UNIVERSITY OF
CAMBRIDGE

Automating representation change across domains for reasoning

Aaron Stockdill
Selwyn College

This dissertation is submitted
on 7 September 2021
for the degree of
Doctor of Philosophy

Declaration

This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except where specified in the text.

This dissertation is not substantially the same as any that I have submitted, or, is being concurrently submitted for a degree or other qualification at the University of Cambridge or any other University or similar institution except as declared in the Preface and specified in the text.

I further state that no substantial part of my dissertation has already been submitted, or, is being concurrently submitted for any such degree, diploma or other qualification at the University of Cambridge or any other University or similar institution except as declared in the Preface and specified in the text.

This dissertation does not exceed the prescribed limit of 60 000 words.

Aaron Stockdill
7 September, 2021

Automating representation change across domains for reasoning

Aaron Stockdill

Representing a problem well can make it trivial to solve; represent it poorly, and it becomes impossible. But what makes a representation suitable for a problem, and how can we automatically choose the most suitable from a set of alternatives? Choosing an appropriate representation is a difficult, long-standing problem in artificial intelligence; we want to support people in making an appropriate representation selection based on the problem they are solving, their own cognitive strengths, and the representational systems available. A large part of the challenge in choosing alternative representations stems from not knowing what is 'the same': which parts in the problem statement correspond to parts of an analogous statement in a different representation. If instead this choice was automated, users could better understand the problem, and work towards a solution when given a more appropriate representation.

This dissertation contributes a novel approach for the identification of alternative representations of problems through the idea of *correspondences*. This is a key step towards being able to select representations that are well-suited to enabling problem solutions. Exploiting correspondences, we demonstrate how to compute the *informational suitability* of alternative representational systems; the practical utility of this is shown with a software implementation. The generality of this theory and implementation is demonstrated by applying both to a domain that is distinct from the one it was developed in. We evaluate our theory and implementation with an empirical study, where we present experts with a similar challenge of evaluating representational system suitability, and comparing their responses with that of our implementation.

The work described in this dissertation creates possibilities for software tools that react to the problem and user: intelligent tutoring systems with multiple ways of explaining concepts to students; or interactive theorem provers that create analogies to help the human prover in finding key insights. The resulting tools centre on the representational needs of the *human*, not the computer.

Acknowledgements

My sincerest thanks to my supervisor Prof. Mateja Jamnik, who has been an infallible mentor, supporter, and guide for these four years. Thank you for giving me this opportunity, and going above and beyond in supporting me during this time; I am forever grateful.

I also wish to thank Dr Daniel Raggi, who has been like a second supervisor. Our conversations have not only directly shaped this dissertation, but my approach to being a researcher. I also greatly appreciate the many wonderful dinners I have shared with you, Dr Žygimantė Tarnauskaitė, and your children.

My thanks to Dr Advait Sarkar, your input and advice have challenged my thinking and introduced new perspectives on this project.

This project would not be half of what it is today without the help of the rep2rep team: Prof. Mateja Jamnik, Prof. Peter Cheng, Dr Daniel Raggi, Dr Grecia Garcia Garcia, Dr Gem Stapleton, and Holly Sutherland. Each has given me their time, ideas, comments, feedback, and patience to improve my work at every step. I want to specifically acknowledge Dr Stapleton, whose detailed feedback on a draft of this dissertation immeasurably improved this work.

I am grateful for my friends in college, who have all been a wonderful distraction from the PhD project, and have made my time in Cambridge all the better. Thank you also to Ed, Chaitanya, Zohreh, Botty, and the rest of the AI Research Group. Our lunches, dinners, coffees, and chats have kept me sane during my PhD. I have made lifelong friends.

Finally, my love and thanks go to my family, Robyn, Andrew, and Chelsea Stockdill (and all the cats!), who have given me endless love and support while so far from home.

This research was supported by the Hamilton Cambridge International Scholarship, provided by the Cambridge Commonwealth, European & International Trust.

Overview

Contents

Introduction 1

Maps, like many other kinds of visualizations,
distort the 'truth' to tell a larger truth.

— *Barbara Tversky*

Common advice when attempting to solve a problem is to 'draw a picture', to form analogies between the problem and the diagram. This can focus attention to a specific instance of a more general problem, make implicit relationships explicit, and allow us to exploit the human visual reasoning system. We have all experienced the sudden realisation that comes with drawing an effective diagram to express our problem. But what is an *effective* diagram?

Choosing an effective diagram is a process of many parts. We must explore what makes up the problem being solved, and similarly what the different representational systems are that we could use to express the problem. Some systems will be able to express all the required parts of the problem, some will not; some will be able to express so much more as to obscure solutions, rather than illuminate them. The person solving the problem comes with their own set of strengths and weaknesses.

This dissertation contributes a novel approach to evaluating and recommending alternative representational systems tailored for specific problems and users. We focus on how we describe problems and representational systems, and how we link the two together to make a suitable recommendation. Figure 1.1 shows two representations of 'summing the integers from 1 to n': the first is algebraic, while the second uses dots. These two *representations* come from two distinct *representational systems*—the first from 'algebra', the second from 'dot diagrams'—and these systems have particular *correspondences*—dot arrangements represent numbers, stacking is like summing, and so on. To consider why the algebraic representation could be represented in the dot diagram system, we need *descriptions* of the representation and representational systems, and sets of correspondences between the systems; the people who create these descriptions and sets we call *analysts*. In parallel, *users* are profiled to determine their cognitive strengths and abilities. The descriptions, correspondence sets, and user profile are then fed into our framework to compute the *informational suitability* and *cognitive cost* of each alternative representational system, so that we have a measure of the overall appropriateness of each system, and can thus make a recommendation to the user. This pipeline, from representations to recommendations, is summarised in Figure 1.2.

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

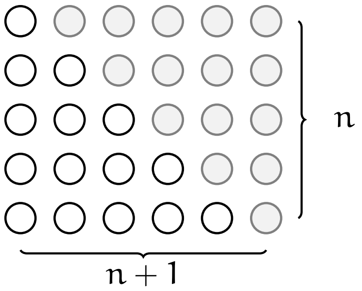


Figure 1.1 Two representations of the 'same' expression in different representational systems: algebra and dot diagrams. The algebraic representation asserts that the sum of integers between 1 and n is equal to the stated quadratic expression. The dot diagram counts the dots in a triangle by vertically stacking rows of dots (the black-edged circles), each one longer than the last, then observing a symmetry to create a rectangle. The annotations assert the generalised size of the rectangle, and so the number of dots in the original triangle is half the number in the rectangle, which itself is the product of the dimensions. We shall return to this example throughout this dissertation.

1.1 Context and motivation

Problem-solving encompasses many high-level reasoning tasks, so supporting people when solving problems can mean supporting many daily tasks. For this project, we consider mathematical problem solving: given a set of assumptions, determine whether a particular result holds. A common first impression of mathematics is that the notation is largely 'formulae': strings of symbols. But a diverse assortment of representational systems exist to encode mathematics: graphs, geometric figures, Venn diagrams, and many more. Why are these representational systems useful, and how are they related to each other?

1.1.1 *Human problem solving*

To solve a problem is, quite simply, to transform the problem statement into a goal statement [Simon et al. 71]. The goal might not be known at the start, but there is usually a way to identify it. The series of transformations is the problem solving process: taking a step away from the problem statement, towards the goal statement, by performing individual actions. In this way we induce a problem solving space—a graph[1] —that we, as problem solvers, traverse. We might walk the wrong way, or take a long path, or paths with high arc costs,[2] but as long as we end in a goal state we have solved the problem.

Even experts are forced to traverse the problem space, but they do not necessarily traverse the *same* space as novices [Condell et al. 10]. They can skip over states by combining arcs, or have strong heuristics when deciding which arc to follow. Experts have a mental map of the problem space that, quite literally, makes the solution path more obvious [Simon et al. 78]. How can we give the same abilities to novices? What strategies

[1] We assume the problem solving space is discrete: the states are countable, although not necessarily finite.

[2] 'High arc costs' denote more difficult actions to take.

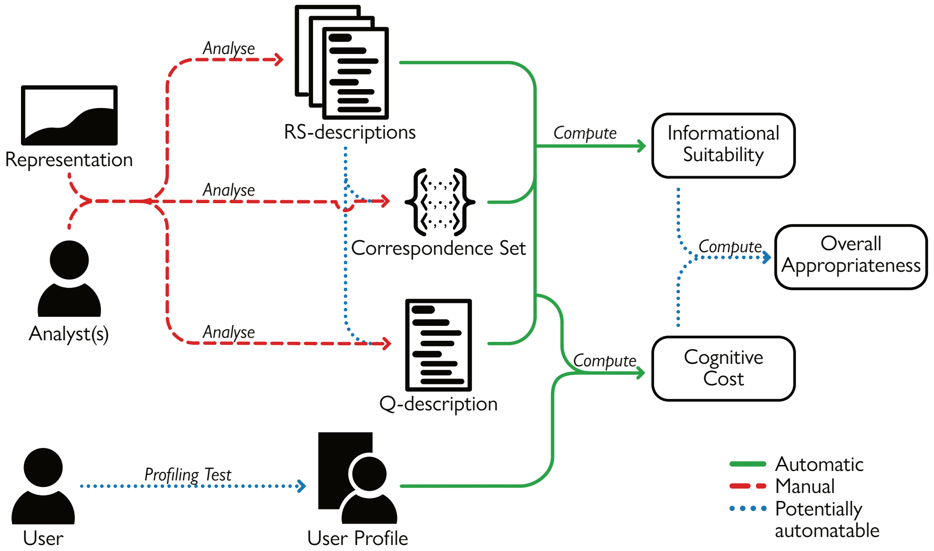


Figure 1.2 The representation recommendation pipeline from beginning to end, including the analysts creating representations, user profile generation, the algorithm computing informational suitability and cognitive cost, and the combination of the two measures. Note that 'Manual' and 'Potentially automatable' are short-term projections; we hope all steps will one day be automatic.

can we use to manipulate the problem space, and so make the problem easier to solve?

People use many different strategies to solve problems, from generalisation to specialisation to analogy [Pólya 57]. While problem solving is the traversal of the problem space, strategies aim to *change* the problem space making it easier to traverse. We might make the steps between states smaller, and easier to grasp; conversely, each step might be more difficult, but result in fewer necessary steps. Perhaps the number of decisions along the path is reduced, making the search less overwhelming. Intelligent problem solving is not searching a large problem space: it is *avoiding* searching a large problem space [Simon et al. 71]. If the user can be given a problem space which they can more easily traverse, then solving the problem also becomes simpler. We must recommend a way to induce a problem space which best matches the problem, and the user solving the problem.

1.1.2 *Heterogeneous reasoning*

An effective way to change the representational system that the problem is stated in: to consider the problem *heterogeneously*.[3] From Pólya's problem solving strategies, we focus on two representational system changes: 'draw a figure,' and 'analogy' [Pólya 57]. We further consider that drawing a figure is constructing an analogy that happens to be to a diagrammatic representation. For now, let us focus on drawing diagrams, and consider how they can be so effective at supporting reasoning.

[3] Restricting the problem to a single representation means it is represented *homogeneously*.

As famously noted in the title of Larkin and Simon's paper, a diagram can be worth 10 000 words [Larkin et al. 87]. They note three advantages of diagrams: spatial grouping, where related objects appear close to each other, so reducing the search for information; reduced labelling, such that things are not referenced, they simply *are*;[4] and 'perceptual inferences' can be had almost for free [Shimojima 96]. These features result in a problem space for which novices have better heuristics, and is lower cost to navigate through [Cheng 04].

Diagrams are also useful not just in isolation, but when considered together. By using several representations—both diagrammatic[5] and sentential[6] —the problem solver can compare and contrast the representations. This juxtaposition can elucidate previously concealed information because of the contrast between explicit and implicit information encoding. Complementary representations have explanatory power because they are analogically related: consider a table and a plot, where the former provides easily indexed, precise values, while the plot highlights trends and patterns. Each is useful, but together they are more effective.

Representations are able to change the problem space that the solver must traverse. By suggesting representational system changes, we can guide the solver to work in problem spaces that they are more effectively able to traverse. That is, by suggesting an appropriate representation change for the problem and user, we can make solving the problem easier for that user.

1.1.3 *Intuiting similarity*

Analogies are an interesting class of relations because they are very broad, and not universal. What might be an obvious analogy to one person might be unclear to another. But at their core, analogy is about the *similarity* of two things, and by inspecting the properties of one thing, we can extend the analogy to the properties of the other. The effectiveness of an analogy for problem solving is non-obvious: a higher degree of similarity is not always better, but missing similarity on specific properties can result in ineffective analogies [Thagard 92].

The effectiveness of an analogy is partly related to its 'depth' [Gentner 83]: surface analogies are weaker than structural analogies. That is, the best analogies link the internal structure of both objects, ensuring the similarity of the objects is carried throughout. If the problem solver has more expertise on one object in the analogy, they can lift that expertise across the analogical link, and so exploit their expertise in a domain where they previously had less. By creating analogous representations of the current problem, the novice can make themselves an expert.

By identifying *how* representations are similar, we can consider how 'strong' the analogy between them is. A representation that makes a sufficiently strong analogy to the problem that the user is currently solving is a candidate to which the user could transform their current problem. If the representation that makes the strong analogy also changes the prob-

[4] For example, a map does not 'state' that object z is at a location (x, y)—the object z *is at* the location.

[5] We interpret 'diagrammatic' broadly; anything visual and not sentential.

[6] String-like, sentence-like.

lem space into one more suitable for the problem and the user, then the user may be able to use the analogy to more effectively solve their problem.

1.2 Research questions

The overarching aim of this research project is to support human reasoning by encouraging and guiding the use of alternative representations. For the purposes of this dissertation, we identify three key themes that direct the path of our research: understanding representations and representational systems, capturing the similarities between systems, and using these similarities to recommend more suitable representations of problems.

Our first strand of research on representations and representational systems is summarised by the question:

> **Question 1.** What constitutes a problem, representation, and representational system, and can we describe each of these in a way that is equally suited to many varieties of representations?

This directly leads to three objectives:

- distinguish between problems, representations, and representational systems;
- identify the fundamental components of a representation, applicable to all representational modalities; and
- arrange these components into descriptions of problems, representations, and representational systems.

Now we consider how representations and representational systems interact with each other and are linked together. Thus our second research question is:

> **Question 2.** How are representational systems—and their components—similar, and can we state which components are similar across systems?

The three resulting objectives are:

- to define a similarity relation on components and descriptions;
- interpret this relation with respect to the underlying representational systems; and
- determine this relationship between two arbitrary representational systems' descriptions, potentially automatically.

Finally, we bring together the strands of work generated by the previous two questions:

> **Question 3.** How can we algorithmically evaluate and rank representational systems based on their ability to be used to solve a particular problem?

So we have three further objectives:

- define a measure of 'suitability' for a representational system with respect to a problem and a user;
- implement this suitability function as practical validation; and
- evaluate the 'correctness' of (our implementation of) this suitability function.

1.3 Contributions

The work in this dissertation is deeply linked with the work by the rep2rep research group, which consists of Prof. Mateja Jamnik, Prof. Peter C.-H. Cheng, Dr Grecia Garcia Garcia, Dr Daniel Raggi, Dr Gem Stapleton, and Holly Sutherland. In this section[7] we state contributions of this dissertation. We emphasise seven key contributions, four of which are uniquely attributable to this dissertation, and three of which are embedded within the rep2rep project.

1.3.1 *Contributions of this dissertation*

1. The novel concept of *correspondence* is a direct contribution of this dissertation. Whilst the specifics were informed by the rep2rep project, this dissertation is where correspondences are defined and developed. This includes the definition, theory, tooling, discovery, and interpretation, which are described in detail in Chapter 4. Correspondences are a route to answer to our second research question.

2. We develop *informational suitability* in Chapter 5, our answer to the third research question; the formalisation and implementation of minimally redundant and maximally covering (MRMC) correspondence sets is a contribution of this dissertation. This concept allows for more comprehensive correspondence sets, while avoiding 'double-counting' when computing informational suitability.

3. The implementation of the recommendation framework as `robin`, and the surrounding family of tools, is a contribution of this dissertation, except for the implementation of *cognitive properties* and *cognitive costs*. We describe this implementation in Chapter 5, as a means to evaluate the practicality of our solution to the third research question. We apply the framework—and implementation—to a detailed example in Chapter 6.

[7] And at the beginning of each chapter.

4. A further contribution of the dissertation is an empirical study that evaluates the representational system recommendations made by the rep2rep framework. The study aims to determine the extent to which the framework makes recommendations that are consistent with those of experts. The results establish that even experts do not have a universally consistent view on which representational systems are most suitable for particular problems and user profiles, reinforcing the need for tools to support heterogeneous reasoning. This study constitutes Section 7.3.

1.3.2 *Contributions as part of the* rep2rep *project*

5. Components and descriptions, the focus of Chapter 3, are unique to the rep2rep project, and originate in that work; they address our first research question. The concepts of components and descriptions have been updated and refined in conjunction with the work contributed by this dissertation, notably during the definition of correspondences. Work relating to the format of descriptions and the computational encoding of components is a contribution of this dissertation, in Sections 3.2.2 and 3.2.3. The work on pseudo-descriptions is part of this dissertation, and is discussed in Section 4.4. Cognitive properties, also part of Chapter 3 in Section 3.4, are not a contribution of this dissertation.

6. Informational suitability, presented in Chapter 5, is the combined contribution of the rep2rep research team and this dissertation. Its definition was developed alongside correspondences, and so was directly influenced by the work presented here. Amongst the rep2rep researchers and me, no one person could be considered the lead contributor. Cognitive costs (Section 5.2.2) build on the cognitive properties, and are not a contribution of this dissertation. Together, informational suitability and cognitive cost address our third research question.

7. The dissertation also presents another evaluation, which we call the ablation study. Whilst the data was initially collected and analysed in collaboration with the rep2rep research group, this dissertation extended that study to include ablated versions of the framework missing the factors *importance* and *correspondence strength*. The results suggest that each factor in our framework contributes important information to the final suitability score, and are presented as Section 7.2 in Chapter 7.

In summary, this dissertation contributes a novel method of linking together representational systems by means of correspondences, which are used to evaluate the informational suitability of said systems for the purpose of problem solving. It provides a proof-of-concept implementation (`robin`) that automates this evaluation and subsequent recommendation, given correctly formatted descriptions and correspondence sets.

This dissertation includes an evaluation of the efficacy of this implementation by way of an empirical study and an ablation study. The result is a system that achieves the stated goal of automatically suggesting alternative representational systems to potentially help a user more easily solve specific problems.

Parts of this dissertation have been already been published:

A. Stockdill, D. Raggi, M. Jamnik, G. Garcia Garcia and P. C.-H. Cheng. 'Considerations in Representation Selection for Problem Solving: A Review'. In: *Diagrammatic Representation and Inference, Diagrams 2021*. Ed. by A. Basu, G. Stapleton, S. Linker, C. Legg, E. Manalo and P. Viana. Vol. 12909. Lecture Notes in Computer Science. Springer, 2021, pp. 1–17. doi: 10.1007/978-3-030-86062-2_4.

A. Stockdill, D. Raggi, M. Jamnik, G. Garcia Garcia, H. E. A. Sutherland, P. C.-H. Cheng and A. Sarkar. 'Correspondence-based analogies for choosing problem representations'. In: *IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2020*. Ed. by M. Homer, F. Hermans, S. Tanimoto and C. Anslow. 2020, pp. 1–5. doi: 10.1109/VL/HCC50065.2020.9127258.

A. Stockdill, D. Raggi, M. Jamnik, G. Garcia Garcia, H. E. A. Sutherland, P. C.-H. Cheng and A. Sarkar. 'Cross-domain correspondences for explainable recommendations'. In: *Proceedings of the Workshop on Explainable Smart Systems for Algorithmic Transparency in Emerging Technologies (ExSS-ATEC)*. Ed. by A. Smith-Renner, S. Kleanthous, B. Lim, T. Kuflik, S. Stumpf, J. Otterbacher, A. Sarkar, C. Dugan and A. Shulner. Vol. 2582. CEUR Workshop Proceedings. CEUR-WS.org, 2020.

P. C.-H. Cheng, G. Garcia Garcia, D. Raggi, A. Stockdill and M. Jamnik. 'Cognitive Properties of Representations: A Framework'. In: *Diagrammatic Representation and Inference, Diagrams 2021*. Ed. by A. Basu, G. Stapleton, S. Linker, C. Legg, E. Manalo and P. Viana. Vol. 12909. Lecture Notes in Computer Science. Springer, 2021, pp. 1–16. doi: 10.1007/978-3-030-86062-2_43.

D. Raggi, G. Stapleton, A. Stockdill, M. Jamnik, G. Garcia Garcia and P. C.-H. Cheng. 'How to (Re)represent it?' In: *IEEE 32nd International Conference on Tools with Artificial Intelligence, ICTAI 2020*. Ed. by M. Alamaniotis and S. Pan. 2020, pp. 1224–1232. doi: 10.1109/ICTAI50040.2020.00185.

D. Raggi, A. Stockdill, M. Jamnik, G. Garcia Garcia, H. E. A. Sutherland and P. C.-H. Cheng. 'Dissecting Representations'. In: *Diagrammatic Representation and Inference, Diagrams 2020*. Ed. by A.-V. Pietarinen, P. Chapman, L. Bosveld-de Smet, V. Giardino, J. Corter and S. Linker. Vol. 12169. Lecture Notes in Computer Science. Springer, 2020, pp. 144–152. doi: 10.1007/978-3-030-54249-8_11.

D. Raggi, A. Stockdill, M. Jamnik, G. Garcia Garcia, H. E. A. Sutherland and P. C.-H. Cheng. 'Inspection and Selection of Representations.' In: *Intelligent Computer Mathematics, CICM 2019*. Ed. by C. Kaliszyk, E. Brady, A. Kohlhase and C. Sacerdoti Coen. Springer, 2019, pp. 227–242. doi: `10.1007/978-3-030-23250-4_16`.

1.4 Dissertation outline

Background & literature We continue this dissertation with Chapter 2 as an exploration of the existing work on problem solving. We tackle this from two directions: the cognitive processes and mental strategies inside the human mind when solving problems; and automated computational reasoning, largely in the domain of theorem proving. We examine *representations* in terms of how we define and classify them, what effect they have on the problem solving process, and how they are designed and evaluated to be *effective*.

Components & descriptions Chapter 3, our entry to novel material, begins with a definition of *components* and *descriptions*. This chapter includes a brief diversion into the philosophy of *why* we take the approach we do on representation description, before shifting to examine a parallel aspect of the rep2rep framework: *cognitive properties*.[8] This chapter addresses our first research question, contributing a language to describe problems, representations, and representational systems.

