/Tumor Problem

I chose the fortress/tumor example because some experimental participants have used visual inferences in solving it [43]. Table 12 shows some of the `visual elements` and their attribute values for the first fortress problem `s-image`.

I represented the fortress story with three `s-images` (see Figure 21.) The first is a representation of the original fortress problem. It has four roads, represented as thick `lines`, radiating out from the fortress, which was a `curve` in the `center` (`curves` are used to represent irregular shapes). I represented the original soldier path as a `thick`

`line` on the bottom road. This `s-image` was connected to the second with a `decompose transformation`, where the arguments were `soldier-path1` for the `object` and `four` for the `number-of-resultants`. The second `s-image` shows the `soldier-path1` decomposed into four thin `lines`, all still on the bottom road. The `lines` are thinner to represent smaller groups.

I represented the start state of the tumor problem as a single `s-image`. The tumor itself is represented as a `curve`. The `ray` of radiation is a thick `line` that passes through the bottom body part.

In the fortress/tumor example, after the `decompose transformation` generates a number of smaller armies (by transforming a thick line into thinner lines), those armies must be dispersed to the various roads, in various locations in the image. In a previous version of this model [12, 13] each army line was `moved-to-location` individually to each road line. This solution was brittle because the number of roads to which the armies moved needed to match exactly the number of body areas the weaker rays moved to in the target.

The model now uses `sets` to address this problem. By grouping the armies, roads, rays, and body parts into their own sets, Galatea adapts the solution in the source analog to accommodate differing numbers of any of these `elements`. Rather than using the `move-to-location transformation` on each army, it uses `move-to-set` to the change the location of the `set` of armies. The argument to this function is a `set` of roads. The `move-to-set` function takes one set and distributes its members around the locations of another set.

I have described in some detail the how the fortress/tumor example was implemented in Galatea. This example shows the system's robustness with respect to transfer when different set sizes come into play.

2.5 The Cake/Pizza Problem

I chose the Cake/Pizza problem to demonstrate Galatea's adaptation ability with respect to the transformation arguments. For this problem Galatea is given the target problem of feeding six people with one pizza. It is also given a source analog in which a cake is cut

into four pieces for four people. The two states connected at the top of Figure 22 illustrate the source analog; the first state at the bottom series in the Figure illustrates the target problem.

The cake analog contains only two `s-images`, the initial and the goal `s-images`, as illustrated in Figure 22. For the cake analog, Galatea represents the initial `s-image` as a `rectangle` of a specific size, and the goal `s-image` as four, smaller `rectangles`. For the target pizza analog, Galatea initially knows only the initial `s-image` and represents the pizza as a `rectangle` (see the left side of the target in Figure 22).

In the cake analog, the `decompose transformation` takes one shape (the rectangle) and a number `n` (an argument of the `transformation`) and results in `n` smaller pieces of the same shape.[4] Galatea initially does not know of any `transformations` for the target pizza problem.

Note that this is all the knowledge that Galatea has about the source analog and the target problem. That is, it does not know of any conceptual hierarchy that specifies that cake and pizza are, for example, similar food categories. It does not even know that there are goals in the source cake analog and the target pizza analog,, let alone that the two goals are similar. The reason that the source cake analog may be used given the target pizza problem is the similarity in their shapes (which is why both are represented as rectangles), and not because of any similarity between their food categories or the goals in the source and target problems. To introduce the latter kind of knowledge would amount to use of non-visual knowledge, and is therefore beyond Galatea's intended scope.

The question now becomes how Galatea can transfer knowledge of the `decompose transformation` from the source cake analog to the target pizza analog. For the current example, this is quite simple: Galatea knows that the `rectangle` representing the pizza in the initial `s-image` of the target problem corresponds to the rectangle representing the cake in the initial `s-image` in the source. Therefore, it attempts to transfer the `s-images` and

[4]The `decompose transformation` cannot break up one shape into `n` smaller *different* shapes, as one might cut a round pizza into roughly triangular shapes. To do this would require changing the `transformation` so that it either had a complex notion of how shapes can be sectioned, or took as an argument the resultant shapes.

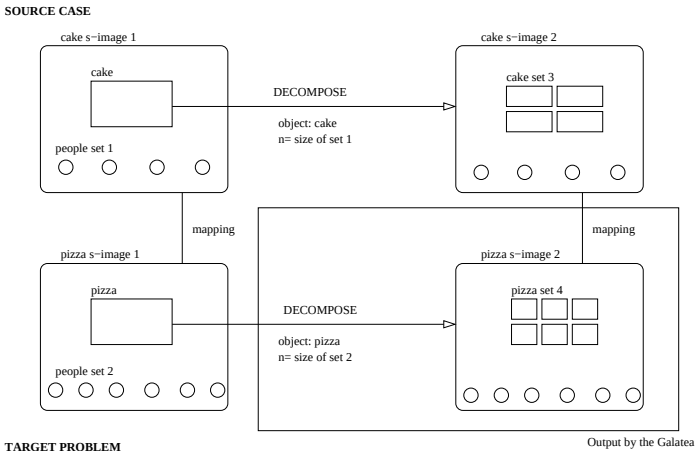

Figure 22: The two `s-images` along the top are the representation of the cake analog. The `decompose transformation` turns the `rectangle` representing the cake into `four` pieces of cake in `s-image` 2. It gets the number four by evaluating the function that gets the size of the set of people. The same thing happens in the target problem, but since the set size is different, it results in `six` rather than `four` pieces in the second `s-image`.

`transformation`s following the initial `s-image` in the source to the target.

One source of difficulty here is that while the number of people in the cake analog is six, the number of people in the pizza problem is four (a different number). Galatea uses the concepts of `sets` and members of a group to address this issue. It explicitly represents the people as belonging to a `set` in the source and target analogs. The `decompose transformation` takes, as an argument, a `set` which is counted for each analog: in the cake analog, this set contains six members, in the target problem it contains four. When the `decompose transformation` is transferred to the target pizza problem, it is instantiated with the count of the set of people in *that* problem. When the `transformation` is executed, it generates six smaller rectangles in the goal `s-image` of the target problem.

2.6 The Maxwell Example

The next example is the model of the construction process James Clerk Maxwell used in deriving the electromagnetic field equations. The interpretation I employed is taken from Nersessian's cognitive-historical analysis of James Clerk Maxwell's problem solving [55, 56, 58, 57, 59]. I will present here only in broad terms and refer you to Nersessian's extensive research for the details.

Maxwell's problem was the mathematization of the electromagnetic field concept. In Maxwell's model of electromagnetism, the ether between magnets swirl into vortices, which are all spinning in the same direction. The spinning causes the vortices to shorten, pulling the magnets together. Maxwell constructed this vortex-fluid model through an analogy with continuum mechanics.

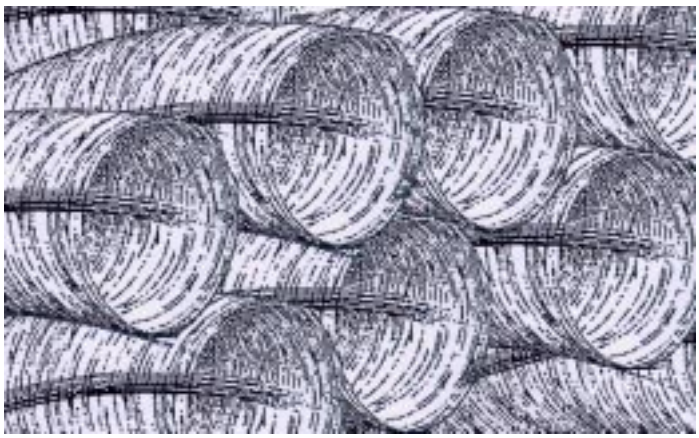

Figure 23: Many vortices packed together.

Figure 23 is drawn from Maxwell's description of the aether and the vortices. I do not assume his mental model had this level of detail. In thinking about how electricity relates to magnetism he needed to consider multiple vortices and their interaction. Nersessian hypothesizes that a generic cross section as drawn in Figure 24 approximates his mental model at this stage of problem solving. Making topological changes of this kind to imagined physical systems has been shown in our earlier work to be useful in problem solving [38, 39, 40].

Since Maxwell assumed the vortices were spinning in the same direction, he found a problem in the model: friction would cause the vortices to slow or stop. Figure 25, drawn by Maxwell, shows his solution to this problem. He introduced what he called "idle wheel particles" spinning in the opposite direction between the vortices. He used this model in the further derivation of the mathematical laws of the electromagnetic field. Nersessian mounts a sustained argument for the generativity of the models in Maxwell's derivation in her work.

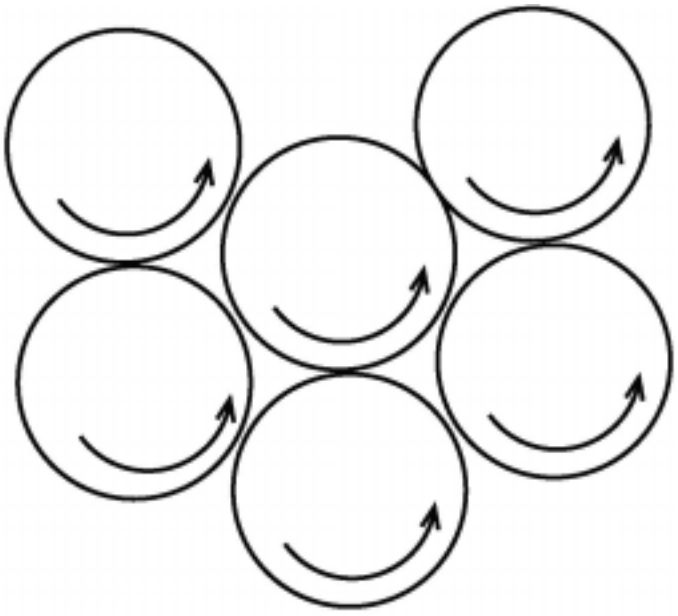

Figure 24: Cross-section of the vortices.

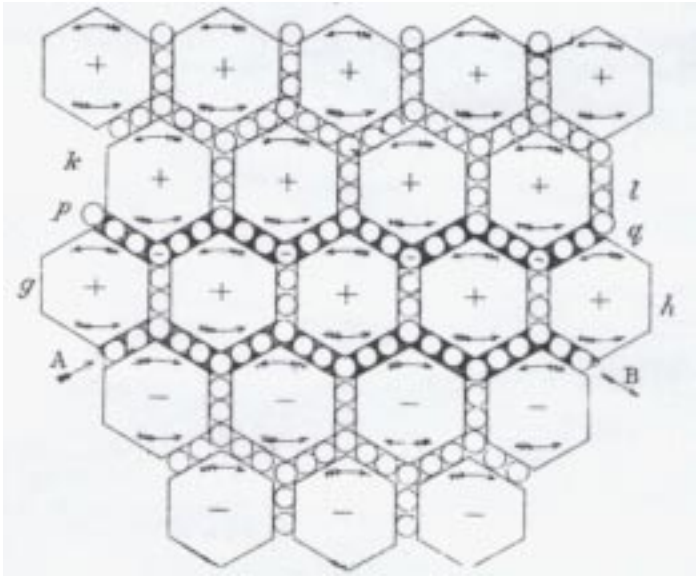

Figure 25: Maxwell's drawing of the wheels in the vortices ([51] p.489).

The next issue was that of how Maxwell got the idea to put in the idle wheel particles. Nersessian hypothesizes that Maxwell used a visual analogy drawn from another model in memory to obtain the notion of the idle wheels and then transfer the notion to the vortex model. Maxwell noted that in machine mechanics such problems are solved with idle wheels [50]. But gear systems and continuum mechanical systems, such as the vortex fluid model, are quite different. She hypothesizes that understanding the cross-sectional model of the vortices generically as "spinning wheels" enabled Maxwell to retrieve his knowledge of gear systems which in turn enabled him to generate the abstraction of "dynamically smooth connectors" and instantiate it as "idle wheels" between vortices in the model.

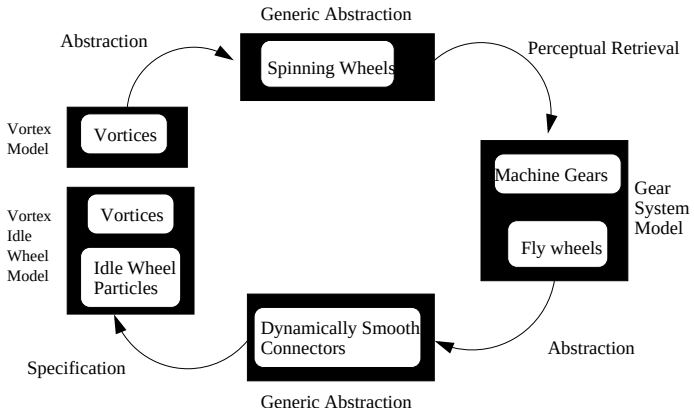

Figure 26: The analysis of how Maxwell transferred the idea of the dynamically smooth connectors from the gear system model to the vortex idle wheel model through the use of a generic abstraction.

Figure 26 summarizes Nersessian's analysis of the process through which Maxwell created the analogy. The vortices in the initial vortex model were abstracted into generic spinning wheels. Then, the abstraction was used as a probe to retrieve the gear system model, which was perceptually similar to it. This model contained the notion of fly wheels acting between gears to keep them moving. The fly-wheel mechanism was abstracted to the generic notion of a dynamically smooth connector. From there it was specified into the idle wheel particles in the new vortex idle wheel model. Galatea models the transfer of the solution from the generic abstraction to the vortex model.

There is reason to think that Maxwell used visual reasoning in this episode because

he used visual language, drew visual representations, and explicitly discussed the analogy when describing the system, e.g., "We have obtained a point of view from which we may regard the relation of an electric current to its lines of force as analogous to the relation of a toothed wheel or rack to wheels which it drives." [51](p472).

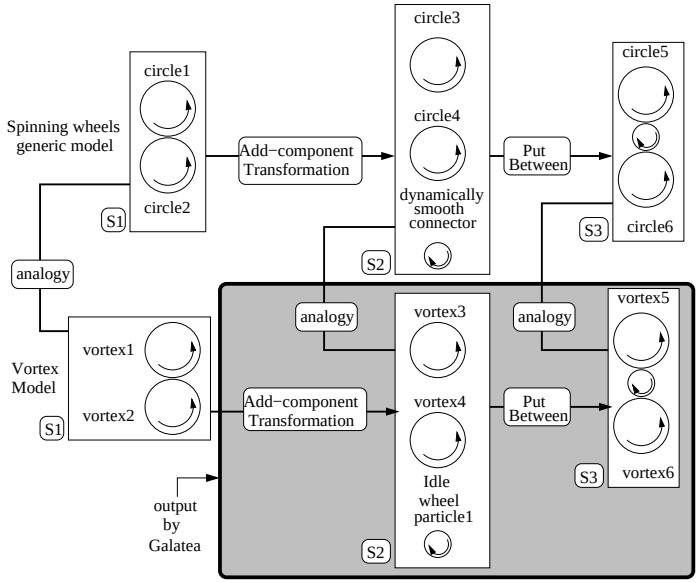

Figure 27: The source and target analogs for the Galatea implementation of the Maxwell example. The top `s-image` series represents the source analog. The shaded area represents the output of Galatea as a result of the analogical transfer.

Figure 27 is a diagram of the input to Galatea for the Maxwell case. S1, S2, and S3 refer to `s-images` for each series. The `circle` represents the generic spinning wheels pictured in the cross section of the vortices (Figure 27). The idle wheels are represented as `circles`. In his drawing of the solution Maxwell exaggerated the deviation of the vortices from circles, rendering them as hexagonal cross sections in order to emphasize the packing of the idle wheel particles between them. However, in the mathematical analysis he treated the vortices as rigid pseudo spheres and a generic cross section of these would be approximately circular, as in Figure 24. I used the `primitive element circle` for this reason. There is an analogy between the two first `s-images`. `Mapping` is enabled by the visual abstraction: Even though spinning vortices will not always look like circles, as discussed above, generically they approximate circles, facilitating the analogy.

The idle wheel particles are added with the `transformation` `add-element`. As shown in Figure 27, the first two `s-images` in the spinning wheels generic model are connected with an `add-element` `transformation`, which adds the dynamically smooth connector, which looks like a small `circle`. Its exact location is unspecified, since one can have an attribution of what something is without knowing exactly where it is (corresponding to the different what/where pathways in the brain.)

The ouput of this instance of `add-element` results in the following propositions:

```
(VORTEX-IDLEWHEEL)
(VORTEX-IDLEWHEEL LOOKS-LIKE-RELATION CIRCLE)
(VORTEX-IDLEWHEEL MAPS-TO NOTHING)
(VP-SIMAGE1-VP-SIMAGE2-MAPPING1
HAS-MAP [VORTEX-IDLEWHEEL_MAPS-TO_NOTHING])
(VP-SIMAGE2 HAS-COMPONENT VORTEX-IDLEWHEEL)
```

The second `transformation`, `put-between`, places the new idle-wheel circle in between the other spinning vortex circles, resulting in the final `s-image`.

The output of the second transformation is:

```
(VORTEX-IDLEWHEEL2)
(VORTEX-IDLEWHEEL2 LOOKS-LIKE-RELATION CIRCLE)
(MAIN-VORTEX3 IS-NOT-CONNECTED-TO 2ND-VORTEX3)
(MAIN-VORTEX3 IS-CONNECTED-TO VORTEX-IDLEWHEEL2)
(2ND-VORTEX3 IS-CONNECTED-TO VORTEX-IDLEWHEEL2)
([MAIN-VORTEX3_IS-CONNECTED-TO_VORTEX-IDLEWHEEL2]
HAS-ARGUMENT TOUCHING)
([2ND-VORTEX3_IS-CONNECTED-TO_VORTEX-IDLEWHEEL2]
HAS-ARGUMENT TOUCHING)
(VORTEX-IDLEWHEEL2 MAPS-TO VORTEX-IDLEWHEEL)
(MAIN-VORTEX3 MAPS-TO MAIN-VORTEX2)
(2ND-VORTEX3 MAPS-TO 2ND-VORTEX2)
```

```
(VP-SIMAGE2-VP-SIMAGE3-MAPPING1
HAS-MAP [VORTEX-IDLEWHEEL2_MAPS-TO_VORTEX-IDLEWHEEL])
(VP-SIMAGE2-VP-SIMAGE3-MAPPING1
HAS-MAP [MAIN-VORTEX3_MAPS-TO_MAIN-VORTEX2])
(VP-SIMAGE2-VP-SIMAGE3-MAPPING1
HAS-MAP [2ND-VORTEX3_MAPS-TO_2ND-VORTEX2])
```

2.7 Summary

I will re-iterate the hypotheses of this work and describe how Galatea relates to them. The first hypothesis is that transfer of strongly-ordered procedures is computationally complex, even given the correct mapping. I discovered that the successful transfer of strongly-ordered procedures *in which new objects are created* is indeed complex. It requires the reasoner to generate intermediate knowledge states and mappings between the intermediate knowledge states of the source and target analogs. Galatea shows why, in detail, this is so. Components of the problem are *created* by the operations, and these components are acted on by later operations. In the tumor problem, for example, the strong ray must be turned into weaker rays before they can be moved. When the reasoner transfers the second operation of moving the soldier paths, how does it know that the corresponding objects in the target are the weaker rays? It must have some mapping to make this inference. And since the weaker rays do not exist in the start state of the tumor problem, this `mapping` cannot be given as input with the initial mapping. The new knowledge state with the weaker rays must be generated, and then a `mapping` must be made on the fly between it and the second knowledge state of the source.

My second and third hypotheses are that visual knowledge alone is sufficient for transfer of problem solving procedures in some domains, and that visual knowledge facilitates transfer even when non-visual knowledge might be available.

Galatea, implemented with four examples, shows that non-trivial problem-solving procedures can be represented visually and transferred successfully across domains. The

cake/pizza example shows transfer for a domain where the visual representation is straightforward, and the fortress/tumor example shows cross-domain analogy where non-visual knowledge might be available to a human reasoner.

The work on Galatea has also resulted in an unexpected discovery: That evaluation requires non-visual knowledge. It appears that evaluation is beyond the abilities of pure visual reasoning. Though Galatea transfers problem-solving procedures, it still has no way of knowing if the transferred solution was *adequate* for the new problem. In the tumor problem, in order for the agent to determine if the tumor was destroyed and the patient was still alive, it needed some causal knowledge. By causal we mean knowledge of how things in a system change as they interact. Pre- and post-conditions are a straightforward way to represent this, but it is difficult to imagine what "visual" pre- and post-conditions might look like. Visual representations alone cannot enable evaluation of the solution.

CHAPTER III

COGNITIVE MODELLING: PART ONE

Galatea is intended to be a partial cognitive model of visual analogical transfer in human beings. To support Galatea with respect to its psychological plausibility I modelled some of the visual aspects of four experimental participants' drawings.

Dr. David Craig ran 34 participants in an analogical transfer experiment [10]. Participants were shown a problem-solving solution about a laboratory, presented with text and a diagram. They were asked to solve an analogous problem with a weed-trimmer, presented with text only. Of these, 17 participants (in three conditions) correctly described the analogous solution. All participants were asked to draw a diagram to illustrate their suggested solutions. The given diagrams were of four slightly different kinds, as described in this chapter and the next.

The source given was a laboratory clean room problem. A single door lets in dirty air, so a vestibule is added, with two doors where one door stayed shut while the other was open (see Figure 28). The target problem is a weed trimmer arm attached to a truck that must be able to pass through street signs. The analogous solution is to design an arm with two latching doors, so that while one is open to let the sign pass, the other stays closed to support the arm and trimmer. Participants produced diagrams describing their solutions to the problems. I modelled four of these experimental participants in Galatea: L14, L22, L15, and L16.

3.1 The Galatea Model of L14

L14 received Condition 1 of the lab problem (see Figure 28). Figure 30 shows what L14 wrote on his or her data sheet during the experiment.

I represented the source analog as a series of `s-images` connected with `transformations`. See the top of Figure 31 for an abstract diagram of the source analog, and see Figure 29 for

a diagram of some of the propositions in its first `s-image`.

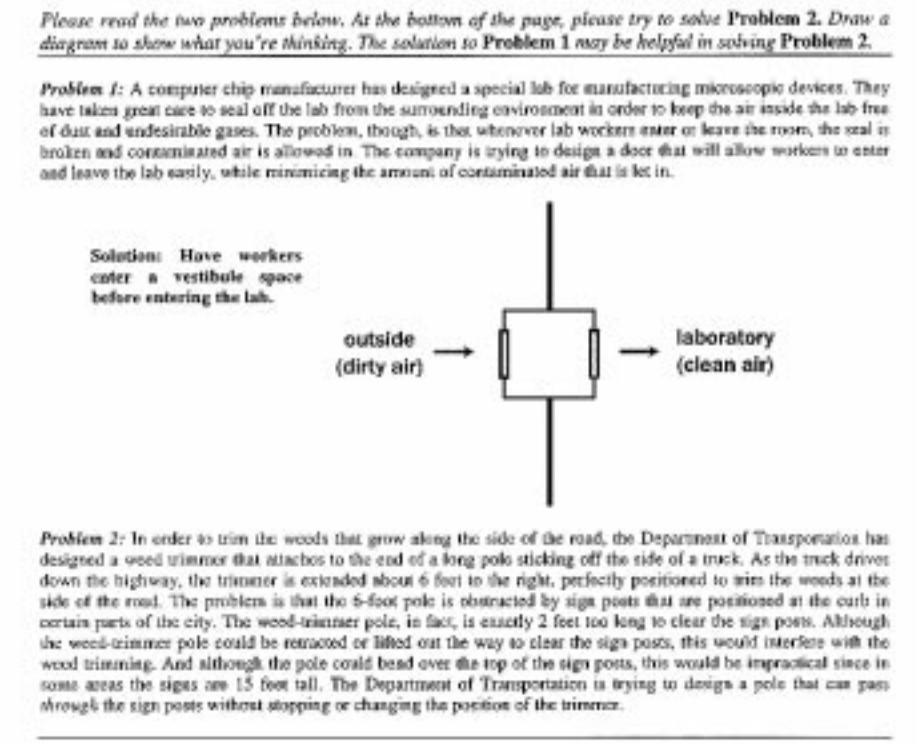

Figure 28: Condition 1: Plan view of lab, with the vestibule centered.

The model of L14 involves five `transformations` (See Figure 31). The first `transformation` is `replicate`. It takes in the `door-set-l14s1` as an argument, generating `door-set1-l14s2` and `door-set2-l14s2` in the next `s-image.`

The second `transformation` is `add-connections` which places the door sets in the correct position in relation to the top and bottom walls.

The third and fourth `transformations` are `add-component`, which add the top and bottom containment walls.

The fifth `transformation`, another `add-connections`, places these containment walls in the correct positions in relation to the door sets and the top and bottom walls.

I will describe the first two `transformations` in detail. The first `transformation` in the lab-base1 source is a `replicate`, which takes two arguments: some `object` and some `number-of-resultants`. In this case the `object` is `door-set-b1s1` (represented as `door-set` in Figure 8. `b1s1` means "base one, `s-image` one.") and the `number-of-arguments`

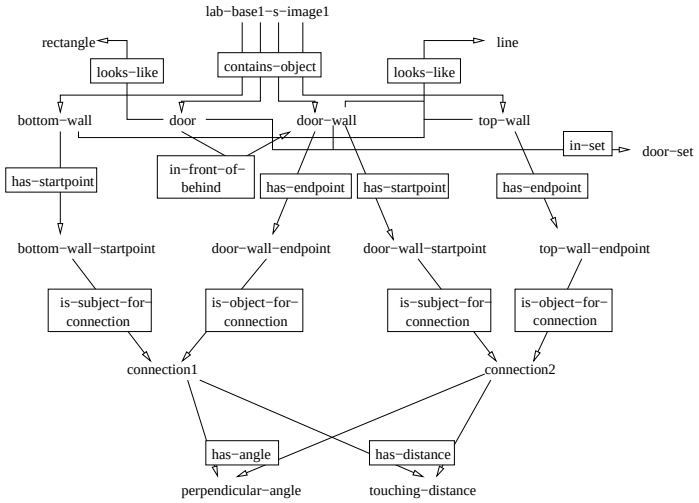

Figure 29: This Figure shows part of the first `s-image` in L14's source `s-image` series. Each `relationship` is represented as an arrow. At the beginning of the arrow is the `ThingX` of the relationship, and at the end of the arrow is the `ThingY`. The boxed text in the middle of the arrow is the `Relation.` Each string of unboxed text is a `concept`.

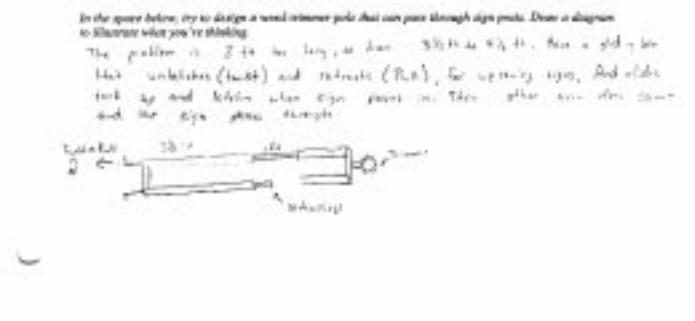

Figure 30: The source data for L14. The drawing above and handwritten text are what participant L14 produced on the experiment sheet.

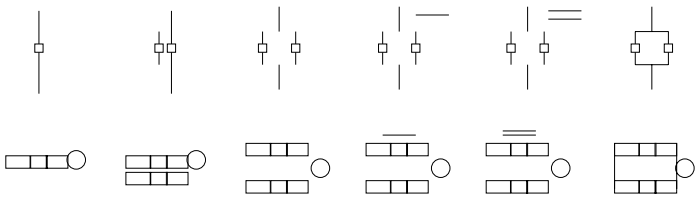

Figure 31: The implementation of L14. The top series of `s-images` represents the source analog (the lab problem) and the bottom series the target. There are six `s-images` for the five `transformations`.

is two. The replicate is applied to the first L14 s-image, with the appropriate adaptation to the arguments: The mapping between the first source and target s-images indicates that the door-set-b1s1 maps to the door-set-l14s1, so the former is used for the target's object argument. The number two is a literal, so it is transferred directly.

Using a function that takes in the name of an element instance or set (in this case door-set-l14s1) and recursively returns all set names and element instances, Galatea retrieves (from memory of the source s-image with the replications in it) all propositions with any of those set names and element instances in the thingX or thingY slots. These propositions are put through a function that creates the same number of new propositions with the same relations and literals, but with new names for the element instances. These new propositions are stored in memory. The effect of this is a replication of the intended structure. This occurs once for each replication.

Galatea chooses an arbitrary name for the superset of door-sets (in this case door-sets-set-l14s2) and connects door-set1-l14s2 and door-set2-l14s2 to it with in-set relations. It makes a map between L14's s-image1 and s-image2, connecting door-set-l14s1 to door-sets-set-l14s2. It also creates maps from door-set-l14s1 to door-set1-l14s2 and another to door-set2-l14s2.

The other propositions from L14's s-image1 are put through a function that finds analagous propositions: literals and relations are kept the same, and element instance names are replaced with new names for the new s-image. For example, the top-door-l14s1 becomes top-door-l14s2.

Maps between the element instances in the target s-image1 and the target s-image2 are stored in memory as well.

The mapping between lab-base1-simage2 and l14-simage2 is automatically generated. Element instances that are results of source transformations are mapped to newly-generated instances in the target. All other maps are carried over to the new s-images with their new names.

The inputs to replicate are the object to be replicated DOOR-SET-L14S1, the number of resultants 2, the current and next target s-images L14-SIMAGE1 and L14-SIMAGE2, the

mapping L14-SIMAGE1--L14-SIMAGE2--MAPPING1 and the memory.

The output is

(L14-SIMAGE1 TRANSFORM-CONNECTION L14-SIMAGE2)

(L14-SIMAGE1--L14-SIMAGE2--MAPPING1)

([L14-SIMAGE1_TRANSFORM-CONNECTION_L14-SIMAGE2] HAS-MAPPING

L14-SIMAGE1--L14-SIMAGE2--MAPPING1)

(DOOR-SET-L14S1 MAPS-TO DOOR-SETS-SET-L14S2)

(DOOR-SET1-L14S2 IN-SET DOOR-SETS-SET-L14S2)

(L14-SIMAGE1--L14-SIMAGE2--MAPPING1

has-map [DOOR-L14S1_MAPS-TO_DOOR1-L14S2])

(DOOR-L14S1 MAPS-TO DOOR1-L14S2)

(L14-SIMAGE1--L14-SIMAGE2--MAPPING1

has-map [DOOR-SET-L14S1_MAPS-TO_DOOR-SET1-L14S2])

(DOOR-SET-L14S1 MAPS-TO DOOR-SET1-L14S2)

(L14-SIMAGE1--L14-SIMAGE2--MAPPING1

has-map [DOOR-WALL-L14S1_MAPS-TO_DOOR-WALL1-L14S2])

(DOOR-WALL-L14S1 MAPS-TO DOOR-WALL1-L14S2)

(L14-SIMAGE1--L14-SIMAGE2--MAPPING1

has-map [DOOR-WALL-SIDE2-L14S1_MAPS-TO_DOOR-WALL1-SIDE2-L14S2])

(DOOR-WALL-SIDE2-L14S1 MAPS-TO DOOR-WALL1-SIDE2-L14S2)

(DOOR-WALL1-L14S2 HAS-SIDE2 DOOR-WALL1-SIDE2-L14S2)

(DOOR-WALL1-L14S2 LOOKS-LIKE-RELATION RECTANGLE)

(DOOR-WALL1-L14S2 IN-SET DOOR-SET1-L14S2)

(DOOR1-L14S2 IN-FRONT-OF DOOR-WALL1-L14S2)

(L14-SIMAGE2 CONTAINS-OBJECT DOOR-WALL1-L14S2)

(DOOR1-L14S2 LOOKS-LIKE-RELATION RECTANGLE)

(DOOR1-L14S2 IN-SET DOOR-SET1-L14S2)

(L14-SIMAGE2 CONTAINS-OBJECT DOOR1-L14S2)

(DOOR-SET1-L14S2 LOOKS-LIKE-RELATION SET)

(L14-SIMAGE2 CONTAINS-OBJECT DOOR-SET1-L14S2)

(DOOR-SET-L14S1 LOOKS-LIKE-RELATION SET)

(DOOR-SET1-L14S2)

(DOOR1-L14S2)

(DOOR-WALL1-L14S2)

(DOOR-WALL1-SIDE2-L14S2)

(DOOR-SET2-L14S2 IN-SET DOOR-SETS-SET-L14S2)

(L14-SIMAGE1--L14-SIMAGE2--MAPPING1

has-map [DOOR-L14S1_MAPS-TO_DOOR2-L14S2])

(DOOR-L14S1 MAPS-TO DOOR2-L14S2)

(L14-SIMAGE1--L14-SIMAGE2--MAPPING1

has-map [DOOR-SET-L14S1_MAPS-TO_DOOR-SET2-L14S2])

(DOOR-SET-L14S1 MAPS-TO DOOR-SET2-L14S2)

(L14-SIMAGE1--L14-SIMAGE2--MAPPING1

has-map [DOOR-WALL-L14S1_MAPS-TO_DOOR-WALL2-L14S2])

(DOOR-WALL-L14S1 MAPS-TO DOOR-WALL2-L14S2)

(L14-SIMAGE1--L14-SIMAGE2--MAPPING1

has-map [DOOR-WALL-SIDE2-L14S1_MAPS-TO_DOOR-WALL2-SIDE2-L14S2])

(DOOR-WALL-SIDE2-L14S1 MAPS-TO DOOR-WALL2-SIDE2-L14S2)

(DOOR-SET2-L14S2 LOOKS-LIKE-RELATION SET)

(DOOR-SET2-L14S2)

(DOOR-WALL2-L14S2)

(DOOR2-L14S2)

(DOOR-WALL2-L14S2 LOOKS-LIKE-RELATION RECTANGLE)

(DOOR2-L14S2 LOOKS-LIKE-RELATION RECTANGLE)

(DOOR-WALL2-L14S2 IN-SET DOOR-SET2-L14S2)

(DOOR2-L14S2 IN-SET DOOR-SET2-L14S2)

(DOOR2-L14S2 IN-FRONT-OF DOOR-WALL2-L14S2)

(DOOR-WALL2-L14S2 HAS-SIDE2 DOOR-WALL2-SIDE2-L14S2)

(DOOR-WALL2-SIDE2-L14S2)

(L14-SIMAGE2 CONTAINS-OBJECT DOOR-SETS-SET-L14S2)

(DOOR-SETS-SET-L14S2)

(DOOR-SETS-SET-L14S2 LOOKS-LIKE-RELATION SET)

The second `transformation` is `add-connections`. The effect of this `transformation` is to place the replicated door-sets in the correct spatial relationships with the other element instances. It takes `connection-sets-set-b1s3` as the `connection/connection-set` argument. This is a `set` containing four connections. Galatea uses a function to recursively retrieve all connection and set proposition members of this set. These propositions are put through a function which creates new propositions for the target. Each proposition's `relation` and `literals` are kept the same. The element instance names are changed to newly generated analogous names. For example, `door1-endpoint-b1s3` turns into `door1-endpoint-l14s3`.

Then, similarly to the `replicate` function, horizontal target maps are generated, and the other propositions from the previous `s-image` are instantiated in the new `s-image`.

The inputs to this transformation are `nothing` (denoting that there is not any thing in the previous `s-image` that is being modified), the connection set `connection-sets-set-b1s3`, the source `s-image lab-base1-simage2`, the current and next target `s-images l14-simage2 and l14-simage3`, the `mapping l14-simage2--l14-simage3--mapping1` and the memory.

The output propositions are

(DOOR-WALL2-ENDPOINT-L14S3--BOTTOM-WALL-STARTPOINT-L14S3--CONNECTION)

(DOOR-WALL2-ENDPOINT-L14S3--BOTTOM-WALL-STARTPOINT-L14S3--CONNECTION
HAS-DISTANCE SHORT-DISTANCE)

(DOOR-WALL2-ENDPOINT-L14S3--BOTTOM-WALL-STARTPOINT-L14S3--CONNECTION
HAS-ANGLE RIGHT-ANGLE-CW)

(DOOR-WALL2-ENDPOINT-L14S3--BOTTOM-WALL-STARTPOINT-L14S3--CONNECTION
IN-NON-VISUAL-SET DOOR-SET2-L14S3-CONNECTION-SET)

(BOTTOM-WALL-STARTPOINT-L14S3 HAS-CONNECTION
DOOR-WALL2-ENDPOINT-L14S3--BOTTOM-WALL-STARTPOINT-L14S3--CONNECTION)
(DOOR-WALL2-ENDPOINT-L14S3 HAS-CONNECTION
DOOR-WALL2-ENDPOINT-L14S3--BOTTOM-WALL-STARTPOINT-L14S3--CONNECTION)
(TOP-WALL-ENDPOINT-L14S3--DOOR-WALL2-STARTPOINT-L14S3--CONNECTION)
(TOP-WALL-ENDPOINT-L14S3--DOOR-WALL2-STARTPOINT-L14S3--CONNECTION
HAS-DISTANCE SHORT-DISTANCE)
(TOP-WALL-ENDPOINT-L14S3--DOOR-WALL2-STARTPOINT-L14S3--CONNECTION
HAS-ANGLE RIGHT-ANGLE-CCW)
(TOP-WALL-ENDPOINT-L14S3--DOOR-WALL2-STARTPOINT-L14S3--CONNECTION
IN-NON-VISUAL-SET DOOR-SET2-L14S3-CONNECTION-SET)
(DOOR-WALL2-STARTPOINT-L14S3 HAS-CONNECTION

TOP-WALL-ENDPOINT-L14S3--DOOR-WALL2-STARTPOINT-L14S3--CONNECTION)
(TOP-WALL-ENDPOINT-L14S3 HAS-CONNECTION
TOP-WALL-ENDPOINT-L14S3--DOOR-WALL2-STARTPOINT-L14S3--CONNECTION)
(DOOR-WALL1-ENDPOINT-L14S3--BOTTOM-WALL-STARTPOINT-L14S3--CONNECTION)
(DOOR-WALL1-ENDPOINT-L14S3--BOTTOM-WALL-STARTPOINT-L14S3--CONNECTION
HAS-DISTANCE SHORT-DISTANCE)
(DOOR-WALL1-ENDPOINT-L14S3--BOTTOM-WALL-STARTPOINT-L14S3--CONNECTION
HAS-ANGLE RIGHT-ANGLE-CCW)
(DOOR-WALL1-ENDPOINT-L14S3--BOTTOM-WALL-STARTPOINT-L14S3--CONNECTION
IN-NON-VISUAL-SET DOOR-SET1-L14S3-CONNECTION-SET)
(BOTTOM-WALL-STARTPOINT-L14S3 HAS-CONNECTION
DOOR-WALL1-ENDPOINT-L14S3--BOTTOM-WALL-STARTPOINT-L14S3--CONNECTION)
(DOOR-WALL1-ENDPOINT-L14S3 HAS-CONNECTION
DOOR-WALL1-ENDPOINT-L14S3--BOTTOM-WALL-STARTPOINT-L14S3--CONNECTION)
(TOP-WALL-ENDPOINT-L14S3--DOOR-WALL1-STARTPOINT-L14S3--CONNECTION)
(TOP-WALL-ENDPOINT-L14S3--DOOR-WALL1-STARTPOINT-L14S3--CONNECTION

HAS-DISTANCE SHORT-DISTANCE)

(TOP-WALL-ENDPOINT-L14S3--DOOR-WALL1-STARTPOINT-L14S3--CONNECTION

HAS-ANGLE RIGHT-ANGLE-CCW)

(TOP-WALL-ENDPOINT-L14S3--DOOR-WALL1-STARTPOINT-L14S3--CONNECTION

IN-NON-VISUAL-SET DOOR-SET1-L14S3-CONNECTION-SET)

(DOOR-WALL1-STARTPOINT-L14S3 HAS-CONNECTION

TOP-WALL-ENDPOINT-L14S3--DOOR-WALL1-STARTPOINT-L14S3--CONNECTION)

(TOP-WALL-ENDPOINT-L14S3 HAS-CONNECTION

TOP-WALL-ENDPOINT-L14S3--DOOR-WALL1-STARTPOINT-L14S3--CONNECTION)

(DOOR-SET2-L14S3-CONNECTION-SET)

(DOOR-SET2-L14S3-CONNECTION-SET LOOKS-LIKE-RELATION NON-VISUAL-SET)

(DOOR-SET2-L14S3-CONNECTION-SET IN-NON-VISUAL-SET

CONNECTION-SETS-SET-L14S3)

(DOOR-SET1-L14S3-CONNECTION-SET)

(DOOR-SET1-L14S3-CONNECTION-SET LOOKS-LIKE-RELATION NON-VISUAL-SET)

(DOOR-SET1-L14S3-CONNECTION-SET

IN-NON-VISUAL-SET CONNECTION-SETS-SET-L14S3)

(CONNECTION-SETS-SET-L14S3)

(CONNECTION-SETS-SET-L14S3 LOOKS-LIKE-RELATION NON-VISUAL-SET)

(L14-SIMAGE3 CONTAINS-OBJECT

CONNECTION-SETS-SET-L14S3)

(L14-SIMAGE3 CONTAINS-OBJECT

DOOR-SET1-L14S3-CONNECTION-SET)

(L14-SIMAGE3 CONTAINS-OBJECT

DOOR-SET2-L14S3-CONNECTION-SET)

(L14-SIMAGE3 CONTAINS-OBJECT

TOP-WALL-ENDPOINT-L14S3--DOOR-WALL1-STARTPOINT-L14S3--CONNECTION)

(L14-SIMAGE3 CONTAINS-OBJECT

DOOR-WALL1-ENDPOINT-L14S3--BOTTOM-WALL-STARTPOINT-L14S3--CONNECTION)

```
(L14-SIMAGE3 CONTAINS-OBJECT
TOP-WALL-ENDPOINT-L14S3--DOOR-WALL2-STARTPOINT-L14S3--CONNECTION)
(L14-SIMAGE3 CONTAINS-OBJECT
DOOR-WALL2-ENDPOINT-L14S3--BOTTOM-WALL-STARTPOINT-L14S3--CONNECTION)
```

We can now examine what made L14 (Figure 30) differ from the stimulus drawing: L14 features a longer vestibule in the drawing than the vestibule pictured in the stimulus. In fact, there is no trimmer arm (analogous to the wall in the lab problem) in the drawing at all that is distinct from the vestibule, save a very small section, apparently to keep the spinning trimmer blade from hitting the vestibule. The entire drawing is rotated ninety degrees from the source. The single lines in the source are changed to double lines in the target. The doors also slide in and out of the vestibule walls. What's interesting about this modification is that it does not appear that this kind of door opening is possible with the diagram given of the lab in the source: Since the door is a rectangle that is thicker than the lines representing the walls, the door could not fit into the walls. In contrast L14 explicitly makes the doors and walls thick (with two lines) and makes the doors somewhat thinner. L14 adds objects to the target not found in the source: a blade and a twisting mechanism to describe how the doors can work. L14 also included numerical parameters to describe the design of the trimmer, to describe length. Finally, L14 includes some mechanistic description of how the trimmer would work.

In summary, these differences are:

1. long vestibule
2. rotation
3. line to double line
4. sliding doors
5. added objects
6. numeric dimensions added

7. mechanisms added

Of these seven differences, Galatea successfully models four of them. The *rotation* of the source is modelled by a rotation in the target start `s-image`. In this `s-image`, all spatial relationships are defined only relative to other element instances in the `s-image`. Each instance is a part of a single set which has an orientation and direction. In the case of `s-image` 1 of the target, it is facing right. Since all locations are relative, there is no problem with transfer and each `s-image` in the model of L14 is rotated to the right.

The *line to double line* difference is accounted for by representing the vestibule walls with rectangles rather than with lines, as it is in the source. Because the `mapping` between the source and target correctly maps the `side1` of the rectangle to the `startpoint` of its analogous line, the rectangle/line difference does not adversely affect processing and transfer works smoothly.

The *long vestibule* difference is accounted for by specifying that the heights of the vestibule wall rectangles are `long`. In the source the vestibule wall lines are of length `medium`, but this does not interfere with transfer.

The trimmer head *added object* is accounted for by adding a circle to the first `s-image` in the target.

Unaccounted for are the two bent lines emerging from the vestibule on the left side, the numeric dimensions and words describing the mechanism. Also, L14 shows one of the doors retracting, and the model does not. The model also fails to capture the double line used to connect the door sections, because the single line is transferred without adaptation from the source. This could be fixed, perhaps, by representing the argument to the `add-component` as a function referring to whatever element is used to represent another wall, rather than as a `line`.

3.2 The Galatea Model of L22

L22 received Condition 2 (see Figure 32.) Figure 34 shows what L22 wrote on his or her data sheet during the experiment. See the top of Figure 33 for an abstract diagram of the source analog in the model.

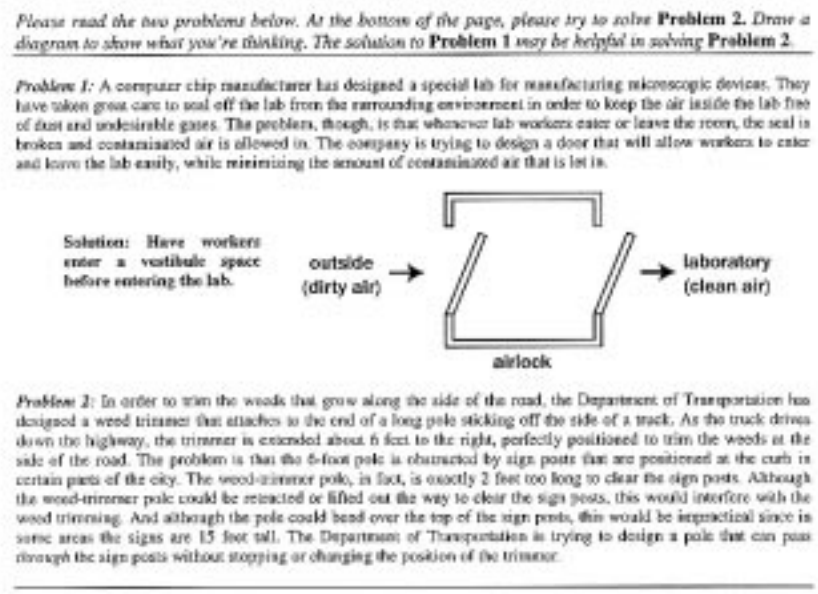

Figure 32: Condition 2: Plan view of lab, with no walls.

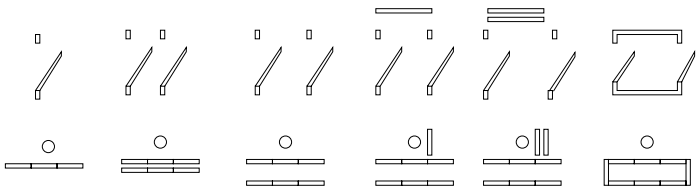

Figure 33: The implementation of L22. The top series of `s-images` represents the source analog (the lab problem) and the bottom series the target. There are six `s-images` for the five `transformations`.

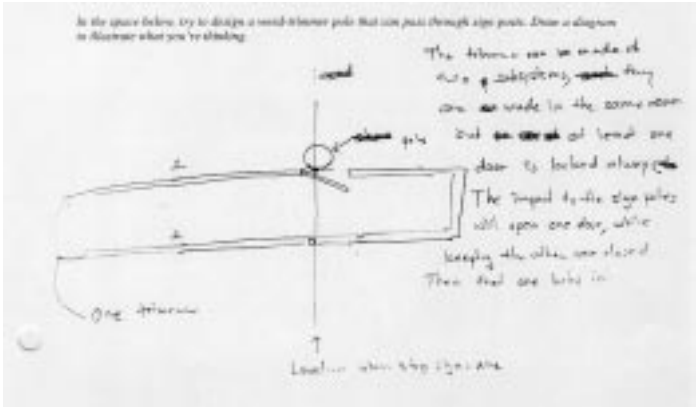

Figure 34: The source data for L22. The drawing above and handwritten text are what participant L22 inscribed on the experimental sheet.