Correspondences With components and descriptions in our vocabulary, Chapter 4 considers how the links between components can be captured, and then understood at the description level. We introduce the formal underpinnings of correspondences and their *strengths*, and describe how this allows us to interactively discover new correspondences based on existing correspondences. Correspondences are our contribution towards the second research question, and understanding how representational systems can capture similar concepts in different ways.

Automated representation recommendation In Chapter 5 we bring together components, descriptions, and correspondences to evaluate the suitability of representational systems, and thus recommend to which the user should switch. We revisit the complete recommendation framework, and define the objective functions for both the *informational suitability* of representational systems, and their *cognitive cost*. We examine the details of filtering correspondences. This chapter covers details in the `robin` codebase, our implementation of the rep2rep framework. The work in this chapter

[8] Cognitive properties are not a focus of this dissertation, but deserve consideration due to their foundational position in the rep2rep framework.

addresses our third research question, providing an algorithmic approach to representational system recommendation.

Applying the framework With the entire framework covered, we review an application of the framework, and show how it applies in new domains. While this dissertation uses examples of representation change from school-level mathematics, Chapter 6 instead exemplifies how programming languages and algorithms can fit into this paradigm, and so each language is evaluated for its suitability to implement each algorithm. This chapter demonstrates the generality of our work and how it can be applied to more than mathematics.

Evaluating the framework To demonstrate the practicality of our work, we evaluate the framework with an ablation study and an expert study, both in Chapter 7. We first ablate the framework of two factors that contribute to informational suitability: component importance, and correspondence strength. We examine the output and determine the influence of each feature on the representational system recommendation, demonstrating that each factor is bringing new information to the recommendation, and is thus important to the framework. Then the user study presents mathematics teachers with the same challenge as we give to our framework: given a problem and hypothetical user, evaluate the suitability of alternative representational systems. We consider their responses quantitatively and qualitatively, and find broad trends based on both problem and user.

Conclusions & future work Finally, Chapter 8 summarises our work and contributions, and reiterates the future research opportunities highlighted throughout the dissertation.

SUMMARY OF CHAPTER I

This dissertation contributes novel methods to the long-standing open problem of recommending alternative representational systems, by way of correspondences. We define, formalise, and implement tools based on correspondences to demonstrate their theoretical and practical utility. The described work opens new approaches to interactive software that reacts and adapts to the problem and problem solver, working with the user to make their problem easier for them to solve by recommending appropriate representational systems.

Background & literature 2

> The more of the context of a problem that a scientist can comprehend, the greater are his chances of finding a truly adequate solution.
>
> — *Russell L. Ackoff*

The goal of this dissertation is to develop a framework that can recommend a representational system for any specific person to solve any specific problem. This is a goal of three parts: the problem, the person, and the representational system. We must understand all three to recommend an effective representational system; in this dissertation we produce a framework and software implementation based on these ideas. This chapter reviews the current research on problem solving with representations, both from a human-centred perspective, and from a software-centred perspective. This chapter forms the basis of our paper published at the *International Conference on the Theory and Application of Diagrams* (Diagrams) 2021 [Stockdill et al. 21].

We begin this chapter in Section 2.1 by exploring the human reasoning system: how people understand and solve problems, and how expertise affects the solving process. In Section 2.2 we consider how representations interplay with human reasoning: the different modalities, their effectiveness, the relationship to analogy, and how human reasoners choose representations for their problems. From the software angle, we consider how problems are solved by automated and interactive theorem provers; Section 2.3 explores different types of theorem provers, and how some incorporate multiple representations to varying degrees.

2.1 Cognition and reasoning

People are excellent reasoners, able to adapt and update their problem solving strategy such that almost any problem can be tackled. But some people are able to solve problems more effectively than others, exhibiting expertise in particular domains. We focus on *problem solving* because of its generality: there is an initial state, some way of identify goal states, and actions that can be taken that modify the state. A wide array of tasks can be modelled as problem solving, so we wish to understand how humans model problem solving, and how expertise is related to this model.

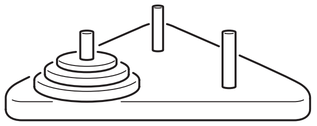


Figure 2.1 A typical three-disc Tower of Hanoi puzzle. The goal is to move all three discs from the starting peg to one other peg. You can move any disc that is at the top of its stack to any other peg, but at no point can a larger disc be on top of a smaller disc.

2.1.1 *Problem solving*

Solving a problem is conjectured to be a tight loop of understanding, planning, executing, and evaluating progress until a condition is met [Pólya 57]. Pólya's influential work on problem solving, *How to Solve It: A New Aspect of Mathematical Method*, lays out these four steps clearly, presents many varied examples of each step, and exemplifies the loop in its entirety. A more formal treatment of problem solving comes from Simon et al., where they introduce the *problem space* [Simon et al. 71]. The problem space is modelled as a (possibly infinite) graph where nodes are the problem state, and the arcs are the actions that allow movement between them; if a series of arcs connects the initial state to some goal state, then there is a solution to the problem. The nature of the problem, and the representation of the problem, determine the problem space. The person solving the problem must traverse the problem state space.

In this dissertation, we choose Simon et al.'s model of problem solving as our foundation. We use this model because it maps cleanly to common models of automated reasoning and theorem proving, making our comparison with these tools simpler. Other ways to frame problem solving (such as Zhang's distributed cognition [Zhang 97], or Johnson-Laird's mental models [Johnson-Laird 06]) may also function as a suitable foundation, but are beyond the scope of this dissertation.

When a person is traversing the problem space, some factors are fixed: the fundamentally serial information processing, small-capacity[1] but rapid-recall short-term memory, and effectively infinite slow-recall long-term memory. But other factors are mutable, such as *how* the space is traversed. Kotovsky et al. presented participants with variations on the 'Towers of Hanoi' problem, recording how they interacted with the problem and made progress towards (and away from) the solution [Kotovsky et al. 85]. The Towers of Hanoi puzzle, depicted in Figure 2.1, involves three discs with holes stacked atop one peg—a small disc on top, then a medium sized disc, then a large disc at the bottom. Alongside, there are two pegs without any discs. The goal is to move all three discs from the first peg to either of the remaining pegs such that they all end up on the same peg, in the same order they started, and at no point is a larger disc on top of a smaller disc. Kotovsky et al. analysed how people perform when presented with isomorphic variants of the Towers of Hanoi puzzle—such as monsters-and-globes, boxes-and-dots, or acrobats-and-flagpoles—and with different types of action: either *moving* objects (as in

[1] Famously, seven plus or minus two *chunks* [Miller 56].

moving a disk from one peg to another) or *changing* their size. Notably, representations involving unfamiliar scenarios (e.g., monsters rather than acrobats), and representations involving changing rather than moving, strongly hindered problem-solving performance, in spite of the problem being isomorphic. When facing an unfamiliar problem, participants tended to *probe* the problem space: they would perform a short sequence of actions with minimal deviation from the planned sequence before returning to the initial state. After these probes had been completed, and the participants were satisfied with their ability to traverse the space, they applied short sequences of actions chained together, rapidly converging on the goal state. These action sequences achieved sub-goals, unblocking the next action to be performed [Kotovsky et al. 85].

The work by Kotovsky et al. has two significant results that impact our work: first, there are two distinct methods of traversing the problem space (the probing back-and-forth approach, and the rapid sub-goal chaining approach); and second, changing the representation of the problem without changing its nature made the Towers of Hanoi-like problems easier or harder. These two results are tightly coupled: the representation of the problem impacted how the participants were able to traverse the problem space, and the participants' relative expertise in the problem space affected how difficult they found the task. To better understand this, we must understand expertise.

2.1.2 *Space traversal and expertise*

In computer science, there are many ways to traverse a graph: breadth first search, depth first search, A* heuristic search, etc. While people are less procedural, Larkin et al. identify two strategies that solvers use to traverse the problem space: *means-ends analysis* and *knowledge development* [Larkin et al. 80]. The former is similar to the behaviour seen by Kotovsky et al., using probing then sub-goal unblocking; the latter uses heuristics to avoid the probing and sub-goal analysis to immediately start chaining actions. Further, the solvers who use each strategy can be identified: means-ends analysis is indicative of *novices* in the problem domain, while *experts* employ knowledge development [Larkin et al. 80].

The strategy of *means-ends analysis*, which is employed by novices, is a type of 'working backwards.' The solver must identify what necessary conditions must be met to move towards the goal, and then work towards this new sub-goal [Kotovsky et al. 85]. Thus the novice begins to probe the problem space, understanding what effect their actions have, and then can begin to achieve their sub-goals. Maintaining this internal sub-goal chain is cognitively demanding, using working memory that could otherwise be devoted to the problem itself, not the 'traversal state'; even small problem spaces overwhelm human working memory [Kotovsky et al. 85].[2] Worse, the high cognitive load required to employ means-ends analysis can inhibit *schema acquisition*, a method of becoming an expert [Sweller 88].

[2] By analogy to computers, we devote 'registers' that would otherwise be used on the problem to maintaining the 'call stack,' but the human brain's 'call stack' capacity is small, and—due to the nature of graph search—easy to overflow.

Expert problem solving is best modelled through *knowledge development*, in which powerful heuristics guide the expert through the problem space [Sweller 88]. Because experts are familiar with the domain—and thus the problem space—there is little to no 'probing' phase; they have seen and solved similar problems in the past. Instead, experts can immediately begin applying *schemas*, which are patterns that the expert can recognise in the new problem space, and so immediately apply actions [Sweller 88]. Not only does this approach eliminate the probing and sub-goal creation, this approach induces less cognitive load—the utilisation of working memory—than means-ends analysis [Sweller 88]; experts will be faster *and* more cognitively efficient.

SUMMARY OF SECTION 2.1

Novices and experts alike solve problems by traversing a problem space, applying actions to change state within the space such that they eventually reach a goal state. But their traversal methods are very different: novices have a costly, means-ends analysis approach to searching the problem space; experts apply powerful heuristics called schema to efficiently work from the start to the goal. Clearly, being an expert is advantageous: can we somehow transfer these advantages to a novice? Or perhaps, can we change the problem space so that our novice is already expert?

2.2 Representation

As Kotovsky et al. discovered, the way a problem is represented can significantly impact how difficult the problem is to solve [Kotovsky et al. 85]. But why is this, and what exactly is involved in the representation of a problem? In this section we consider representation, and what it means for a problem to be represented effectively.

2.2.1 *Modalities of representations*

A representation is a view of a problem: the problem is expressed using some representation. The representation itself belongs to some representational *system*: a collection of syntax and rules that generate some agreed-upon notation. This is sometimes called an *external* representation because it exists outside the mind; there is a corresponding *internal* representation that exists within the mind of the problem solver [Scaife et al. 96].[3] Cheng links internal and external representations in two directions: an appropriate external representation can induce an effective internal representation, while an effective internal representation encourages external representation generation [Cheng 16]. We shall return to *effectiveness* in the next subsection.

Restricting ourselves to external representations, we can classify representations further. A common distinction is between 'sentential'—a sequence of characters composed only through concatenation [Stenning

[3] In this dissertation, unless explicitly qualified as an internal representation, we take representation to mean 'external representation'.

et al. 01]—and 'diagrammatic' representations.[4] Despite their apparent value of '10 000 words' [Larkin et al. 87], diagrammatic representations are often second-class in mathematics, even in highly visual domains such as graph theory. Informally, diagrammatic representations are widely used by mathematicians; formally, diagrams are often stripped from the discussion, because mathematicians consider them unsuitable for proof [Inglis et al. 09]. Even educational materials such as textbooks present only sentential solutions to problems, obscuring any intuition that a diagram can provide [Zazkis et al. 16]. Perhaps it is because diagrammatic systems are difficult to define: what makes a diagrammatic representation *diagrammatic*?

Taken in the extremes, there is obvious consensus on which representations are 'sentential' and which are 'diagrammatic': in mathematics, standard propositional logic notation[5] is sentential, while Euler diagrams[6] are diagrammatic. But as we drift away from these extremes, the boundary becomes indistinct: positioning limits on a summation is not concatenative, and hints towards some low-to-high relationship; a table filled with words uses space and positioning to encode information, but uses strings extensively. The distinction is difficult because, as Giardino observes, there is no distinction to be made [Giardino 13]. Representations exist on a continuum, some with more diagrammatic aspects than others; when we discuss diagrammatic representations we are referring to representations exhibiting four diagrammatic aspects:

- direct encoding,
- syntactic constraints,
- syntactic plasticity, and
- heavy use of geometric and spatial attributes and relations.

Let us consider each of these in more detail.

direct encoding

Diagrammatic representations *directly* encode types, structures, and relations of problems, rather than using indirect association as in sentential representations [Stenning et al. 01].

Example 2.1. Consider a relation 'to the right of': we can easily state an instance of this sententially:

a is to the right of b

while observing that a is visibly *left* of b. By comparison,

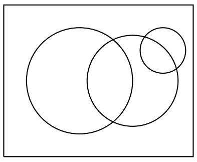
b   a

is a more *direct* encoding: a is literally to the right of b. This extends to all levels: rather than using the word 'square', diagrams can include

[4] We consider only *visual* representations; representations that are audial or tactile, for example, are beyond the scope of this dissertation.

[5] That is, $\wedge, \vee, \neg, \rightarrow$, and so forth.

[6] For example,



squares; rather than explain how nodes and arcs form a graph, we can draw the graph. But this can also enforce specificity: we can sententially state that ‘a zebra has some stripes,’ without making any claim to how many stripes, but any particular drawing of a zebra has a *fixed number* of stripes. This makes the representation easier to process at the cost of reducing generality [Stenning et al. 95].[7] ▽

syntactic constraints

Shimojima observed that rules of a representational system come in two broad classes: intrinsic, and extrinsic[8] [Shimojima 01]. An *intrinsic* (or *syntactic*) constraint is imposed by the syntax of the representational system: the geometry, topology, or physics of the representation enforce the rules. An *extrinsic* constraint is imposed by the problem solver: the representational system allows for statements that the solver wishes to avoid.

Example 2.2. Going back to our ‘to the right of’ example, let us assume a system where we can write $\mathsf{a} >_{\mathsf{r}} \mathsf{b}$, meaning a is to the right of b.[9] Then we can state the following three facts:

$$\mathsf{a} >_{\mathsf{r}} \mathsf{b}, \mathsf{b} >_{\mathsf{r}} \mathsf{c}, \text{and } \mathsf{c} >_{\mathsf{r}} \mathsf{a}$$

Now, if we try to represent this in our ‘positional’ notation from earlier, we hit an *intrinsic* constraint: we cannot arrange the letters on the page such that this is true! The representational system has prevented us from representing some state. So on a plane, the sentential notation is too permissive: we failed to apply the *extrinsic* constraints necessary to identify a nonsense statement. If we are working on a sphere, then the positional representation is overly restrictive: the intrinsic constraints are preventing us from encoding a valid state. ▽

syntactic plasticity

Closely related to syntactic constraints is *syntactic plasticity*: the ability for a representation to allow correct actions, but discourage incorrect actions [Cheng 02]. While syntactic constraints are static expressiveness limits of a representational system, syntactic plasticity is the dynamic *malleability* provided by a representational system. A syntactically plastic representational system allows actions to be executed that traverse a problem space, while restricting the number of actions to prevent the solver from becoming ‘lost’ in the problem space. In more graph theoretic terms: a syntactically plastic representational system has a low out-degree for each state in the problem space—the choice of actions is restricted—while still ensuring there is a path from the current state to a goal state.

geometry and space

Finally, diagrammatic representations make use of *geometry* and *space* [Stenning et al. 01]. The benefit of this is that it exploits the human

[7] Throughout this dissertation, we end examples with a ▽.

[8] Shimojima identified many variants of this divide, but all are sufficiently similar for our discussion.

[9] Note again that, visually, a is *left* of b.

visio-spatial reasoning system—the Towers of Hanoi variants presented by Kotovsky et al. to participants consistently demonstrated that participants more efficiently solved the 'physically plausible' variants [Kotovsky et al. 85]. Humans evolved in a physical world that obeys particular rules: we are well-adapted to manage systems that follow these rules. But geometry and space are limiting; just as we identified in direct encoding and syntactic constraints, we forfeit *abstraction* and *generality* by following the physical rules.

2.2.2 *Effective representations*

With a diverse range of representational systems at our disposal, some with more diagrammatic aspects than others, we must consider: what makes a representation *effective*? In the context of problem solving, there are quantifiable *results* we might be interested in: lower cognitive load, shorter times to generate a solution, or shorter solution paths. But in this subsection we look at the representations themselves, not the results they generate: in order to achieve these results, what properties do our representations have?

We consider effective external representations in relation to the internal representations they induce. Green et al. created the 'Cognitive Dimensions' framework as a guide on creating representations, but note that it is not intended for a deep analysis of existing representations [Green et al. 98].[10] Instead we consider Cheng's criteria for effective representations [Cheng 16]. While 19 criteria are listed, we consider the five categories in which the criteria exist:

- direct encoding,
- low-cost inference,
- conceptual transparency,
- syntactic plasticity, and
- conceptual-syntactic compatibility.

Let us explore these, and compare them to the diagrammatic aspects.

DIRECT ENCODING

Cheng's first criterion for effective representations is that it *directly encodes* the types, structures, and relations of the problem [Cheng 16]. This is the same benefit that diagrammatic representational systems provide. But *why* is this necessary for a representation to be effective? Consider, for example, Duncker's 'candle problem': given a box of tacks, some matches, and a candle, attach the candle to the wall [Weisberg et al. 73]. Participants will attempt to tack the candle to the wall, or melt some wax to use as glue, neither being effective; rarely do they consider they could pin the tack box to the wall and sit the candle in the box [Weisberg

[10] Although work that builds upon these dimensions (for example, [Blackwell et al. 01]) often include concepts very close to those we are about to discuss.

et al. 73]. Condell et al. call this inability to re-contextualise the tack box *functional fixedness*: the 'type' of the box is wrong, because in the 'representation' people have, the box is a container for *tacks*, not a container for *candles* as required for the problem [Condell et al. 10]. Tversky highlights a similar point in regards to structure: people have a mental hierarchy to categorise their environment, and benefit when the representation follows the same hierarchy [Tversky 11].[11] Thus a representation that more directly encodes a problem is likely to be more effective than those that encode the problem indirectly.

LOW-COST INFERENCE

The cost of inference in representations is a combination of factors: while the inferential actions themselves should be low-cost to perform, they must also be low-cost to identify [Cheng 16]. In diagrammatic representational systems, this is a mixture of geometric and spatial aspects, syntactic constraints, and syntactic plasticity. One notable variety of low-cost inference is the *free ride*—an inference that can be made without specifically taking steps to make that inference [Shimojima 96]. Stapleton et al. extend this to *observational advantages*: some representations allow information to be observed 'for free' that would require purposeful inference in other representations [Stapleton et al. 17]. A 'free' inference is certainly low-cost; representations exhibiting observational advantages are likely to be more effective than their disadvantaged counterparts.

CONCEPTUAL TRANSPARENCY

More difficult to define, conceptual transparency is the ability to 'see through' the representation to its underlying meaning. Cheng decomposes conceptual transparency into five aspects: coherence and unambiguity; small conceptual gulf; integration of conceptual perspectives; integration of granularity scales; and the comparing and contrasting of typical, special, and extreme cases [Cheng 02]. These are themselves difficult to resolve; we consider them in more detail in Section 3.4.[12]

SYNTACTIC PLASTICITY

We have already directly addressed syntactic plasticity in relation to diagrammatic representational systems: a representation should allow just enough manipulation to reach a solution, but not so much as to allow the solver to become lost.

CONCEPTUAL-SYNTACTIC COMPATIBILITY

Cheng's final criterion for effective representations is conceptual-syntactic compatibility. In diagrammatic representational systems, this is related to the idea of syntactic constraints: a close relationship between 'expressible' and 'valid' ensures a more effective system [Cheng 16]. By analogy, in computer science we discuss making illegal states unrepresentable [Minsky 11]

[11] In this case, the hierarchy is that the box is restricted to tacks, and there is no hierarchical relationship to the candle; the *necessary* hierarchy has box restricted to *objects*, which includes tacks and candles. When the tacks and box are given *separately*, the participants succeed much more often [Weisberg et al. 73].

[12] See particularly 'concept mapping'.

for the same effect: if you *cannot* say something incorrect, then you have reduced the ways in which you can make a mistake. Other mechanisms through which representations have conceptual-syntactic compatibility is by having a construction process which mirrors the problem solving process, and by allowing for distinct phases in encoding, interpreting, and making inferences [Cheng 16].

Bringing our conversation back to problem solving, we can consider an effective representation to be one that provides a problem space in which the solver is sufficiently expert: they already have access to low cost inferences and powerful schema.

2.2.3 *Analogical reasoning*

When people change the problem's representation—by creating a diagram, or otherwise—they are creating an analogy between the original representation of the problem and the new representation of it. Indeed, one distinction on diagrammatic representations by Sloman is explicitly noted to be between 'Fregean'[13] and 'analogical' representations [Sloman 75]. A restricted form of analogy is *translation*, wherein a statement in one representational system can be formally translated to a statement in another representational system, and remain equivalent.[14] While this form of analogy can be useful, it only applies in limited situations: many representational systems are not sufficiently rigorous that we can assert that two statements are equivalent.

Analogy can also be used for *informal* reasoning: there may not be a formal translation between representational systems, but by constructing a 'similar enough' statement in alternative representational systems the solver can exploit their knowledge of the analogical system to support their problem solving process in the original system. Thagard concisely breaks down the three aspects of analogy: pragmatics, semantics, and structure [Thagard 92]. The first suggests that the purpose of the analogy must be clear: to solve a problem, or to understand a concept, or to persuade. This feeds into whether the analogy is 'within-domain'—the analogical representational system is the same as the original representational system—or 'between-domain'—the analogical representational system is distinct from the original. We focus on the latter. At the semantic level, the surface attributes of both representations are considered: which terms are shared, which features are preserved. Thagard uses the analogy of pandas and televisions [Thagard 92, original quote from Dolnick 89]:

> In evolution, as in television, it's not necessary to be good.
> You just have to be better than the competition.

Superficially, pandas are black and white, while some television programmes are black and white; this does not help the analogy. Terms like 'competition' *do* carry across directly. But it is the third criterion, structure, that makes the analogy successful: there is a thing in a competitive environment, and scarce resources that are necessary, so for the

[13] Comparable to our 'sentential'; named for Gottlob Frege, an early contributor to symbolic logic.

[14] Alternatively, the new statement could be *stronger* such that it implies the original statement.

thing to continue existing it must have successfully acquired enough resources from the environment [Thagard 92]. Whether the thing is a panda and the resources food and reproduction, or a television show needing viewers and money, the structure of the problem remains.