The model of L22 involves five `transformations` (See Figure 33). The first `transformation` is `replicate`. It takes in the `door-set-l22s1` as an argument, generating `door-set1-l22s2` and `door-set2-l22s2` in the next `s-image`. Note that the door set replicated here is different from the door set replicated for L14. In this case, there are three connected rectangles, corresponding to the top wall, door, and bottom wall. In the case of L14, the door set is made of a single long rectangle (representing the wall) with another rectangle (representing the door) in front of it. But because `replicate` can work on any set of element instances, Galatea can accomodate the kind of doorway L22 had in mind.

The second `transformation` is `add-connections` which places the door sets in the correct position in relation to each other. Unlike for L14, there are no top and bottom walls.

The third and fourth `transformations` are `add-component`, which add the top and bottom containment walls.

The fifth `transformation`, another `add-connections`, places these containment walls in the correct positions in relation to the door sets.

The processing and adaptation of these `transformations` resembles the processing done with L14.

We can now examine what made L22 (Figure 34) differ from the stimulus drawing: The entire drawing is rotated ninety degrees from the source. An object is added to the target that has no analog in the source: the trimmer. L22 features a proportionately longer vestibule than in the source, and has some explicit simulation diagrammed. Of these differences, all but the last were modelled by changing the nature of the start `s-image` for L22.

Observed differences:

1. rotation
2. added objects
3. long vestibule
4. explicit simulation

L22 shows that Galatea's models of these participants work with different source as well as target analogs. The modelling of L15 and L16 were modelled similarly. For *all* of these models, no core processing code was changed. Only `transformations` were added to code. All differences I was able to accomodate I did by changing the input representation, not the code itself.

3.3 The Galatea Model of L15

As shown in Figure 35, L15 does not distinguish between the vestibule and the doors leading into it. The drawing is rotated, and the lines depicting the walls are turned into double lines. Added objects include: truck, pole, hinges, and the trimmer head.

Most interestingly, at the bottom is a set of states, like a film strip, describing a simulation of how the pole could move through the trimmer.

Observed differences:

1. rotation
2. line to double line
3. added objects
4. explicit simulation
5. no vestibule/doors distinction

Figure 35: The data for L15.

Implementation. My model uses the same source analog as L14. As seen in Figure 36, the rotation and added objects are accounted for by the input target. The no vestibule/doors distinction is accounted for by what is replicated. It does not account for the simulation, nor some of the details of the shape of the door mechanism (particularly the angle of the doors).

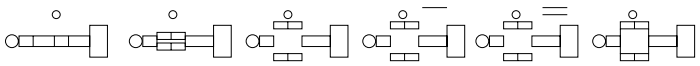

Figure 36: The model of L15.

3.4 The Galatea Model of L16

L16 (figure 37) features a rotated trimmer, and includes an arrow showing the direction of the motion of the truck. The pole is added, the lines are thickened to double lines, and the mechanism is described, including one door open and one shut.

Observed differences:

1. rotation
2. line to double line
3. added objects
4. mechanism added

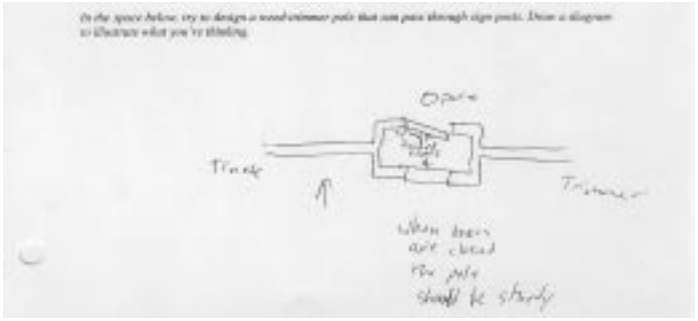

Figure 37: The drawing produced by participant L16.

Table 13: Differences accounted for in Galatea's participant modelling.

Participant	Differences Accounted For
L14	4/7
L22	3/4
L15	4/5
L16	3/4

Implementation. The door mechanism, which includes doubled lines in the initial target, gets replicated in the second `s-image`. As in the case of L14 and others, the results of the connection `transformations` result in single line transfers. This is because the `add-component` function takes the `line` literal as an argument. Thus when Galatea transfers it, it remains a `line`, even though the rest of the structure in the target is `rectangles`.

Figure 38: The implementation of L16

3.5 What the Implementation Shows

Above I described models of some of the visual aspects of four experimental participants. Specifically, I have modelled the visual input and output for this participant data–a good start to a full cognitive model. Though people likely use non-visual as well as visual knowledge in analogical problem solving, this work shows how visual knowledge alone *could* be used.

L14, L15, L16, and L22 are representative of some of the more difficult experimental participants to be modelled. They were given a source analog diagram and produced drawings describing their solutions. My models of them show how the analogical transfer could be done using only visual knowledge. The drawings produced by the participants differed from the stimulus diagrams in many ways, and in all four cases my models accounted for most of these differences, as seen in Table 13.

There are, however, some important differences between the inputs that the participants

and Galatea received. Participants were given a diagram representing the solution state of the lab problem, and a text description of the initial problem. Galatea, in contrast, takes in a series of knowledge states connected with transformations, the last of which is the diagram that participants see. Galatea also is given a mapping between the source and target, which, I assume, successful participants had to generate.

The model predicts that people *generate* the previous knowledge states in the source, generate a mapping, and then proceed to do transfer transformation-by-transformation. Future empirical work can test these predictions.

The previous section showed how the implementation of Galatea addressed the three hypotheses of this work. Modelling these four participants shows that the claims from the previous chapter are supported for the modelling of human cognition as well.

CHAPTER IV

COGNITIVE MODELLING: PART TWO

In the previous chapter I described the four experimental participants modelled with the computer program Galatea. This chapter offers an analysis of the drawings produced by the other successful participants in Dr. Craig's experiment. Though they managed to correctly transfer the problem solving solution to the weed-trimmer problem, each drawing was different from the source diagram in several ways. In this chapter I describe how they were different, and suggest reasons why these differences might have arisen.

These differences are interesting, because the weed-trimmer problem can be solved analogically without making any changes at all to the spatial structure of the laboratory problem's solution.

I will describe each participant in the lab/weed-trimmer problem who got the analogous solution, as well as the differences found between the source diagram and the drawing produced. In a later section I provide a classification of these differences and speculate on the psychological reasons why they might have occurred.

What am I calling a "difference?" I am assuming that the simplist diagram would be to copy the lab source drawing, labeling it, perhaps, with elements of the weed trimmer problem. Any deviations from this are adaptations to the lab source and require explanations.

I will also describe how Galatea, as is, could account for the data.

4.1 Complex Elements.

To better account for the data, rather than just using primitive elements, I will introduce the notion of `complex elements`. `Complex elements` are composed of two or more `visual elements` (primitive or complex). Like `primitive elements`, `complex elements` have attributes. Unlike `primitive elements`, many `complex elements` can be domain-specific[27], and differ from reasoner to reasoner.

Table 14: Suggested `complex elements` based on an analysis of the Craig data.

Complex element
Multiple-pass line
Dual line arrow
Semi-circle
arrowhead-isosoles-triangle
arrow-curve
arrow-line
fringe
two-line line
curly-cue
shading
dotted line
wiggly line
circle
dotted circle
dotted curve
curved fringe
box
curly braces
zig-zag
cylinder
box door closed
box door open
line door closed
line door open
triangle
text alphabet

The following is list of what the `complex elements` might be, based on my analysis of Craig's diagram data. Together with the `primitive elements` they account for all the components of these participants.

4.2 Modelled Participants

4.2.1 Participant L1 (condition 3)

Figure 39 shows L1's drawing. The most striking thing about this drawing is that it is rotated ninety degrees from the source.

Observed differences:

1. center

2. rotation

3. added objects

4. line to double line

5. mechanisms added

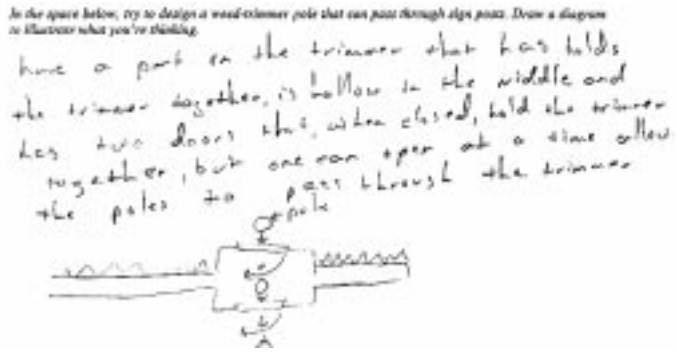

Figure 39: The drawing produced by participant L1.

Participants L1 and L2 were in Condition 3 (see Figure 41), L1's vestibule is centered on the weed-trimmer, even though the vestibule is off to the side in the source. Furthermore, L1 added objects: a pole, motion lines, and a zig-zag line representing, presumably, the blades. The trimmer arm is drawn thickly, with two drawn lines representing the two sides of it. In contrast the source depicts the analogous wall as a single line.

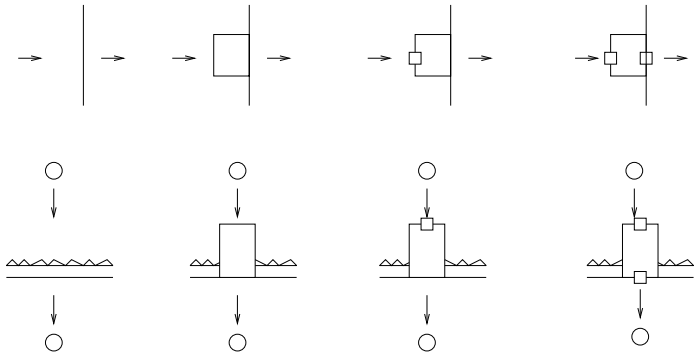

Figure 40: The model of the implementation of L1. Along the top are the source s-images, along the bottom the target. In this model the transformations are all *add-element.*

Implementation. The first target `s-image` in the model of L1, as seen in Figure 40, consists of a `double line` for the trimmer, a `zig-zag line` for the blade, `circles` for the pole, and `arrows` showing its motion. The input target is in a different orientation than the source. This change in input accounts for the rotation of the solution as drawn by L1. Galatea could not account for the vestibule being centered in the target.

Transformations:

1. *add-element:*

   object: square;

   location: adjacent to the wall line.

2. *add-element:*

   object: square;

   location: on top of the top part of the vestibule.

3. *add-element:*

   object: square;

   location: on top of the bottom part of the vestibule.

The transformations get transferred literally, and the model accounts for two of the five differences. Left behind are the added mechanisms (door swing arrows and the text description), the centering of the vestibule, and one of the added objects: the pole in the center of the vestibule. Galatea was able to account for the rotation, most of the added objects, and the line to double line differences.

4.2.2 Participant L2 (condition 3)

L2 (Figure 42), like L1, centered the vestibule. Interestingly, participants L1 and L2 are the only participants in Condition 3 who got the analogous solution, and *both* centered the vestibule.

Also like L1, L2 added some objects: An abstract sketch of a truck, and some lines presumably indicating weed-trimming blades.

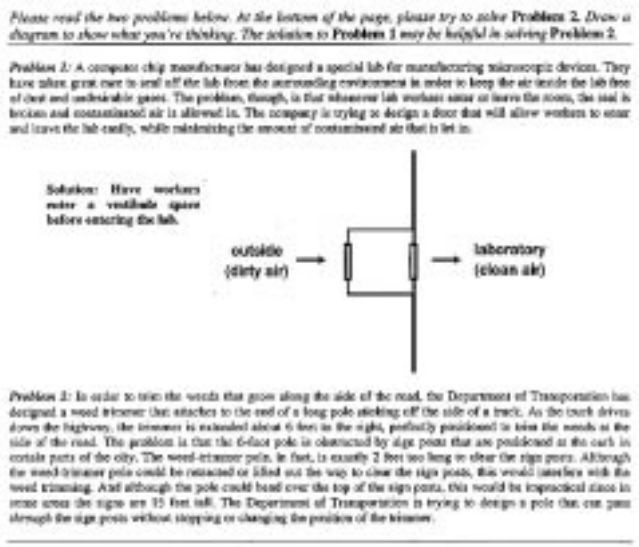

Figure 41: Condition 3: Plan view of lab, with the vestibule on the side.

Another difference L2 exhibits is that the doors and the walls of the vestibule are indistinguishable. In this design the walls themselves would have to move to open.

Observed differences:

1. center
2. no vestibule/doors distinction
3. added objects
4. line to double line

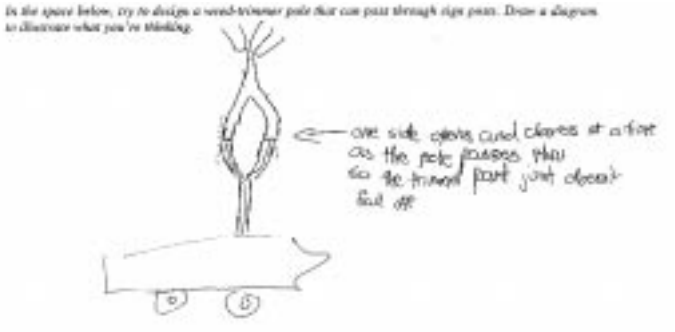

Figure 42: The drawing produced by participant L2.

Implementation. The model of L2 involves a different source than that of L1 (See Figure 43). Rather than adding a vestibule as a whole, the door section is a complex object that gets duplicated (L14, L15, L16, and L22 were modelled with Galatea using this strategy). The first source `s-image` appears in the diagram to be a single line, but it is actually in three sections: A top, bottom, and door section, the part that gets duplicated in the first transformation.

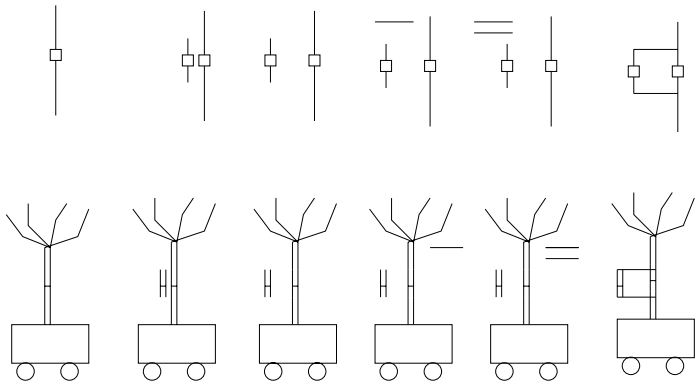

Figure 43: Model of L2. The source simages are along the top, the target are along the bottom.

Here is the justification for this method. If, as modeled in L1, a vestibule were added, more adaptations would need to occur: the vestibule is shaped differently, and then the "doors" (if you can say there are doors at all in the target,) look different as well. Rather than implementing these adaptations, by having them input at the start and duplicated, the model can account for more of the observed differences.

Transformations:

1. *replicate:*

   object: door section;

   number of resultants: 2;

2. *add-connections:*

   connection1: top of door1 to bottom part of top trimmer arm, angle: 90cw distance: short;

   connection2: top part of bottom trimmer arm to bottom of door1, angle: 90ccw,

distance: short;

3. *add-element:*

   type: line;

4. *add-element:*

   type: line;

5. *add-connections:*

   connection1: left part of line1 to top of door1

   connection2: right part of line1 to top of door2

   connection3: left part of line2 to bottom of door1

   connection4: right part of line2 to bottom of door2

The second transformation only connects the first door section because the other one is, by default, already in place. Replicated objects have the same locations as the original.

The model accounts for three out of the four observed differences. Only the centering of the vestibule was unaccounted for.

4.2.3 Participant L11 (condition 1)

L11 is the most straightforward of the solutions to the lab/weed-trimmer problem. The doors, represented in Condition 1 as rectangles, are simple lines in L11's drawing (Figure 44.) The other difference is that the doors are open and the doorways are missing–that is, the walls continue where there should be doorways. Other than that, the diagram is remarkably like that in the stimulus (see Figure 45).

Observed differences:

1. rectangle to line: door

2. doors open, walls remain

Implementation. My model of L11 is the simplest. The source and target analogs are identical, but still manage to account for most of the drawing that L11 did, though it accounts for none of the differences.

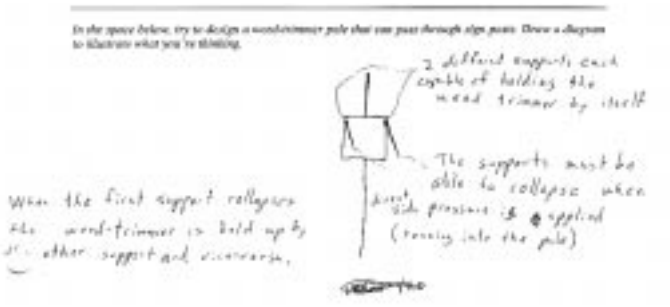

Figure 44: The drawing produced by participant L11.

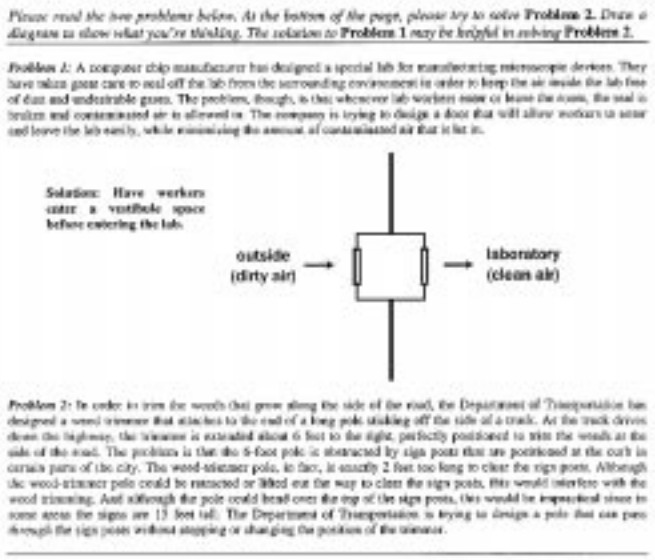

Figure 45: Condition 1: plan view with the vestibule centered.

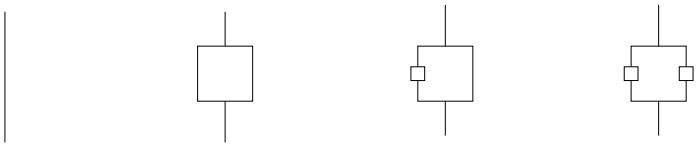

Figure 46: The model of L11. This `s-image` sequence describes both the source and target.

Transformations:

1. *add-element:*

   object: square

   location: centered on wall

2. *add-element:*

   object: small square

   location: one side of vestibule

3. *add-element:*

   object: small square

   location: other side of vestibule

The transformations are applied directly, with no adaptation. This model accounts for the orientation of the wall, the location of the vestibule on the wall, the location of the wall with respect to the vestibule, and the location of the doors. It fails to account for the doors being open, and that the doors are written as lines.

4.2.4 Participant L12 (condition 1)

In L12, we have a return to the rotation. Again, the truck is moving upward on the page. And like L11, the rectangular doors are drawn as lines. The truck, curb, blades, and road lines are also added. See Figure 47.

Observed differences:

1. rotation

2. rectangle to line: door

3. added objects

Implementation. See Figure 48 for a digram of the implementation of L12. The rotation and added objects can be accounted for with the input target problem.

Transformations:

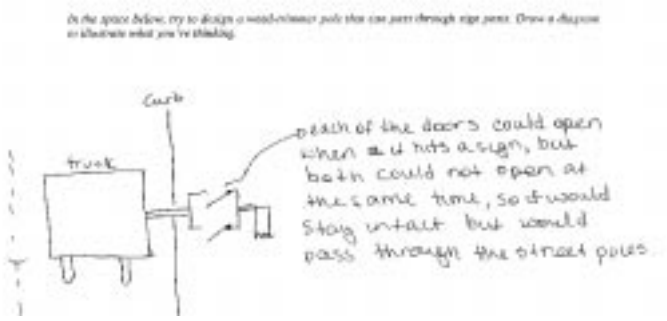

Figure 47: The drawing produced by participant L12.

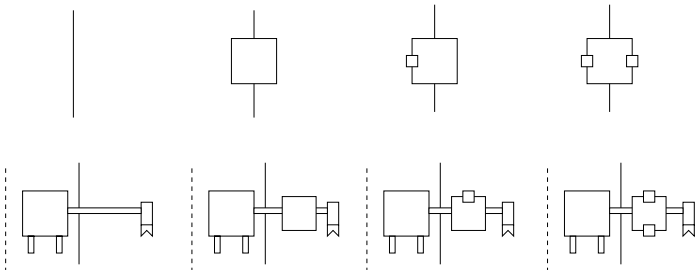

Figure 48: The model of L12. The top series of s-images are the source, the bottom are the target.

1. *add-element:*

   object: square

   location: centered on wall

2. *add-element:*

   object: small square

   location: one side of vestibule

3. *add-element:*

   object: small square

   location: other side of vestibule

Two of the three differences are accounted for. The open doors, drawn as lines, are not accounted for in the implementation.

4.2.5 Participant L13 (condition 1)

L13 (Figure 49) features rotation, and another instance of turning the rectangular doors into lines. On top of that, the door lines are dotted. The added objects for this participant are the truck, curb, road line, the trimmer, the sign, and the blades. Like L1 and L2 the trimmer arm is turned into a double line in a zoomed part of the diagram: There are really two diagrams, one of which is a magnified view of a piece of the broader-scoped diagram.

Observed differences:

1. rotation
2. rectangle to line: door
3. dotted object
4. added objects
5. zoom
6. line to double line

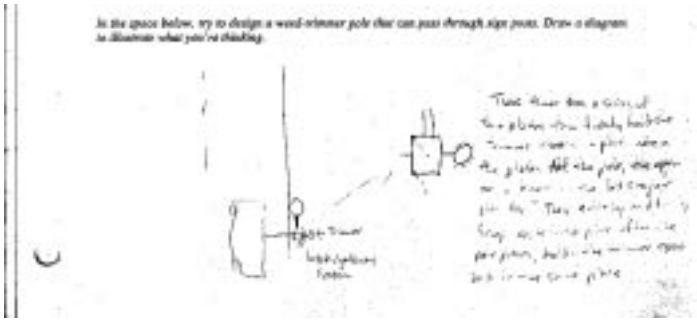

Figure 49: The drawing produced by participant L13.

Implementation. L13's model is identical to that of L12 except that the actual blade is a circle rather than a rectangle with a zig-zag line under it. See Figure 50. L13's case is somewhat more complicated than L12's, though, and as a result the model does not account for as much in the participant's drawing: The zoom in and out is ignored and treated as a single representation, the doors are squares and not dotted lines. The added objects and

orientation are accounted for by the nature of the target input. The model accounts for three out of six of the observed differences.

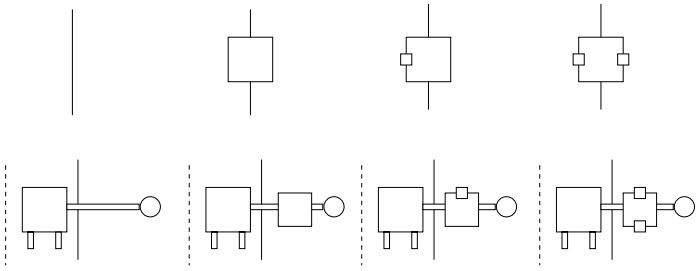

Figure 50: The model of L13.

Transformations:

1. *add-element:*

   object: square

   location: centered on wall

2. *add-element:*

   object: small square

   location: one side of vestibule

3. *add-element:*

   object: small square

   location: other side of vestibule

4.2.6 Participant L19 (condition 2)

L19 is in condition 2 (see Figure 51). It differs from conditions 1 and 2 in that the doors are shown to be open, the walls are double lines, and the wall in which the vestibule is embedded is not shown. It is the same condition L22 was in.

Rather than simply having two doors, L19 drew nine (see Figure 52). Also added were the blade, and hinges to describe the mechanism. The motion is explicitly simulated with an additional diagram below. The moving doors in this diagram are dotted, perhaps to show that their position is dynamic. As a reverse of what some of the earlier participants have done, L19 makes a move from double lines in the source to single lines in the target.

Observed differences:

1. multiple doors
2. added objects
3. mechanism added
4. double line to line
5. dotted object
6. explicit simualtion

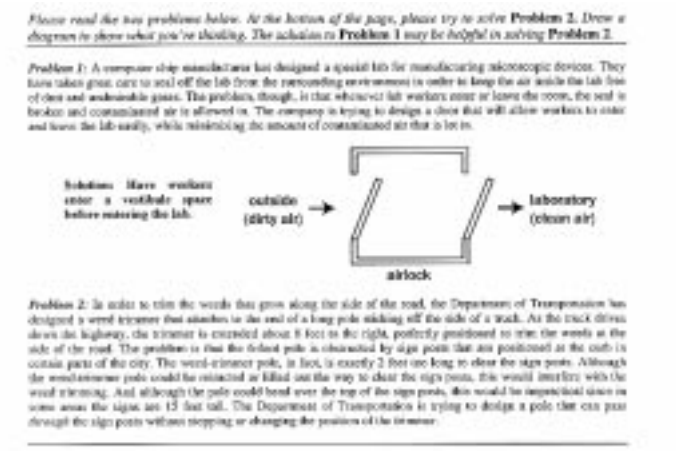

Figure 51: Condition 2: plan view with no walls aside from the vestibule's walls.

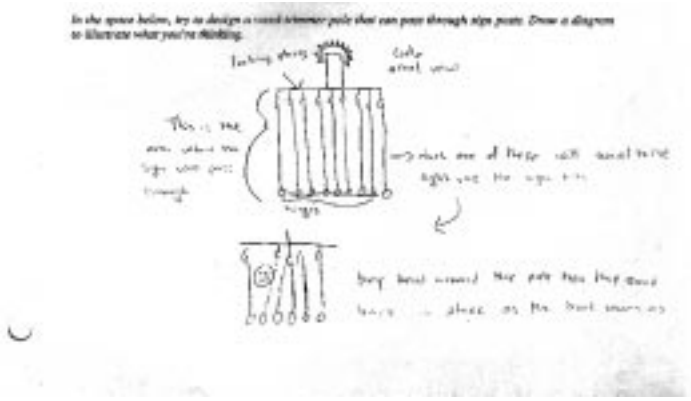

Figure 52: The drawing produced by L19.

Implementation. The source for L19 involves replication and connection, see the top of Figure 53, but involves some different primitives: The walls are represented with double lines rather than lines. L19 has a single door represented as a single line, which gets replicated in a manner simiar to previous participants. When the *connect* transformation gets transferred, though, a double-lined connector gets transferred rather than a single line, causing the same problem that the model for L14 has, but in reverse. See Figure 53. Since a double line is an aggregate object, it does not come built in with "ends as a *line* does. To be able to connect the ends of the double lines, the model imposes ends on the complex object.

The model does not account for are the other seven doors, the dotted lines, the simulation, and the mechanism in it. In sum it accounts for two out of the six observed differences.

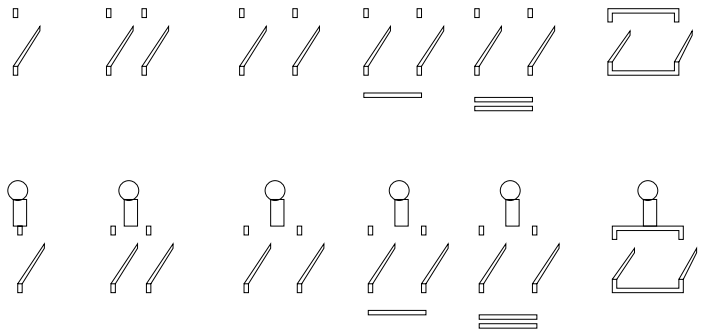

Figure 53: The model of L19.

Transformations:

1. *replicate:*

   object: door section;

   number of resultants: 2;

2. *add-connections:*

   connection1: top of door1 to bottom part of

   top trimmer arm, angle: 90cw distance: short;

   connection2: top of door2 to bottom part of

   top trimmer arm, angle: 90ccw distance: short;

3. *add-element:*

   type: line;

4. *add-element:*

   type: line;

5. *add-connections:*

   connection1: left part of line1 to top of door1

   connection2: right part of line1 to top of door2

   connection3: left part of line2 to bottom of door1

   connection4: right part of line2 to bottom of door2

4.2.7 Participant L20 (condition 2)

L20 (see Figure 54) drew four diagrams, showing a simulation of how the mechanism worked. The truck and blade were added, and the entire apparatus was rotated. Like L19 double lines were turned to lines.

Observed differences:

1. rotation

2. added objects

3. double line to line

4. explicit simulation

Implementation. See Figure 55. The door mechanism is replicated, resulting in `s-image` two. When the connections are made one by one, we get an inconsistency with the data: there are double lines where single lines are in L20's sketch. The input target state accounts for the rotation, added objects, and double line to line. The considerable simulation pictured in L20's sketch, as all examples of simulation in this domain, are unexplained by my model. In total, the model accounts for two out of four differences.

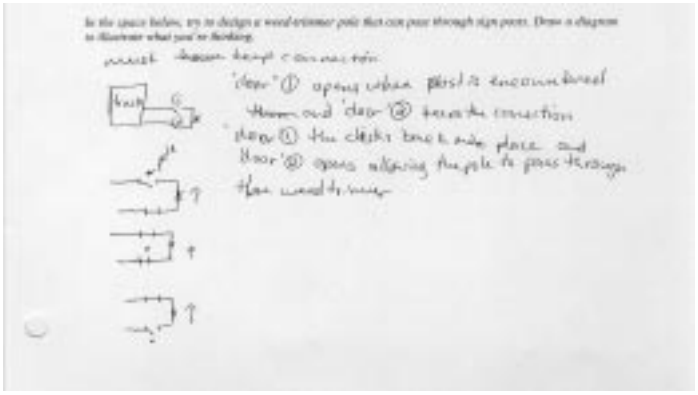

Figure 54: The drawing produced by participant L20.

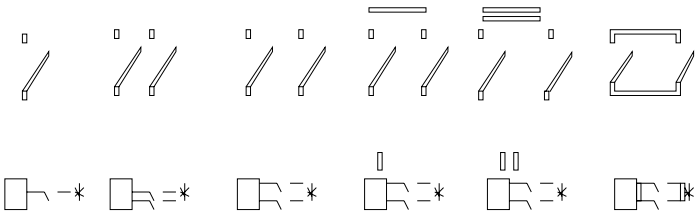

Figure 55: The model of L20.

4.2.8 Participant L21 (condition 2)

The truck dominates L21's diagram (Figure 56), which shows an elevation of the system, including the trimmer and the blade. It is difficult to show that Galatea would be able to model data presenting an elevation view. Perhaps this is the reason L21 continued to make a zoomed plan view, below, which looks more like the drawings of other participants. In this diagram the pole was added, and double lines are turned into lines.

Observed differences:

1. double line to line
2. added objects
3. zoom
4. point of view change
5. explicit simulation

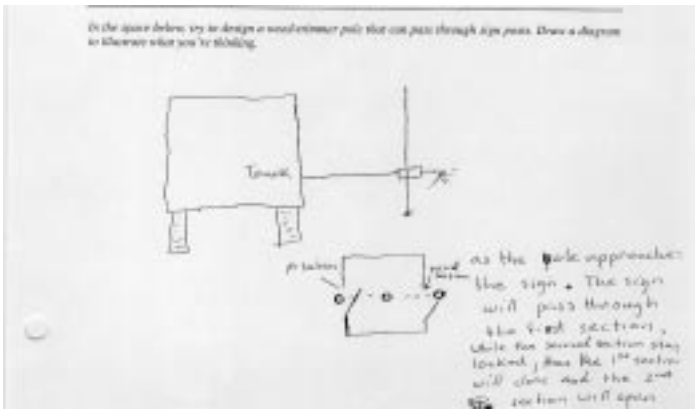

Figure 56: The drawing produced by participant L21.

Implementation. I only modelled the zoomed portion of the L21's drawing (Figure 57). Only the door section, the pole, at different locations, and the dotted line between the pole representations are in the target start state. The doorway is replicated, and the double line connector is addded. The single line connector in L20's sketch is not accounted for, nor is the zoomout, nor point of view change. The simulation is accounted for in that the markings on the paper are accounted for, even though the model has no sense of any simuation going on. That is, to the model, those simualtive diagram elements are equivalent to other diagram elements. The model is able to account for two out of five observed differences.

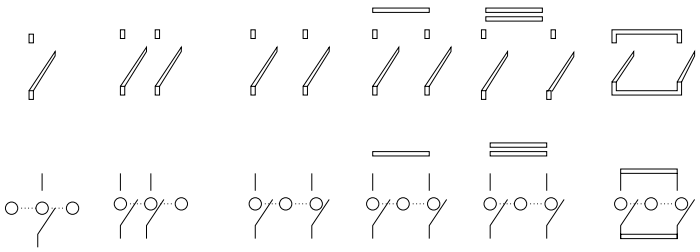

Figure 57: The model of L21.

Transformations:

1. *replicate:*

   object: door section;

   number of resultants: 2;

2. *add-connections:*

   connection1: top of door1 to top of door2, angle: 90cw distance: short;

   connection2: bottom of door1 to bottom of door2, angle: 90ccw distance: short;

3. *add-element:*

   type: line;

4. *add-element:*

   type: line;

5. *add-connections:*

   connection1: left part of line1 to top of door1

   connection2: right part of line1 to top of door2

   connection3: left part of line2 to bottom of door1

   connection4: right part of line2 to bottom of door2

4.2.9 Participant L24 (condition 2)

L24 shows a frame-by-frame simulation of the mechanism involved. The truck, blades, and pole are added (see Figure 58.) Double lines are turned to lines and the system is rotated.

Observed differences:

1. explicit simulation

2. added objects

3. double line to line

Implementation. The trimmer is modelled as a double line, connected to a rectangle representing the truck (See Figure 59). The door section is modelled as a single line, with two horizontal lines indicating the sides of the door. A single circle off to the left represents the pole. L24's four-part simulation is not modelled; the model most closely approximates part two of it. Missing is the explicit simulation, and, once again, the single line connector present in the drawing. The model accounts for two out of the three observed differences.

Transformations:

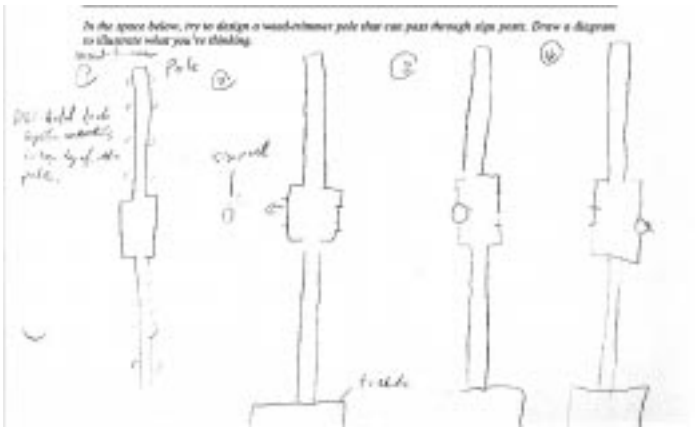

Figure 58: The drawing produced by participant L24.

1. *replicate:*

   object: door section;

   number of resultants: 2;

2. *add-connections:*

   connection1: top of door1 to bottom part of top trimmer arm, angle: 90cw distance: short;

   connection2: top part of bottom trimmer arm to bottom of door1, angle: 90ccw, distance: short;

3. *add-element:*

   type: line;

4. *add-element:*

   type: line;

5. *add-connections:*

   connection1: left part of line1 to top of door1

   connection2: right part of line1 to top of door2

   connection3: left part of line2 to bottom of door1

   connection4: right part of line2 to bottom of door2

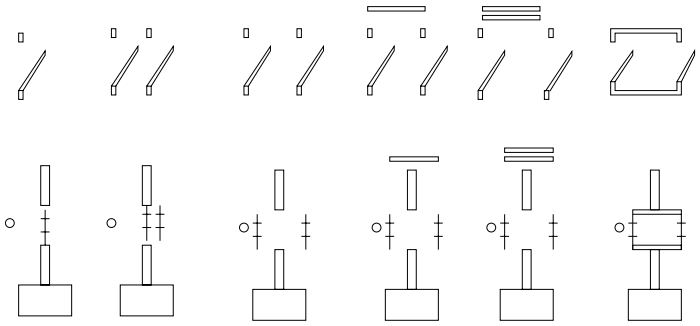

Figure 59: The model of L24.

4.2.10 Participant L27 (condition 4)

Like L21, L27 has a point of view change from an elevation to a plan, which might make more sense in this case because the source in this condition is also an elevation (see Figure 60.) The truck, blades, and motor are added to the rotated (see Figure 61.) This rotated and zoomed diagram also features numeric dimensions for the length of the trimmer.

Observed differences:

1. point of view change
2. zoom
3. added objects
4. rotation
5. numeric dimensions added

Implementation. L27 is an interesting case in that there are no connections. To account for this, the source was changed so that the walls that are connections in other models are present (as a floor and celing) in the first source state, rather than being added. Thus the only transformation is a single replication of the door.

This model accounts for the two doors in the drawing (Figure 62), their orientation, and the lack of connections. However, the model fails to account for a good many things in L27's drawing: The change in point of view, the zoom, the many objects and numeric dimensions

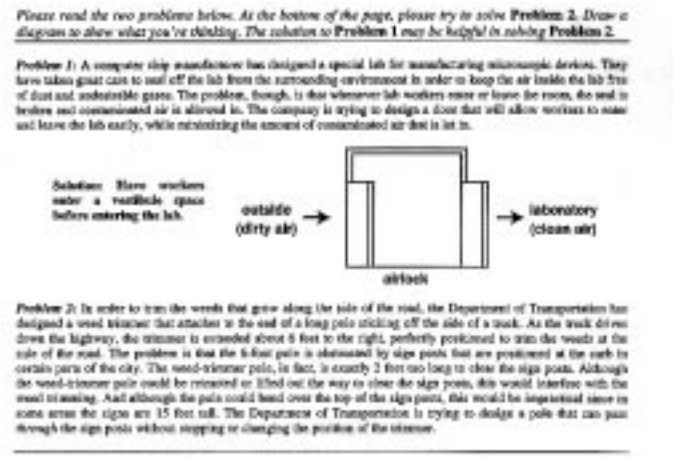

Figure 60: Condition 4: elevation view with no walls aside from the vestibule's walls.

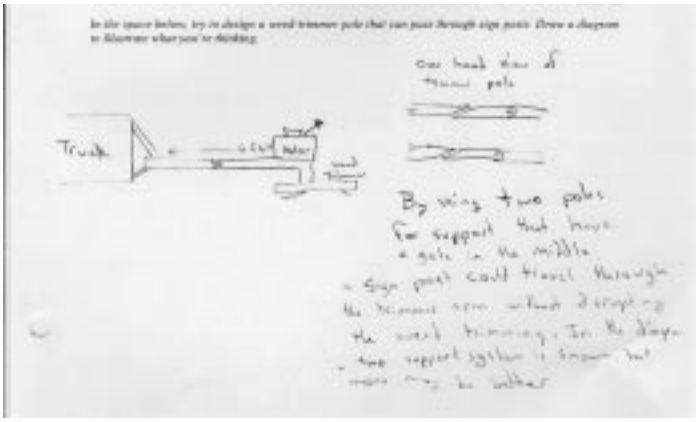

Figure 61: The drawing produced by participant L27.

in the zoom out, and the exact shape of the doors, including the hinge and diagonal line where the doors meet. In total two of the five differences can be accomodated.

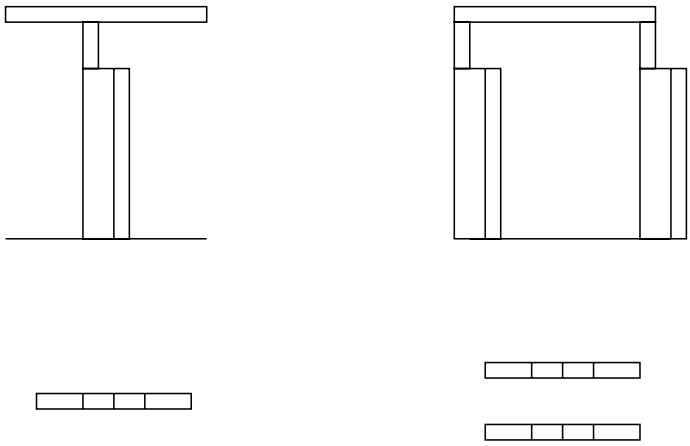

Figure 62: The model of L27.

Transformations:

1. *replicate:*

   object: door section;

   number of resultants: 2;

4.2.11 Participant L28 (condition 4)

L28 (Figure 63) changed the point of view to a plan, and had the whole vestibule open for the added pole. Again, there are four diagrams showing the simulation of the system.

Observed differences:

1. point of view change
2. no vestibule/doors distinction
3. added objects
4. explicit simulation

Implementation. As in the case of L27, L28 has a four part simulation sketch that I do not model. As shown in Figure 63, the model most resembles the first diagram in L28's

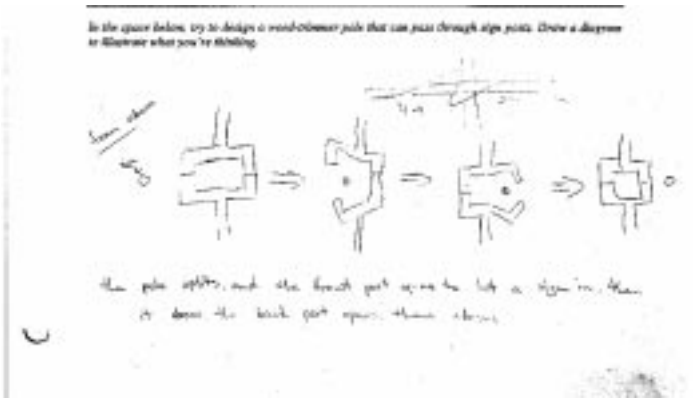

Figure 63: The drawing produced by participant L28.

sketch. The added pole object is accounted for in the first target s-image. The door is replicated and connected at the top and bottom with a double line and a single line, as in the source. This causes a slight inconsistency with the sketch: In the sketch all lines are double lines, and they smoothly connect. My model does not make a distinction between the nature of the connection. The no vestibule/door distinction behavior is accounted for by the source analog, which has no explicit representation of a vestibule. The point of view change is elegantly handled by the representation of the target in the input. Three of the four differences are accomodated.

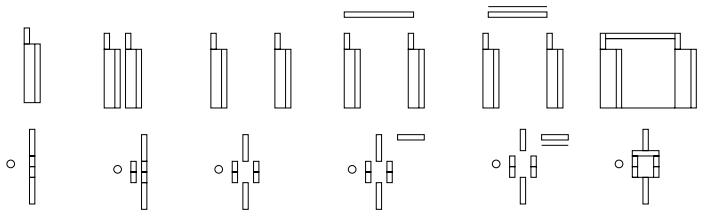

Figure 64: The model of L28.

Transformations:

1. *replicate:*

   object: door section;

   number of resultants: 2;

2. *add-connections:*

connection1: top of door1 to bottom part of top trimmer arm, angle: 90cw distance: short;

connection2: top part of bottom trimmer arm to bottom of door1, angle: 90ccw, distance: short;

3. *add-element:*

   type: rectangle;

4. *add-element:*

   type: line;

5. *add-connections:*

   connection1: left part of rectangle to top of door1

   connection2: right part of rectangle to top of door2

   connection3: left part of line to bottom of door1

   connection4: right part of line to bottom of door2

4.3 Differences

In the previous section I described how the successful participants changed their diagrams from what a straightforward copy from the source. For each participant, I listed the differences, with similar names for similar differences. This section explores this ontology of differences in more detail.

Table 15 shows the observed differences and which participants produced them.

I will describe each difference in detail. For each difference, I list possible *influences*, which are suggested causes for the difference. I will explore these influences in detail in the next section.

4.3.1 Added Objects.

4.3.1.1 Participants.

[L1, L2, L12, L13, L14, L15, L16, L19, L20, L21, L22, L24, L27, L28]

Table 15: Differences observed in each of the participants.

Difference	L1	L2	L11	L12	L13	L14	L15	L16	L19	L20	L21	L22	L24	L27	L28	total
added objects	X	X		X	X	X	X	X	X	X	X	X	X	X	X	13
center	X	X														2
doors open, walls remain			X													1
dotted object					X				X							2
double line to line							X		X	X	X		X			5
explicit simulation									X	X	X	X	X		X	6
line to double line						X	X	X								3
long vestibule						X						X				2
mechanism added						X		X	X							3
multiple doors									X							1
no vestibule/doors distinction		X					X								X	3
numeric dimensions added						X								X		2
point of view change											X			X	X	3
rectangle to line: door			X	X	X											3
rotation	X			X	X	X	X	X		X		X		X		9
sliding doors						X										1
zoom					X						X			X		3
total	3	2	2	3	5	6	4	3	6	4	5	4	4	5	4	

4.3.1.2 Description.

The addition of objects with no analogous elements presented in the laboratory problem diagram, such as blades, the truck, a road, lines in road, etc. perhaps to make the solution more realistic or embodied.

4.3.1.3 Influences.

1. demand characteristics: explication purposes
2. simulative concerns: completeness of mental model

One way this could happen is by the participant adding the objects to the *source* before adaptation. This explanation is not good for objects for which there are no analogs in the lab problem, such as the blades.

Another is that the source stays as it appears, but somehow through adaptation a `transformation` to add objects gets put into the target. Currently Galatea cannot model this.

A third way for this to happen is to simply add the objects to the target start state. All suggested models for this difference used this method.

4.3.2 Center.

4.3.2.1 Participants.

[L1, L2]

4.3.2.2 Description.

Vestibule is moved to center of line representing the long part of the trimmer.

4.3.2.3 Influences.

1. aesthetic concerns: centered on line
2. simulative concerns: completeness of working mental model (possibly in this case centering made it easier to imagine only one door at a time being open)
3. differences in input structure: vestibule explicit

All participants who got the analogous solution in the vestibule on the side condition (condition 3) centered the vestibule. Since there is no vestibule in the start target state, the centered-ness of the vestibule must be a result of a transformation. Either the location of the addition of the vestibule is adapted to be centered on the trimmer, or the participant modifies the source such that the vestibule is centered in it.

4.3.3 Doors Open, Walls Remain.

4.3.3.1 Participants.

[L11]

4.3.3.2 Description.

Participant drew open doors, yet the wall continues where the closed doors would be.

4.3.3.3 Influences.

1. Lazy drawing: object drawn whole

This difference implies that the vestibule is drawn whole, and that when the doors are added, the part of the vestibule they replace are not erased.

4.3.4 Dotted Object.

4.3.4.1 Participants.

[L13, L19]

4.3.4.2 Description.

The lines used to draw an object are dotted or dashed.

4.3.4.3 Influences.

1. prior knowledge: diagrammatic convention

The participant might retrieve the idea of a dotted line rather than a line when drawing it due to interference from the diagrammatic convention of drawing dotted lines for line-shaped objects of indeterminite location.

4.3.5 Double Line To Line.

4.3.5.1 Participants.

[L19, L20, L21, L24]

4.3.5.2 Description.

Double lines in source are drawn as single lines in target.