Like Thagard, Gentner concluded that representations are best considered analogous through their internal structure [Gentner 83]: that the deeper 'shape' of the problem is more important than the surface details of the problem. Her work on structure-mapping constructs links between two analogous concepts by unifying their internal associations; this requires well-defined, hierarchical internal structure [Falkenhainer et al. 89]. Gentner's 'Structure-mapping engine' works on this philosophy: it proceeds bottom-up, finding simple structural analogies and propagating these up to higher level structural analogies that capture more abstract relationships [Falkenhainer et al. 89].

2.2.4 *Recommending a representation*

We have seen that representations can affect how difficult a problem is to solve; that diagrammatic representations often exhibit favourable aspects for problem solving; and that analogies provide a strong foundation for changing representation. Undeniably, changing to an effective representation is useful [Ainsworth 08; Cheng 02; Cox 99; Grawemeyer 06]—the problem is that students do *not* change to a more effective representation [Superfine et al. 09; Uesaka et al. 10]. We want to work towards supporting these students, and helping them change representation. Ideally, we want to support students in changing representation to one that is appropriate for the problem and for their expertise. But how should they be guided to change towards effective representations?

In the restricted domain of extracting information from a database, Grawemeyer's External Representation Selection Tutor (ERST) was able to recommend an information visualisation to users to answer queries [Grawemeyer 06]. The visualisations were scatter plots, sector graphs, pie charts, bar charts, tables, and Euler diagrams; when supported by ERST in choosing an effective representation, participants were more effective at answering the queries [Grawemeyer 06]. But to consider tasks beyond information extraction, the literature on representation recommendation becomes scarce. To *solve* a problem, we explore the representation *design* literature: what factors are important when designing representational systems, which we may consider for representation recommendation?

Representation design, and for our work representation recommendation, is a product of three factors: what is the problem, who is approaching it, and why are they working on it? This combination of factors determines the *cognitive fit* of a representation [Moody 09; Vessey 91]. Vessey introduces cognitive fit as the combination of the specific problem under consideration, and the overarching task and context in which the problem is encountered, which together influence the internal representation a person constructs [Vessey 91]. But implicit in Vessey's discussion is that the *person* influences the internal representation; as we saw earlier, an expert and a novice will be operating with different internal representations: the novice's internal representation is tuned for search, and the expert's for heuristics [Larkin et al. 80]. Moody makes this explicit: cognitive fit is the interaction between the problem, the person, and the task [Moody 09]. It is from this point where we launch our work: this dissertation focuses on the problem, and how it influences the choice of appropriate representation; we briefly discuss the cognitive considerations of the person, as context of the surrounding rep2rep project; neither this dissertation nor the rep2rep project are yet considering the task in which a problem has been encountered, but it is expected to be future work.[15]

SUMMARY OF SECTION 2.2

A representation is a complex thing: it is an encoding of information into the real world, which induces a specific *internal* representation in people. A wide range of factors determine representational efficacy, and diagrammatic aspects of representations align to allow for effective representations. By considering representation change as *analogy*, we have a model to understand how and why people change representation; through cognitive fit, we can begin to understand how to recommend a representation based on the problem being solved, the person solving the problem, and the task and context in which the problem was encountered.

2.3 Automated heterogeneous reasoning

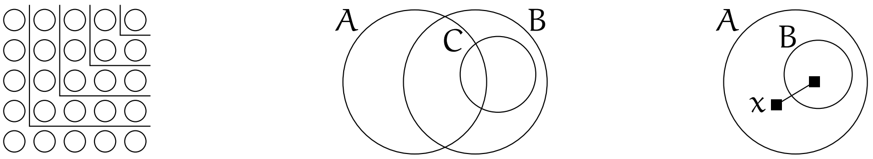
In the previous two sections we considered why and how representations are evaluated and recommended.[16] In this section, we explore the use of representations in computational systems. While artificial intelligence researchers have attempted to build general problem solvers for a long time—consider the aptly named 'General Problem-Solving Program' [Newell et al. 59]—most success has been had in solvers specialised to particular domains. We focus on interactive and automated theorem provers, as this class of software is forced to consider concerns similar to ours: solving problems, representing them effectively, and considering their users.

2.3.1 *Theorem provers*

Theorem provers are used by people to solve a very specific type of problem: given some assumptions, derive a specific conclusion. To make progress, the set of assumptions is updated using already-proved theorems (or axioms) through tactics. This maps directly to the problems space we discussed: the current state is the current set of assumptions, a goal state is any set of assumptions which contains the desired conclusion, and the actions to move between states are the tactics. So the difference

[15] We discuss these factors in Section 5.2.

[16] Note that this is a manual process; a person performs the evaluation and recommendation.



(a) $\sum_{i=1}^{5}(2i-1) = 5^2$ (b) $A \cap B \neq \varnothing, A \cap C \neq \varnothing, C \subset B$ (c) $B \subset A, \exists x \in A$

Figure 2.2 Representations, with their meaning in sentential notation underneath. The representational systems are (a) Dot diagrams, (b) Euler diagrams, and (c) Spider diagrams. In the spider diagram, the 'spider' means x could be in either indicated region. That is, $(x \in A \wedge x \notin B) \vee x \in B$.

between the theorem provers is the state space they model—and the representations they exploit.

Most theorem provers are *homogeneous*: that is, they use a single representational system. This single representational system is usually sentential, but the details vary. One family of theorem provers are those based on Martin-Löf type theory: two notable members are Coq [Huet et al. 04], and Nuprl [Constable et al. 86]. These systems use Martin-Löf type theory as their representational system, and proofs are constructions of a value that has the type which is an encoding of the theorem to prove. A second notable family of theorem provers are those with HOL/LCF ancestry [Gordon et al. 79]: HOL4 [Slind et al. 08], HOL Light [Harrison 09], and Isabelle/HOL [Paulson 89].[17] These systems use a small core of actions that is easily verified, and all other actions must be built on top of this core. Both families use a syntax that is programming-language-like, and purely sentential.

Equally homogeneous, but no longer sentential, are *diagrammatic* theorem provers. Diamond focuses on diagrammatic proofs of arithmetic using grids of dots (Figure 2.2a), and ways of partitioning the grid [Jamnik et al. 99]. The high-level approach of Diamond is different to that of the sentential provers mentioned earlier: it works with instances of a proof and generates a generalised version automatically, rather than expecting the person proving the theorem to work in the most general case at all times. Edith, and its successor Speedith, focus on Euler diagrams (Figure 2.2b) and Spider diagrams (Figure 2.2c), respectively [Stapleton et al. 07; Urbas et al. 12]. Their proof structure more closely resembles that of the sentential systems: from some diagrams you can construct a new diagram; analogously, from some assumptions you derive a new conclusion. These systems show that software is capable of supporting diagrammatic representational systems, but they do not yet push the bounds to *heterogeneous* reasoning: exploiting multiple representations.

2.3.2 *Slight heterogeneity*

Homogeneous theorem provers continue to grow in sophistication and power, but their generality comes at the cost of speed: some problems are best left to dedicated tools that have a better representation for that

[17] Isabelle (without 'HOL') is a *meta*-logic system: a developer tailors Isabelle to work in their particular logic. For example, there is an Isabelle/ZF which allows people to use ZF set theory rather than higher-order logic. This is an interesting step for Isabelle towards heterogeneity, but the different logics are inaccessible from each other.

problem. In the HOL/LCF tradition, these tools are integrated as *hammers* [Blanchette et al. 16]. For example, Isabelle/HOL uses 'Sledgehammer' to transform a higher-order logic problem into a first-order logic problem before passing the transformed problem (along with relevant lemmas) to automated first-order provers; the proof is returned to Isabelle/HOL, and validated in the verified core like any other proof [Paulson et al. 10]. While not obviously heterogeneous—every representational system involved is sentential—Isabelle/HOL exploits a system with a more effective problem space by transforming the problem.

Isabelle has a second means of heterogeneous reasoning: the *Transfer* package. The Transfer package was designed for code generation—taking theorems and proofs to generate executable code for software development. Raggi et al. repurposed Transfer into a heterogeneous reasoning toolkit [Raggi et al. 16], such that the 'transfers' happen between object definitions. Raggi et al. defined different definitions of natural numbers—for example, successors of zero, multisets of primes, and classes of finite sets—such that different definitions produced novel proofs with varying lengths. The formalised translations between the different definitions allowed for free movement between the definitions, producing a heterogeneous approach to theorem proving while remaining in purely sentential representational systems.

2.3.3 *Analogical proofs*

Remaining briefly with homogeneous, sentential theorem provers, we consider another mechanism by which they introduce heterogeneity: analogy. We earlier touched on the power of analogy for human reasoners; here we examine how discovering analogies can improve the effectiveness of automated theorem provers.

Formal proofs in theorem provers often resemble trees: the conclusion is the root while the branches are logical antecedents, continuing recursively until the leaves are either the theorem assumptions, axioms, or tautologies. For any given theorem, there are potentially many proof trees; any are valid. But the trees can also be *analogous*: by substituting predicates from one domain (for example, number of dots in a diagram) with another (the value of a number), we can re-use the same proof tree for a different, but analogous theorem [Boy de la Tour et al. 14]. This approach generalises to partial proofs, allowing mixing analogies to construct proofs for novel theorems.

Theorem provers' analogies are similar to human analogies, in that they are most effective when there are *structural* mappings: the overall 'shape' of the proof is what matters, not the superficial similarities of the theorems or representations. Melis et al. implement and demonstrate the efficacy of ABALONE [Melis et al. 99], an analogical proof constructor and extension to the CLAM proof planner [Bundy et al. 90]. ABALONE was able to construct proofs for theorems that CLAM was not; as with human reasoners, analogies allow the automated reasoner to exploit its expertise from one domain in another domain.

2.3.4 *Openproof, HETS, and MixR*

We move now from homogeneous or purely sentential systems to heterogeneous reasoning systems. An early and notable heterogeneous system, Hyperproof, was an educational tool for first-order logic that used a three-dimensional chessboard environment alongside a more typical sentential representational system [Barwise et al. 96-a]. The actions available in the two representational systems were different, as would be expected; proofs in the sentential first-order logic system are often more verbose than their chessboard counterparts [Barwise et al. 96-b]. Barker-Plummer et al. generalised Hyperproof to Openproof, a framework allowing heterogeneous reasoning with many representational systems [Barker-Plummer et al. 08]. The framework avoids an *inter-lingua*[18] but maintains a common proof state; this avoids some 'lowest-common-denominator' expressiveness concerns while maintaining a valid proof state. But as a result, there is a tight coupling between the representational systems in Openproof: there is a one-to-one correspondence between the objects and relations in each representation, and formal translations between them.

An alternative heterogeneous reasoning framework is the Heterogeneous Tool Set (HETS) [Mossakowski et al. 07]. HETS uses a graph of automated and interactive theorem provers, and so mixes the representational systems of those provers. The proof state and goals are sent to the provers as representations in their own representational system, meaning there is no tight coupling between the representational systems in a HETS-derived prover. The complication is the need for *comorphisms* between each of the representational systems: formal translations *must* link each system [Mossakowski et al. 07].

MixR is a heterogeneous theorem proving framework that grew out of a desire to integrate Speedith, the spider diagram reasoner, with Isabelle [Urbas et al. 14]. The MixR framework consisted of two parts: one theorem prover that 'owned' the proof state, and many 'working' theorem provers that could modify the proof state. Much like HETS, MixR aimed to reuse existing theorem provers, rather than develop specialist heterogeneous theorem provers; unlike HETS, MixR allowed for unsound transformations between the representational systems used by each of the 'working' theorem provers. MixR also introduced heterogeneous statements—using multiple representational systems simultaneously—through *placeholders*.[19] MixR, like all the heterogeneous systems we have discussed, provides the *option* for heterogeneous reasoning. But it does not *encourage* or *guide* heterogeneous reasoning: representation selection is a human-driven process.

SUMMARY OF SECTION 2.3

This section explored how current software, designed to work towards

[18] 'Common language'.

[19] An example use of placeholders would be $(x = \square) \rightarrow (\mathrm{shape}(x) = \texttt{square})$, physically placing the square in the statement.

solving problems alongside a person, manages the issue of representation. For the most part, software systems maintain a single representational system, whether sentential or diagrammatic. Some software is heterogeneous, notably HETS and MixR: these allow the user to combine multiple representational systems together to solve a single problem, with the latter allowing for informal transformations between representations. But the decision on which representational system to use at any given point is driven by the user—while multiple representations may be available, the user is *not* helped to use them.

SUMMARY OF CHAPTER 2

This chapter has examined how human problem solving can be modelled as traversing a problem space: the solver is attempting to reach goal states by applying actions to the current state. The expertise of the solver impacts their ability to navigate the problem space, but by selecting an effective representation we can induce a problem space in which the solver is already expert. The nature of the representation determines its effectiveness, and specific aspects—each with trade-offs—are generally agreed to be better; conveniently, these align with diagrammatic aspects of representational systems. We considered how analogical thinking allows for this kind of representation change, and how people struggle with representation change; to support them, we examined the three factors in representation recommendation: problem, solver, and task. We also discussed how representations are used in software, specifically theorem proving software: few support heterogeneous reasoning, and those that do fail to support the user in selecting an appropriate representational system. We proceed with a clear need to fill: to dissect and analyse problems and representations so that effective representational systems can be automatically recommended to the human problem solver.

Components & descriptions 3

What are numbers and
what should they be?
— *Richard Dedekind*

The foundation of our framework is the *component*.[1] In this chapter we address the first of our three research questions: what are problems, representations, and representational systems, and how can we describe them? We motivate and define what we mean by *component*; introduce *descriptions*, which are structured collections of components; and introduce *cognitive properties* and their relationship to components. We also discuss why we use components as our encoding system for representations and representational systems. In Chapter 4 we will consider relationships between components in different representational systems; in this chapter we consider each system in isolation.

Describing and comparing representations is a difficult, long-standing open problem [Cheng 16], where distinct grammatical and inferential components must be weighed against how they could potentially be used. As such, the first step must be to define a common language that allows us to describe and compare vastly different representations. When developing the framework introduced in this chapter, we followed three guiding principles:

- We must accommodate representations with different levels of *formality*, such that formal languages and informal ad hoc systems are equally describable;
- Representations of any *modality* must be describable, meaning the framework features must not favour a particular way of composing the constituent pieces; and
- The framework should capture the *structure* within representations.

We shall address how our formulation of components and descriptions addresses these principles in Section 3.3.

This chapter contributes a language to describe problems, representations, and representational systems in terms of their components in a general way;[2] thus, we address our first research question. Section 3.1 develops components, the building blocks of representations; Section 3.2 arranges components into descriptions, and thus allows us to distinguish problems, representations, and representational systems; Section 3.3 considers the philosophical position of components and descriptions as a general

[1] Our early publications named components *properties*.

[2] This is our 'fifth' contribution, see Section 1.3. Note that this contribution is joint work with the rep2rep research group.

representation encoding; and Section 3.4 describes cognitive properties, a concept related to components that are central to the rep2rep project. Early versions of the ideas in this chapter were presented at the *Conference on Intelligence Computer Mathematics* (CICM) 2019 [Raggi et al. 19], and later versions appeared at the *International Conference on Tools with Artificial Intelligence* (ICTAI) 2020 [Raggi et al. 20-A]. A set of extended examples appeared at the *International Conference on the Theory and Application of Diagrams* (Diagrams) 2020 [Raggi et al. 20-B]. Cognitive properties, in Section 3.4, were extended upon at Diagrams 2021 [Cheng et al. 21].

3.1 Defining components

Components are the building blocks of representations: breaking representations down into smaller and smaller units eventually yields simple pieces that can be composed together. For example, we can break down

$$n(n+1) \div 2$$

into n, 1, 2, $+$, and $\div$, amongst other things; these are candidates to become components. A component has structure that captures information about pieces of a representation, and so consists of three parts: kind, value, and attributes. Components can be classified into families, which are the *kinds*: primitives, types, patterns, laws, and tactics. The value of the attribute is a unique identifier, and for primitive components this is often the 'thing' itself. Finally, attributes are a set of associated objects: sometimes components, sometimes metadata.

3.1.1 *Kinds, values, and attributes*

Consider the '+' symbol in an algebraic representational system. As a fundamental concept,[3] it cannot break down into anything else. The component that we use to capture $+$ consists of three parts: it is of kind 'primitive,' the value is the icon $+$, and it has attributes such as a type. Together, the kind and value uniquely identify the component, while the attributes anchor it in the context of a representation.

The 'kind' of a component is a classification that groups components based on their cognitive status. We define five kinds: primitives, types, patterns, laws, and tactics. Each will be explored in the following sections, but to summarise:

- Primitives are the elements of a representation;
- Types describe the grammar of a representation;
- Patterns describe emergent groupings of primitives in a representation;
- Laws are the foundational true statements of a representational system; and

[3] Ignoring the definition of $+$ in terms of a successor relationship; typically algebra is not broken down that far. Components are 'relative,' in that when something can or cannot be broken down depends on the point of view; we shall explore this in Section 3.1.2.

- Tactics are the actions that can be taken within a representational system.

Each component has exactly one kind.

Component values describe a specific instance of a kind. For example, a value associated with a component of kind 'type' is the type's name:

$$(\text{type}, \text{real}, \varnothing)$$

is a valid component of kind type and value real. We adopt a more succinct notation for components, dropping the parentheses and commas, and omitting attributes completely if there are none. The above component is thus written

$$\text{type real}$$

in our notation.

Attributes give context and information about a component, and are sets of key-entry pairs where the keys are strings and the entries are arbitrary. A common attribute is 'type'[4] relating a primitive with its type. Another attribute is 'occurrences', counting how often the component is observed to be used in a representation.

Putting all this together, we define a component [Raggi et al. 20-A].

Definition 1 (component)**.** A component is a triple (k, v, a) where k is the *kind* of the component, v is the *value* of the component, and a are the attributes of the component.

Example 3.1. Consider a very simple representation, $1 + 1 + 1$. From this representation, we can extract a component like

$$\text{primitive } 1 : \{\text{type} := \text{real}; \text{occurrences} := 3\}$$

which includes a kind (primitive), a value (1), and two attributes (type, assigned real, and occurrences, assigned 3). This tells us the representation this component is associated with has a symbol 1 that we interpret as a real number, and it occurred three times. ▽

The pieces of components are now in place, so we turn to considering the different kinds in greater detail.

3.1.2 *Primitives, types, and patterns*

The component kinds can be split into two groups: *grammatical* kinds, and *inferential* kinds. The first group—consisting of primitives, types, and patterns—has the components which encode the representation notation. The second group—consisting of laws and tactics—has the components which make the representational system useful as a problem solving tool.

[4] The attribute 'type' is not the kind 'type'. But the attribute type $:= t$ is used as a link from some component to the type-kinded component with the value t.

PRIMITIVES

Primitives are the obvious building blocks of a representation: these are the parts we sense (see, hear, feel, smell, or taste)[5] and often register first when reading and understanding a representation. Primitives do not decompose: '$x + y$' is not a primitive, but an empty table cell is. Filled table cells are not primitives because they decompose into an empty cell and a contained value. The point at which you decide something no longer decomposes depends on the purpose of the representation. The number 42 may or may not be a primitive: in a representation like algebra, 42 is a primitive; in a representation describing how Hindu-Arabic numerals are constructed, 42 would not be a primitive because it decomposes into a 4 and a 2, both of which *are* primitives. We will usually work with the former case, where 42 is a primitive.

Sometimes with primitive components[6] we can use the literal value (using again our example of $+$, or x, or 52.7), but sometimes a primitive cannot be written directly: line segments, or a circle, or shading. In these cases we mark the value with a $ sigil, reminding us that this is the name of a primitive, not the primitive itself.

Primitives can have many different attributes, the two most common being 'occurrences' and 'type'. The former tells us how often a primitive occurs in a representation, while the latter associates the primitive with its type. Other possible attributes include: size, colour, opacity, etc., depending on whether these are important to the representation.[7]

Definition 2 (Primitive components)**.** A *primitive* component is a component with kind 'primitive'; the value is a conceptually indivisible building block of a representation.

TYPES

A type component describes a grammatical role within a representation. The values of two components with the same 'type' attribute could be 'swapped' with each other in their representation, and their contexts would still make grammatical sense. Simple examples would be type number, type vertex, or type real.[8] Type components consist of 'simple' types, and the value of these components is the name of that type. In the framework we define in this dissertation, the name of the type is not interpreted: creating a type component with value 'real' creates a unique type identifier whose name has meaning to the reader, but we make no assumptions about the properties that the terms of this type might have.

Simple types like number or vertex are sufficient for some parts of a representation, and are the only types that occur as type-kinded components; but simple types are too limiting when dealing with representations that involve functions or relations, so the 'type' attribute may contain *composite* types [Gordon et al. 79, p. 46]. The type of the primitive $+$ is not a simple type, instead we assign it the composite type

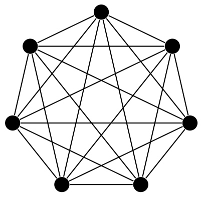
$$\text{number} \times \text{number} \to \text{number};$$

[5] The representations we consider are all visual; heard representations would be something like an audio book; felt representations would be something like braille; smelt or tasted representations are rarer.

[6] We use the term 'x component' to mean a 'component with kind x'. In this case, we mean a component with kind 'primitive'.

[7] These attributes can influence the cognitive properties (Section 3.4) of a representation, and future work might consider them. For now, we do not explicitly use these attributes.

[8] We shall shortly see how type number and type real relate when we discuss patterns.



The complete graph K_7; every vertex is connected to every other vertex. Figure 3.1

that is, $+$ has a function type from a pair of numbers to a single number.[9] Unlike simple types, we do interpret composite types: the symbols $\times$ and $\rightarrow$ are taken as pair and function, respectively; brackets have grouping and precedence semantics; juxtaposition is parametric type nesting, for example the type 'number set' is the type of a set consisting of numbers. These composite types allow us to construct larger expressions and so begin to describe the grammar of the representation.[10]

Types rarely have attributes.

Definition 3 (Type components)**.** A *type component* is a component with kind 'type'; the value is either a label for a simple type, or a label for a parametric type along with a type variable. That is, the value of a type component is defined as

$$\begin{aligned} \textit{type-value} &\coloneqq \textit{simple-type} \mid \textit{parametric-type} \\ \textit{simple-type} &\coloneqq \textit{label} \\ \textit{parametric-type} &\coloneqq \textit{type-variable label}. \end{aligned}$$

We denote type variables with a Greek letter.

Definition 4 (Composite types)**.** A *composite type* is constructed from the grammar

$$\begin{aligned} \textit{type} \coloneqq\ & \textit{simple-type} \\ \mid\ & (\textit{type})\ \textit{parametric-type} \\ \mid\ & (\textit{type}) \times (\textit{type}) \\ \mid\ & (\textit{type}) \rightarrow (\textit{type}) \end{aligned}$$

where $\times$ is left-associative and $\rightarrow$ is right-associative, and brackets can be dropped as necessary: parametric type constructors bind with higher precedence than $\times$, which has higher precedence than $\rightarrow$. Note that for the parametric type, $(\textit{type})$ is substituted for the type variable from Definition 3.

PATTERNS

Pattern components perform roles that both primitives and types play, but extend the potential of both. A *pattern* describes an arrangement of primitives, for example a complete graph (Figure 3.1) is a salient arrangement of vertices and arcs. To experienced readers, these arrangements can behave as if they were a primitive [Koedinger et al. 90]; the pattern

[9] This notation is taken from the Hindley-Milner typing tradition. Readers with knowledge of a language from the ML family will find it familiar.

[10] These composite types are actually short-hand for *patterns*, which we will see below.

describes a class of these arrangements. These arrangements also fill a grammatical role: when all these pieces are in place, an object with a potentially different type emerges. Thus, a pattern must have holes, which primitives[11] can 'fill,' and a type, which is the grammatical role the complete arrangement of primitives will fill. Patterns can also have associated primitives, which must be present for the pattern to exist; these are not holes, because they cannot be substituted for something of the same type: it *must* be this primitive.

Example 3.2. Consider two patterns: $x+y$, and $f(x)$, where f, x, and y are 'meta-variables'—we could replace them with any valid expression. In the first pattern we have the primitive $+$. Its type, number × number → number, tells us how the pattern is formed. Thus this primitive $+$ has an associated pattern[12]

$$\begin{aligned}\text{pattern} + : \{\ &\text{type} \coloneqq \text{number};\\ &\text{holes} \coloneqq [(\text{number}, 2)];\\ &\text{primitives} \coloneqq [+]\ \}\end{aligned}$$

which captures the same idea: there is a pattern (named $+$) that consists of two numbers and a $+$ primitive which then behave as a number. Pattern components that are based on primitives with a composite type can be automatically derived from the primitive component.[13]

The second pattern, $f(x)$, has no 'generating' primitives: there is no primitive component with a compound type from which we can automatically derive the pattern component. So we create the pattern component ourselves:[14]

$$\begin{aligned}\text{pattern functionApplication} : \{\ &\text{type} \coloneqq \beta;\\ &\text{holes} \coloneqq [(\alpha, 1), (\alpha \to \beta, 1)];\\ &\text{primitives} \coloneqq [\,(,)\,]\ \}.\end{aligned}$$

This is a function application pattern that consists of a function, some input, and opening and closing parentheses which then behaves as a value of the same type as the function output.[15] ▽

There is a third common type of pattern, *sub-typing*. This kind of pattern is used to 're-interpret' a term as a different type. For example, all integers are numbers, so we have a pattern

$$\text{pattern integerAsNumber} : \{\text{type} \coloneqq \text{number}; \text{holes} \coloneqq [\text{integer}]\}.$$

This example allows integers to be used where any number would be appropriate. Consider the utility of this in typing an operator like '+': rather than needing to define 'separate' addition operators for each numerical type—plus-for-integers, plus-for-reals, and so on—we can instead define it for one super-type 'number,' and require types such as integer or real be sub-types.

[11] Or *terms*, which we shall introduce shortly.

[12] The paired 2 in the holes attribute denotes there are two holes of type number.

[13] Our implementation, `robin`, does this: it interprets composite types to automatically produce patterns.

[14] We use the type variables α and β so that the pattern operates over any function.

[15] Type variables are quantified over all attributes of the component.

Definition 5 (Pattern component)**.** A *pattern component* is a component with kind 'pattern'; the value is a label denoting a salient arrangement of primitives. A pattern *must* have the attributes 'type' and 'holes'; the multiset associated with the holes attribute must not be empty.

We divert briefly to introduce *terms*. Intuitively, terms are a simple concept: $2 + 3$ is a term, or Figure 3.1 is a term.[16] Typically, terms are defined as combinations of symbols arranged so they adhere to some grammar [Chomsky 56], for example using types [Coquand 06]. While this would be a term in our system, we consider constructions more broadly. We define terms to be *fully instantiated patterns*. That is, a pattern in which all the holes have been filled with either primitives or terms.[17] Terms are not components themselves, but are directly derived from components; as such, they play an important role in the rep2rep framework.

Between primitives, types, and patterns, we have described the grammatical side of a representation: we have the 'pieces' (primitives) of the representation, and the rules for combining them (types and patterns). In the next subsection we turn to the inferential side of a representational system, and explore how components can capture the utility of representations as problem solving tools.

3.1.3 *Laws and tactics*

The inferential aspects of a representation are captured by the law-kinded and tactic-kinded components. Laws are like theorems at a certain level of abstraction: the things that are assumed to be true.[18] For example, 'dots' might have a *cardinality preservation* law where combining two dot arrangements does not change the overall number of dots. Similarly, a Bayesian Algebra representational system would include Bayes' Theorem[19] as a law. Tactics, in contrast, are the things you do with the laws and patterns of a representation [Paulson 89]. Our dots might have the tactic 'regroup', where the dots are moved or reinterpreted as different collections. Our Bayesian Algebra representational system—like most algebraic systems—will have a 'rewrite' tactic, where an expression is rewritten by applying particular laws.

LAWS

A law component usually has very few attributes. It may have a count of its occurrences in a problem, and sometimes an informal description as metadata.

Definition 6 (Law component)**.** A *law component* is a component with kind 'law'; the value is a label denoting a known true statement for the representational system.

[16] Although, this is determined by the representational system, as we hinted at the beginning this subsection.

[17] So our definition is recursive.

[18] In practice, we limit laws to axioms and important theorems; listing all theorems is impossible.

[19] $\Pr(a \mid b) = \Pr(b \mid a) \cdot \Pr(a) / \Pr(b)$

TACTICS

Tactic components are expected to have two attributes: 'laws', defining how many laws the tactic is parameterised over;[20] and 'patterns', similarly defining how many patterns the tactic is parameterised over. Consider our rewrite tactic in algebraic representational systems:

$$\text{tactic rewrite} : \{\text{laws} := 1; \text{patterns} := 1\}.$$

That is, to fully apply the rewrite tactic one must choose one pattern (the thing you want to rewrite) and one law (the equality—or implication—that we apply). For example, applying the rewrite tactic to the pattern $x + y$ and the law '+-commutativity' produces the pattern $y + x$.[21]

Definition 7 (Tactic components)**.** A *tactic component* is a component with kind 'tactic'; the value is a label denoting a method of manipulating a representation. A tactic may have the attributes 'laws' and 'patterns'—both integers—which must be supplied if the tactic is conceptually parameterised over a non-zero number of laws or patterns.

We have already called out one common tactic, rewrite, but there is another common tactic worth introducing: 'observe'. Observation is extracting information from a representation without formal inference. Only some representations exhibit *observational advantages* [Stapleton et al. 17], so the tactic of observation can range in utility from pointless to fundamental—at one extreme, you can observe only what is explicitly stated, nothing else; at the other extreme, representing the problem statement allows us to immediately observe the solution. That observation occurs without inference is key to its potential as a tactic: no laws are necessary, simply patterns. We illustrate this with the following example.

Example 3.3. Consider the following sequence of probability statements.

Assumption:

$$\begin{aligned} \Pr(X) &= 0.5 \\ \Pr(Y) &= 0.6 \\ \Pr(X \cap Y) &= 0.3 \end{aligned}$$

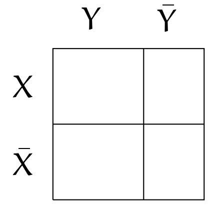
Deduction:

$$\Pr(X \cap Y) = \Pr(X) \cdot \Pr(Y) \Rightarrow \text{ X and Y are independent.}$$

Not a difficult chain of working, but it does require inference—specifically the final line. Now consider the geometric representation of the same probability statements in Figure 3.2. Note that the two dividing line segments meet at a single point: the segments do not get 'split'. This observation is sufficient to conclude that X and Y are independent events; no calculations or inference is necessary. That is, the pattern of 'line segments crossing', without any other laws, is sufficient. By representing the assumptions correctly, the conclusion comes 'for free'. ▽

[20] We parameterise over *how many* laws because we do not have a clear way of 'typing' the laws: some laws might not fit a particular tactic, but disambiguating them is future work.

[21] Our rep2rep framework does not apply tactics, it only describes them. We leave the application to either humans or automated systems.



A geometric representation of two independent events, X and Y. Figure 3.2

Observation is a simple, powerful tactic that is widely used in diagrammatic—and sometimes sentential—representations, and captures the expressive capabilities of representations which exhibit observational advantages.

SUMMARY OF SECTION 3.1

Components are the building blocks of representations, defined as triples of *kind*, *value*, and *attributes*. The kind partitions components into five categories, three notational—primitives, types, and patterns—and two inferential—laws and tactics. Attributes give information about components, linking them with other components through relationships such as typing. Representations may be described by many components, or very few components; some representations, when broken down, will have components in common, while others will not share any.

3.2 Representations, systems, and descriptions

The essence of representations, and representational *systems*, can be captured by sets of components; these sets are called *descriptions*. Descriptions come in three flavours: RS-descriptions for representational systems, R-descriptions for specific representations, and Q-descriptions for problems posed within representations. But before we get to descriptions, we must disambiguate what we mean by *representation* and *representational system*.

3.2.1 *Representations and representational systems*

A representational system characterises a class of representations, while a representation is a specific instance of a representational system. Using our example in Figure 1.1 from the introduction, the specific algebraic equation

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

is a representation, while 'algebra' is the representational system in which it is expressed. The *system* can be *instantiated* into many different representations: algebra can be used to construct many expressions and equations.

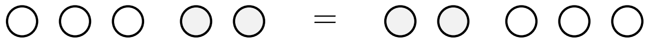
Representations[22] are an encoding of information into a medium for the purpose of communicating that information. Each representation is specific to the information it is communicating. The pieces present may or may not have meaning, and they may or may not be important; yet each piece is interpreted by the reader. This interpretation is guided by rules of grammar, and the validity of a representation is determined by its adherence to that grammar. An effective representation follows the grammar, uses pieces of syntax that convey important meaning, and successfully communicates information to the reader that is equivalent to the information intended by the author.

In order for the representation to succeed—to be interpreted correctly by another party—the 'rules' of extracting its information must be at least partially standardised. When a representation adheres to a grammar, we ask where the grammar comes from. When we consider the pieces of syntax in a representation, we ask where those pieces of syntax come from. This abstracts over individual representations to form a family of related representations that all use subsets of the same syntax and adhere to the same grammar. This abstraction is the representational *system*. Representational systems, unlike representations, are not about a single instance. No representational system is best at everything, and there are always trade-offs: ideas that are succinct and obvious in representations constructed in one representational system can become verbose and confusing when encoded as a representation in another representational system.

Example 3.4. Addition is commutative, and this can be shown by induction (assuming addition is already shown to be associative).[23] For some fixed a and induction variable k, the base case when $k = 0$ is trivial: $a + 0 = a = 0 + a$. Assume $a + k = k + a$. The recursive case is also straightforward:

$$\begin{aligned} a + (k+1) &= (a+k) + 1 && \text{(By associativity)} \\ &= (k+a) + 1 && \text{(By induction hypothesis)} \\ &= k + (a+1) && \text{(By associativity)} \\ &= k + (1+a) && \text{(By induction hypothesis)} \\ &= (k+1) + a && \text{(By associativity)} \end{aligned}$$

This is quite a lot of work for a simple property of addition. Instead, we could represent natural numbers as dots, and present the obviously generalisable solution:



That is, for a row of $a + b$ dots, it does not matter if we split the row a on the left and b on the right, or b on the left and a on the right. Formally, this generalises with the constructive ω-rule [Baker et al. 92],[24] but this is trivial for the human visual reasoning system. So to prove commutativity of addition on natural numbers, a dot representational system is superior

[22] We consider only *external* representations: those that can be shared between people. *Internal* representations—those inside people's minds—are an important class of representations, but are outside the scope of this dissertation. We discussed them briefly in Section 2.2.

[23] This proof is only for natural numbers, but let us keep it simple!

[24] The constructive ω-rule is similar to induction, broadly stating that if you have a procedure to generate a proof for *any* $n \in \mathbb{N}$, the then proof holds for *all* $n \in \mathbb{N}$.

to an algebraic representational system. Obviously, to state that the dots system is superior to the algebraic system is *not* universally true. ▽

We must capture a representational system in our framework to determine which is appropriate to solve problems. Thus for representational systems we create *RS-descriptions*, which contain the components describing the associated representational system; similarly for representations we create *R-descriptions* [Raggi et al. 20-A]. In practice, an RS-description can be thought of as the union of multiple R-descriptions: each are sets of components, one is just 'bigger.' In R-descriptions, components *may* have the attribute 'occurrences,' while components in an RS-descriptions must *not* have this attribute: RS-descriptions are describing the abstract *system*, not a specific representation instance. We have already seen the occurrences attribute: this captures how often a representation uses a particular component. For example, the R-description of $x + y + z$ has the components

$$\begin{aligned}\text{primitive} + : \{\, &\text{type} \coloneqq \text{number} \times \text{number} \to \text{number};\\ &\text{occurrences} \coloneqq 2 \,\}\end{aligned}$$

and

$$\text{primitive}\, x : \{\text{type} \coloneqq \text{number}; \text{occurrences} \coloneqq 1\}$$

(and others) whereas in the RS-description of algebra the same components are included without the occurrences attribute.

Definition 8 (RS-description)**.** An *RS-description* is a set of components that are conceptually grouped as part of the same representational system. Components must not include an 'occurrences' attribute.

Definition 9 (R-description)**.** An *R-description* is a set of components that are conceptually grouped as part of the same representation. The components may have an 'occurrences' attribute.

Descriptions are sets, so each component is in the description at most once. Occurrences allow for a sort of indirect multiplicity. We could not simply allow multiplicity through multisets—including a component in the set more than once—as this introduces two ambiguities: we cannot include components with zero occurrences (a potentially useful ability to capture background understanding), and we cannot distinguish between 'occurs once' and 'occurs some unknown number of times,' the latter of which we can encode by not including *any* occurrences attribute. By annotating components with this attribute, we allow for more expressive descriptions that better reflect the representation being described.

A particular subset of representations are *problems*, representations that have a specific purpose: to present specific information that must be used to derive new information, or verify existing information. We introduce the *Q-description*,[25] an R-description equipped with an *informational importance* function (*importance*, for brevity).[26] Importance reflects the necessity of a piece of information in understanding and solving

[25] Q for 'question.'

[26] In practice, Q-descriptions do not consist of components, instead they consist of importance-component pairs.

the problem represented by the representation. The value of importance ranges from 0 to 1, where 0 is irrelevant and 1 is essential. In our $x+y+z$ example we say the + primitive has importance 1: that is, the + is essential to the problem. The task of assigning importance is challenging; currently, analysts assign importance while constructing descriptions, which we shall expand on in Section 5.1.1.

Definition 10 (Q-description)**.** A *Q-description* q is an R-description equipped with an importance function $\mathrm{importance}_q$[27] that maps each component to a real value in the interval $[0,1]$. That is, for a given R-description r, we define the Q-description q derived from r as the pair $(r, \mathrm{importance}_q)$ where

$$\mathrm{importance}_q : r \rightarrow [0,1]$$

is defined by an analyst.

Descriptions, in a practical version of our framework, exist in two states: as a textual data format that we work with, and as a data structure within the `robin` software. These are equivalent in that one can be converted to the other without losing information, but are conceptually slightly different. We shall consider each in turn, beginning with the data structure because it is simpler, and then turning to the data format.

3.2.2 *Descriptions as a data structure*

The data structure backing descriptions is the set. Components have no inherent order, nor should the same component be listed twice (where 'the same' means equal kind and value).[28] By using a set, we are able to quickly find common components between representations (intersection), add and remove components, as well as determine if components are present (contains and subset).

Within the set, components are stored as a triple consisting of a kind, a value, and an attribute list.[29] The kind is a datatype

```
datatype Kind = Primitive | Type | Pattern | Law | Tactic;
```

and so is limited to exactly these values. A value is more complex, because it can contain more types of data. We break it down into five main categories:

```
datatype Value = Label of string
               | Number of int
               | Boolean of bool
               | Type of Type.T
               | Raw of string;
```

where `Type.T` is our custom 'type' datatype. While components should contain only base 'types' (i.e., strings) using a consistent 'type' datatype allows us to operate directly with the value at other places in the code.

Attributes are a list of 'associations' that we break into six categories:

[27] We drop the q subscript when the context is obvious or unnecessary.

[28] Types are considered equal up to *unification*.

[29] Remember that in Q-descriptions we store importance-component *pairs*.

```
datatype Attribute = Type of Type.T
                   | Holes of Type.T M.multiset
                   | Primitives of string list
                   | NumFunction of string * real[30]
                   | StringFunction of string * string[31]
                   | Feature of string;
```

We have seen the first three before: these are the type, holes, and primitives attributes from patterns. The holes' multiset type collects the types that fill the holes, but retains the multiplicity of each type. The final three are more generic, taking an arbitrary label and associating it with a number, a string, or nothing.

Using these types and a set functor, we are able to efficiently encode and operate on descriptions. This gives us an interface to develop against when algorithmically recommending suitable representational systems.

3.2.3 *Descriptions as a format*

We represent descriptions in a plain text format.[32] A file typically contains a single description (either an RS-, R-, or Q-description) that lists the components. There are also specific commands to import and extend existing descriptions. The syntax is based on Standard ML.

A description is declared with a `representation` block, where the assigned name is the name of the representational system.

```
representation Algebra = rep
    import primitives from LatinAlphabet;
    import terms as primitives from RealNumerals;
    ...
end;
```

The first few lines are 'imports', showing how this description is built on existing descriptions. The first import lifts the letters directly from the `LatinAlphabet` representational system; the second import lifts the numbers—which are terms, which are fully-instantiated patterns—from the `RealNumbers` representational system, but in this representational system they are primitives.

Next come the component declarations. Components with the same kind can be written on the same line. We group all components with the same attributes:

```
types integer, real, number, bool, formula, proof;
primitives |, =, >, <
    where type = number * number -> bool;
```

These are converted into the expected components, for example

$$\text{primitive} < : \{\text{type} := \text{number} \times \text{number} \to \text{bool}\}$$

was derived from the < in the description. This extends to larger structures where there are multiple attributes; for example, the earlier function application pattern component could be written:

[30] E.g., `occurrences := 4`

[31] E.g., `registration := icon`

[32] We present here an idealised version of our data format. The actual implementation has syntactic differences.

```
pattern functionApplication
    where holes = ['a: 1, 'a -> 'b: 1],
          primitives = [(,)],
          type = 'b;
```

In this way a complete representational system or representation can be described in a machine-readable way.[33]

Analysts use five specific keywords to assign importance:[34] `essential`, `instrumental`, `relevant`, `circumstantial`, and `noise`. In our $x + y + z$ example, the primitive $+$ is essential:

```
essential primitive +
    where type = number * number -> number,
          occurrences = 2;
```

These keywords correspond to values ranging from 1 down to 0.

3.2.4 *Moving between descriptions*

RS-descriptions, R-descriptions, and Q-descriptions are very closely related, so the obvious question arises as to whether we can transform one to the other. The answer is *partially*: we can go from an RS-description to an R-description (although it would lack occurrences), but we may need many R-descriptions to form an RS-description.

By creating an R-description from an RS-description, we are specifying the representation of a specific statement as it would appear using a representational system. By analogy to human language, we are creating a specific phrase by using particular aspects of the language. Much like we do not use every word in the language to write a single sentence, we do not use every component in the RS-description to create an R-description. By selecting just a subset of the components, we specialise the description to be about a specific instance. R-descriptions include an occurrences attribute: analysts[35] must add these to each component.

Conversely, we cannot necessarily create an RS-description from a single R-description.[36] Instead, we must union many R-descriptions that describe representations that belong to one representational system. The union of the descriptions creates a more comprehensive description of the representational system: the more diverse the R-description components are within that one representational system, the more comprehensive the subsequent RS-description will be. Because RS-descriptions do not have the occurrences attribute this is discarded from the R-descriptions when unioning.

Moving between R- and Q-descriptions is a matter of stripping or including an importance function—or more concretely, pairing each component with its importance.

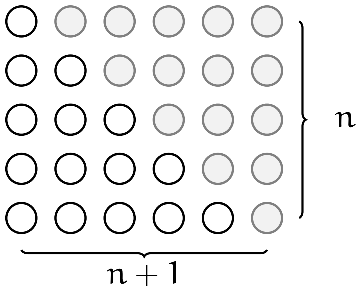
Example 3.5. Consider the following representation of the sum of consecutive integers (a fragment of Figure 1.1).

[33] Note that type variables like α are written `'a`.

[34] In practice, we have found it simpler to work with five discrete levels of importance, rather than a real number from 0 to 1.

[35] Or automated tooling, eventually.

[36] We *could*, but it would be a very poor RS-description with limited applicability.



We have a rectangle of dots, which we divide in two diagonally, forming two triangles. Thus we have a component of the 'dot-diagram' representational system:

$$\begin{aligned}\text{pattern dot-triangle} : \{\ & \text{type} \coloneqq \text{dot-arrangement};\\ & \text{holes} \coloneqq [(\text{dot}, O(n))]\ \}\end{aligned}$$

This pattern component would exist within the RS-description; to relate it to *this* representation, we add the 'occurrences' attribute:

$$\begin{aligned}\text{pattern dot-triangle} : \{\ & \text{type} \coloneqq \text{dot-arrangement};\\ & \text{holes} \coloneqq [(\text{dot}, n)];\\ & \text{occurrences} \coloneqq 2\ \}\end{aligned}$$

That is, we have a triangle of dots *twice*: once in white, once in grey. This component can now exist within an R-description.

Finally, if we bring our representation back to the context of computing the sum of consecutive integers, we can consider this component within a Q-description, where we assign it an importance. Giving the name t to the dot-triangle component, we have

$$\text{importance}(t) = 1.0$$

indicating that the triangle is *essential* to the problem. ▽

SUMMARY OF SECTION 3.2

A set of components all derived from the same source form a description. We use RS-descriptions to capture representational systems, which hold all components that any instance representation might be described using; these components must not have the 'occurrences' attribute. A representation is captured through an R-description, which is primarily a subset of some RS-description, but with the 'occurrences' attribute populated appropriately. Finally, problems are described with a Q-description, which is an R-description with an importance function, assigning each component a value between 0 and 1 based on how critical it is to capturing the problem. In our implementation, we use sets of component triples for R- and RS-descriptions, while we use a set of pairs consisting of the component triple and the importance to encode Q-descriptions. We also provide a textual format for descriptions, which we use as an input mechanism for our implementation. Complete examples using this format are in Appendices F and G, from the extended example in Chapter 6.