4.3.5.3 Influences.

1. lazy drawing: fewer lines

2. prior knowledge: diagrammatic convention

The participant may retrieve a single line rather than a double because of interference from diagrammatic convention. Another explanation could be that the single line is an easier-to-draw version of a double line, and in some effort-reward calculation, the participant finds it not worth it to draw the double line, and changes to the less energy-intensive single line.

4.3.6 Explicit Simulation.

4.3.6.1 Participants.

[L15, L19, L20, L21, L22, L24, L28]

4.3.6.2 Description.

Shows some diagrammatic description of how mechanism would work, e.g. steps, motion lines.

4.3.6.3 Influences.

1. demand characteristics: explication

2. engineering bias: mechanistic description

If the goal of the participant is not simply to draw the solution, but to show the experimenter how the system works, then the participant could determine that simply drawing

the final solution does not fulfill the goal. Drawing the states is a diagrammatic convention for showing how something works–this could be retrieved to help attain the goal. Also, simulation makes the solution appear more "real-world" and embodied.

4.3.7 Line To Double Line.

4.3.7.1 Participants.

[L14, L15, L16]

4.3.7.2 Description.

A line represented as single in source is represented as double in target.

4.3.7.3 Influences.

1. simulative concerns: completeness of mental model
2. prior knowledge: diagrammatic convention

The participant may be experiencing the influence of the diagrammatic convention bias as described in *double line to line*, but with a different diagrammatic convention.

4.3.8 Long Vestibule.

4.3.8.1 Participants.

[L14, L22]

4.3.8.2 Description.

The vestibule drawn is rectangular, and not square in shape.

4.3.8.3 Influences.

1. Simulation concerns: completeness of mental model

Upon evaluation of the transferred solution, the participant may realize that the sliding doors (in the case of L14) will not work unless the vestibule is long enough to accomodate them when open. This would cause the L14 to lengthen the vestibule.

In the case of L22, the vestibule needs to be longer because the trimmer needs to be long, and would fail on the evaluation phase because of that. The reason the "wall" part cannot be longer is because it is not present in condition 3 (see Figure 51.)

4.3.9 Mechanism Added.

4.3.9.1 Participants.

[L14, L16, L19]

4.3.9.2 Description.

Detailed description of how the weed trimmer contraption will work.

4.3.9.3 Influences.

1. demand characteristics: details
2. engineering bias: mechanistic description

These mechanisms could be added in the same way the simulation described above would be added.

4.3.10 Multiple Doors.

4.3.10.1 Participants.

[L19]

4.3.10.2 Description.

Participant draws more than two doors.

4.3.10.3 Influences.

1. simulative concerns: completeness of working mental model

Upon simulation, the participant probably didn't think two doors would be sufficient to hold the trimmer together. Failing on this count, L19 went back and added more doors, reasoning that since the existing doors gave support, more doors would mean more support.

4.3.11 No Vestibule/Doors Distinction.

4.3.11.1 Participants.

[L2, L15, L28]

4.3.11.2 Description.

The walls and doors are indistinguishable, and this object is what moves to allow the pole to enter.

4.3.11.3 Influences.

1. simulation: completeness of working mental model
2. differences in input structure: vestibule explicit
3. differences in input structure: addition of complex object

The vestibule/door solution in the source could be abstracted functionally into some symbol representing a redundant mechanism. When adapted to the target, it could get replaced with a different specification of that abstraction.

For this to happen the source needs to represent the vestibule explicitly, as its own symbol, as opposed to, say, an unaggregated set of lines.

4.3.12 Numeric Dimensions Added.

4.3.12.1 Participants.

[L14, L27]

4.3.12.2 Description.

Participant draws numbers indicating count, length, weight, etc.

4.3.12.3 Influences.

1. demand characteristics: details
2. entineering bias: quantitative description

This would work the same way as the simulation and mechanism additions described above.

4.3.13 Point Of View Change.

4.3.13.1 Participants.

[L21, L27, L28]

4.3.13.2 Description.

Drawn point of view is different from that of the source's. E.g. plan to elevation view shift.

4.3.13.3 Influences.

1. demand characteristics: explication
2. differences in input structure: target at wrong point of view
3. simulative concerns: possibly enhances imaginative simulation

For L27 and L28, the point of view change is necessary to explain, and possibly to think about the solution. The source or the target would be changed in point of view to be able to match the other.

For L21, the elevation is probably there for explanatory reasons. See the demand characteristics explanation for simulation.

4.3.14 Rectangle To Line: Door.

4.3.14.1 Participants.

[L11, L12, L13]

4.3.14.2 Description.

Doors in source are rectangles, in target are drawn as lines.

4.3.14.3 Influences.

1. demand characteristics: horizontal space
2. simulative concerns: kinesthetic metaphors

This would work the same way as *double line to line.*

4.3.15 Rotation.

4.3.15.1 Participants.

[L1, L12, L13, L14, L15, L16, L20, L22, L27]

4.3.15.2 Description.

Entire analog is rotated to a horizontal position.

4.3.15.3 Influences.

1. demand characteristics: horizontal space

2. simulative concerns: kinesthetic metaphors

Rotation is a change of orientation for a visual object. To change orientation, the object's orientation must be represented.

The difficult way to do this is to represent the absolute locations of the primitive objects that compose the complex object. The relation of the relations of these primitive objects to each other implicity represents the orientation of the complex object. Changing the orientation means a painstaking transformation of all the locations.

The simpler way to do this is to have an explicit representation of the primitive objects to one another (e.g. connected perpendicularly). These relations are rotation-invariant, so rotations will not require their updating. With this, though, an explicit notion of orientation is needed, which requires an ontology of directions, as Covlan does. Associating a part of a complex object to a direction gives directionality. To rotate something requies a bit more, however. Rotation requires the amount rotated (or and end position) as well as a center of rotation.

Thus the compact way to represent rotation is by relating the components of a complex object with rotation-invariant predicates, specifying a center and a "front," and noting which direction the "front" is facing. Galatea models rotation in this way. The initial target is rotated, and the transformation transfer works smoothly.

4.3.16 Sliding Doors.

4.3.16.1 Participants.

[L14]

4.3.16.2 Description.

Doors slide rather than swing.

4.3.16.3 Influences.

1. differences in input structure: sliding doors in source

2. simulative concerns: completeness of working mental model

Since, for L14, it is ambiguous in the source diagram whether the doors slide or swing, we can assume L14 imagined they swung to account for his swinging doors in the target.

Alternatively, L14 might have thought swinging doors would not work. Under simulation, L14 could have discovered this and retrieved a more fail-safe solution serving the same function: sliding doors.

4.3.17 Zoom.

4.3.17.1 Participants.

[L13, L21, L27]

4.3.17.2 Description.

At least two drawings are presented, one of which is a close-up of a part of the other.

4.3.17.3 Influences.

1. demand characteristics: explication

2. differences in input structure: target at wrong magnification

This would work for the same reasons as the simuation and mechanistic explanation differences.

4.4 The Influences

In the previous section I described the differences found in the data. With them I listed possible *influences*: my conjectures of psychological reasons why these differences might have appeared. In this section I describe the conjectured influences, which are classified with seven categories.

4.4.1 Aesthetic Concerns

An aesthetic influence is one where the change is made because the resulting spatial layout is preferred because it is more pleasing to the eye. Since it only accounts for a single difference, and this difference can be accounted for by other influences, this influence will not be further explored in this work.

4.4.1.1 centered on line

1. difference: vestibule centered on trimmer

4.4.2 Demand Characteristics

Demand characteristics are influences from the experimental setup. Some might be based on what the experimental participant's perception of the experimenter's expectation, or on the structure of the stimuli.

4.4.2.1 horizontal space

Participant thinks a horizontally-oriented picture is more appropriate for a horizontally-oriented space, since the space given on the experiment sheet is larger horizontally than it was vertically.

1. difference: rotation

4.4.2.2 explication purposes

Representations are not necessarily a part of problem solving, but are merely added to the diagram drawn to explain something to the experimenter.

1. difference: Zoom

2. difference: Added objects

3. difference: Explicit simulation

4. difference: Point of view change

4.4.2.3 Details

Participant thinks experimenter expects to see details with respect to the mechanisms involved and the dimensions, possibly because the data were taken at a technical university.

1. difference: Numeric dimensions added

2. difference: Mechanisms added

4.4.3 Differences in Input Structure

Differences in input structure group those changes that occur as a result of differences in the representational content of the source case or target problem.

4.4.3.1 Explicitly represented vestibule

This means that the vestibule is represented as a complex object, rather than implied by the existence of more primitive objects (e.g. a set of unaggregated lines.)

1. difference: vestibule centered on line

2. difference: no door/vestibule distinction

4.4.3.2 Addition of Complex Object

The whole vestibule object must be a unit for it to get adapted to something different with the same function.

1. difference: no door/vestibule distinction

4.4.3.3 Target at Wrong Magnification

Participant starts with one zoom level, similar to lab scale. But because scale is in appropriate for transfer, must generate zoomed in version (see L13).

1. difference: Zoom

4.4.3.4 Target at Wrong Point of View

The analog might be represented in a point of view in which the solution is not visible (because, perhaps, the important mechanisms are occulded).

1. difference: Point of view change

4.4.3.5 Sliding doors in source

If the source is represented as having a sliding door, then it makes sense that the participant would represent sliding doors in the target.

1. difference: Sliding doors

4.4.4 Engineering Bias

4.4.4.1 Quantitative Description

As opposed to the demand characteristics, things like numeric dimensions may be added because of some internal motivation as a result of the participant's engineering training.

1. difference: numeric dimensions added

4.4.4.2 Mechanistic Description

The participant may add simulation or mechanism descriptions because that is the participant's idea of a proper solution, due to their training at a technical school.

1. difference: mechanisms added

2. difference: explicit simulation

4.4.5 Lazy Drawing

4.4.5.1 Object drawn whole at first needs missing parts later in drawing

This is evidence of discreteness of transformations: adding vestibule, then doors.

1. difference: doors open, walls remain

4.4.5.2 Fewer Lines

Due to some cost/benefit calculation, more lines may be deemed superflous.

1. difference: Double line to line

4.4.6 Prior Knowledge

4.4.6.1 Diagrammatic convention

We all make diagrams, and certain conventions exist for them in our culture. These biases may affect how things get conceptualized or drawn in the target.

1. difference: rectangle to line: door
2. difference: dotted object. (Either because opening and closing doors looks like this (non-visual explanation) or because moving objects look like this (visual explanation).)
3. difference: line to double line
4. difference: double line to line

4.4.7 Simulative Concerns

4.4.7.1 Visual simulation

Participant thinks lines are better for rotating on a hinge than rectangles

1. difference: rectangle to line: door

4.4.7.2 Kinesthetic Metaphors

Ten participants rotated their diagrams. It might be that the horizontally oriented space they were given in which to draw might have been a factor. However this is questionable because of the consistency with which the participants described the direction of the weed trimmer. All them showed the weed trimmer as going up the page (or, similarly, the pole going down the page, either by showing the pole explicitly or through the direction of the door openings), save L14, who showed no direction. This means that not a single participant who rotated made the weed trimmer going down the page. This suggests that there is something about "going up" that is priviledged for this problem situation.

When watching a truck drive away, it appears to go up toward the horizon. When moving forward, you are heading from where to are to a point higher up in your visual plane. These kinethetic metaphors may account for this consistency [48].

1. difference: rotated (people prefer to think of weed-trimmer as moving up).

4.4.7.3 Completeness of Working Mental Model

The participant may be attempting to construct a mental model, and in doing so, might add diagrammatic elements that reflect objects added to that model with the function of making that model more complete, or to make it internally "run" more properly.

1. difference: adding objects (Non-visual. Airflow is represented as arrows in the source. In the target, though, the entire apparatus moves. The truck is the causal source of the motion. Perhaps that is why it is represented. [L12])
2. difference: Centered Vestibule
3. difference: no door/vestibule distinction
4. difference: line to double line (doors need to slide into them)
5. difference: sliding doors (better chance of actually working.)
6. difference: multiple doors
7. difference: long vestibule [L14](doors need to slide into them)
8. difference: long vestibule [L22] (since in the (plan, no walls) condition there are no walls, the Ss need to make the vestibule longer for it to make sense.)

4.5 Summary Of Models

The preceding subsections described "pen-and-paper" models of the remaining eleven participants who got the analogous answers in Dr. Craig's experiment. Table 16 shows the proportions of the differences the models accomodated. Over all, the models accounted for about half (49%) of the differences.

Table 16: Differences accounted for in pen-and-paper models.

Participant	Accomodated Differences	Percentage
L1	2/5	40%
L2	3/4	75%
L11	0/2	0%
L12	2/3	67%
L13	3/6	50%
L19	2/6	33%
L20	2/4	50%
L21	2/5	40%
L24	2/3	67%
L27	2/5	40%
L28	3/4	75%
Total	**23/47**	**49%**

I have every reason to think that these models could be implemented in Galatea without changing Galatea's core code, or, indeed, even adding any `transformations`. Only the `complex elements` would need to be added. Though these participants were not modelled computationally, these models suggest how Galatea could model them without significant changes to the code. Thus these pen-and-paper models show some support for Constructive Adaptive Visual Analogy.

The second section of this chapter described the differences that appeared in more detail, and the third section speculated on the cognitive processes that might have gone on to produce these differences.

This chapter as a whole contributes a detailed analysis of visual analogy in human beings, and presents a sketch of what kinds of psychological factors might be involved in the production of the data. I suggest that some form of these differences and infuences should be a part of any complete cognitive model of these data.

CHAPTER V

PSYCHOLOGIAL EXPERIMENTATION

As Constructive Adaptive Visual Analogy is a cognitive theory, I tested the theory with a psychological experiment. In the previous chapters I described Galatea and the models created with it. The focus of Galatea is on the transfer subtask of analogy. Implicit in this formulation is the idea that there is difficulty in analogical problem solving above and beyond the difficulty associated with mapping. I tested this idea in the experiment as well as the third of my main hypotheses: that visual knowledge facilitates transfer even when non-visual knowledge might be available.

In this experiment, participants are given Gick and Holyoak's classic tumor problem to solve, using the fortress problem as an analogy. Experimental participants read a story about a general who must overthrow a dictator in a fortress. His army is poised to attack along one of many roads leading to the fortress when the general finds that the roads are mined such that large groups passing will set them off. To solve the problem, the general breaks the army into smaller groups, and they take different roads simultaneously, arriving together at the fortress. Participants are then given a tumor problem, in which a tumor must be destroyed with a ray of radiation, but the ray will destroy healthy tissue on the way in, killing the patient. The analogous solution (which in this document I will call the "correct" solution) is to have several weaker rays simultaneously converging on the tumor [33, 15].

Much of the analogical problem solving research with the fortress/tumor problem assumes that the difficult parts of analogy are retrieval and mapping. Studies of this sort manipulate retrieval hints, manipulate changes in the fortress story, use completely different source stories, manipulate the timing of the source story [33], force participants to make comparisons, or change instructions. Analogy involves many tasks; these experiments sometimes distinguish between the retrieval stage and later ones, but not between, for example,

mapping and transfer. Novick and Holyoak [60] however found that for math word problems only around 40% of participants (50% in one experiment, 32% in the next) were able to find the analogous solution even when the mapping was given as a part of the stimuli. This suggests that the mapping stage is not the only difficult analogical subtask.

This work hypothesizes that transfer of strongly-ordered procedures is computationally complex, even given the correct mapping. To get an idea of how difficult analogical problem solving is above and beyond and mapping, this experiment manipulated whether or not the participants were *given* the mapping between the source and target. If `mapping` is the only/major source of difficulty in analogical reasoning, then experimental participants given the correct mapping in a cross-domain analogical problem-solving task should have little difficulty successfully transferring the solution. The experiment investigates whether this is the case for cross-domain analogical problem solving.

Diagrams have been shown to help in analogical problem solving in general (e.g. [4]), but not specifically with analogical transfer. The main hypothesis of this experiment is that visual knowledge facilitates transfer even when non-visual knowledge might be available.

5.1 Method

5.1.1 Participants.

Eighty undergraduate students received extra class credit in exchange for taking part in the experiment. They were randomly assigned to one of the six experimental groups.

5.1.2 Design.

Each participant read a description of the fortress problem and how it was solved: "A small country fell under the iron rule of a dictator. The dictator ruled the country from a strong fortress. The fortress was situated in the middle of the country, surrounded by farms and villages. Many roads radiated outward from the fortress like spokes on a wheel. A great general arose who raised a large army at the border and vowed to capture the fortress. His troops were poised at the head of one of the roads leading to the fortress, ready to attack. However, a spy brought the general a disturbing report. The ruthless dictator had planted mines on each of the roads. The mines were set so that small bodies of men could pass over

them safely, since the dictator would then destroy many villages in retaliation. A full-scale direct attack on the fortress therefore appeared impossible."

Participants in `diagram` conditions (groups A and D) were given a diagram (see Figure 65) with the following text: "Here is an abstract diagram that describes the problem the general faced, and what he did to solve it. The arrows represent the groups of soldiers marching on roads to the fortress in the center."

Participants in the `draw` condition (group C) were asked to "Please draw a diagram or diagrams that describes the problem the general faced (NOT the solution–we will ask for a drawing of that later.) Please make it abstract. So please don't draw realistic drawings of the fortress, for example."

Then all participants read the solution to the fortress problem: "The general, however, was undaunted. He divided his army up into small groups and dispatched each group to the head of a different road. When all was ready he gave the signal, and each group charged down a different road. All of the small groups passed safely over the mines, and the army then attacked the fortress in full strength. In this way, the general was able to capture the fortress and overthrow the dictator."

This text is from Gick and Holyoak [33].

Figure 65: The experimental fortress story diagram used in the diagram conditions (groups A and D.)

Participants in the `draw` condition (group C) were then asked to "Please draw an abstract diagram or diagrams that describes the general's solution to this problem."

All participants looked at the tumor problem: "Suppose you are a doctor faced with a patient who has a malignant tumor in his stomach. It is impossible to operate on the

patient, but unless the tumor is destroyed the patient will die. There is a kind of ray that can be used to destroy the tumor. If the rays reach the tumor all at once at a sufficiently high intensity, the tumor will be destroyed. At lower intensities the rays are harmless to healthy tissue, but they will not affect the tumor either. What type of procedure might be used to destroy the tumor with the rays, and at the same time avoid destroying the healthy tissue?"

Participants in the `draw` condition (group C) were then asked to "Please draw a diagram that describes the above problem (NOT the solution–we will ask for a drawing of that later.) Again, please make it abstract. So please don't draw realistic drawings of a tumor, for example."

Participants in the `mapping` conditions (groups A, B, C, and E) read "These problems are analogous. In these stories, the tumor is like the fortress, and the ray of radiation is like the big army that wants to march. The expolding mines are like the patient's body getting hurt by radiation."

Participants in the `diagram` conditions (groups A and D) were then shown a diagram of the tumor problem, shown in Figure 66.

Participants in all groups read "How would you solve the tumor problem? What type of procedures might be used to destroy the tumor with the rays, and at the same time avoid destroying the healthy tissue? *Use the fortress story as an analogy to help you solve the tumor problem.* Give as many possible solutions as you can think of. This is a difficult problem that requires creativity to solve–you may need to work at it."

Participants in the `draw solution` conditions (groups A, B, C, and D) were then asked to "Please draw diagrams to accompany your written solutions."

Table 17 shows each group (A through F) in this design. The table further shows the number of participants in each group, whether that group gave the participants the mapping, whether diagrams were given, whether they asked to draw diagrams, and whether they were asked to draw solutions, as described above.

The specific hypotheses for this experiment are: First, there will be no large effect of mapping. Second, there will be a positive effect of being in the `diagram` condition. Third,

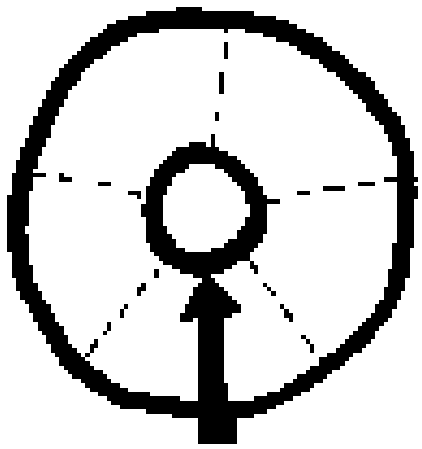

Figure 66: The experimental diagram of the tumor problem used in the diagram condition (groups A and D.)

there will be a positive effect of being in the `draw-solution` condition. The `draw` condition does not have a hypothesis associated with it because participants in it tended to draw the solution rather than the problem. This condition was discontinued halfway through the experimentation process.

5.1.3 Procedure.

Participants signed a consent form, and were given a sheet of paper with the stimuli (described in the previous section) printed on it. They were asked to take their time and to follow the instructions on the sheet. No participant took more than 30 minutes to complete the experiment. After they finished, they were asked if they had ever heard of the fortress/tumor problems before. They were then debriefed and shown out.

5.1.4 Analysis and Scoring.

A given participant was classified as getting the correct answer if any of his or her descriptions of the tumor solution (drawn and written) described 1) multiple rays, 2) weaker rays, and 3) coming in from multiple directions. Those missing any one of these three criteria

Table 17: Experimental results by group.

Group ID	N	Mapping	Diagram	Draw	Draw-Solution	Correct	%
A	16	x	x		x	15/16	94%
B	14	x			x	14/14	100%
C	15	x		x	x	12/15	80%
D	12		x		x	12/12	100%
E	10	x				7/10	70%
F	11					10/11	91%

Table 18: Experimental results by condition. The only significant difference found was for those with and without the `draw solution` manipulation.

Condition	with	without
Mapping	87% (48/55)	96% (22/23)
Diagram	96% (27/28)	86% (43/50)
Draw Solution	93% (53/57)	81% (17/21)

were classified as having gotten an incorrect answer.

5.2 Results

The results are shown in tables 17 and 18. Two participants were excluded from the analysis because they reported having encountered the fortress/tumor problem before.

It is difficult to see the pure effects the conditions by looking at the results tables because it is not a between-subjects design. That is, most participants participated in multiple conditions. The statistical results reported are from methods that control for co-variation, allowing for *statistical* control such as an ANCOVA, or Analysis of Covariance, and regression. These methods use statistical control of conditions when experimental control is impossible. So, for example, when calculating the correlation between `mapping` and `correct`, for example, it is a *partial* correlation that is meaningful; it is measured controlling for the variables associated with the other factors.

The first goal of this experiment is to investigate the effect of mapping for a cross-domain

analogical problem-solving task. This experiment showed no effect of mapping. Controlling for the diagram and draw-solution conditions, the partial correlation between mapping and correctness is negative: -.171 The probability that there was an effect of mapping is insignificant (p=0.144). Even if this result were significant, it is in the wrong direction. That is, those given the mapping fared (insignificantly) worse than those without. The 95% confidence interval for the effect of mapping on correctness is -.296 to .044.[1] Because the interval crosses zero, it is statistically indistinguishable from zero. A regression of mapping on correctness is also shown to be insignificant: r squared (.010) $F(1,61)$=.625, p=.432.

The mapping groups had 87% correct; the non-mapping groups had 96% correct. Because I am relying on a null result, it is important to have enough power to detect a true difference if there is one.

This experiment has to power to detect a medium-sized effect (.31). Thus the positive effect of mapping cannot be *more* than .31. Because 50% or greater is considered a large effect, we are 95% confident that there is no large effect, casting doubt on the overwhelming importance placed on mapping.

Another hypothesis is that the `diagram` condition will help. Groups with diagrams (A and D) have 96% correct (n=28) while those without diagrams have 86% correct. On the face of it it looks like it should be significant. But the result is confounded with draw solution (all subjects in the `diagram` condition also have the `draw solution` condition). Controlling for `draw solution` and `mapping` leaves the partial correlation between `diagram` and correct at .101, and not distinguishable from zero (p=.390). A regression of `diagram` on correct is insignificant when it is the only variable in the equation $F(1, 76)$=2.124, p=.149 and remains an insignificant contributor to the model after `mapping` is added (p=.219) and remains insignificant after `draw solution` is added to the equation (p=.876). Though the difference is insignificant, the results are in the predicted direction: Those shown diagrams fared (insignificantly) better than those not shown diagrams.