3.3 Guiding principles

Components and descriptions allow us to encode representations and representational systems in a flexible way that is suitable for computer processing. But we made many decisions to reach this point, each with benefits and costs. Recall the motivation of this work: we wish to be able to suggest appropriate representation changes between representational systems based on the problem and the person. In this section we discuss the three guiding principles we have followed when designing the rep2rep framework:

- Support for various degrees of system formality;
- Agnostic towards modality; and
- Allow structure to be captured when appropriate.

3.3.1 *Formal and informal systems*

Changing representation is greatly simplified if there is an automated translation between representational systems. For example, spider diagrams are a formal representational system which has been implemented as an interactive reasoning system [Urbas et al. 12]. Spider diagrams are equivalent to a fragment of first-order predicate logic [Stapleton et al. 04], and so can be mechanically translated to this fragment. In general, given equivalent representations in each representational system, we would be able to define measures of suitability at a cognitive level; we would discard a representational system that cannot be translated to, because we have 'perfect' options.

But translations are not simple to construct, even when two representational systems are provably expressively equivalent. Further, many systems do not have a formal definition to compare when evaluating expressiveness. Instead, people construct fuzzy, ad hoc transformations between representations, which are a type of analogy. In addition to formal representational systems such as logics, this is what we aim to capture: informal representations that may or may not be formalisable, and transformations between them.

Rather than fight the informality of most representations, and thus exclude them as candidate re-representation targets, we choose to embrace it when designing our framework. We do not require complete, accurate descriptions of the representation, and instead give a flexible set of tools to describe the representations as they are and as they are used, whether this is logically sound or not. We do not validate the consistency of a representational system, and we do not map it to some underlying logic of our choosing. As a result, descriptions are not unique; the decisions made when creating descriptions influences the final representational system recommendation. While these decisions can mean changes in the output of our algorithm,[37] the changes and subsequent recommendation better reflect the intent of the analyst encoding the representation.

3.3.2 *Mode-agnostic encoding*

Many, and most well understood, representations are *sentential*. That is, they are written as strings of symbols, and are read in a linear fashion:[38] the symbols are read one after the other, and the only rule of construction is juxtaposition (writing the symbols next to each other). Thus sentential representations are simple to describe, and there are many tools for doing so. Further, sentential representations are the 'default' on computing systems.[39] As a result, it is easy to slip into a sentential-first mentality—to the detriment of other modalities, potentially leading to an unfortunate feedback loop where sentential representations are better supported and so get described more frequently so become better supported. Our framework must avoid this sentential-first trap if it is to accurately describe non-sentential representations.

Let us explore when sentential-first assumptions break down. Sentential representations are conceptually *rooted trees*, sometimes explicitly with parentheses, sometimes implicitly through the grammar. This is true whether we consider programming languages,[40] formal languages, or natural languages [Carnie 07]. But trees have a restriction: every child has exactly one parent, and there can be no cycles. In representations, this implies they have a disjoint 'sub-sentence' structure. Such a structure is not universal.

Example 3.6. Consider a matrix with m rows and n columns.

$$\begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{pmatrix}$$

Converting this to a tree, there are two obvious truths: there is one root (the entire matrix), and there are $m \times n$ leaves (the x_{ij} elements). But there is clearly more structure than just a matrix and the elements: there are rows and columns. But each element exists in both a row *and* a column: it has two parents, and neither takes precedence over the other.[41] This produces a parse *graph*, rather than a parse *tree*. Other equally valid decompositions also exist, for example splitting the matrix into quadrants of size $m/2 \times n/2$. There is no one correct way to read the matrix, which is part of the strength of this representation. $\triangledown$

To avoid introducing assumptions that representations might break, we have designed the framework to be flexible. We do not demand

[37] We will address this again in Chapter 5.

[38] For western audiences, we usually read top-left to bottom-right row-first. Many sentential systems do not do this: ancient Greek, for example, could be written boustrophedonically—that is, alternating directions on each line [Sampson 85].

[39] Consider programming languages: visual programming languages are rarer than textual languages.

[40] Lisp is the obvious example: `(+ 1 (* 2 3) (- 4 5))`.

[41] Programming languages must make (often incompatible) matrix bias decisions. For example, C is a 'row-major' language, so a matrix consists of rows of elements; conversely, Fortran is a 'column-major' language, so a matrix consists of columns of elements.

a grammar, nor a consistent type system. Through patterns you can introduce ambiguous grammars and sub-typing. While this restricts our options for automated tools to parse representations, the gains in flexibility mean we have a viable framework for describing a diverse range of representations.

3.3.3 *Degrees of structure*

Finally, we need to capture the *structure* of a representation and representational system. We consider structure to be encoded as *relationships* between components in a description. Structure is extremely diverse, and can be difficult to capture: sometimes it is the way in which terms are composed (the *grammar* structure), while sometimes it is how pieces come together (a *pattern* structure). Structure is a form of *bias*[42] in a representation, which is both helpful and harmful. With no structure (and no bias), the components in the description are unrelated; with too much structure, valid interactions between components are precluded. So how and where do we encode structure to descriptions?

The rep2rep framework provides two means to encode structure: attributes, and patterns. Attributes are used for simpler relationships, the obvious example being typing. If we imagine structure as a graph, then the attributes encode the arcs in the graph. Patterns encode more sophisticated relationships, when many components come together to act as a new unit. In our hypothetical structure graph, patterns are subgraphs. These two tools allow analysts to capture the structure within their representations and representational systems within their Q-, R-, and RS-descriptions.

Example 3.7. Grammar—whether through types or patterns—is the most obvious form of structure, as we outlined in the previous point on mode-agnostic encoding. We use the attribute 'type' to explicitly link a primitive or pattern component to a type component. But there are other varieties of structure. Consider a representation such as a line chart: the relationship between the legend and the plot is not easily encoded through types, but through patterns[43] we can state the relationship quite succinctly:

$$\begin{aligned}\text{pattern keyLineAndLegend} : \{\, &\text{type} \coloneqq \text{relationship};\\ &\text{holes} \coloneqq [(\text{line}, 1), (\text{legend-entry}, 1)]\,\}\end{aligned}$$

That is, there is a relationship between lines and legend entries, which acts as a key to read the chart. The type 'relationship' is a dummy type, existing at the level of the rep2rep framework: it is more for the author of the description than any part of the framework. The significant part is the *holes*: the pattern creates a context in which the two hole types are related. ▽

We defer the decision of how *much* structure is appropriate to the analysts encoding representations and systems. For example, we do not

[42] We use 'bias' in a very general sense; this is a vast topic in machine learning, and we direct the reader to *Machine Learning Bias, Statistical Bias, and Statistical Variance of Decision Tree Algorithms* for further information [Dietterich et al. 95].

[43] A pattern is an arrangement of primitives and terms: the result of the arrangement can be a relationship.

require every primitive to have a type. Instead, we aim to provide a framework that allows analysts to *gradually* add the structure in the representation, as and when it becomes necessary. Similarly to our earlier point about mode-agnostic encoding, we try to make few assumptions about the structure that representations may have. Patterns and attributes are general-purpose structure descriptors, where we encourage analysts to add as many or as few patterns or attributes as necessary. We do not force the use of hierarchy trees or strict well-typed expressions. As we will explore in Chapter 4, we attempt to exploit what structure is present, but do not penalise a lack of structure.

SUMMARY OF SECTION 3.3

Components and descriptions are intended to allow us to consistently catalogue problems, representations, and representational systems that are extremely diverse. We worked to ensure that the framework followed three principles: formal and informal representations and systems were equally supported; no modality (sentential, diagrammatic, or otherwise) was favoured above any other; and the structure of the representation or system could be encoded as loosely or strictly as necessary. This ensures our framework is widely applicable, whether in our specific domain of interest—mathematics education—or more broadly.[44]

3.4 Cognitive properties

The components and descriptions we have outlined so far capture the informational aspects of representation: the grammar, the primitives, the knowledge, and the inference rules. But components are not sufficient to describe how people understand representations: we need to capture aspects of interpretation and understanding. This process of transforming a representation into an *internal* mental model dictates how understandable and effective a representation will be for each specific person.

This section covers material that is parallel to the main point of this dissertation. We provide it as context and motivation for some of the decisions we make. Cognitive properties, and work around including the user, is a prominent focus of concurrent and future research done by the rep2rep research group [Cheng et al. 21].

3.4.1 *Map of cognitive properties*

We wish to capture important aspects of the mental state that a representation allows when being used by particular people.[45] To do this, we consider a representation along two dimensions: a *level of granularity*, a spatial consideration; and *process time-scale*, a temporal consideration [Cheng et al. 21]. Within these two dimensions we identify nine cognitive properties, which we will break down shortly. These properties

[44] We explore how the framework behaves in a different domain in Chapter 6.

[45] This framework was developed in the context of problem solving. Some of the properties, such as solution depth, are not directly applicable in other contexts, but may have analogues (e.g., argument length).

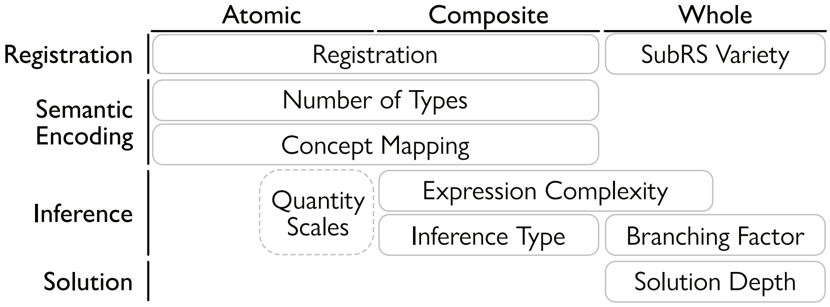


Figure 3.3 The cognitive properties considered by the rep2rep framework. Each column is a representational level of granularity, while the rows are different time scales of cognitive processes.

are positioned according to their *granularity* and their *cognitive process time scale* in Figure 3.3 [Raggi et al. 20-A].

To understand how the use of space impacts the cognitive processes of the user, we consider the *levels of granularity*. This broadly correlates to the level of detail the user is considering, ranging from the specific pieces of the representation, through the structures that emerge, to the big picture of an entire representation.[46]

The second dimension, the cognitive process time scale, uses the approximate time it takes for a process to occur to determine the method through which is occurs [Anderson 02]. For example, reactions (in the sense of reacting instinctively to stimuli) happen in the space of milliseconds, and at a biological level. We start further up the scale, above biological reactions, but still observe such time scale distinctions. Registration processes are recognition tasks that occur in under a second; semantic encoding associates meaning after registration, and takes seconds; inference can take many tens of seconds; and complete solution paths to problems can take minutes[47] [Anderson 02].

Now we look into the individual cognitive properties.

Registration This is the process of identifying and locating a particular term within the representation. The difficulty of this is typically dominated by the method of locating: pop-out effects are faster than using an indexing system,[48] which is in turn faster than search. 'Distractors' also impact registration: similar-appearing terms distract from and delay the registration of the true target term [Alexander et al. 12].

SubRS Variety This determines the 'heterogeneity' of a representation. Representations that mix diverse representational systems exhibit a greater difficulty through context switching and referencing. The more similar the mixed representational systems, the less costly this variety is [Someren et al. 98].

Number of Types Assigning types is part of the semantic decoding of a representation, and many types typically indicates more complex

[46] For a computer science analogy, consider these as layers in a parse tree: the leaves are the individual pieces of a representation, then the internal nodes are the structure, and finally the root node abstracts the entire representation.

[47] Or much longer!

[48] An example of an indexing system is a coordinate grid: you do not need to search every point on the grid, instead you use knowledge of the system to jump to the right place.

semantic processing. Limiting the number of types makes for a simpler representation.

Concept Mapping While the number of types captures how complex the grammar or semantics might be, concept mapping captures how effectively the semantics map to the syntax [Zhang 97]. There are five types of concept mapping: redundancy, where two or more different components of a representation are used for one concept; overload, where one component is used for two or more concepts; excess, where there are components without any associated concept; deficit, where there are concepts without any associated components; and bijection, where one component maps to one concept.

Expression Complexity To understand when a concept is encoded using large expressions in a representation, we estimate how large plausible parse trees of expressions can become: expressions with large parse trees are typically more difficult to understand than expressions with small parse trees.

Inference Type We classify tactics into five different inference types: assign (to give a name to a term), match (to unify two terms), substitute (to replace 'name primitives' with appropriate terms), calculate (to apply domain logic to rewrite terms), and transform (to substitute subterms with terms) [Anderson 02; John et al. 96].[49] The difficulty increases as we move along the scale.

Branching Factor When applying tactics to reach a solution, the user must decide which tactic to apply. This relates to the branching factor of the solution space, where more choices means a more difficult decision on which tactic to apply.[50]

Solution Depth Similarly to branching factor, we need to consider how many steps it may take to reach a solution. The farther away the solution, the more difficult it may be to reach.

Quantity Scales Specifically when dealing with quantitative representations, we can partition numerically-typed terms into four classes: nominal, for incomparable[51] values; ordinal, when the values have an inherent order; interval, when the magnitude between values is meaningful; and ratio, when there is a meaningful multiplicative relationship [Zhang et al. 95].

Cognitive properties have a certain *cost* that is specific to the combination of a representation and the person using the representation. For example, a representation consisting of a simple expression has a low cost for the expression complexity cognitive property, with the cost even lower for expert users of that representation. These costs influence the effectiveness of a representation for a particular user.

[49] Also in unpublished work by Cheng.

[50] We shall explore how human expertise is accounted for in Section 5.2.2.

[51] They *can* be compared for equality, but *only* equality.

Example 3.8. Consider the algebraic notation expression

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}.$$

What might the cognitive properties look like for specific features of this representation?

Registration The individual tokens appear in fixed positions, there is no need to search. For example, we know where the limits are in the summation.

SubRS Variety This is a purely sentential representational system, so there is no 'variety.'

Number of Types This system has a single ground type, integer, which greatly simplifies understanding the representation. The right side also uses only three binary operators, meaning there is just one arity to consider—there is no mix of unary and binary operators. The left side is much more sophisticated, using summation notation, which is ternary over sophisticated types.[52]

Concept Mapping Algebraic notation largely maps a single symbol to a single concept: a 1 is a 1, for example. Algebraic notation does have three components for multiplication: $\times$, $\cdot$, and juxtaposition. This results in some redundancy.

Expression Complexity While this particular expression is quite constrained, algebraic notation can become exceedingly complicated to parse.[53]

Inference Type We are able to apply many inferences in algebraic notation, the most salient being substitute, calculate, and transform. For example, we can transform $n(n+1)$ into $n^2 + n$.

Branching Factor Algebraic notation has a wide branching factor, as there are many possible tactics available at any step. Further, these tactics are not intrinsic to the representational system: the tactic of 'cancelling zeros' in $30/20 = 3/2$ is often generalised to 'cancelling digits' yielding $32/22 = 3/2$, allowing for an even wider branching factor, including incorrect inferences.

Solution Depth Algebraic solutions can be extremely verbose, meaning most solutions are far away. However, powerful tactics can dramatically shorten the solution depth. This depends on the expertise of the solver.

Quantity Scales Algebraic notation, and this expression, deal with ratio values, the most sophisticated quantity scale.[54] ▽

[52] Summation notation has two variants: here we have the ternary variant, which takes a lower bound, an upper bound, and a function of the variable introduced by the binding; a binary variant takes a set and a function on the elements of the set, e.g. $\sum_{x \in X} f(x)$.

[53] Consider the perpetual confusion over operator precedence and BEDMAS/BODMAS/PEMDAS.

[54] Note that although integers are not closed under division, they are still considered to be 'ratio' values because they have a meaningful multiplicative relationship: six is double three, for example.

3.4.2 *Capturing cognitive properties*

We capture the cognitive properties of a representation in three ways: estimate, annotate, and state. When possible, we *estimate* the cost of a cognitive property from the components. Where we cannot estimate the cost of a cognitive property from the appropriate components, we *annotate* the components with attributes that capture the cost of the cognitive properties. When the cognitive property has no suitable component that we can annotate, we must *state* the cost of the cognitive property in the R-descriptions.

Example 3.9. The *expression complexity* cognitive property considers the shape of parse trees for a representation.

Within the components of a representational system, we record primitives (along with their types), and the patterns that compose the primitives into terms. We also have the number of times each primitive and pattern occurs in a representation through the occurs attribute. Thus we can recursively apply the patterns and instantiate them with primitives and other patterns until we have exhausted the number of occurrences.

This procedure gives us an estimate of the breadth and depth of the potential expressions in the representation, from which we can compute the expression complexity cognitive cost. The cost balance between broad-but-shallow and narrow-but-deep expressions will be based on empirical studies. For example, do many levels of grouping help the reader structure their thoughts, or obscure relationships between deeply nested components? ▽

Example 3.10. A pattern defines the context of a primitive, and so the cost to *register* that primitive depends on the pattern. To determine the cognitive cost of registering the primitive, we annotate the patterns which it is used in: the 'registration' attribute has one of four values, increasing in how costly they are:

- emergent, such that gestalt principles support registration;
- spatial index, exploiting coordinates or other positioning;
- notation index, allowing for ordering and keying such as in dictionaries; and
- search, forcing slower registration with limited support.

For example, registering a red dot from a collection of black dots is fast due to the pop-out effect, so the pattern it occurs in is annotated 'emergent'; registering the letter 'S' in a grid of *5*'s is a slow search problem, so the pattern it occurs in is annotated 'search.' ▽

Estimating the cost of cognitive properties is our most favoured approach, as it scales well over many representations and can be updated easily by just tweaking some algorithm for deriving the value. The cognitive properties for which we can estimated costs are number of types, concept

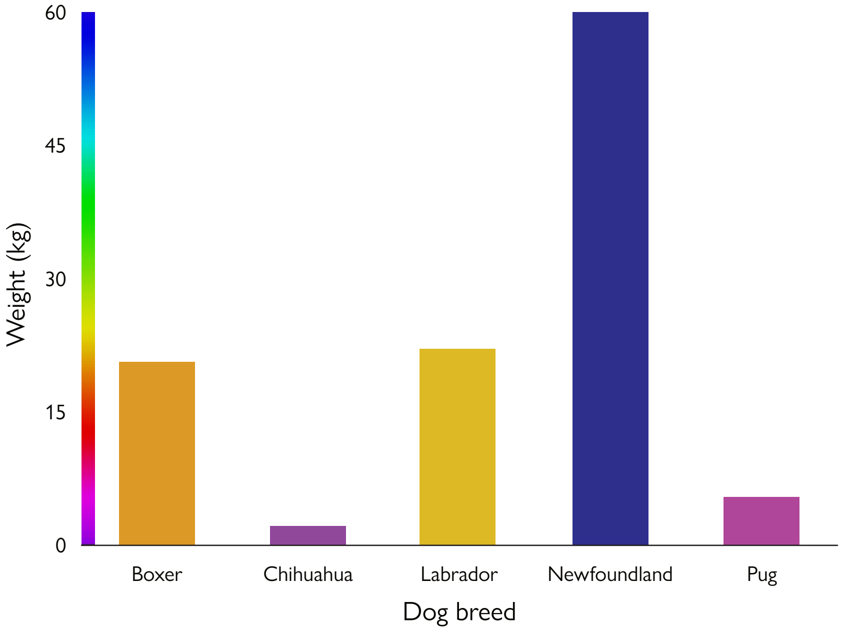


Figure 3.4 Typical weights of dog breeds. This plot uses both colour and height to encode the same information about weights.

mapping, expression complexity, branching factor, solution depth, and quantity scales.[55] The number of types is estimated by counting the number of type-kinded components, while expression complexity, branching factor, and solution depth are estimated by using the existing attributes of pattern and tactic components and calculating how large the resulting instantiations can become. Costs for quantity scales and concept-mapping are estimated by inspecting *correspondences* to a *reference description*; a reference description is taken to be the 'semantic' description: it expresses all and only the concepts for the problem.[56]

Example 3.11. Consider Figure 3.4. This plot is conveying the how much common breeds of dogs weight in a coloured bar chart, but it is using both colour and height for the same information. Against some reference description (for example, a table) we would observe there is *redundancy*: two different components, colour and height, map to the underlying concept of weight. In some reference description, weights would map to exactly one component. ▽

Annotating components with attributes means many cognitive property costs can be recorded in a systematic way. Sometimes, these attributes apply universally, and so are inserted into RS-descriptions. Otherwise, the attributes are inserted into R-descriptions. RS-description attributes are preferred because they are more general. Registration is inferred from attributes on patterns, and inference type from tactic attributes.

One cognitive property remains: subRS variety. The cost of this cognitive property is neither estimable from components, nor suitable as an attribute on any existing components. So we must state the subRS variety in any R-description: we introduce a new component kind 'modes'

[55] Some estimates are more accurate than others. Number of types will be exact, but the expression complexity comes from a heuristic based on the patterns and occurrence count attributes.

[56] Such a description is, in practice, impossible; we request the best possible approximation.

which describes the *heterogeneity* of a representation. The value is an integer from 1 to 6,[57] where 1 is no heterogeneity (the system has exactly one 'modality'), and 6 is extremely heterogeneous. Most representations are 1 or 2: a table filled with formulae would be 2, because it contains a grid, which uses space to encode information, and equations, which are sentential and use symbols and juxtaposition. This then becomes the cost of the subRS variety cognitive property.

SUMMARY OF SECTION 3.4

While components capture the more 'formal' aspects of a representation, cognitive properties capture the interaction between the representation and the human interacting with it. We categorise these cognitive properties along two axes: the granularity of the property, from properties at the atomic level of the representation through to properties over the entire representation; and the time scales of cognitive processes, from sub-second registration processes to multiple-minute solution generation. We consider the cost of the cognitive properties for a specific representation and user, and how we might estimate this cost by building on top of the description components.

SUMMARY OF CHAPTER 3

Between components and cognitive properties, we can construct descriptions of representational systems, representations, and the problems those representations denote. This goes towards answering our first research question, breaking down problems, representations, and representational systems in a way that allows us to describe them consistently. This work forms the 'fifth' contribution of this dissertation, which is also part of the rep2rep research project. The RS-, R-, and Q-descriptions discussed in this chapter capture the details necessary to discuss representations, but we are still short of *comparing* representations. We need to understand how the representations are similar, and subsequently different, to assess the suitability of representations. For this, we introduce *correspondences*.

[57] We have done some work towards identifying six core 'modes,' hence values 1 to 6. The work is largely undeveloped, and outside the scope of this dissertation.

Correspondences 4

Why is a raven like a writing desk?
— *The Hatter (in Lewis Carroll's* Alice in Wonderland*)*

We have seen that representations and representational systems are different, one being specific to a problem, and the other describing a class of representations; both can be described using sets of components. But knowing the representations and systems in isolation is insufficient to make a recommendation: what would *this* problem look like in the *other* representational systems?

This chapter introduces *correspondences*, which intend to capture 'intuitive similarity.' We introduce this informally, then contribute a formalised, realisable definition that we can use to build a representation recommendation system. We also consider how correspondences can be discovered in a semi-automated manner, and how we can construct descriptions of representations we have not seen.

This chapter addresses our second research question: how representational systems, and their components, are similar. It also develops the first novel contribution of this dissertation, correspondences. In Section 4.1 we define correspondences, with more detail on the *strength* of correspondences in Section 4.2. Section 4.3 explores how we can support analysts by interactively discovering new correspondences. Finally, Section 4.4 applies correspondences to construct *pseudo-descriptions*. Most of the work in this chapter was presented at the *IEEE Symposium on Visual Languages and Human-Centric Computing* (VL/HCC) 2020 [Stockdill et al. 20-a] and the workshop on *Explainable Smart Systems for Algorithmic Transparency in Emerging Technologies* (ExSS-ATEC) 2020 [Stockdill et al. 20-b]. The work in Section 4.4 is yet to be published.

4.1 Sameness and component formulae

Correspondences model the 'similarity' of concepts across representations. We examine how different components in distinct representational systems would be considered the same by people, despite having potentially different characteristics—correspondences capture *analogies*. In this section we start with an informal definition, and build up the pieces we need to define what it means for components to correspond, and for representations to satisfy a correspondence. Formally, a correspondence is a triple $\langle a, b, s \rangle$, but this section considers only a and b.

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

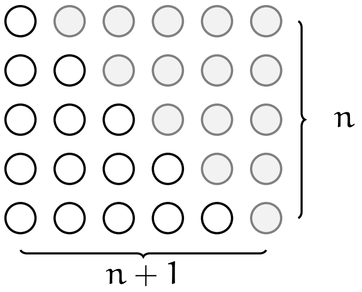


Figure 4.1 This figure is a reproduction of Figure 1.1. Two representations of the 'same' expression in different representational systems: algebra and dot diagrams. The algebraic representation asserts that the sum of integers between 1 and n is equal to the stated quadratic expression. The dot diagram counts the dots in a triangle by vertically stacking rows of dots (the black-edged circles), each one longer than the last, then observing a symmetry to create a rectangle. The annotations assert the generalised size of the rectangle, and so the number of dots in the original triangle is half the number in the rectangle, which itself is the product of the dimensions.

4.1.1 *Sameness and purpose*

What does it mean for two things to be 'the same'? We take a purpose-driven view of sameness: if two things fill equivalent roles in each representation, then they correspond. For example, in Figure 4.1, the dot-arrangements are filling the role of numbers, while stacking is filling the role of adding, and partitioning (by colour, in this case) is filling the role of dividing. The intuition is that these things are 'the same' (have equivalent semantics) in different representational systems.

Let us break down 'purpose' further. The purpose of a component is related to its kind,[1] where primitives and patterns serve to encode ideas, types and patterns provide structure, laws hold truths, and tactics allow change and progress. The boundaries between kinds are not sharp, particularly primitives/patterns and types/patterns, so it is not uncommon to have correspondences between components of different kinds.

A particularly important class of correspondences are those that cross the primitive/pattern boundary, and are valuable in both directions. If a primitive (formally) corresponds to a pattern, this is *decompositional*: the pattern encoding can reveal deeper structure in the 'opaque' primitive; conversely, when patterns correspond to primitives, this is *abstraction*: the details are concealed within a single, higher-level concept. For example, $+$ is a primitive, while stacking is a pattern. This analogical link highlights properties of addition, such as associativity and commutativity, that are not immediately apparent from the symbol $+$, yet are obviously true when stacking dots. Importantly, the same concept is behind both the pattern and the primitive.

[1] For more on *kinds*, see Section 3.1.1.

4.1.2 *Source and target*

We begin by considering the first two of the three pieces that make up a correspondence: the *source* and the *target*. At their simplest, each is a single component: source component $\mathtt{a}$ in RS-description $\mathtt{A}$ corresponds to target component $\mathtt{b}$ in RS-description $\mathtt{B}$. The meaning of these two terms changes slightly depending on the interpretation we assign to correspondences: explanations for existing analogies, or requirements for creating analogies.

Consider correspondences as part of an explanation of an analogy: the source (the $\mathtt{a}$ in $\langle \mathtt{a}, \mathtt{b}, \mathtt{s} \rangle$) is part of an R-description of the content in the original representation; the target (the $\mathtt{b}$) is part of an R-description of the content in the analogous representation. 'Part of a description' is literal: $\mathtt{a}$ and $\mathtt{b}$ are components[2] within R- and Q-descriptions as we defined them in Chapter 3. With two representations for which we have descriptions, we can inspect the correspondences that link them. The correspondences identify which components in each description are related, and how strongly the two components are linked (the strength $\mathtt{s}$). By presenting the set of correspondences between the descriptions to the user—potentially run through tools to improve the user experience, such as natural language generation—the analogy can be *explained*.

Explaining an existing analogy requires both the original and analogous representation descriptions be given. If we do not have the analogous representation, and instead assume a single R- or Q-description as the source, and an RS-description as a *potential* target, we have a different interpretation: the correspondences give *requirements* to the RS-description that it must fill to form an analogy with the original representation. By considering the correspondences from the given R- or Q-description to the RS-description, we can select exactly those components that would be in an R- or Q-description, essentially creating a new R-description.[3]

Example 4.1. Inspecting Figure 4.1, there are 'correspondences' between the representations: for example, numbers are like dot arrangements.[4]

$$\langle \mathtt{type\ number},\ \mathtt{type\ dot\text{-}arrangement},\ 0.9 \rangle.$$

By examining the correspondences that are satisfied[5] by both of these representations, and elucidating the strongest correspondences, we can signal to a reader *how* the representations are similar. The correspondence from numbers to dot arrangements is strong, so we present it in some way to explain the analogy in Figure 4.1.

Alternatively, imagine the second representation, the dot-diagram, was not present. Then the correspondences from the original algebraic representation into the dot-diagram representational *system* carry the informational *requirements* of the representation to the new system. Any components of the original expression that do *not* have analogues in the dot-diagram representational system are 'unsatisfied'—and if in the Q-description they are important, then we know the new representational system is deficient. Using our number correspondence above, we can

[2] Almost; $\mathtt{a}$ and $\mathtt{b}$ are slightly more complex than that, as we will see in Section 4.1.4.

[3] We shall extend this point in Section 4.4.

[4] For now, ignore the 0.9—it just means the correspondence is 'good'.

[5] We define 'satisfaction' in Section 4.1.3.

suggest that an appropriate representation encoded as a dot diagram will use those dots to represent the numbers. ▽

We draw attention here to work by Gentner on analogy, and specifically the focus on *structural alignment* [Gentner 02]. The point of structural alignment is to observe the similarities between not just surface features (such as colour, shape, or size) but how the components work together to form larger structures. Similarly, correspondences aim to capture the larger structural similarity, but *also* the surface-level details. Correspondences act at all conceptual levels of a representation: from primitives to types to patterns to tactics and laws, each moving from a 'simple' analogy to a more complex, structural analogy. We will see later[6] that we use ideas similar to the structure-mapping engine [Falkenhainer et al. 89].

4.1.3 *Satisfiability and covering*

Correspondences and descriptions need to be connected to each other. To form this connection, we define *satisfaction*. For a description to *left-satisfy* a correspondence it must satisfy the *source* component, and similarly to *right-satisfy* a correspondence it must satisfy the *target* component. Thus if the pair of descriptions (A, B) where A left-satisfies a correspondence c and B right-satisfies c, then we state that c is satisfied by (A, B). We delay a formal definition of satisfaction until Section 4.1.4.

We are not (yet[7]) attempting to construct descriptions that satisfy correspondences; we are only checking that existing descriptions satisfy correspondences.[8]

When we attempt to make a representational system recommendation, we will inspect the correspondences between two representational systems. But we do not consider *all* the correspondences: some may not be relevant to the problem at hand. But what if we choose the subset

$$\{\langle a, x, s_1\rangle, \langle a, y, s_2\rangle\}$$

where a is 'covered' twice? *Covering* is the 'dual' operation of satisfying: descriptions can satisfy a correspondence, while correspondences cover a description. A correspondence *left-covers* components that occur in the *source* component; similarly it *right-covers* components that occur in the *target* component. The *covered set* is the set of all components from a description that are either left or right covered by a correspondence.

From a set of correspondences, we preserve the *covering multiplicity*: how many times a component in a description is covered by correspondences in some set of correspondences. If a component has a covering multiplicity greater than one, then there will be some redundancy in our representation. Consider our set with a corresponding to x and y in separate correspondences: the covering multiplicity of a is 2. Do we reward the target representation, because it can encode a really well? This does not make sense: if we can encode a as x, then having an alternative way (y) does not improve the situation.[9] Instead, we say sets

[6] Section 4.3.

[7] See Section 4.4 for pseudo-descriptions.

[8] We can do this very efficiently, at most $O(t)$ time where t is the number of terms in a disjunctive normal form component formula (see next section): this is *not* the SAT problem.

[9] Here we are speaking *informationally*. Cognitively, it might make things worse! See *excess* concept mapping in Section 3.4.1.

of correspondences are minimally redundant and maximally covering (MRMC) if:

- the number of times a is covered in the correspondence subset is as small as possible (minimal); and
- as many components as possible (maximal) from the source representation are covered by the set of correspondences.

Such a set may not uniquely exist, so we must approach this as an optimisation problem. We optimise the covering condition first, and the redundancy condition second. We shall revisit this concept in Section 5.3.

4.1.4 *Component formulae*

Relationships between representations are rarely so simple as to be accurately captured by a correspondence from one component to another single component. Thus we allow the correspondence source and target to be component *formulae*, rather than components alone, meaning we have correspondences between *combinations* of components.

Example 4.2. Within our dots example, we can think about how summation relates to stacking. Within the rules of our dot representational system, we may stack dot arrangements horizontally or vertically—it does not matter which is chosen generally, as both are the same as summing integers.[10] Thus we can say that summation corresponds to stacking horizontally *or* stacking vertically. The correspondence is

$$\langle\, \text{primitive} \textstyle\sum,\ \text{pattern stack-horizontal } \textsc{or} \text{ pattern stack-vertical},\ 1\,\rangle,$$

indicating that either or both types of stacking is sufficient to capture the same information as summation. ▽

The three connectives of correspondence formulae are AND, OR, and NOT. These broadly behave as expected:

- AND denotes that both components are required to capture the same information as the correspondence target;
- OR denotes that one or both components are sufficient to capture the same information as the correspondence target; and
- NOT denotes that this component precludes the correspondence. For example, $\langle\, \textsc{not}\ a,\ x,\ s\,\rangle$ is only left-satisfied if component a is *not* in the source description, while x *must be* in the target description.

The connectives can be combined following the grammar below when describing a correspondence, but we always consider the formula to be in *disjunctive normal form*:[11] a formula consists of a disjunction of clauses (at the top level, there are only OR connectives), clauses are conjunctions

[10] A specific representation might use one or the other for a particular reason, but within the representational *system* they are equivalent.

[11] This makes the theory and implementation simpler, and the `robin` implementation always makes this transformation.

of terms (there will be no OR connectives within AND connectives), and terms consist of only components or negated components (in disjunctive normal form, the NOT can only occur in front of components, not formulae). As an example: $(w \text{ OR } x)$ AND NOT$(y \text{ OR } z)$ would have to become $(w$ AND NOT y AND NOT $z)$ OR $(x$ AND NOT y AND NOT $z)$ to be in disjunctive normal form.

Definition 11 (Component formulae)**.** A *component formula* is a set of components joined by AND, OR, and NOT connectives. The grammar is:

$$\begin{aligned} \mathit{formula} &\coloneqq \mathit{component} \\ &\mid \textsc{not}(\mathit{formula}) \\ &\mid (\mathit{formula})\ \textsc{and}\ (\mathit{formula}) \\ &\mid (\mathit{formula})\ \textsc{or}\ (\mathit{formula}) \end{aligned}$$

Parentheses may be dropped when there is no ambiguity. The order of precedence has NOT binding tightest, then AND, and finally OR. A formula in disjunctive normal form follows a different grammar:

$$\begin{aligned} \mathit{formula} &\coloneqq \mathit{clause} \\ &\mid (\mathit{formula})\ \textsc{or}\ (\mathit{formula}) \\ \mathit{clause} &\coloneqq \mathit{term} \\ &\mid (\mathit{clause})\ \textsc{and}\ (\mathit{clause}) \\ \mathit{term} &\coloneqq \mathit{component} \\ &\mid \textsc{not}\ \mathit{component} \end{aligned}$$

As before, parentheses may be dropped when the formula is unambiguous; the order of precedence is the same.

We earlier delayed a formal definition of satisfaction, because we must consider the case where we have component formulae forming the source or target of our correspondence. We can now define satisfaction.

Definition 12 (Satisfaction)**.** Assume that component formula f is in disjunctive normal form. Then f is *satisfied* by a description d if, for any clause t in f, every non-negated component c in t (that is, the component is not preceded by a NOT connective) is present in d, and no negated component c' in t is present in d. We thus define

$$\mathrm{sat}_f(d) \Leftrightarrow \exists_{t \in \mathrm{clauses}(f)} [\forall_{c \in \mathrm{positive}(t)} c \in d \wedge \forall_{c' \in \mathrm{negative}(t)} c' \notin d]$$

where $\mathrm{clauses}(f)$ is the set of clauses in f, $\mathrm{positive}(t)$ is the set of non-negated components in term t, and $\mathrm{negative}(t)$ is the set of negated components.

A description satisfies a component formula if all the non-negated components[12] —and none of the negated components—in at least one clause of the formula are present in the description. We always work with formulae in disjunctive normal form.

[12] Component formulae are in disjunctive normal form, so the NOT operator only appears in front of components.

Let us consider this more informally. The connective AND behaves as would be expected: if either component is missing from a description, then this component formula is not satisfied. The NOT connective is also straight-forward: if the component being 'negated' *is* present in a description, then this component formula is not satisfied.

Example 4.3. The description consisting of $\{a, b, c, d\}$ would satisfy the formulae

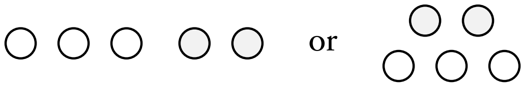
$$(a \text{ AND } b) \text{ OR } e \quad \text{and} \quad (a \text{ AND NOT } e) \text{ OR } (b \text{ AND } f),$$

but it would not satisfy the formulae

$$a \text{ AND } e \quad \text{or} \quad (a \text{ AND NOT } b) \text{ OR NOT } d. \qquad \triangledown$$

When considered as a single correspondence, the OR connective follows expected semantics: if either or both of the components is in a description, then this component formula is satisfied. But the OR connective has more complex behaviour when we consider sets of correspondences. Then the two correspondences $\langle a, b, s \rangle$ and $\langle a, b', s \rangle$ seem similar to the single correspondence $\langle a, b \text{ OR } b', s \rangle$.[13] But when a, b, and b' are matched, something interesting happens: with the two correspondences, the 'strength' of the relationship between the two descriptions s is considered *twice*; with a single correspondence, the strength s is considered only once. The semantic difference here is whether two is better than one—is it better to have b *and* b', or is having both no better than having either? We decide by considering whether the components would still occur independently in descriptions: if b and b' are independent *given* a, then have two correspondences; otherwise, have only one correspondence describing the relationship between a and both b and b'.

Example 4.4. Returning to our earlier Example 4.2, in algebra, we can sum values: $\sum x$, for example. Using dot notation, we can 'stack' arrangements together:



Note we had a choice: we could stack the dots horizontally, or we could stack the dots vertically. To associate these with addition, we might chose to have these be two correspondences:

$$\langle \text{primitive} \textstyle\sum, \text{ pattern stack-horizontal}, 1 \rangle$$

and

$$\langle \text{primitive} \textstyle\sum, \text{ pattern stack-vertical}, 1 \rangle.$$

But in this case, the two types of stacking are not independent: either is sufficient to encode summation. Having *both* patterns of stacking is not 'better' than having *either* pattern. They are not independent given

[13] Similarly for $\langle a, b, s \rangle$ and $\langle a', b, s \rangle$ versus $\langle a \text{ OR } a', b, s \rangle$.

summation: if we are exploiting the ability to stack horizontally, we can also exploit the ability to stack vertically. Thus, we prefer the combined correspondence using the connective OR:

$$\langle \text{primitive} \sum, \text{ pattern stack-horizontal OR pattern stack-vertical}, \ 1 \rangle$$

ensuring that having both is not 'better' than having either. ▽

SUMMARY OF SECTION 4.1

We can combine the *source* component formula and the *target* component formula to express a relationship: that these two things are analogous. For representations we say that a correspondence aims to capture how information can be re-encoded across representations. One caveat of this relationship is that it is not 'all-or-nothing': the information can be preserved to a certain degree, ranging from perfect down to completely lost. We need a way to capture the *strength* of this relationship.

4.2 Strength

Correspondences are triples, and we have now explored the *source* and *target* parts; we now introduce the third and final part of the correspondence triple, *strength*. The strength of a correspondence is a measure of how 'good' the correspondence is: a value close to 0 means the correspondence is poor, while a value close to 1 means the correspondence is excellent.

4.2.1 *Component probabilities*

We define the strength of a correspondence in terms of the probability of component formulae. To simplify, we begin with considering correspondences only over components, not formulae.

In an RS-description, there are many components. In each R- and Q-description, we have only a subset of these components, and how often they occur in these descriptions reflects how often the concepts encoded by these components occur generally within representations. Thus a component has some baseline probability of occurring within an R-description, and whether it occurs or not is binary: it is there, or it is not. Thus we model components as Bernoulli random variables,[14] such that they are either present in an R-description or they are absent from an R-description, and equip an RS-description with a function $\Pr(\cdot)$ which assigns to each component a probability of being present in any R-description.

But how do we define the output of the probability function? This is a difficult, ongoing problem. Presently, we assign the probability of a component being present heuristically. This has the advantage of requiring no other resources to compute, but does reflect biases of the

[14] A Bernoulli random variable takes on only two values, one with probability p and the other with $1-p$.

experts setting the heuristic values.[15] Alternatively, we could define the probability in a more direct, frequentist manner. Consider a dataset of triples (r, p, d) where r is a label for a representational system, p is a label for a specific problem, and d is an R-description for problem p using representational system r.[16] If we partition the dataset by r, each partition contains descriptions of many problems, each encoded in the same representational system; if we partition instead on p, each partition contains many descriptions of the 'same' problem, but encoded in many different representational systems.

Consider first a partition that gives us all descriptions within each representational system: then each component has a probability of occurring in a random description: this is the baseline probability, and thus for a component a we have $\Pr(a)$. We do *not* assume that the components are independent: the probability of combinations of components would need to be captured as well; this could be stored in a Bayesian network. Thus for the component formula a AND a' we can compute $\Pr(a \text{ AND } a')$.

If instead we consider the partition that gives us the 'same' problem in different representational systems, we can calculate different probabilities. Given that component a has occurred in an R-description which encodes a representation from representational system A, we can calculate how often component b occurs in other R-descriptions encoding representations from representational system B. Thus we have the conditional probability $\Pr(b \mid a)$ *across* representational systems.

Let us consider this more rigorously, and more generally. Consider for some problem we have Q-description q_i which is in written in some representational system with RS-description R. This problem can be *transformed* into the 'same' problem in alternative representational systems. This is more general than *translations*, which are provably equivalent.[17] This set of transformed problems starting from q_i is given by $T(q_i)$. We restrict this set to be only those Q-descriptions which are instantiations of some new R-description R' such that

$$T_{R'}(q_i) = \{q_j \in T(q_i) \mid q_j \text{ is an instantiation of } R'\}.$$

Note that this is still a *set*, because there can be many equally valid ways to encode the same problem into a new representational system. Assuming some predicate $\mathrm{sat}_a(q_i)$ is true when the component a is present in q_i,[18] then we have

$$\Pr(b \mid a) = \frac{\sum_i \left|\left\{q'_j \in T_{R'}(q_i) \mid \mathrm{sat}_a(q_i) \wedge \mathrm{sat}_b(q'_j)\right\}\right|}{\sum_i \left|\left\{q'_j \in T_{R'}(q_i) \mid \mathrm{sat}_a(q_i)\right\}\right|} \tag{4.1}$$

where we sum over all problems q_i that contain component c. As before, this generalises to component formulae.

Now that we have probabilities defined over components, we extend this to component formulae. Because of how we define component formulae using AND, OR, and NOT, the operations from probability map

[15] In this case, the experts are also the rep2rep research group.

[16] Note that d could also be a Q-description, but we have no need of importance so consider R-descriptions.

[17] Defining 'same' is difficult; we take as reference the judgement of a reasonable human expert.

[18] We could write $a \in q_i$, but we keep it general as a can be extended to component formulae.

exactly as one expects: AND is equivalent to $\cap$, OR is equivalent to $\cup$, and NOT is equivalent to $\overline{}$. The full details how these sets of connectives relate to each other, and the construction required to make this happen, are in Appendix A.

4.2.2 *Defining strength*

Strength is intended to capture how well two different groups of components fill the same role in different representations. We can approximate this by how well the presence of one component formula based on the R- or Q-description for the original representation *predicts* that the other formula will be satisfied by a description for an analogical representation, or how confident you would be in observing a set of components that satisfy $\mathtt{b}$ given that you have observed a set of components that satisfy $\mathtt{a}$. The conditional probability of $\mathtt{b}$ *given* $\mathtt{a}$ is almost what we want, but we need to also consider how likely the probability of $\mathtt{b}$ was *regardless* of $\mathtt{a}$: is the conditional probability higher, or lower; and by how much? This gives us a *change* in probability of $\mathtt{b}$ based on the observation of $\mathtt{a}$; if $\mathtt{b}$ is more likely, this value is large; if $\mathtt{b}$ is no more likely, then this value is zero.[19] Thus we have $\Pr(\mathtt{b} \mid \mathtt{a}) - \Pr(\mathtt{b})$, the change in likelihood of $\mathtt{b}$ when observing $\mathtt{a}$. But this value also has problems: probabilities are bounded between 0 and 1, meaning an already likely component $\mathtt{b}$ has very little 'room to grow'. Thus we scale the change by the *potential* change: the upper bound on probabilities is 1, so the largest possible change is $1 - \Pr(\mathtt{b})$. We normalise by $1 - \Pr(\mathtt{b})$ so that the strengths are comparable: if the probability increased to fill half of the remaining 'head room', we make the strength 0.5, regardless of the original probability. Thus the strength of a correspondence is as in Definition 13.