[1]This means that if you performed this experiment 100 times, the true population mean would fall between these 95% of the time.

The second hypothesis is that drawing the solution helps participants get the correct solution. Controlling for `mapping` and `diagram`, the partial correlation between `draw solution` and correctness is significant (.180, p=.024); and the 95% confidence interval is .034 to .479. 93% of the people in the `draw-solution` conditions got it correct. For those not asked to draw the solution the percent correct is 81%. Even controlling for `mapping` and `diagram`, this difference is significant. Not only does it appear that the `draw solution` condition improves performance, but because the confidence intervals do not overlap, the effect of `draw solution` is significantly greater than the effect of `mapping`.

5.3 Experimental Conclusions

In conclusion, this experiment has two results: the participants given the mapping did not perform better than those who were not given it, and those asked to draw their solution to the tumor problem outperformed those were not asked to draw it, supporting the claims that there is difficulty in analogical problem solving above and beyond the difficulty associated with mapping and that visual knowledge facilitates transfer even when non-visual knowledge might be available.

Researchers have found other manipulations to this task that have facilitated the participants' finding the analogical soultion. Catrambone and Holyoak [8] facilitated transfer by 1) specifically asking participants to compare the analogs and 2) manipulating the wording in the stimuli such that the solution-relevant information was more salient.

The hypotheses come directly from this work's main three hypotheses. I found that though groups given diagrams did not benefit, those asked to draw their solutions did, partially supporting the notion that visual knowledge facilitates transfer even when non-visual knowledge might be available.

In terms of visual stimuli, animations have been found to be helpful [63]. Gick and Holyoak [34] used diagrams similar to the ones I used to facilitate tranfer, but did not find an effect. A follow up study by Beveridge and Parkins [4] found an effect using diagrams with translucent ray representations where the cumulative effect can be perceptually identified. The similarity of my stimulus to those of Gick and Holyoak could account for why my study

did not find an effect of `diagram`. It may also be that perhaps it is the act of *creation* of the visual representation that helps more than a given diagram because the act of creation is more likely to be associated with the correct things in memory. Further investigation is needed to fully understand this discrepancy.

CHAPTER VI

FURTHER THEORY: VISUAL RE-REPRESENTATION

This work has shown that visual knowledge can be useful in analogical problem solving. This chapter provides theoretical conjectures for the Constructive Adaptive Visual Analogy theory explaining *why* this might be so. As I will describe in more detail in the conclusion, the theory in this chapter will be the subject of my future investigations.

This dissertation is based on the general idea that visual representations provide a level of abstraction at which two otherwise dissimilar domains may appear more alike.

Galatea's design opts for some higher-level visual abstractions when possible. I could have, for example, used a complex hypothetical shape, say `s1`, to accurately describe the shape of a fortress, and a different different shape, say `s14`, to accurately represent the shape of a tumor. In that case, the tumor query might retrieve only other similar-looking tumors, ignoring even different-looking tumors, let alone fortresses. That is, a more detailed and more accurate visual representation would make analogical remindings, mappings and transfer harder.

There are many theories that also resolve ontological mismatches by finding similarities at a higher level of abstraction. For example, in conceptual dependency theory [66], verbs are categorized into ACTs, which are abstractions of actions. Bhatta and Goel's Generic Teleological Mechanisms [6] cover different instantiations of mechanisms that perform the same function. Falkenhainer's Minimal Ascension rule[18] uses a generalization hierarchy to determine the distance between concepts. This dissertation suggests that visual abstraction, too, is a useful mechanism for analogical problem solving.

A major issue, then, is under what conditions are visual analogies useful? I conjecture that one of the ways visual representations are useful is in resolving *ontological mismatches* in analogical problem solving. For example, imagine a reasoner needs to find similarity between a door and a television set. A functional description would not find this similarity.

Table 19: Knowledge states, entities, and manipulations.

	Knowledge State	**Entity**	**Manipulation**
non-visual	nv-state	object	action
Covlan	s-image	element	transformation

The symbols representing the objects and their descriptions are different. However, they have a visual similarity: they can both be seen as rectangles from one point of view. This is a simple example of how a visual abstraction can find similarity when there is an ontological mismatch. This is something humans routinely do in problem solving that AI theories of analogy need to accomodate. Ontological mismatches encountered in non-visual representations are resolved by providing a level of visual abstraction at which two different symbols are similar.

Problems and solution procedures can be represented non-visually and visually. In the non-visual representation, the knowledge states are called *nv-states*, and the manipulations are called *actions*. Both actions and nv-states can be transformed into visual representations. This is done using Covlan, as described in previous chapters.[1]

Unsolved target problems are represented as single nv-states or s-images.

6.1 Resolving Ontological Mismatches

Analogy is one among many ways to find a problem solution. For example, if an identical problem has been encountered before, that solution might be retrieved directly. In Figure 67, analogy is a *method* for a problem solving *task*. Analogy consists of several steps: *retrieval* of a candidate source analog in memory; *mapping* the components of the analogs; *transfer* of knowledge from source to target; *evaluation;* and *storage* of the target in memory, perhaps to be used as a source analog later. My proposal involves changing the representations of

[1]The terms "entity," "element," "object," "manipulation," "action," and "transformation" are used merely to differentiate the visual, non-visual, and super-ordinate counterparts of things and operators. Their common sense meanings in English do not have everything to do with which term gets used with which meaning.

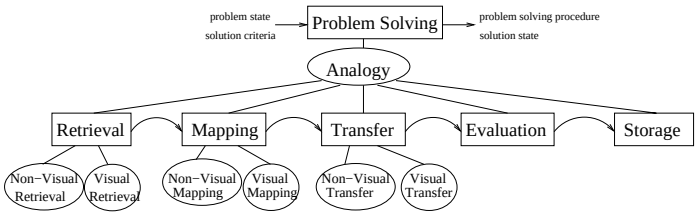

Figure 67: This Figure shows a high level description of my the CAVA theory of the role of visual reasoning in analogical problem solving. Straight horizontal arrows represent input (arrows entering a box) and output (arrows exiting a box.) Boxes represent complex actions to be taken by the agent. Curved arrows represent an ordering relation. A series of boxes connected with curved lines represent a series of ordered subtasks of the higher task, connected with a vertical line. The order is from left to right. Boxes below a task that are unconnected to each other are not subtasks but alternative method for achieving the task in the box above it. Realistically, there is looping in the analogical process; this will be detailed in the model.

the analogs, which is often a useful process to prepare the analogs for one of the above core steps. I will describe steps of analogy and how visual knowledge ebables it.

6.1.1 Retrieval and Mapping.

Visual information can be used to retrieve memories. Since visual cueing can occur, it is reasonable to think that some memories are encoded in terms of perceptual information. It has even recently been hypothesized that *all* memories are encoded in terms of perceptual information [2].

In analogical problem solving, a visual representation of the target could be used to retrieve visual representations of potential sources. An analog can be represented in terms of process, causality, uses, etc.; the visual representation is one of many possible forms of representation. The mind will use these connected, different representations for retrieval when relevant.

Visual representations can be generated to aid in retrieval. However, in trying to understand the situation the reasoner might generate a visual abstraction to represent it. For example, the demands might be represented as converging red lines on a circle, representing the reasoner, perhaps triggering other memories regarding convergence.

Mapping two analogs involves aligning their elements. Since retrieval is based on matching, to some extent, it is reasonable to suppose that the processing done to retrieve an analog

could be used to guide mapping [44, 19].

Since I am dealing with problem solving, and the analogical transfer of problem solving solutions, retrieval queries are based on the initial problem state and the solution constraints. See Figure 68.

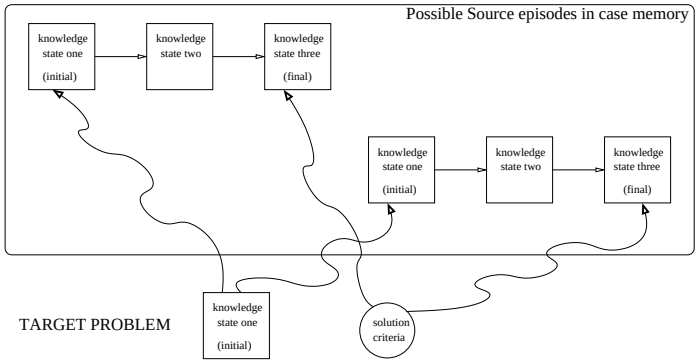

Figure 68: This Figure shows a target analog problem and how it could relate to the potential analogs in the memory. The items in the case memory are represented here as a series of knowledge states (represented by boxes) connected with manipulations, which are changes to the knowledge state (represented by straight arrows pointing right.) The last box in a series is the solution state, and the arrows, in order, represent the solution procedure. The target only has a single knowledge state because there is no solution procedure yet. There is also no solution state, but rather a set of criteria that must be fulfilled. The initial target knowledge state is compared to the initial states of the cases in memory for similarity (shown as wiggly arrows coming from the target knowledge state). Also, the solution criteria is compared to the solution states in memory to see if they fulfill the criteria (shown as wiggly arrows coming from the target solution criteria). Based on these measures cases could be retrieved.

Mapping visual analogs may differ from mapping non-visual analogs in two ways. First, visual analogy could use visual knowledge, such as an abstraction hierarchy (e.g. a square is a kind of rectangle). Such knowledge could be used to find matches based on similarity. Second, the reasoner's perceptual system could be brought to bear to inform the mapping. Seeing a truck may have something to do with matching its parts to the parts of some episode of truck experience in memory. Likewise, one might help guide mapping by using perception on generated mental images [47].

6.1.2 Transfer and Adaptation.

As in some other theories of analogical problem solving, I hypothesize that problem solving strategies are transferred. Other theories of *visual* analogy do not do this.

As shown in Figure 68, the possible source analogs are represented as solution procedures of connected knowledge states. In analogical problem solving, transfer is applying a source analogy's solution strategy to a target. This could be done with both visual and non-visual representations.

Transfer works as follows: In the mapping stage, a mapping is found between the source and target initial problem states. The manipulation that connects the first to the next knowledge state in the source is transferred to the target. The parts of the target that the manipulation affects are those analogous parts of what get affected in the source.

When this is done with visual representations, transfer of manipulations can work because they are sufficiently abstract such that they can apply equally well to many different `visual primitives`. For example, a manipulation that `moves` something can apply to `lines` as well as `circles`. This means that the same `move` manipulation that worked in the source with a `circle` could work with the `line` in the target.

This process can repeat unhindered for the entire sequence, transferring the manipulation from the source and generating new knowledge states in the target. Sometimes, however, there can be problems with ontological mismatches. For example, in the fortress/tumor example, imagine that in the reasoner's mind the army is decomposed using a `break-up` action that will not work on the ray because the ray does not have constituent parts.

Visual representations can be used as an intermediate level of abstraction to do plan *adaptation*. To follow the example, imagine the advancing army gets visually instantiated as a `line`, and the ray does as well. The manipulation, too, gets visually instantiated as the `decompose` visual `transformation`, which applies fairly broadly to visual elements. In the generated visual representation, the transfer of the manipulation occurs smoothly, as `decompose` can apply equally well to both lines. I call this visual representation an intermediate step because it must be turned back into the non-visual again, because, I assume, "solving" the problem in the visual abstraction is really not informative of actions

that must be taken in the real world (more on this in the next section). When the visual `transformation` is turned into a manipulation for the ray in the non-visual representation, it becomes a different manipulation: `disperse-energy`. Because `transformation`s can specify into different actions, strategies can be adapted to new instances, and steps in the problem solving process do not need to be transferred literally.

6.1.3 Solution Evaluation.

In my theory the target problem, at the start, has only a single knowledge state. The final "goal state" is not represented at the start. I am dealing with insight problems, for which, I assume, a clear picture of the goal state is often all one needs to solve the problem. Most of the work is in finding out what the goal state is, as opposed to how to get there. Rather, the "solution" is represented in terms of criteria that are used to determine whether the problem is solved.

For many problems the reasoner cannot tell if the problem is solved by examining the uninterpreted visual representation at the level of abstraction I've been discussing. An agent needs to turn it back into a non-visual representation and run a *simulation* to determine the effectiveness of the manipulations made. For example, moving the weaker rays and pointing them toward the tumor cannot be identified as an adequate solution to the problem unless the agent understands that the result of this would be that the tumor is destroyed while leaving the healthy tissue unharmed but notions of "harm" are not a part of the visual representation. Then the goal criteria can be applied. Once the knowledge state is non-visual again, its workings need to be simulated to be able to test it against the goal criteria. The reasoner maintains correspondences between elements of the knowledge states throughout the transfer process, and is able to use that information to return the final solution state back into a non-visual representation. Simulation in this sense means predicting the behavior of a system given the knowledge of how it works.

Thus to simulate, the agent needs causal knowledge. Neither this causal knowledge nor the goal criteria can be represented with only visual information, as causality consists of more than visual relationships between things.

By causal I mean knowledge of how things in a system change as they interact. Pre- and post-conditions are a straightforward way to represent this, but it is difficult to imagine what "visual" pre- and post-conditions might look like. For the reasons above I suggest that visual representations alone cannot enable evaluation of the solution.

6.1.4 Solution Storage.

Newly created knowledge state series are stored just like source analogs so they can be used as such in the future.

6.2 Inference and Control

The theme of this chapter is that visual information will prove to be useful in resolving ontological mismatches. In this section I will specify this theme further: Visual information will help resolve two different kinds of ontological mismatches in analogical problem solving. First, it can help with the generation of mappings. Take, for instance, where the reasoner needs to find an alignment between the army and the ray of radiation. For the sake of simplicity, let's say that, in the non-visual representation, the army is represented with the token `army` and the radiation is represented with the token `ray`. The tokens are not identical, and the system cannot align them. The situation is depicted in Figure 69.

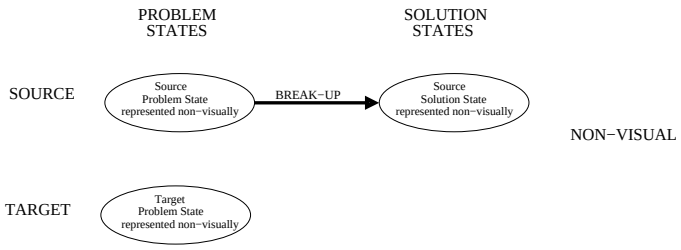

Figure 69: The ovals along the top represent the solved problem in memory. For the sake of simplicity, imagine that the solution to the problem involves only a single `transformation`. The top left oval represents the start nv-state. The action `break-up`, splits the army up into smaller groups, resulting in many smaller armies. Every oval in this figure represents a non-visually represented `knowledge state`. The problem state on the bottom is the tumor problem.

Now imagine the reasoner has knowledge that both the `ray` and the `army` share the visual abstraction `line`, because a `ray` is shaped like a `line`, and the `army`'s motion can

be abstracted into a `line`-shaped path. The reasoner generates a visual representation of both analogs and finds that there is a `line` in each, and makes the alignment as a result of this found similarity. This alignment is brought back to the non-visual representation and the analogical problem solving process continues in the non-visual representation.

The second use I predict is in the visual abstraction of actions. Let's say that to solve the problem the reasoner needs to break up the `army` into smaller armies, and the action it uses to do this is `break-up`, which takes a set of things as an argument and outputs smaller groups. Further, suppose `break-up` works by finding the constituent parts of the idea, and dividing them into `n` groups. If the `ray` of radiation is represented such that it does not have constituent parts, then the action `break-up` can not be applied to it. You can see the state of the reasoner's knowledge at this point in Figure 70.

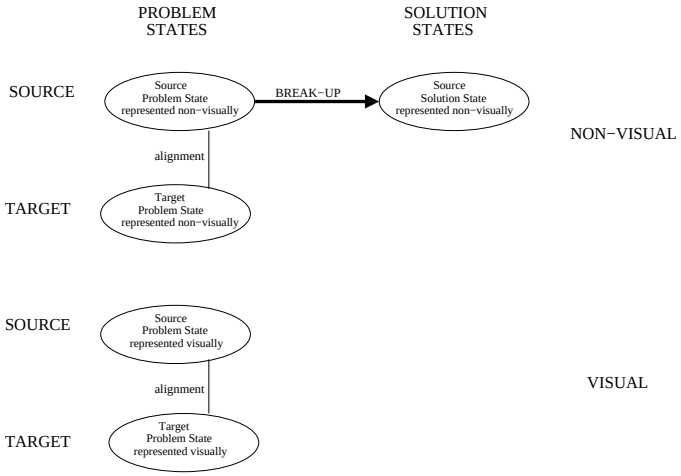

Figure 70: The bottom two ovals are the problem states of the target and source, represented in terms of their visual information. As a result of their visual similarity, the reasoner can find an alignment, or mapping. This allows the reasoner to hypothesize the analogous alignment in the non-visual representation. From here, the reasoner can attempt the transfer process in the non-visual representation. It will fail because the action `break-up` cannot be applied to the `ray` of radiation.

The reasoner can visually instantiate the `transformation` into one that can be applied to `line`. Let's suppose it visually instantiates into the visual `decompose transformation`, which will turn a `line` into several thinner `line`. This visual `transformation` can be applied to both source and target `s-images`, because they both contain `lines`.

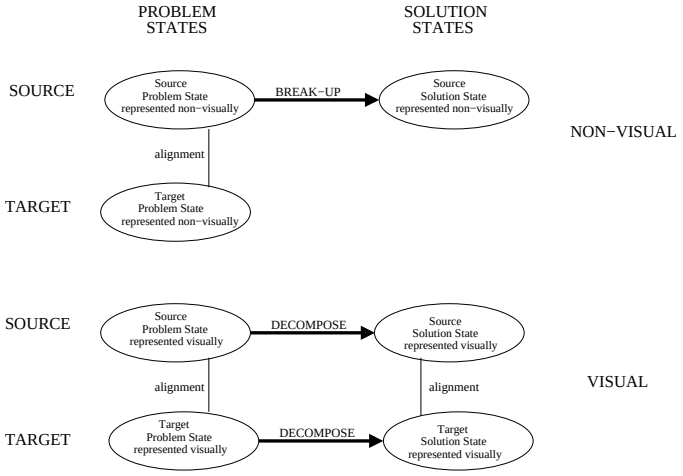

Figure 71: To resolve this transfer of strategy problem, the reasoner continues fleshing out the visual representation, visually instantiating the `break-up` action into the visual `decompose transformation`. Since the `ray` and the soldier path are abstracted as `line`, the decompose function can be transferred from the source to the target in the visual. The visual solution state can be generated. But solving the problem in a visual abstraction does not mean solving it in the more realistic non-visual representation. But if `decompose` specifies to `break-up`, then the visual representation does no good, because we could have just substituted `break-up` in the non-visual to begin with.

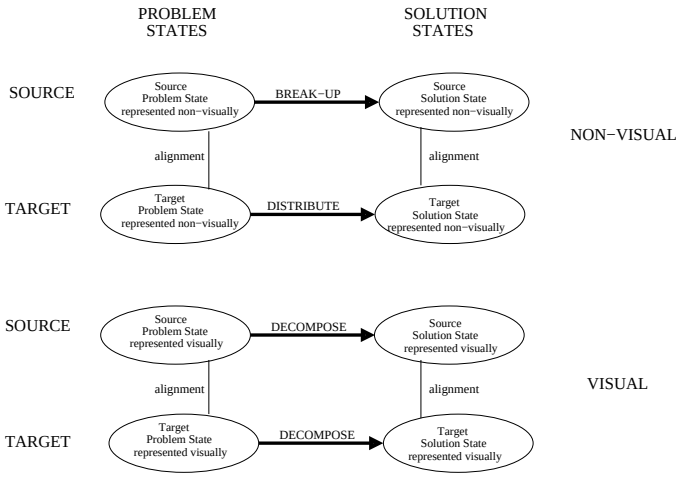

Figure 72: The `decompose` function specifies to more than one non-visual `transformation`. The reasoner chooses the correct specification based on the kind of object it is modifying. In this case, it's energy, so it chooses the more appropriate `distribute transformation`. It decomposes the `ray` correctly. The target solution state represented non-visually is generated and evaluated. The problem is solved.

Next comes `specification`, which is translating the `transformation` back into an action in the non-visual representation, which I assume is a more realistic representation. That is, if you can't translate it back from the visual abstraction, then you won't really know what to do to the radiation, only a `line` representing it.[2]

But if you can trivially translate back and forth, then there is no need for the visual abstraction. What makes the visual `transformation` useful is that it does not specify back into `break-up`. The `decompose transformation` specifies into `break-up` when dealing with, perhaps, entities with constituent parts, but when dealing with something like energy, whose intensity might be represented by a number, it specifies into a different action. Let's call this new `transformation distribute`, which divides an intensity level by some number m and allocates the intensity to several sources of energy. This final nv-state is depicted in Figure 72.

In summary, according to visual re-representation theory the visual abstractions can provide an intermediate representation through which otherwise dissimilar entities and manipulations can be aligned.

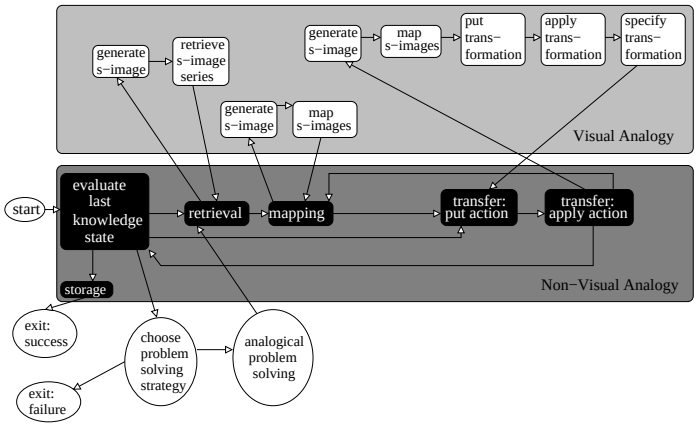

Figure 73: Flow diagram demonstrating control in visual re-representation theory.

Here is the main algorithm. It takes as input at least the following items: A memory of

[2]It may well be that you cannot understand how to solve a problem in a physical system without some perceptual representation of it. The visual representations I'm dealing with in this dissertation are more abstract than a full-blown, pictorial image. You can imagine how to decompose a ray of radiation quite realistically, and solve a problem with this in mind, but the visual level I'm talking about is more akin to a sketched diagram than an instructional video. They are so abstract that they are often ambiguous as to what they represent (e.g. a circle representing a person).

potential source series, success conditions, and a series consisting only of a single problem nv-state (the target problem). Words in bold represent functions that will be described in more detail later.

1. **Evaluate.** Run **evaluate**, where the knowledge-state argument of evaluate is the last nv-state currently in the target problem, and the input success conditions are the specification conditions. If the goal conditions are met, exit, and the problem is solved. If not, set the target nv-state to current-target-knowledge-state, then go on.

2. **Choose problem solving strategy.** Choose a problem solving strategy from those that have not failed for this target problem. If all have failed, exit and fail. If analogy is chosen then go on.

3. **Retrieve.** This is the first step of *non-visual analogical problem solving*. Run **retrieve**, where the single argument is the input target nv-state. If retrieve fails, mark analogy as having failed for this problem and return to 2 and choose another strategy.

4. **Find mapping.** Run **find-mapping** with the following arguments: the current-target-knowledge-state and current-source-knowledge-state. There may be an input suggestion from a visual mapping that was found. If find-mapping fails to find a mapping, and visual knowledge state abstraction has not failed, go on. If it succeeds, Go to 8 to try to transfer the actions. If it fails because there are no more states to evaluate, or because visual knowledge state abstraction has failed, go to step 3 and try another retrieval.

5. **Generate new s-images.** This is the first step of *knowledge state visual instantiation*. There are multiple ways to visualize something. Each time this step is reached, search through the space of visual instantiations for the source and target, creating new visual instantiations by running **generate-s-image** where its argument is the current target knowledge state. It will generate the current target `s-image`. Then run **generate-s-image** where its argument is the current-source-knowledge-state. It

will generate the current-source-s-image. If there are no more new visual instantiations to be made, mark knowledge state visual instantiation as having failed for this mapping and go to 4.