Definition 13 (Correspondence strength)**.** We define the strength of a correspondence $\langle \mathtt{a}, \mathtt{b}, \mathtt{s} \rangle$ as

$$\mathtt{s} = \frac{\Pr(\mathtt{b} \mid \mathtt{a}) - \Pr(\mathtt{b})}{1 - \Pr(\mathtt{b})} \tag{4.2}$$

where $\Pr(\cdot)$ is the probability function for a representational system.[20] We assume that neither $\Pr(\mathtt{a})$ nor $\Pr(\mathtt{b})$ are 0 or 1: if either are 0, then the component formula is never satisfied by any description, and thus is not worth considering in correspondences; if either are 1, then we have a component formula that will be satisfied in every description, which carries no information (in both the informal sense and the information theory sense) and thus not worth considering in correspondences. We also assume $\Pr(\mathtt{b} \mid \mathtt{a}) \geq \Pr(\mathtt{b})$—that is, the component $\mathtt{b}$ is more likely to occur in the analogous description if we already know that $\mathtt{a}$ is in the original description.

Example 4.5. Consider two components $\mathtt{a}$ and $\mathtt{b}$ such that $\mathtt{a}$ *guarantees* $\mathtt{b}$—that is, if $\mathtt{a}$ is a component of the source representation description, then $\mathtt{b}$ *must* be a component in any analogous representation description.

[19] We will handle the case where $\mathtt{b}$ is *less* likely shortly.

[20] Covered in the previous subsection.

Then we have that $\Pr(b \mid a) = 1$. By the definition of correspondence strength,

$$\begin{aligned} s &= \frac{\Pr(b \mid a) - \Pr(b)}{1 - \Pr(b)} \\ &= \frac{1 - \Pr(b)}{1 - \Pr(b)} = 1 \end{aligned}$$

assuming $\Pr(b) \neq 1$. That is, a correspondence $\langle a, b, 1 \rangle$ means that a guarantees b. ▽

Note that a correspondence with NOT is *not* the same as a 0 strength correspondence:[21] these concepts are orthogonal. A strength of 0 means there is no relationship between the source and target of the correspondence; a component x being absent from a description (that is, NOT x) might *strongly* indicate some component formulae will be satisfied in the target representation.

Example 4.6. Consider again components a and b, but now a *has no bearing* on b—if a is a component of the source representation description, then have no new information on whether b will be present in any analogous representation descriptions. Then $\Pr(b \mid a) = \Pr(b)$, and by the definition of correspondence strength,

$$\begin{aligned} s &= \frac{\Pr(b \mid a) - \Pr(b)}{1 - \Pr(b)} \\ &= \frac{\Pr(b) - \Pr(b)}{1 - \Pr(b)} = 0 \end{aligned}$$

assuming again that $\Pr(b) \neq 1$. Thus the correspondence $\langle a, b, 0 \rangle$ means that a and b are unrelated. ▽

Example 4.7. Take two components, n = type number in the algebraic representation, and d = type dot-arrangement in the dots representation. Assume that from a dataset we have computed $\Pr(n) = 0.86$ and $\Pr(d) = 0.9$, but we also have that $\Pr(d \mid n) = 0.99$. Then the strength of the correspondence $\langle n, d, s \rangle$ is

$$\begin{aligned} s &= \frac{\Pr(d \mid n) - \Pr(d)}{1 - \Pr(d)} \\ &= \frac{0.99 - 0.9}{1 - 0.9} \\ &= 0.09/0.1 = 0.9, \end{aligned}$$

and so we have the correspondence

$$\langle \text{type number}, \ \text{type dot-arrangement}, \ 0.9 \rangle. \qquad ▽$$

A similar logic applies when b becomes *less* likely after observing a. If component formula b has become less likely after observing component

[21] We will introduce strength in the following section.

formula a, then we again want the difference—but reversed, so the value is positive—and divide this by the 'room to fall'; in this case, potentially down to zero. Converting this to a formula, we have $\Pr(b) - \Pr(b \mid a)$ in the numerator, and normalise by the potential *drop*: a probability is bounded below by 0, so we normalise by $\Pr(b) - 0 = \Pr(b)$. Thus the strength of an 'inverse' correspondence (one in which a component formula becomes less likely) is

$$s = \frac{\Pr(b) - \Pr(b \mid a)}{\Pr(b)}.$$

We can rewrite $\Pr(b)$ as $1 - \Pr(\textsc{not}\, b)$ and $\Pr(b \mid a)$ as $1 - \Pr(\textsc{not}\, b \mid a)$, assuming that component formula connectives behave the same as set connectives.[22] This yields

$$\begin{aligned} s &= \frac{(1 - \Pr(\textsc{not}\, b)) - (1 - \Pr(\textsc{not}\, b \mid a))}{1 - \Pr(\textsc{not}\, b)} \\ &= \frac{\Pr(\textsc{not}\, b \mid a) - \Pr(\textsc{not}\, b)}{1 - \Pr(\textsc{not}\, b)} \end{aligned}$$

which is the strength of the correspondence between a and NOT b. Thus a separate definition for inverse correspondences is unnecessary; we can instead redefine the correspondence to be between a and NOT b, rather than a and b.

Example 4.8. Consider our components a and b again, but now a *precludes* b—if a is a component of the source representation description, then b *never* occurs in analogous representation descriptions. Then we have $\Pr(b \mid a) = 0$. Applying the definition of strength naïvely yields

$$\begin{aligned} s' &= \frac{\Pr(b \mid a) - \Pr(b)}{1 - \Pr(b)} \\ &= \frac{-\Pr(b)}{1 - \Pr(b)} < 0 \end{aligned}$$

violating the assumption that $0 \leq s \leq 1$. Instead, we consider what it means for a to correspond to NOT b: taking that $\Pr(\textsc{not}\, b) = 1 - \Pr(b)$ and $\Pr(\textsc{not}\, b \mid a) = 1 - \Pr(b \mid a) = 1$, we have

$$\begin{aligned} s &= \frac{\Pr(\textsc{not}\, b \mid a) - \Pr(\textsc{not}\, b)}{1 - \Pr(\textsc{not}\, b)} \\ &= \frac{1 - \Pr(\textsc{not}\, b)}{1 - \Pr(\textsc{not}\, b)} = 1 \end{aligned}$$

when $\Pr(b) \neq 0$. So we have a perfect correspondence between a and NOT b. If instead we consider the formula stated earlier for when a component becomes less likely, we have

$$\begin{aligned} s &= \frac{\Pr(b) - \Pr(b \mid a)}{\Pr(b)} \\ &= \frac{\Pr(b) - 0}{\Pr(b)} = 1 \end{aligned}$$

[22] See Appendix A.

again assuming $\Pr(b) \neq 0$. So the correspondence $\langle\, a,\ \textsc{not}\, b,\ s \,\rangle$ reflects the same strength as we would expect from a definition for 'inverse' correspondences. ▽

Finally, with strength defined, we are ready to complete the definition of a correspondence.

Definition 14 (Correspondence)**.** We define a correspondence to be

$$\langle\, a,\ b,\ s \,\rangle \tag{4.3}$$

where a is the *source*, b is the *target*, and s is the *strength*. The source and target a and b are *component formula*, while s is a real number between 0 and 1.

From the point of view of representations, rather than components and descriptions, correspondences capture how information can be preserved between representations, and the degree to which that information is preserved. They do so by linking together representational systems, expressing relationships that transcend a single pair of representations. Correspondences are a central contribution of this dissertation.

Correspondence describe an *implication* relationship: if this, then that. Except correspondence strength changes this subtly: if this, then *maybe* that. There is an asymmetry from the definition of strength between a corresponding to b and b corresponding to a;[23] this similarly occurs with implication, where $\alpha \to \beta$ does not necessarily mean $\beta \to \alpha$.

But implication does exhibit *contraposition*: if $\alpha \to \beta$, then $\neg\beta \to \neg\alpha$. For correspondences: if we have $\langle\, a,\ b,\ s \,\rangle$, then we must also have $\langle\, \textsc{not}\, b,\ \textsc{not}\, a,\ s \,\rangle$.[24]

Theorem 1 (Correspondences exhibit contraposition)**.** *If, for any component formulae* a *and* b*, we have the correspondence* $\langle\, a,\ b,\ s \,\rangle$*, then the correspondence* $\langle\, \textsc{not}\, b,\ \textsc{not}\, a,\ s' \,\rangle$ *has the same strength; that is,* $s = s'$.

Proof. Using $\langle\, \textsc{not}\, b,\ \textsc{not}\, a,\ s' \,\rangle$, by the definition of strength we have

$$\begin{aligned} s' &= \frac{\Pr(\textsc{not}\, a \mid \textsc{not}\, b) - \Pr(\textsc{not}\, a)}{1 - \Pr(\textsc{not}\, a)} \\ &= \frac{(1 - \Pr(a \mid \textsc{not}\, b)) - (1 - \Pr(a))}{\Pr(a)} \\ &= \frac{\Pr(a) - \Pr(a \mid \textsc{not}\, b)}{\Pr(a)}. \end{aligned} \tag{$*$}$$

[23] This also makes sense before we consider strength: just because b can fill every role a does, does not mean a can fill every role b does.

[24] Note the same s.

Eliminating the NOT b in the condition is slightly more complicated.

$$\begin{aligned}
\Pr(a \mid \textsc{not}\, b) &= \frac{\Pr(a \textsc{ and not } b)}{\Pr(\textsc{not}\, b)} \\
&= \frac{\Pr(a \textsc{ and not } b)}{1 - \Pr(b)} \\
&= \frac{\Pr(\textsc{not}\, b \mid a) \cdot \Pr(a)}{1 - \Pr(b)} \\
&= \frac{(1 - \Pr(b \mid a)) \cdot \Pr(a)}{1 - \Pr(b)}
\end{aligned}$$

Substituting this back into (*) we get

$$\begin{aligned}
s' &= \frac{1}{\Pr(a)} \cdot \left(\Pr(a) - \frac{\Pr(a) \cdot (1 - \Pr(b \mid a))}{1 - \Pr(b)} \right) \\
&= 1 - \frac{1 - \Pr(b \mid a)}{1 - \Pr(b)} \\
&= \frac{(1 - \Pr(b)) - (1 - \Pr(b \mid a))}{1 - \Pr(b)} \\
&= \frac{\Pr(b \mid a) - \Pr(b)}{1 - \Pr(b)} = s
\end{aligned}$$

where s is the strength of the correspondence $\langle\, a,\ b,\ s \,\rangle$. Thus the strength of the contrapositive of the correspondence is equal to the strength of the correspondence. □

4.2.3 *Alternative measures*

Our definition of strength does not exist in the literature. But two alternative measures need exploring: mutual information, and Kullback-Leibler divergence. This subsection makes use of concepts from Shannon's information theory[Shannon 48], which we briefly cover here for clarity. An outcome of an event has a certain amount of *information* associated with it, and the amount is related to how *surprising* the outcome is. A likely outcome has a small amount of information, while an unlikely outcome has a large amount of information. The information content of outcome x is

$$I(x) = -\log \Pr(x).$$

The logarithm is often taken to be base-2, and so the measure of information content is *bits*. If we consider a random variable X which has many possible outcomes x_i, then we need to consider all the different outcomes together. This is the *entropy* of a random variable,

$$H(X) = -\sum_{x_i} \Pr(X = x_i) \log \Pr(X = x_i)$$

which is the *expected* information content of all outcomes. Both information content and entropy can extend to be *conditional*:

$$I(x \mid y) = -\log \Pr(x \mid y)$$

and

$$H(X \mid y) = -\sum_{x_i} \Pr(X = x_i \mid y) \log \Pr(X = x_i \mid y),$$

where we are considering x and X, respectively, given the already known outcome y. For a more thorough treatment of information theory, see 'Elements of Information Theory' [Cover et al. 05].

Mutual information [Cover et al. 05] captures the shared information content of two random variables. Equivalently, it tells us how much information we have about a second random variable after observing the first. One definition is in terms of entropy:

$$I(a; b) = H(b) - H(b \mid a) \tag{4.4}$$

where I is the mutual information, and H is the (possibly conditional) entropy.

We immediately see similarities to our definition of strength, where we consider the difference between a measure of b and a measure of b given a. But the similarities end there, and there is one significant drawback: mutual information is *symmetric*. That is, $I(a; b) = I(b; a)$, such that the strength of $\langle a, b, s \rangle$ is the same as $\langle b, a, s \rangle$. This breaks an intuition of correspondences: they are not necessarily symmetric.[25]

In Equation (4.4), mutual information was defined in terms of entropy. It could be defined in terms of the Kullback-Leibler (KL) divergence [Kullback et al. 51; Cover et al. 05]

$$I(a; b) = D_{KL}(\Pr_{a \cap b} \parallel \Pr_a \cdot \Pr_b)$$

where $D_{KL}(\cdot \parallel \cdot)$ is the KL divergence, and $\Pr_X$ is the probability distribution for random variable X. That is, the mutual information of a and b is the divergence between their product (the 'independent' conjunction) and their conjunction probability (the true conjunction). The KL divergence is defined as

$$D_{KL}(P \parallel Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)} \tag{4.5}$$

where P and Q are distributions; we are measuring the divergence of Q from P.[26]

Because mutual information is inappropriate, perhaps we can use the KL divergence differently. KL divergence is asymmetric, so we ensure that the parameters P and Q are asymmetric in their parameters: the resulting divergence measure will then be asymmetric. Instead of calculating the divergence between $\Pr(a \cap b)$ and $\Pr(a)\Pr(b)$, we can calculate the divergence between $\Pr(b \mid a)$ and $\Pr(b)$. Thus we can define a modified strength s_{KL}:

$$s_{KL} = \frac{D_{KL}(\Pr_{b|a} \parallel \Pr_b)}{I(b)}. \tag{4.6}$$

[25] Why are they not symmetric? Consider dot-arrangements and numbers. Every dot-arrangement is a number, but not every number is a dot-arrangement—π, for example. Thus we expect the correspondence from dot-arrangements to numbers to be stronger than the correspondence from numbers to dot-arrangements. We considered this for Theorem 1, contrapositivity, and we will see this again in Section 4.3.2.

[26] In a Bayesian learning context, KL divergence is often used to measure the information gain by moving from a model Q to a model P.

We introduce the normalisation by $I(b)$, the information content of b: we assume a and b behave as Bernoulli random variables, so the KL divergence between $\Pr(b \mid a)$ and $\Pr(b)$ is between 0 and $I(b)$; normalising by $I(b)$ bounds the s_{KL} between 0 and 1.

But s_{KL} loses nice properties that we have with strength. First, inversion is different: if the probability of b goes down after observing a, KL divergence still produces a positive value. But there is no obvious relationship between the s_{KL} of the correspondence between a and b to the s_{KL} between a and $\textsc{not}\, b$. Second, we no longer have contraposition: there is no obvious relationship between $\langle a, b, s \rangle$ and $\langle \textsc{not}\, b, \textsc{not}\, a, s' \rangle$. These are two intuitive properties of correspondence which do not exist when using s_{KL}.

Example 4.9. Consider two component formulae, a and b, such that $\Pr(a) = 0.7$, $\Pr(b) = 0.8$, and $\Pr(b \mid a) = 0.85$. We can compute $\Pr(\textsc{not}\, a) = 0.3$, $\Pr(\textsc{not}\, b) = 0.2$, $\Pr(\textsc{not}\, b \mid a) = 0.15$, and $\Pr(\textsc{not}\, a \mid \textsc{not}\, b) = 0.475$.

Using Definition 13, we compute the strength of the correspondence $\langle a, b, s \rangle$ as

$$\begin{aligned} s &= \frac{\Pr(b \mid a) - \Pr(b)}{1 - \Pr(b)} \\ &= \frac{0.85 - 0.8}{1 - 0.8} = 0.25. \end{aligned}$$

The contrapositive correspondence, $\langle \textsc{not}\, b, \textsc{not}\, a, s' \rangle$ has strength

$$\begin{aligned} s' &= \frac{\Pr(\textsc{not}\, a \mid \textsc{not}\, b) - \Pr(\textsc{not}\, a)}{1 - \Pr(\textsc{not}\, a)} \\ &= \frac{0.475 - 0.3}{1 - 0.3} = 0.25 \end{aligned}$$

which is equal to s.

Now we perform the same computation, but using s_{KL} from Equation (4.6). First, we compute $D_{KL}(\Pr_{b|a} \,||\, \Pr_b)$:

$$\begin{aligned} D_{KL}(\Pr_{b|a} \,||\, \Pr_b) &= \sum_b \Pr(b \mid a) \log \frac{\Pr(b \mid a)}{\Pr(b)} \\ &= 0.85 \log_2 \frac{0.85}{0.8} + (1 - 0.85) \log_2 \frac{1 - 0.85}{1 - 0.8} \\ &\approx 0.0121 \end{aligned}$$

We then compute the normalising factor $I(b)$:

$$I(b) = -\log_2 \Pr(b) \approx 0.3219$$

So the final strength is $s_{KL} \approx 0.0375$. (This is incomparable to the above values.)

Consider now the contrapositive strength s'_{KL}. As before we start by computing the Kullback-Leibler divergence:

$$\begin{aligned} D_{\mathrm{KL}}(\Pr_{\textsc{not}\ a \mid \textsc{not}\ b} \parallel \Pr_{\textsc{not}\ a}) &= \sum_{a} \Pr(\textsc{not}\ a \mid \textsc{not}\ b) \log \frac{\Pr(\textsc{not}\ a \mid \textsc{not}\ b)}{\Pr(\textsc{not}\ a)} \\ &= 0.475 \log_2 \frac{0.475}{0.3} + 0.525 \log_2 \frac{0.525}{0.7} \\ &\approx 0.0970 \end{aligned}$$

Then, computing the normalising factor $I(\textsc{not}\ a)$:

$$I(\textsc{not}\ a) = -\log_2 \Pr(\textsc{not}\ a) \approx 1.7370$$

So the final strength is $s'_{\mathrm{KL}} \approx 0.0559$, which is not the same as s_{KL}. So using strength as defined in Definition 13 ensures correspondences are contrapositive, but if we were to use the alternative definition of strength proposed in Equation (4.6), we would lose this property. ▽

Equation (4.6) has one further drawback which relates to the next section: it lacks many of the nice properties we will use to discover correspondences and strengths. We would lose the ability to compose strengths without needing to know the conditional distribution of the constituent component formulae; we would also lose the ability to reverse correspondences with a simple odds multiplication.[27] By needing to recompute strength from first principles, we lose *encapsulation*: we must continually return to the artefacts which defined strength, rather than the strength itself, to operate on correspondences. This complicates reasoning about strengths and correspondences.

We will continue to use the definition in Definition 13 for two reasons:

- Strength is asymmetric; and
- Strength abstracts the underlying conditional probability.

Based on this analysis, neither mutual information nor Kullback-Leibler divergence are appropriate.

summary of section 4.2

We have defined a correspondence as a relationship capturing the ability for two component formulae to fill the same roles in their respective descriptions. The degree of similarity is captured by the *strength* of the correspondence. Strength is defined in terms of the probability of an R-description containing a set of components that satisfy the component formula, and can be derived from a dataset of R-descriptions. Correspondences are contrapositive, and their strengths are bound between 0 and 1, allowing direct strength comparisons.

[27] 'Odds' are a ratio: odds for an event with probability p is defined as $p/(1-p)$; odds against is $(1-p)/p$.

4.3 Discovering correspondences

Correspondences are central to understanding how representational systems relate to each other, and we shall use them extensively when constructing our representation recommendation system. Ensuring a high-quality set of correspondences is vital: missing or excessive correspondences can skew the results one way or another. We take steps[28] to avoid these situations, with the goal of producing high quality recommendations.

In this section, we will describe four rules which allow us to automatically suggest new correspondences: *identity*, for linking components which are the same; *reversal*, constructing the correspondence $\langle b, a, s' \rangle$ from $\langle a, b, s \rangle$; *composition*, for chaining multiple correspondences together; and *relation*, incorporating the wider context of descriptions and components into discovering new correspondences.

4.3.1 *Identity*

The rule of *identity* states that a component formula corresponds with itself perfectly: $\langle a, a, 1 \rangle$. We use the notation from natural deduction to visually represent our rules:

$$\frac{a \equiv b}{\langle a, b, 1 \rangle}\ [\textsc{idy}]$$

That is, if $a \equiv b$, then a perfectly corresponds to b. We define $\equiv$ to be true when there is a one-to-one mapping between the components in a and b where the kind and value of each component pair are the same, and the associated formula is equivalent.

Identity serves as a starting point for future derivations, and allows us to construct correspondences even when no 'seed set' is available.[29] But the rule only applies in limited cases where we have the same components; this is appropriate for naturally overlapping systems such as variants of algebra and symbolic manipulation systems, but is less appropriate when linking representational systems of different modalities.

4.3.2 *Reversal*

If there is a relationship between a and b, then there is some relationship between b and a. Thus, the rule of *reversal*. In our natural deduction notation, we have

$$\frac{\langle a, b, s \rangle}{\langle b, a, s' \rangle}\ [\textsc{rev}]$$

Note the modified strength s', rather than the original strength. Much like in classical logic where $a \rightarrow b$ does not tell us $b \rightarrow a$, the strength of the correspondence between a and b is not necessarily the strength of the correspondence between b and a.

The modified strength s' of the reversed correspondence is not unknowable: using Definition 13, we can derive the strength s' from s.

[28] We consider automatically deriving correspondences in this section; selecting minimally redundant and maximally covering correspondence sets is covered in Section 5.3.

[29] The 'seed set' is the set of correspondences provided by the analyst. Correspondences from this set provide the premises for all the other discovery rules until further correspondences are derived to build upon.