6. **Find visual mapping.** Map the s-images by running **find-mapping** with the following arguments: The current target `s-image` and the current-source-s-image. Upon failure, go to 5 and try to get a different visual instantiation to try. Upon success, go on.

7. **Transfer mapping.** Run **transfer-mapping** with the following arguments: the new visual mapping and its `s-images`, and their corresponding `nv-states`. It will generate a suggested mapping for the `nv-states`. Use this suggestion, going back to 4, the mapping stage. This is the last step in knowledge state visual instantiation.

8. **Transfer actions.** Run **transfer-manipulation**, attempting to transfer the current source nv-state's action to the target. Go on to try to apply it.

9. **Apply action.** Run the associated action on the current target `nv-state`. If it works, generating a new nv-state, go to 1 and evaluate. If it does not work, go on to try manipulation visual instantiation at 10. If visual instantiation has been marked as failed for this mapping, go back and try another mapping at 4.

10. **Generate s-images.** Generate new `s-images` as in step 5, if 1) there are no `s-images` for the current `nv-states`, or 2) the current `s-images` have been marked as having failed.

11. **Generate transformation.** Run **generate-transformation** and find the visual analog of the action of the current source nv-state. If this function fails, mark visual manipulation abstraction as having failed for this source and target nv-state series and go back to apply action at 9.

12. **Apply-manipulation.** Apply this `transformation` to the corresponding element in the current source visual s-image. This makes a new s-image.

13. **Transfer manipulations.** Run **transfer-manipulation**, attempting to transfer the current source s-image's `transformation` to the target. If the `transformation` cannot be applied, go back to 5, marking this visual instantiation as failed. Else go on.

14. **Apply transformation.** Apply this `transformation` to the current target s-image. This makes a new s-image. Run **apply-manipulation** to do this.

15. **Generate action.** Run **generate-action** to specify the abstract `transformation` into an action that can be taken on the current target nv-state. If no unused specifications remain, mark visual manipulation abstraction as having failed for this source and target `s-images`. Go back to 10 and generate new `s-images`. If an action is generated, associate that action with the nv-state and go to 9 to apply it.

Now I will describe the functions referred to above in more detail.

- **Name:** Evaluate
- **Input:** Success Conditions, nv-state (nv-state)
- **Output:** [success — failure]
- **Process:**
    1. Evaluate will run a simulation of the system as it stands in the input `nv-state`, and returns "success" if the simulation meets the goal criteria, else it returns failure.
- **Name:** Retrieve
- **Input:** knowledge state (knowledge state)
- **Output:** knowledge state-series (knowledge state-series)
- **Process:**

1. If the input knowledge state is an `s-image`, then this function will return series represented in Covlan. If the input knowledge state is a `nv-state`, it will return `nv-state` series.
2. The function will reject any series that has been marked as failed for the input knowledge state's series.
3. If there is a conflict, the best matching series is returned. The details of how retrieval happens will be fleshed out over the course of the dissertation and is not important to my theoretical claims.
4. If all potential analogs have been marked as failed, mark analogical problem solving has having failed for this source `nv-state` series and exit.
5. If it succeeds, set the retrieval problem state to current-source-knowledge-state.

- **Name:** find-mapping
- **Input:** knowledge state1 (knowledge state), knowledge state2 (knowledge state)
- **Output:** a mapping
- **Process:**
  1. This theory has no theoretical ties to any particular mapping mechanism; one of the many published means are possible.
- **Name:** generate-s-image
- **Input:** knowledge-state (`nv-state`)
- **Output:** an `s-image`, visual instantiation connections between the `nv-state` and the `s-image`
- **Process:**
  1. The reasoner has knowledge of what each object looks like. This means that each object is associated with an element or complex of elements and relations.

Default values (such as where something will be placed in an `s-image`) will be determined by the relations of objects with other objects in the `nv-state`. There is psychological data [65] showing how actions are associated with image placement that could be used to constrain how this works.

- **Name:** transfer-mapping
- **Input:** `mapping` (mapping), knowledge state1 (knowledge state), knowledge state2 (knowledge state)
- **Output:** `mapping` or failure
- **Process:**
  1. Generate a new symbol for the `mapping`.
  2. Associate with the new `mapping` new versions of all the maps.
  3. Change the referents of all the maps to what is connected to them with the visual instantiation connections.
- **Name:** transfer-manipulation
- **Input:** knowledge state1 (knowledge state), manipulation (manipulation), knowledge state2 (knowledge state)
- **Output:** The manipulation associated with knowledge state2.
- **Process:**
  1. Take the manipulation connected to knowledge state1 and connect it to knowledge state 2. Specifically, connect the manipulation to the analogous entity or entities in knowledge state2.
  2. Transfer all manipulation arguments to the new manipulation. If an argument has an analog in knowledge state2, use that. If it it does not, transfer it literally.
- **Name:** apply-manipulation

- **Input:** knowledge state1 (knowledge state), manipulation (manipulation)
- **Output:** another knowledge state in knowledge state1's series
- **Process:**
    1. Generate a new knowledge state, connected in series to knowledge state1. This new knowledge state is like knowledge state1 except it has the manipulation applied to it. If this cannot be done, exit and fail. Else exit with success.
- **Name:** generate-action
- **Input:** transformation1 (transformation), knowledge-state1 (knowledge-state)
- **Output:** an action associated with knowledge-state1
- **Process:**
    1. Retrieve an unused candidate action from the specifications of transformation1. Take into account the `transformation`, and what it will be applied to in knowledge-state1.
- **Name:** generate-transformation
- **Input:** knowledge-state1 (knowledge-state), action1 (action)
- **Output:** transformation, and possibly s-images.
- **Process:**
    1. If there is no `s-image` associated with knowledge-state1, make one.
    2. If there is no `s-image` associated with knowledge-state1's target `s-image`, make one.
    3. Abstract action1 into a visual `transformation` appropriate to the visual abstraction in the `s-image`.

In conclusion CAVA is a computational theory that uses specific data structures, algorithms and control architectures for visual analogical transfer. The task and method breakdown of CAVA is in Figure 67.

CAVA also consists of the following claims:

1. A primary function of visual abstraction is for the resolution of ontological mismatches.

2. Visual abstraction can be used for the resolution of ontological mismatches at several stages of analogy, including retrieval, mapping, transfer, and adaptation.

3. Transfer for strongly-ordered procedures in which new objects get created involve construction of intermediate knowledge states and mappings.

4. A useful level of visual abstraction is that same level used by Covlan in Galatea (shapes, lines, etc.)

5. Transfer of strongly-ordered procedures is computationally complex, even given the correct mapping, because the successful transfer of strongly-ordered procedures in which new objects are created requires the reasoner to generate intermediate knowledge states and mappings between the intermediate knowledge states of the source and target analogs.

6. Visual knowledge alone is sufficient for transfer of problem solving procedures in some domains.

CHAPTER VII

RELATED WORK

In this chapter I will discuss other relevant implemented computer systems. I will descibe systems that do some kind of visual analogy, systems that specifically do analogical problem solving, and finally analogy systems of other kinds.

7.1 Other Visual Analogy Systems

The closest system to my own is Letter Spirit. Like my theory, LetterSpirit is a model of analogical transfer [52, 64]. It takes a stylized seed letter as input and outputs an entire font that has the same style. It does this by determining which letter is presented, determining how the components are drawn, and then drawing similar components of other letters the same way. Like Galatea, the analogies between letters are already in the system: the vertical bar part of the letter *d* maps to the vertical bar in the letter *b,* for example. The mapping is implicit, though, in that the parts of each letter are mapped to "roles" (e.g. crossbar). That is, rather than mapping the crossbar of `t` to the crossbar of `f`, the system simply knows that both parts are instances of `crossbar.` During transfer, for example, the seed letter may be interpreted as an *f* with the cross-bar suppressed. When the system makes a lower-case *t,* by analogy, it suppresses the crossbar.

LetterSpirit transfers single transformations/attributes (e.g. crossbar-suppressed) and therefore cannot make analogical transfer of procedures (e.g. moving something, then resizing it) which Galatea can do. In contrast, one can see how Galatea might be applied to the font domain: The stylistic guidelines in LetterSpirit, such as "crossbar suppressed" are like the visual transformations Galatea: it would be a transformation of removing an element from the image, where that element was the crossbar and the image was a prototype letter *f.* Then the transformation could be applied to the other letters one by one. In this way my theory has more generality than LetterSpirit.

Copycat [42] shares Letter Spirit's underlying theory, but operates on *strings* of letters. For example `abc : abd :: ijk : ?` Where the agent is expected to apply the same procedure to `ijk` that was done to `abc`. Copycat searches stochastically through rules ("codelets" in the "coderack") to find rules that, in this example, transform `abc` into `abd`. These rules are applied to `ijk`, resulting in, perhaps, `ijl`. Sometimes a reasoner must find similarity between non-identical relationships. Copycat accomplishes this with the hand-coded "slipnet."

Like Letter Spirit, Copycat does not transfer strongly-ordered procedures during which new elements are created, and is limited in its domain. Both share a similar underlying theory, however, but for both strongly-ordered procedures are not transferred. There can be several transformations tranferred, but they are order-independent because later transformations do not rely on previous transformations.

ANALOGY is an early visual analogy program [16] that solves multiple choice analogy of the kind found on intelligence tests (e.g. `A:B::C:?`). All analogs are represented with semantic networks. ANALOGY chooses the best answer by describing how to turn `A` into `B` (this transformation is also represented with a semantic network), then how `C` turns into all the choices. It matches the A to B transformation semantic net to the nets of the choices. The best match determines ANALOGY's choice for the answer. Like CAVA, ANALOGY had a visual language consisting of primitives (e.g. dot, circle, square, rectangle, triangle), relations (above, left-of, inside) and transformations (rotate, reflect, expand, contract, add, delete). Galatea's representation language, Covlan, has considerable overlap with ANALOGY's ontology.

The PAN system [61], like ANALOGY, uses graph-like representations of abstract diagrams and outputs transformations that will turn one into another.

ANALOGY and PAN have many differences from CAVA. They have no sense of absolute location in its visual representation. They describe only meaningless images, without any tie to what they represent (indeed, the domains are intentionally non-representational). They can only describe transformations that occur in a single step. That is, they cannot represent strongly-ordered procedures. ANALOGY has no sense of transfer.

GeoRep [23] takes in line drawings and outputs the visual relations in it. First it uses the LLRD (low-level relational describer). Its visual primitives are: line segments, circular arcs, circles, ellipses, splines, and text strings. It finds relations of the following kinds: grouping, proximity detection, reference frame relations, parallel lines, connection relations, polygon and polyline detection, interval relations, and boundary descriptions. Then the HLRD (high-level relational describer) finds higher-level, more domain-specific primitives and relations. GeoRep's content theory is at the low level–the higher level primitives are left up to the modeler. Covlan has considerable overlap with GeoRep's primitives, though the processing goal of GeoRep is quite different: GeoRep generates visual relations from a more primitive input. Galatea uses a given visual input representing a procedure and transfers it to a new, visually represented problem.

MAGI [24] takes visual representations and uses the Structure-Mapping Engine (SME, described in a later section) to find examples of symmetry and repetition in a single image. JUXTA [22] uses MAGI in its processing of a diagram of two parts, and a representation of the caption. It outputs a mapping between the images, and notes distracting and important differences. It models how humans understand repetition diagrams. Both MAGI and JUXTA use GeoRep as the visual representation. The focus of these systems is on mapping. Something akin to JUXTA could potentially be used by a system like Galatea to automatically generate the transformations between sequential `s-images` in a series in that JUXTA can detect important differences from which transformations could be inferred.

The VAMP systems are visual analogical mappers [70] as well. VAMP.1 uses a hierarchically organized symbol/pixel representation. It superimposes two images, and reports which components have overlapping pixels. VAMP.2 represents images as agents with local knowledge. Mapping is done using ACME/ARCS [44], which is described in a later section. The fortress/tumor problem was one of the examples to which VAMP.2 was applied. Like my theory, MAGI, JUXTA, and the VAMPs use visual knowledge. But unlike my theory their focus is on the creation of the mapping rather than on transfer of a solution procedure. JUXTA's and my theory are compatible: a JUXTA-like system might be used to create the mappings that my theory uses to transfer knowledge. The theory behind the VAMPs is

incompatible because they use a different level of representation for the images.

DIVA [11] is another analogical mapper that uses visual representations. Specifically, it uses the Java Visual Object System. Like the VAMPs, it uses the ACME architecture for mapping. One of its examples is the fortress/tumor problem. The system does no transfer of the problem solving procedure, however.

FABEL [31] uses diagrammatic case-based reasoning in the domain of architectural design. It uses domain-specific heuristics to guide pattern extraction and transfer.

7.2 Other Analogical Problem Solving Systems

Derivational Analogy theory, [72, 71, 67] implemented in the Prodigy system, models transfer using memories of the justifications for each step, allowing for adaptation of the transferred procedure. Traces, called *derivations*, are scripts of the steps of problem solving, along with justifications for why the steps were chosen over others. One way my work differentiates itself is that in derivational analogy, the intermediate knowledge states are not saved in the case memory, only the record of the changes made to them. This means that the states can be inferred, but are not explicitly present in memory. Prodigy is able to avoid generation of intermediate mappings because the examples with which it has been implemented do have procedures that create new objects.

CHEF [41] is a case-based reasoner that adapts cooking recipies from a source to a target. Like Prodigy, CHEF does not create intermediate knowledge states, and, also like Prodigy, does not transfer procedures that create new objects. Two other case-based reasoning systems are ARCHIE [62] and AskJef [1]. They have case memories of graphic representations (buildings and user interfaces) indexed symbolically. Like FABEL they are intended for use by people, and the analogical transfer task must be done by the human user.

The Process of Induction (PI) model [43] is the only implemented computational model, other than my own, that solves the fortress/tumor problem analogically. It uses a production system to solve the fortress problem, and the activation resulting guides the finding of the solution to the tumor problem. Interestingly, PI does not even do transfer in the common sense of the term. Rather than taking a procedure directly from one analog to another,

it searches for a solution, guided by the activation trace left from the activation of that procedure by the source. After that PI generates an abstract schema that works as a single rule that can apply to both problems in the future.

7.3 Other Analogy Systems and Theories

SME is based on the Structure-Mapping Theory [32]. It constrains the mapping problem with the empirically validated systematicity principle, the one-to-one mapping principle, parallel connectivity, and identicality. [19]. The systematicity principle holds that high order relational similarities are preferred. SME finds many possible mappings, then evaluates them according to the map rules. Similarity is based on analogy (described above), literal similarity (where both relational and object predicates are mapped), mere-appearance (where primarily only the object descriptions are mapped), and abstraction mapping (where the entities in the base domain are variables rather than objects). These correspond to different match rules that can be used with SME.

I-SME [28] (also known as SME 3) and the Incremental Analogy Machine (IAM) [46] are incremental mappers. An incremental mapper generates a mapping as objects in a given analog are introduced to the mapper one at a time. They are intended to model experimental effects found with human participants. The focus of SME and I-SME are on mapping, where the focus of Galatea is on transfer of problem-solving steps. However, Galatea does incremental mapping of a different sort: It *modifies* a mapping as a result of changes made to an `s-image`. Though they are different, both tasks are important.

The task of the PHINEAS [17] system is create an explanation for some phenomenon using analogical reasoning. It uses Qualitative Process Theory [26] for its knowledge representation. It evaluates using simulation, where past reasoning traces are summarized by storing with each state in an observation the collection of theories that were used to explain it. E.g., with an example of alcohol evaporating from a flask, it may store theories about evaporation and containment. PHINEAS can explain new behaviors based on its knowledge of old behaviors, and it transfers knowledge from source to target. The transfer problem-solving *procedures* is outside of the system's domain. In its attempt to create

explanations, it can hypothesize that some un-represented objects might exist for a target analog based on the existence of some object in the source analog (skolem objects). Though the creation of skolem objects is an important part of analogical reasoning, it is different from how Galatea creates knowledge states with new objects as a result of transformations. PHINEAS represents changes that the system undergoes as a result of how the system works (e.g. simulating how boiling water will evaporate). In contrast Galatea represents the changes some agent makes to the system (e.g. placing an egg in the boiling water.) Thus PHINEAS's skolem objects are hypothesized objects to generate alignments with unmapped entities in the source analog. The new objects in Galatea are objects added to *both* analogs as a result of transformations.

The Analogical Constraint Mapping Engine, or ACME [44], is a mapping engine based on the theory that mapping is a result of structural, semantic, and pragmatic constraints. Structure, in this sense, does not necessarily mean a physical makeup, but the nature of the representation: elements are structurally similar if they share the same relational structure with other elements. Semantic similarity means elements are either identical symbols or share predicates (e.g. a common super ordinate). Pragmatic constraints involve relative importance of some propositions in the representation given the goals of the agent. The mapping is generated as a result of a constraint-satisfaction spreading activation network. Transfer in ACME involves transferring relations and postulating new elements from the source analog, but it does not have a mechanism for the transfer of a solution procedure. That is, it is made to transfer facts, not procedures.

LISA [45] is another cognitive model of analogical mapping. Propositions are made up of units that spread activation to each other. Arguments of propositions fire in synchrony with the case roles to which they are bound, and out of synchrony with other case roles and arguments. Through spreading activation, the best map is found.

In Model-Based Analogy, or MBA [6], model-based representations include information about the structure of the domain, as well as the behaviors and functions of its component parts. To make analogies with these kinds of representations, the reasoner takes a part of the source domain and puts it in the target domain mechanism. For example, if a device

isn't delivering enough power, the reasoner might transfer the amplifier concept from the source device. The transfer is of structures, behaviors, and functions of the target (the SBF language).

MBA is implemented in the IDeAL system [6, 5], which used a language of SBF, generic physical principles and generic teleological mechanisms, which are useful units of analogical transfer in creative device design. Generic teleological mechanisms provide a taxonomy of functional and causal transformations to physical devices. IDeAL transfers conceptual strategies, not procedures, and has more of a focus on adaptation than Galatea. Candidate designs in IDeAL are evaluated with qualitative simulation.

The ToRQUE2 system [38, 40, 39] uses a taxonomy of generic structural transformations that can be applied to physical systems. Like Galatea, ToRQUE2 was used to model experimental participants. These transformations were found to be useful in modeling a protocol of a human subject solving a problem dealing with spring systems. Structural representations are different from visual representations: They describe a system's physical composition but typically include only the information directly relevant for predicting the causal behaviors of the system. Structural knowledge, like a schematic, shows the components of the system and the connections among them but leaves out other visual information, such as what a component wire looks like, which side of a pump is up, etc. ToRQUE2 applies changes to analogs, but the changes are not transferred from a source as they are in Galatea. The changes are taken from an ontology of Generic Structural Transformations.

Winston [73] created an analogical mapper with a content account of its domain, including causation. All possible mappings are generated, then scored.

REBUILDER [37] is a case-based reasoner that does analogical retrieval, mapping, and transfer of software design class diagrams. The diagrams are represented structurally, not visually, however. This means that, for example, what the connection is between two nodes is more important than the length of direction of that connection. A school has a relationship with a teacher, but it is not represented as a left-of/right-of connection, for example.

FAMING [20] is a case-based reasoning system that uses cases describing physical mechanism parts. FAMING uses the SBF (Structure-Behavior-Function) to describe the cases. The structure is described in terms of a metric diagram (a geometric model of vertices and connecting edges), a place vocabulary (a complete model of all possible qualitative behaviors of the device), and configuration spaces (a compact representation of the constraints on the part motions.) Shape features can involve two objects, expressing, for example, one part's ability to touch another part. Human desingers are necessary for FAMING's processing. The designer chooses which cases and functions should be used, which dimensions the system should attempt to modify, and which shape features should be unified. It uses qualitative kinematics to propose design solutions for the desired function following the designer-suggested idea. Though not described as a visual system, the important parts of physical mechanisms of the sort FAMING uses inevitably contain much knowledge that could be construed as visual. The point of FAMING is to modify cases according to shape substitution, and, unlike Galatea, makes no attempt to transfer strongly-ordered procedures of any sort.

7.4 Diagrammatic Reasoning Work

This section describes non-analogical visual reasoning systems research. The diagrammatic reasoning literature is large; I will only spotlight a few systems that represent the range of schemes for visual representation.

Larkin and Simon [49] created a system that could reason about, among other things, pulley systems. In the diagrammatic representation, objects are not represented explicitly, only locations. When one location is attended to, all information there is attended to. To answer a question about a pulley system, the agent uses some non-visual knowledge of physics along with the visual representation.

Forbus's Qualitative Spatial Reasoning [25] shows that for visual reasoning about physical systems, an agent needs both a metric diagram representation and a place vocabulary representation. A metric diagram shows the quantitative aspects of the system, like sizes, as expressed in numbers, etc. Perceptual processes can be applied to it. The place vocabulary

is a qualitative representation of where things are and their shape, as is relevent to the task at hand. A place is a region of space where some important property (e.g. being in contact with something) is constant. The paper puts forth the poverty conjecture: that there is no problem-independent, purely qualitative representation of space or shape. As an example of when a qualitative representation breaks down, you can represent that a robot can get through a certain door, but if it is carrying something, to figure out whether it can get through with it the robot would need to reason at the metric level. But the qualitative is important too, and the place vocabulary can make a graph of what the robot can do– it's a task specific representation. Because this switching needs to happen, qualitative and quantitative information needs to be tightly coupled. It is implemented in a system called FROB [27].

Like FROB, the task of Narayanan, Suwa and Motoda's model [54] is to predict the behavior of physical systems. Its workings are based on protocol studies of people predicting physical behaviors based on given diagrams. The visual knowledge is represented with diagram frames (representing lines and spaces and connections between them) and occupancy array representations (representing, for each pixel, what kind of object is located there). Though the diagram frames represent only lines, they are similar in character to and at the same level of abstraction as the representations in Covlan.

The NIAL system [35] distinguishes between *depictive* and *descriptive* representations (corresponding to bitmap-style and propositional style), as well as a distinction between *visual* and *spatial* (corresponding to *where* something is and *what* something is.) The descriptive representation is stored in memory, and the depictive is generated as a working memory structure as needed. This system has been applied to molecular scene analysis.

WHISPER [30] is an AI problem solver that can request observations from and make changes to depictive diagrams of a blocks world. It knows about stability and falling objects. It can visualize something rotating in the diagram and determine when it will hit another object. The system's goal is to move all blocks until they are stable. It moves them, then simulates how they will act (in a bitmap "retina") for evaluation.

The retina in WHISPER has best resolution at the center. It can focus on different parts

of the diagram. The retina is made up of bubbles that are grouped as rings and wedges. Bubbles also communicate with their nearest neighbors. The retinal supervisor tells the bubbles what to process.

The perceptual primitives are: find the center of a shape, find the points of contact between a shape of one color and the shape of another, examine curves for abrupt slope changes, test a shape for symmetry, test the similarity of shapes, and visualize the rotation of a shape while watching for a collision with another shape.

To rotate, all bubbles with the object in it ask the next wedge to turn on, then turn themselves off. Collisions are detected when you ask a bubble to turn on with some object when it's already on with another.

7.5 Summary of Related Work

To summarize, there are a variey of systems, each aiming to understand different parts of the analogical process (see Table 20). Though they use visual representations, MAGI, JUXTA, VAMP.1, VAMP.2, and DIVA are all addressing the problem of analogical mapping. They are all extensions of non-visual analogical mappers: MAGI and JUXTA are built on SME and GeoRep (a visual language and inference engine); VAMP.1, VAMP.1, and DIVA are all built on ACME. Other non-visual mappers are LISA and Winston's analogy work.

Some systems are for augmenting the analogical abilities of human beings as systems to be operated by a user. FABEL, ARCHIE, and AskJef fall into this category.

Other analogy systems, like Galatea, attempt problem solving. Galatea transfers problem-solving solution procedures, like Prodigy, CHEF, and PI. However none address the problem of when new objects are created that must be acted upon by later operations. Visual problem solvers, Letter Spirit and ANALOGY, as well as non-visual ones, IDeAL, ToRQUE2, PHINEAS, and Copycat, do not attempt to transfer procedures at all.

Table 20 summarizes the systems and their features.

Many of the systems described above deal with visual and spatial reasoning. Though

Table 20: Comparison of analogy systems. SOP refers to the transfer of strongly ordered procedures in which new objects are created.

System	retrieval	mapping	transfer	visual knowledge	SOP
Galatea			tran	vis	SOP
Letter Spirit	ret	map	tran	vis	
ANALOGY				vis	
PAN		map		vis	
Georep				vis	
MAGI		map		vis	
JUXTA		map		vis	
VAMP.1		map		vis	
VAMP.2		map		vis	
DIVA		map		vis	
FABEL	ret			vis	
Prodigy	ret	map	tran		
CHEF	ret	map	tran		
ARCHIE	ret				
AskJef	ret				
PI		map			
SME		map			
I-SME		map			
PHINEAS		map	tran		
ACME		map			
LISA		map			
IDeAL		map	tran		
ToRQUE2		map			
Winston		map			
Copycat		map	tran		
REBUILDER	ret	map	tran		
FAMING	ret	map	tran		

the systems represent many things, including, sometimes, non-visiospatial things, the visiospatial things represented by all fall under the categories of *what* is there, *where* it is (corresponding to the what (visual) and where (spatial) brain pathways) and finally if and how the components of the image are related (e.g. above/below relationships).[1]

Where the systems differ is in their modes of representation. Some use a purely symbolic or propositional representation (e.g. Galatea, GeoRep), some use a pixel or occupancy array representation (e.g. NIAL, WHISPER, Narayanan et. al's), some use a hybrid, such as a symbolic array (e.g. NIAL and VAMP.2), and finally one (FROB) uses quantitative measures, such as lengths and distances. As stated in the visual re-representation chapter and in many other works (e.g. [35, 21, 47]), there is good reason to think that a variety of representations schemes come into play in cognition. In terms of visual representation, Covlan's `primitive visual elements` resemble GeoRep's [23] "primitive shapes." Covlan's `connection` ontology allows orientation-independent transfer of operations in the cognitive modelling, where many experimental participants rotated the target ninety degrees.

All of these systems use symbols at a higher level of abstraction than pixels. Even those that use pixel representations use them *in addition to* higher-level symbolic representations. However this describes a range of abstraction levels: occupancy arrays use very-high-level symbols such as `chair`, as opposed to more generic shapes. What general principle can be used to determine the correct level of abstraction? I conjecture that an agent should use the highest level of abstraction one can which still allows component similarity to be found between the examples your system uses. For example, if a system is reasoning about room layout, the symbol `chair` might be appropriate. However if the system needs to see similarities between chairs and, say, cardboard boxes, then a lower-level shape vocabulary might be appropriate. Since Galatea is intended to transfer accross domains, it, like other systems, uses a symbolic shape vocabulary. Higher level abstraction means more ambiguity, which Do and Gross [14] have found to be an important aspect of diagrams in the architectural design domain.

[1] It could be argued that relations are a part of the "where" class of information, but "where" information is typically conceived as being a location relative to an image, rather than in relation to other visual objects.

More novel are Covlan's `transformations`. Though most diagrammatic reasoning systems include ways to change visual knowledge, Covlan's `transformations` are intended to represent steps in problem-solving procedures that are reasoned about by the system. Griffith's Generic Structural Transformations (GSTs), [38, 40, 39] though not specifically visual in nature, are somewhat similar in that they are transformations that are chosen by the system to be applied to a representation in an effort to solve a problem.

This brings us beyond visual representation and into visual *reasoning*. Diagrammatic reasoning systems tend to reason for one of the following four broad tasks:

First is simulation (e.g. [49, 27, 30, 54]) in which the system uses visual representations of physical systems to predict how the represented systems will behave. Second is recognition and visual inferencing (e.g. [23, 35]). Third is geometrical proving, which includes tasks such as proving geometric math problems [3] and reasoning about, for example, Euler circles [69]. Fourth is analogical reasoning, such as Galatea and Letter Spirit. Within the class of systems, Galatea is the first and only system to use visual knowledge and reasoning to transfer problem-solving procedures.

CHAPTER VIII

CONCLUSION

8.1 Claims

This work describes the Constructive Adaptive Visual Analogy theory, which deals with visual knowledge used to transfer problem-solving procedures.

Following is a summary of my hypotheses:

1. Transfer of strongly-ordered procedures is computationally complex, even given the correct mapping.

2. Visual knowledge alone is sufficient for transfer of problem solving procedures in some domains.

3. Visual knowledge facilitates transfer even when non-visual knowledge might be available.

In conclusion, the evaluation supported all three of the hypotheses, and resulted in one unexpected discovery, for a total of four claims:

Claim One: Visual knowledge alone is sufficient for transfer of problem-solving procedures in some domains.

The Galatea implementation shows that problem-solving procedures for inherently visual domains like the cake/pizza problem can be represented visually, and solutions can be transferred successfully. In light of this research I can speculate for which domains visual knowledge might be sufficient for transfer of problem-solving procedures: those domains, the solution procedures of which *could* be adequately described with descriptions of changes to spatial properties. A way to think about this is that if the important differences between the problem and the solution are reflected in *visual* differences, then that problem is likely to fall within the intended class. I refer to this class of problems as "physical systems"

the solutions typically involve physical changes (as opposed to, changes in ownership, the issuing of commands, etc.)

Claim Two: Visual knowledge facilitates transfer even when non-visual knowledge might be available.

The fortress/tumor example is an example of a domain which need not be visually represented. Galatea shows that visual knowledge of it can be used to transfer a non-trivial procedure across domains.

The implemented models of L14, L15, L16, and L22 show how Galatea's model of visual processing can account for human participant data as well, and provides details of how visual problem-solving transfer might work. The pen-and-paper models of the rest of the participants in Dr. Craig's experiment show how Galatea might model even more, using only visual knowledge, as well as describing the limits of visual knowledge.

The experiment partially supported the claim in that those who were asked to draw the solutions were more likely to get the analogous answer.

The third hypothesis of this work that visual knowledge facilitates transfer of strongly-ordered procedures. It turns out that the computational details involved in transfer of strongly-ordered procedures appear to bear no relationship with visual knowledge. However, in the course of building Galatea and the models in it, I discovered something about analogical transfer in general:

Claim Three: The successful transfer of strongly-ordered procedures in which new objects are created requires the reasoner to generate intermediate knowledge states and mappings between the intermediate knowledge states of the source and target analogs.

The first hypothesis states that transfer of strongly-ordered procedures is computationally complex. Galatea shows why, in detail, the first hypothesis may be right. A characteristic of strongly-ordered procedures is that components of the problem are *created* by the operations, and these components are acted on by later operations.

The psychological modelling shows how this might work for human cognition: The doorway is replicated, then moved, then sealed with containing walls. For the transfer of

multi-step, strongly-ordered procedures it was necessary for Galatea to generate intermediate knowledge states and mappings.

Claim Four: Evaluation appears to require non-visual knowledge

Though Galatea transfers problem-solving procedures, it still has no way of knowing if the transferred solution was *adequate* for the new problem. In the tumor problem, in order for the agent to determine if the tumor was destroyed and the patient was still alive, it needed some causal knowledge. By causal we mean knowledge of how things in a system change as they interact. Pre- and post-conditions are a straightforward way to represent this, but it is difficult to imagine what "visual" pre- and post-conditions might look like. Visual representations alone cannot enable evaluation of the solution. Other visual reasoning work that does evaluation, such as Funt [30], must use causal knowledge about things such as the force of gravity to make its evaluative simulations.

8.2 Future Work

My research theme is the study of the use of multiple representation schemes in intelligent systems.

In this dissertation I was able to create a cognitive model of visual analogical problem solving. The focus is on knowledge—what kinds of knowledge are needed, and the function of visual knowledge.

Intelligent agents can change their knowledge representations when needed. Though this is clearly a fundamental part of cognition, little attention has been paid to the details of how and why this happens. My research addresses this problem. Specifically, I look at how intelligent systems use visual knowledge in analogy and problem solving. This has led to the Constructive Adaptive Visual Analogy theory.

My future work will proceed in two directions: automatically generating visual representations from non-visual ones, and exploring the role of depictive visual representations. I will describe each in turn.

The **Visual Re-Representation Theory** is that two situations that appear dissimilar

non-visually may appear similar when re-represented visually: that one use of visual knowledge is to resolve ontological mismatches. An ontological mismatch is when two similar ideas are not perceived by the reasoner as such because they are represented with different symbols. In this computational theory of multi-modal analogy, visual re-representation enables analogical transfer in cases where there are ontological mismatches in the non-visual representation.

In the future, I will expand Galatea to be able to do analogical problem solving with non-visual knowledge. Upon encountering ontological mismatches, however, it will automatically change the non-visual knowledge into Covlan. Galatea will use this visual re-representation to resolve these mismatches. Once the correct connections are made using the visual knowledge, those inferences will be translated back into the non-visual knowledge representation.

This work will involve several modules: 1) The non-visual representation language and the version of Galatea that attempts transfer using it, 2) the visual instantiation module, which will create visual knowledge, 3) the current version of Galatea, which transfers visual knowledge, 4) the specification module, which turns visual knowledge back into non-visual, and 5) the evaluation module, which will determine if the generated solution is adequate.

The non-visual representation language will be one of causal relationships, functions, goals, behaviors, and structure. The non-visual transformations will be more specific to the kinds of objects they are changing—for example, the transformation that breaks up an army into small armies will take a group with constituent parts and break it up into some n number of groups. A transformation that affects a laser will disperse energy into several lasers with less energy in each. If the reasoner needs to transfer the `break-up` of the army to the `disperse-energy` of the laser, there is an ontological mismatch for the transformation.

The visual instantiation module will use knowledge of how things are visualized to create the visual representation. The most straightforward instantiations will be shapes of physical objects. But recent studies are beginning to provide constraints on the visual representations associated with more abstract concepts such as respect and argumentation. This module, which in some sense is a model of imagination and visualization, will be my

future work's first focus.

The specification module will use much of the same knowledge as the instantiation module, as it is re-representing in the opposite direction: from visual back to non-visual. However, the complication is that the visual primitives must at times specify into different symbols than those that originally were used to generate the visual knowledge. For example, the army's `break-up` transformation visually instantiates into `decompose`, but when specified to a be used on a laser, it specifies to `disperse-energy`. With this module's completion, the cycle of re-representation for ontological mismatch resolution will be complete.

The second avenue of my future research is more ambitious: I will explore the role of depictive imagery in knowledge representation and analogy. Most work on visual analogy, including my own, uses visual knowledge represented symbolically. Though the mental imagery debate continues, there appears to be excellent evidence of mental imagery playing a role in problem solving. But what role does it play beyond the roles of symbolically-represented, descriptive visual knowledge, such as the kind Galatea currently uses? My work so far has used visual knowledge represented symbolically—for this research direction I will work with lower-level images made of pixels, to try to discover exactly why they are useful for intelligent systems.

One advantage that imagery could provide is a means for the reasoner to use the powerful perceptual mechanisms at its disposal for detecting emergent properties of symbolically-represented visual knowledge. For example, imagine how a reasoner might represent people who start at a house and walk in different directions away from the house. Even if the trajectories of the people are represented, the distances between them, the distances they've traveled, etc., the reasoner might need to recognize that the shape the people form is that of a growing circle–an emergent property not explicit in the symbolic representation. Using visual perception on bitmap depictive imagery is one way to detect such properties. In short, perceptual mechanisms can facilitate detection of patterns in bitmaps that are not explicit in the symbolic representation that generated the bitmap. This is another kind of visual re-representation—this time at the depictive level. Again, for certain analogical

problems, such re-representation could facilitate analogy.

The theme of my research is in the changing of knowledge through different modes of representation, and the whens, whys and hows of these modes and changes between them. Given the discoveries from the Constructive Adaptive Visual Analogy theory described in this document, the future of this line of research is promising.

REFERENCES

[1] Barber, J., Jacobson, M., Penberthy, L., Simpson, R., Bhatta, S., Goel, A., Pearce, M., Shankar, M., and Stroulia, E., "Integrating artificial intelligence and multimedia technologies for interface design advising," *NCR Journal of Research and Development*, 1992.

[2] Barsalou, L. W., "Perceptual symbol systems," *Behavioral and Brain Sciences*, 1999.

[3] Barwise, J. and Etchemendy, J., "Heterogeneous logic," in *Diagrammatic Reasoning: Cognitive and Computational Perspectives* (Glasgow, J., Narayanan, N. H., and Chandrasekaran, B., eds.), pp. 211–234, Cambridge, UK: MIT Press, 1995.

[4] Beveridge, M. and Parkins, E., "Visual representation in analogical problem solving," *Memory & Cognition*, 1987.

[5] Bhatta, S. R. and Goel, A., "Learning generic mechanisms for innovative strategies in adaptive design," *The Journal of the Learning Sciences*, 1997.

[6] Bhatta, S. R. and Goel, A. K., "Design patterns: A computational theory of analogical design," in *Proceedings of IJCAI-97, workshop on Using abstraction and reformulation in analogy.*

[7] Casakin, H. and Goldschmidt, G., "Expertise and the use of visual analogy: Implications for design education," *Design Studies*, 1999.

[8] Catrambone, R. and Holyoak, K. J., "Overcoming contextual limitations on problem-solving transfer," *Journal of Experimental Psychology*, 1989.

[9] Chandrasekaran, B., Glasgow, J., and Narayanan, N. H., *Diagrammatic Reasoning: Cognitive and Computational Perspectives*, ch. Introduction. AAAI Press/MIT Press, 1995.

[10] Craig, D. L., Catrambone, R., and Nersessian, Nancy, J., *Model-Based Reasoning: Science, Technology, & Values.*

[11] Croft, D. and Thagard, P., *Model-Based Reasoning: Science, Technology, & Values*, ch. Dynamic Imagery: A computational model of motion and visual analogy. Kluwer Academic: Plenum Publishers, 2002.

[12] Davies, J. and Goel, A. K., "Visual analogy in problem solving," in *Proceedings of the International Joint Conference for Artificial Intelligence 2001.*

[13] Davies, J., Nersessian, N. J., and Goel, A. K., *Foundations of Science 2002, special issue on Model-Based Reasoning: Visual, Analogical, Simulative*, ch. Visual models in analogical problem solving. 2002.

[14] Do, E. Y.-L. and Gross, M. D., "Thinking with diagrams in architectural design," *Artificial Intelligence Review*, 2001.

[15] Duncker, K., "A qualitative (experimental and theoretical) study of productive thinking (solving of comprehensible problems)," *Journal of Genetic Psychology*, 1926.

[16] Evans, T. G., *Semantic Information Processing*, ch. A heuristic program to solve geometric analogy problems. MIT Press, 1968.

[17] Falkenhainer, B., *A unified approach to explanation and theory formation*, ch. 6. Morgan Kaufman, 1990.

[18] Falkenhainer, B., "Learning from physical analogies," tech. rep., Department of Computer Science, University of Illinois at Urbana-Champaign, 1988.

[19] Falkenhainer, B., Forbus, K. D., and Gentner, D., "The structure-mapping engine: Algorithm and examples," *Artificial Intelligence*, vol. 41, pp. 1–63, 1990.

[20] Faltings, B. and Sun, K., "FAMING: supporting innovative mechanism shape design," *Computer-aided Design*, vol. 28, no. 3, pp. 207–216, 1996.

[21] Farah, M. J., *Spatial Cognition– Brain bases and development*, ch. The neuropsychology of mental imagery: Converging evidence from brain-damaged and normal subjects. Erlbaum, 1988.

[22] Ferguson, R. W. and Forbus, K. D., *Advances in Analogy Research*, ch. Telling juxtapositions: Using repetition and alignable difference in diagram understanding. New Bulgarian University, 1998.

[23] Ferguson, R. W. and Forbus, K. D., "Georep: A flexible tool for spatial representation of line drawings," *Proceedings of the 18th National Conference on Artificial Intelligence*, 2000.

[24] Ferguson, R. W., "Magi: Analogy-based encoding using regularity and symmetry," in *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society.*

[25] Forbus, K. D., "Qualitative kinematics: A framework," in *Readings in Qualitative Reasoning About Physical Systems.*

[26] Forbus, K. D., "Qualitative process theory," *Artificial Intelligenc*, 1984.

[27] Forbus, K. D., *Diagrammatic Reasoning: Cognitive and Computational Perspectives*, ch. Qualitative spatial reasoning framework and frontiers. AAAI Press/MIT Press, 1995.

[28] Forbus, K. D., Ferguson, R. W., and Gentner, D., "Incremental structure-mapping," in *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society.*

[29] Forbus, K. D., Mostek, T., and Ferguson, R., "An analogy ontology for integrating analogical processing and first-principles reasoning," in *Proceedings of American Association for Artificial Intelligece 2002.*

[30] Funt, B. V., "Problem-solving with diagrammatic representations," *Artificial Intelligence*, 1980.

[31] Gebhardt, F., Voss, A., Grather, W., and Schmidt-Belz, B., *Reasoning with Complex Cases*. Kluwer, 1997.

[32] Gentner, D., "Structure-mapping: A theoretical framework for analogy," *Cognitive Science*, vol. 7, no. 2, pp. 155–170, 1983.

[33] Gick, M. L. and Holyoak, K. J., "Analogical problem solving," *Cognitive Psychology*, 1980.

[34] Gick, M. L. and Holyoak, K. J., "Schema induction and analogical transfer," *Cognitive Psychology*, 1980.

[35] Glasgow, J. and Papadias, D., *Mind Readings*, ch. Computational imagery. MIT Press, 1998.

[36] Glasgow, J. I., Fortier, S., Conklin, D., and Allen, F., "Knowledge representation tools for molecular scene analysis," in *Proceedings of the 28th Annual Hawaii International Conference on System Biotechnology Computing Track*.

[37] Gomes, P., Seco, N., Pereira, F. C., Paiva, P., Carreiro, P., Ferreira, J. L., and Bento, C., "The importance of retrieval in creative design analogies," in *Creative Systems: Approaches to Creativity in AI and Cognitive Science. Workshop program in the Eighteenth International Joint Conference on Artificial Intelligence*.

[38] Griffith, T. W., *A Computational Theory of Generative Modeling in Scientific Reasoning*. PhD thesis, College of Computing, Georgia Institute of Technology, 2000.

[39] Griffith, T. W., Nersessian, N. J., and Goel, A. K., "Function-follows-form transformations in scientific problem solving," in *Proceedings of the Twenty-second Annual Conference of the Cognitive Science Society*.

[40] Griffith, T. W., Nersessian, N. J., and Goel, A. K., "The role of generic models in conceptual change," in *Proceedings of the Eighteenth Annual Conference of the Cognitive Science Society*.

[41] Hammond, K. J., "Case-based planning: A framework for planning from experience," *Cognitive Science*, 1990.

[42] Hofstadter, D. R. and Mitchell, M., *Fluid Concepts and Creative Analogies*, ch. The copycat project: A model of mental fluidity and analogy-making. Basic Books, 1995.

[43] Holyoak, K. J. and Thagard, P., *Similarity and analogical reasoning*, ch. A computational model of analogical problem solving. Cambridge University Press, 1989.

[44] Holyoak, K. J. and Thagard, P., "The analogical mind," *American Psychologist*, 1997.

[45] Hummel, J. and Holyoak, K. J., "Lisa: A computational model of analogical inference and schema induction," in *Proceedings of the Eighteenth Annual Conference of the Cognitive Science Society*.

[46] Keane, M. T. and Brayshaw, M., "The incremental analogy machine," in *Proceedings of the Third European Working Session on Learning*.

[47] Kosslyn, S. M., *Image and Brain: The Resolution of the Imagery Debate*. MIT Press, Cambridge, MA, 1994.

[48] Lakoff, G. and Johnson, M., *Metaphors We Live By*. Chicago: University of Chicago Press, 1980.

[49] Larkin, J. and Simon, H., "Why a diagram is (sometimes) worth ten thousand words," *Cognitive Science*, 1987.

[50] Maxwell, J. C., "On faraday's lines of force," *Scientific Papers*, 1855-6.

[51] Maxwell, J. C., *The Scientific Papers of J. C. Maxwell*, ch. On physical lines of force. Cambridge University Press, 1861-2.

[52] McGraw, G. and Hofstadter, D. R., "Perception and creation of alphabetic style," tech. rep., AAAI, 1993.

[53] Monaghan, J. M. and Clement, J., "Use of computer simulation to develop mental simulations for understanding relative motion concepts.," *International Journal of Science Education*, 1999.

[54] Narayanan, H. N., Suwa, M., and Motoda, H., "How things appear to work: Predicting behaviors from device diagrams," in *Proceedings of the 12th National Conference on Artificial Intelligence*.

[55] Nersessian, N. J., *Faraday to Einstein: Constructing Meaning in Scientific Theories*. Kluwer.

[56] Nersessian, N. J., *Cognitive Models of Science*, ch. How do scientists think? Capturing the dynamics of conceptual change in science. University of Minnesota Press, 1992.

[57] Nersessian, N. J., *Idealization and Abstraction in Science*, ch. Abstraction via generic modeling in concept formation in science. Rodopi, 1994.

[58] Nersessian, N. J., "Opening the black box: Cognitive science and the history of science," *Constructing Knowledge in the History of Science*, 1994.

[59] Nersessian, N. J., *Essays in the History and Philosophy of Science and Mathematics*, ch. Maxwell and "the Method of Physical Analogy": Model-based reasoning, generic abstraction, and conceptual change. Open Court, 2002.

[60] Novick, L. R. and Holyoak, K. J., "Mathematical problem solving by analogy," *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 1991.

[61] O'Hara, S. and Indurkhya, B., "Incorporating (re)-interpretation in case-based reasoning," in *Proceedings of the First European Workshop on Case-Based Reasoning (EWCBR-93)*.

[62] Pearce, M., Goel, A. K., Kolodner, J. L., Zimring, C., Sentosa, L., and Billington, R., "Case-based design support: A case study in architectural design," *IEEE Expert: Intelligent Systems & their Applications*, 1992.

[63] Pedone, R., Hummel, J. E., and Holyoak, K. J., "The use of diagrams in analogical problem solving," *Memory & Cognition*, 2001.

[64] Rehling, J. A., *Letter Spirit (Part Two): Modeling Creativity in a Visual Domain*. PhD thesis, Indiana University, 2001.

[65] Richardson, D. C., Spivey, M. J., Edelman, S., and Naples, A. J., ""language is spatial": Experiemental evidence for image schemas of concrete and abstract verbs," in *Proceedings of the Twenty-third Annual Meeting of the Cognitive Science Society*.

[66] Schank, R. C., "Conceptual dependency: A theory of natural language understanding," *Cognitive Psychology*, 1972.

[67] Schmid, U. and Carbonell, J., "Empirical evidence for derivational analogy," in *Proceedings of the 21st Annual Conference of the Cognitive Science Society*.

[68] Shepard, R. and Cooper., L., *Mental Images and their Transformations*. MIT Press, 1988.

[69] Stenning, K. and Inder, R., "Applying semantic concepts to analyzing media and modalities," in *Diagrammatic Reasoning: Cognitive and Computational Perspectives* (Glasgow, J., Narayanan, N. H., and Chandrasekaran, B., eds.), pp. 303–338, Cambridge, UK: MIT Press, 1995.

[70] Thagard, P., Gochfeld, D., and Hardy, S., "Visual analogical mapping," in *Proceedings of the 14th Annual Conference of the Cognitive Science Society*.

[71] Veloso, M. M., "Prodigy/analogy: Analogical reasoning in general problem solving," in *EWCBR*, pp. 33–52, 1993.

[72] Veloso, M. M. and Carbonell, J. G., "Derivational analogy in prodigy: Automating case acquisition, storage, and utilization," *Machine Learning*, 1993.

[73] Winston, P. H., "Learning and reasoning by analogy," *Communications of the ACM*, 1980.