Theorem 2 (Correspondence strength reversal). *Given the correspondence* $\langle \mathtt{a},\ \mathtt{b},\ \mathtt{s} \rangle$, *the strength of the reversed correspondence* $\langle \mathtt{b},\ \mathtt{a},\ \mathtt{s}' \rangle$ *is*

$$\mathtt{s}' = \mathtt{s} \cdot \frac{\Pr(\mathtt{a})}{1 - \Pr(\mathtt{a})} \cdot \frac{1 - \Pr(\mathtt{b})}{\Pr(\mathtt{b})}.$$

Proof. For $\langle \mathtt{b},\ \mathtt{a},\ \mathtt{s}' \rangle$, by Definition 13 we have that

$$\mathtt{s}' = \frac{\Pr(\mathtt{a} \mid \mathtt{b}) - \Pr(\mathtt{a})}{1 - \Pr(\mathtt{a})}.$$

From Definition 13, $\Pr(\mathtt{a}), \Pr(\mathtt{b}) \notin \{0, 1\}$. Further, by Bayes' Theorem,

$$\Pr(\mathtt{a} \mid \mathtt{b}) = \Pr(\mathtt{b} \mid \mathtt{a}) \cdot \Pr(\mathtt{a}) / \Pr(\mathtt{b}).$$

Thus

$$\begin{aligned}
\mathtt{s}' &= \frac{\Pr(\mathtt{b} \mid \mathtt{a}) \cdot \Pr(\mathtt{a}) / \Pr(\mathtt{b}) - \Pr(\mathtt{a})}{1 - \Pr(\mathtt{a})} \\
&= (\Pr(\mathtt{b} \mid \mathtt{a}) / \Pr(\mathtt{b}) - 1) \cdot \frac{\Pr(\mathtt{a})}{1 - \Pr(\mathtt{a})} \\
&= \frac{\Pr(\mathtt{b} \mid \mathtt{a}) - \Pr(\mathtt{b})}{\Pr(\mathtt{b})} \cdot \frac{\Pr(\mathtt{a})}{1 - \Pr(\mathtt{a})} \\
&= \frac{\Pr(\mathtt{b} \mid \mathtt{a}) - \Pr(\mathtt{b})}{1 - \Pr(\mathtt{b})} \cdot \frac{1 - \Pr(\mathtt{b})}{\Pr(\mathtt{b})} \cdot \frac{\Pr(\mathtt{a})}{1 - \Pr(\mathtt{a})} \\
&= \mathtt{s} \cdot \frac{\Pr(\mathtt{a})}{1 - \Pr(\mathtt{a})} \cdot \frac{1 - \Pr(\mathtt{b})}{\Pr(\mathtt{b})}
\end{aligned}$$

as required. □

The reversed strength $\mathtt{s}'$ is the original strength $\mathtt{s}$ multiplied by the *odds for* $\mathtt{a}$, and the *odds against* $\mathtt{b}$. 'Odds for' is a measure of the likelihood of something occurring, while 'odds against' is the likelihood of something not occurring. Thus the strength of the reversed correspondence is the strength of the forward correspondence multiplied by the likelihood of the original *source* component being present, and the likelihood of the original *target* not being present; under the reverse perspective, we multiply by the odds against the *new source*, and the odds for the *new target*. One interpretation of this is that by reversing direction, we are 'undoing' one odds bias and replacing it for the opposite.

Example 4.10. We see this kind of asymmetric relationship frequently between representational systems when one is more powerful than the other. For example, every dot-arrangement is equivalent to a real number, but not every real number is equivalent to a dot-arrangement—it thus makes sense for the correspondence from the type component for dot-arrangements to the type component for numbers to be much stronger than that from numbers to dot-arrangement. Take the following example probabilities we could compute from a dataset, where $\mathtt{n}$ is the type number component and $\mathtt{d}$ is the type dot-arrangement component:

$$\begin{aligned}
\Pr(\mathtt{n}) &= 0.86\\
\Pr(\mathtt{d}) &= 0.9\\
\Pr(\mathtt{d} \mid \mathtt{n}) &= 0.99\\
\Pr(\mathtt{n} \mid \mathtt{d}) &= \Pr(\mathtt{d} \mid \mathtt{n}) \cdot \Pr(\mathtt{n}) / \Pr(\mathtt{d}) = 0.946
\end{aligned}$$

That is, we have the prior and cross-representational-system component probabilities for both $\mathtt{n}$ and $\mathtt{d}$. To disambiguate the strengths, we use $s_{\mathtt{n}\to\mathtt{d}}$ for the strength of the correspondence from $\mathtt{n}$ to $\mathtt{d}$. The strength of the correspondence $\langle \mathtt{n},\ \mathtt{d},\ s_{\mathtt{n}\to\mathtt{d}} \rangle$ is

$$\begin{aligned}
s_{\mathtt{n}\to\mathtt{d}} &= \frac{\Pr(\mathtt{d} \mid \mathtt{n}) - \Pr(\mathtt{d})}{1 - \Pr(\mathtt{d})}\\
&= 0.09/0.1\\
&= 0.9.
\end{aligned}$$

Similarly for $\langle \mathtt{d},\ \mathtt{n},\ s_{\mathtt{d}\to\mathtt{n}} \rangle$:

$$\begin{aligned}
s_{\mathtt{d}\to\mathtt{n}} &= \frac{\Pr(\mathtt{n} \mid \mathtt{d}) - \Pr(\mathtt{n})}{1 - \Pr(\mathtt{n})}\\
&= 0.086/0.14\\
&\approx 0.614.
\end{aligned}$$

Now applying [REV] to $s_{\mathtt{n}\to\mathtt{d}}$ we have

$$\begin{aligned}
&s_{\mathtt{n}\to\mathtt{d}} \cdot \frac{\Pr(\mathtt{n})}{1 - \Pr(\mathtt{n})} \cdot \frac{1 - \Pr(\mathtt{d})}{\Pr(\mathtt{d})}\\
&= 0.9 * (0.86/0.14) * (0.1/0.9)\\
&\approx 0.614 = s_{\mathtt{d}\to\mathtt{n}}
\end{aligned}$$

as expected. So applying the [REV] rule computes the same correspondence strength as would be computed directly from the dataset that the original probabilities are drawn from. ▽

4.3.3 *Composition*

If $\mathtt{a}$ corresponds to $\mathtt{b}$, and $\mathtt{b}$ corresponds to $\mathtt{c}$, then $\mathtt{a}$ corresponds to $\mathtt{c}$; this chaining of correspondences is *composition*.

$$\frac{\langle \mathtt{a},\ \mathtt{b},\ s \rangle \quad \langle \mathtt{b},\ \mathtt{c},\ s' \rangle}{\langle \mathtt{a},\ \mathtt{c},\ s \cdot s' \rangle}\ [\textsc{cmp}]$$

For example, if the type component for dot-arrangements corresponds to the type component for numbers, and the type component for numbers corresponds to the type component for arcs in a graph, then the type component for dot-arrangements corresponds to the type component for arcs.

Because correspondences are between component formulae, and not just between components, we generalise the [CMP] rule by loosening the restriction on b: the two bs are not necessarily the same, but b_1 must logically imply b_2. Thus [CMP] becomes

$$\frac{\langle a,\, b_1,\, s\rangle \quad b_1 \rightarrow b_2 \quad \langle b_2,\, c,\, s'\rangle}{\langle a,\, c,\, s \cdot s'\rangle}\ [\textsc{cmp}]$$

So we can chain correspondences between formulae safely even when the 'joining' formulae are not equivalent.

In this generalised [CMP] rule, we assume $b_1 \rightarrow b_2$ is purely syntactic on components; that is, we treat components as symbols, not pieces of a representation. The most typical use for this is 'discarding conjunctions': $b_1 = x$ AND y, while $b_2 = x$, so we can use this rule to discard the unnecessary y component. It is possible there are implications between component formulae based on the interpretations of their underlying representations, but because these might not actually be realised in the representations, we do not use them.

Example 4.11. Take b_1 to be two dots, ∘∘, and b_2 to be a single dot, ∘. We *do not* have that $b_1 \rightarrow b_2$ in the case of [CMP], because the implication is *not* syntactic on components. The implication relationship between b_1 and b_2 is part of the representational system, not part of our language of component formulae.

Conversely, if b_1 were ∘ and b_2 were ∘ OR ⋆, then we *do* have $b_1 \rightarrow b_2$: the implication exists in our language of component formulae. ▽

With composition, the power of strength from Definition 13 becomes apparent. The strength of a composition of correspondences is the product of the individual correspondence strengths: naïvely chaining together correspondences produces monotonically non-increasing strengths, so that the further you walk from the 'start,' the weaker your strength becomes. We first show this is true for the simple case without implication.

Lemma 3 (Correspondence strength composition)**.** *Given the correspondences $\langle a,\, b,\, s\rangle$ and $\langle b,\, c,\, s'\rangle$ such that the probability that a and c are satisfied in their respective descriptions is independent given that b is satisfied in its description, then the correspondence obtained through composition $\langle a,\, c,\, s''\rangle$ has strength $s'' = s \cdot s'$.*

Proof. This is a sketch of the proof. We lay out the complete argument in Appendix B.

By assuming a and c are independent given b, we have

$$\Pr(a \text{ AND } c \mid b) = \Pr(a \mid b) \cdot \Pr(c \mid b).$$

By the definition of correspondence strength, we have that

$$s'' = \frac{\Pr(c|a) - \Pr(c)}{1 - \Pr(c)},\ s = \frac{\Pr(b|a) - \Pr(b)}{1 - \Pr(b)},\ \text{and } s' = \frac{\Pr(c|b) - \Pr(c)}{1 - \Pr(c)}$$

By marginalising over b we can show that

$$\Pr(\mathsf{c} \mid \mathsf{a}) = \frac{\Pr(\mathsf{c}) + \Pr(\mathsf{c} \mid \mathsf{b}) \Pr(\mathsf{b} \mid \mathsf{a}) - \Pr(\mathsf{c}) \Pr(\mathsf{b} \mid \mathsf{a}) - \Pr(\mathsf{c} \mid \mathsf{b}) \Pr(\mathsf{b})}{1 - \Pr(\mathsf{b})}.$$

Substituting this into the definition of s'' we have

$$\begin{aligned} \mathsf{s}'' &= \frac{\Pr(\mathsf{c} \mid \mathsf{b}) - \Pr(\mathsf{c})}{1 - \Pr(\mathsf{c})} \cdot \frac{\Pr(\mathsf{b} \mid \mathsf{a}) - \Pr(\mathsf{b})}{1 - \Pr(\mathsf{b})} \\ &= \mathsf{s}' \cdot \mathsf{s} \end{aligned}$$

as required. □

This can generalise to component formulae related by implication, preserving the result that strengths multiply under composition.

Theorem 4 (Correspondence strength composition of formulae)**.** *Given the correspondences* $\langle\, \mathsf{a},\ \mathsf{b}_1,\ \mathsf{s} \,\rangle$ *and* $\langle\, \mathsf{b}_2,\ \mathsf{c},\ \mathsf{s}' \,\rangle$, *where* $\mathsf{b}_1 \rightarrow \mathsf{b}_2$ *and* a *and* c *are conditionally independent given* b_2, *then the composed correspondence* $\langle\, \mathsf{a},\ \mathsf{c},\ \mathsf{s}'' \,\rangle$ *has strength* $\mathsf{s}'' = \mathsf{s} \cdot \mathsf{s}'$.

Proof. To allow for the case where $\mathsf{b}_1 \not\equiv \mathsf{b}_2$, we construct a correspondence $\langle\, \mathsf{a},\ \mathsf{b}_2,\ \mathsf{s} \,\rangle$. Because b_2 is a consequence of b_1, we have $\Pr(\mathsf{b}_2 \mid \mathsf{b}_1) = 1$, and so $\langle\, \mathsf{b}_1,\ \mathsf{b}_2,\ 1 \,\rangle$. Further, and for the same reason, we have that

$$\Pr(\mathsf{a} \text{ AND } \mathsf{b}_1 \text{ AND } \mathsf{b}_2) = \Pr(\mathsf{a} \text{ AND } \mathsf{b}_1).$$

So a and b_2 are conditionally independent given b_1:

$$\begin{aligned} \Pr(\mathsf{a} \text{ AND } \mathsf{b}_2 \mid \mathsf{b}_1) &= \frac{\Pr(\mathsf{a} \text{ AND } \mathsf{b}_1 \text{ AND } \mathsf{b}_2)}{\Pr(\mathsf{b}_1)} \\ &= \frac{\Pr(\mathsf{a} \text{ AND } \mathsf{b}_1)}{\Pr(\mathsf{b}_1)} \\ &= \Pr(\mathsf{a} \mid \mathsf{b}_1) = \Pr(\mathsf{a} \mid \mathsf{b}_1) \cdot \Pr(\mathsf{b}_2 \mid \mathsf{b}_1). \end{aligned}$$

We derive $\langle\, \mathsf{a},\ \mathsf{b}_2,\ \mathsf{s} \,\rangle$ from $\langle\, \mathsf{a},\ \mathsf{b}_1,\ \mathsf{s} \,\rangle$ and $\langle\, \mathsf{b}_1,\ \mathsf{b}_2,\ 1 \,\rangle$ using Lemma 3.

From $\langle\, \mathsf{a},\ \mathsf{b}_2,\ \mathsf{s} \,\rangle$ and $\langle\, \mathsf{b}_2,\ \mathsf{c},\ \mathsf{s}' \,\rangle$ we can derive $\langle\, \mathsf{a},\ \mathsf{c},\ \mathsf{s} \cdot \mathsf{s}' \,\rangle$, again by Lemma 3. □

In Lemma 3, we assumed that the component formulae a and c are independent given b (and similarly in Theorem 4 for b_2). This simplifying assumption allows us to multiply strengths, but we should question it: are a and c really independent given b? Likely, no. Further, we cannot conclude whether the result is an upper or lower bound on the true strength between a and c—if there is an underlying reason that relates all of a, b, and c, then the computed strength will be an underestimate; if there is some otherwise unknown preclusion between a and c that does not effect b, then the computed strength will be an overestimate. However, the product serves as a good approximation in the case where better estimates (either from datasets or a human expert) are not available.

4.3.4 *Relation*

The final rule is the most powerful, but least rigorous. We call this the rule of *relation*, because it inspects the relationships between components and 'copies' correspondences between them. We begin with the binary relation rule,

$$\frac{R(a,x) \quad R(b,y) \quad \langle a,\ b,\ s\rangle}{\langle x,\ y,\ s\rangle}\ [\textsc{rel}]$$

That is, if a relates to x and b relates to y, and a corresponds to b, then perhaps x relates to y.[30]

Example 4.12. The key example of using the relation inference rule is in typing; that is, $R(\cdot,\cdot)$ is the relationship 'has-type'. If 2 has type number, and ∘∘ has type dot-arrangement, and we know[31]

$$\langle \text{primitive}\, 2,\ \text{primitive}\, \circ\circ,\ 0.9\rangle,$$

then we infer that

$$\langle \text{type number},\ \text{type dot-arrangement},\ 0.9\rangle.$$

The 'has-type' relationship comes from the attributes of the components: both components have the same attribute (type), the content of which is another component, so we have a relation between components which we can derive automatically. In this case, we copy the strength of the original correspondence: this is a heuristic we justify below. ▽

This 'contextual' inference looks to extract correspondences not from the components themselves, but from the context they occur in using the correspondences between *other* components. It is an appeal to *abductive* reasoning: given the overall relationship and the extant correspondences, the best explanation is that all the components correspond. Under this interpretation the generalised rule becomes more obvious. We want to consider how a collection of components—with some already in correspondence—can inform the correspondences for the remaining components in that context. The generalised rule of relation is

$$\frac{\begin{array}{c} R(x_1,\dots,x_n) \quad R(y_1,\dots,y_n) \\ \forall_{i\in\mathcal{R}}\left[\langle x_i',\ y_i',\ s_i\rangle \wedge (x_i \to x_i') \wedge (y_i' \to y_i)\right] \end{array}}{\forall_{j\notin\mathcal{R}}\langle x_j,\ y_j,\ s'\rangle}\ [\textsc{rel}]$$

where

$$s' = \frac{1}{n-1}\sum_{i\in\mathcal{R}} s_i$$

and $\mathcal{R}$ is the subset of indices from 1 to n such that the ith x-y pair correspond.[32]

When determining the derived strength s', we must consider how the relation and given correspondences influence the strength of the

[30] The rule is equally valid if a/x and b/y swap—we can derive a correspondence on the first arguments to R just as we can on the second.

[31] Omitting attributes for space.

[32] They technically do not need to correspond, only satisfy the condition wrapped in the 'for all' in the hypotheses of [REL]. Much like in composition, syntactic implication lets us work with component formulae.

resulting correspondences. There is little way to know a priori what the strengths will be, so we discount the strengths based on the evidence available. That is, if we 'know' many of the correspondences *and* the known correspondences are strong, then the resulting correspondences will be strong. Conversely, if we know very few of the correspondences *or* the known correspondences are not strong, the resulting correspondences will not be strong. So the derived strength s' is not necessarily equal to any of the hypotheses' strengths; instead, it is an average of all hypotheses' strengths, weighted such that if few of the components correspond then the strength is lower, while if many of the components correspond then the strength is higher. Thus the normalising factor is $1/(n-1)$ rather than $1/|\mathcal{R}|$.

The [REL] rule is a *heuristic*, rather than a theorem like the first three. It is possible that the estimated strength does not match what would be computed from any dataset. Indeed, the derived correspondences themselves have a chance of not being attested in any representations: applying our typing example in reverse can yield incorrect results. Thus this rule is best paired with domain-specific 'filtering' functions which eliminate spurious correspondences: for example, in practice we apply a 'no-duplicates' filter to ensure that components already in a one-to-one correspondence does not appear in *another* one-to-one correspondence. This remains largely future work, and will necessarily need to be tailored to the domain that correspondences are being applied to.

Context and relationships are the basis of *structure* in representational systems, and in agreement with Gentner we find the most potent correspondences emerge when we consider the representations 'in the large' [Gentner 83]. By considering the components in their contexts—which we provide through patterns and attributes—we can structurally align the components. These deeper, structural correspondences drive analogy and discovery better than surface correspondences [Gentner 02].

SUMMARY OF SECTION 4.3

Together these four rules allow for automatic correspondence discovery. Analysts tasked with finding correspondences can find this a challenge, as the space of *potential* correspondences is huge and difficult to conceptualise. We have constructed a tool, `findcorr`, which allows for analysts to provide *seed* correspondences, and then interactively accept, accept with different strength, or reject correspondences suggested by the tool. Large correspondence sets emerge, filtered by a human to avoid bad correspondences; the analyst can add more correspondences (perhaps inspired by `findcorr`'s suggestions) to give the program more correspondences to work with. We will use this tool in Chapter 6.

4.4 Pseudo-descriptions

Correspondences act as a knowledge base of 'analogical building blocks,' telling us that a representational system is likely to be capable of encoding some of the ideas from the original representation. But we do not necessarily know what the new representation, in the target representational system, is going to look like. To address this, we create *pseudo-descriptions*—approximations of what the R-description of the analogous representation would look like. These allow us to consider the cognitive cost[33] of a representation that we do not necessarily have, or do not have an R- or Q-description for. Pseudo-descriptions are *approximations* of real descriptions, and give us options for working with descriptions that analysts have not yet created.

4.4.1 *Constructing pseudo-descriptions*

A pseudo-description is constructed in four steps, from a given initial R-description $\mathtt{r}$ for the representation for which we want an analogy, and an RS-description $\mathtt{R}$ for the target representational system.

1. Collect the appropriate set of correspondences $\mathtt{C}$ between $\mathtt{r}$ and $\mathtt{R}$ such that as much of $\mathtt{r}$ is covered as possible, and there is as little redundancy as possible for each description.[34] We use an arbitrary strength threshold of $\mathtt{0.5}$ to prevent weak correspondences contributing components to the pseudo-description that are less surely present.

2. Select the right-covered set of components from $\mathtt{R}$: this is the set of components that make up the target R-description $\mathtt{r}'$.

3. To each component in $\mathtt{r}'$ we must add the 'occurrences' attribute. we 'carry this through the correspondence': if, in correspondence $\langle \mathtt{a}, \mathtt{b}, \mathtt{s} \rangle$, $\mathtt{a}$ has $\mathtt{n}$ occurrences, then we assign $\mathtt{n}$ occurrences to $\mathtt{b}$. This is a heuristic that is likely wrong, but gives us *some* occurrences value to work with.

4. If the initial $\mathtt{r}$ is in fact a Q-description, we carry through importance to $\mathtt{r}'$ as well. For correspondences where the left component formula is a single component, this is trivial. For non-trivial component formula, we take the maximum importance of all left-covered components from the correspondence, excluding those that are preceded by NOT in disjunctive normal form of the left component formula. As with 'occurrences,' this is a heuristic designed to give us *some* importance, even if it is not a perfect importance.

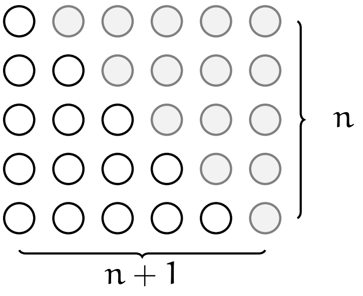
Component formulae complicate this procedure slightly: the target of a correspondence might be a formula, rather than a component. As a result, constructing pseudo-descriptions becomes NP-hard: we are attempting to select a set of components that satisfy the component formulae, which is a SAT problem. Fortunately, most of the formulae

[33] See Section 5.2.2.

[34] This is the MRMC construction, Section 5.3.

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

(a)



(b)

Figure 4.2 The original and transformed representation, from algebra (4.2a) to dot-arrangements (4.2b). This is a reproduction of Figure 4.1.

we must deal with are simple: only conjunctions or disjunctions, rarely both, over very few components, usually less than five, that we can refute quickly. We have built a tool, `pseudodesc`, which automates the pseudo-description-generation procedure. Despite the theoretically exponential run-time, we have found pseudo-descriptions typically build in less than one second.

Example 4.13. Consider again our problem of summing the first n integers, reproduced in Figure 4.2a. In Listing 1 we present a complete description of the representation in Figure 4.2a. Similarly, Listing 2 is a complete description of the representation in Figure 4.2b. Finally, Listing 3 is the generated pseudo-description based on the description of the original representation and a set of correspondences. Listings 2 and 3 are not the same. Ignoring the superficial differences[35] we see there are four key changes: missing components, extra components, incorrect attributes, and changed importances.

Missing components Absent from the *generated* description, Listing 3, are the pattern triangle and law count_invariant components. We see these components included in Figure 2. The first is a significant shortcoming: the identification of triangles is necessary to understand the representation.

Extra components The generated description includes 'extra' components it has classified as 'essential': five 'cut' tactics. Indeed, the tactic cut_diagonal is a valuable tactic to include, and perhaps should be in the hand-written description. However, the other tactics are irrelevant. Inspecting the diagnostic output of the code, the component primitive $div is the cause: the correspondences indicate that division relates to cutting, but we cannot be sure which kind of cutting.

Incorrect attributes Most attributes are copied automatically from the associated RS-description for the target representational system,

[35] The representation is un-named, and there are no comments.

such as ‘type’ and ‘holes,’ but ‘occurrences’ is not. Thus the occurrences attribute of primitive $dot is 2 rather than 30. This is understandable: there is no way to know the size of the instance we will be working with. But this would impact the computation of cognitive properties, for example.[36]

Changed importances Finally, we see some interesting shifts in importance between Listings 2 and 3. Where before we had the stacking tactics as instrumental, we now see them listed as essential. We see no problem with this. Conversely, setting the nonsense tactics as essential *is* problematic: if we used correspondences as hints for students, this could lead them astray. ▽

[36] See Section 5.2.2.

A human-written Q-description of the representation in Figure 4.2a. Importance is one of essential, instrumental, relevant, circumstantial, or noise. Listing 1

```
representation IntSum = rep
    (* Sum of integers between 1 and n
       in an algebraic representational system *)
    modes 1;
    essential type integer;
    instrumental type proof;
    instrumental primitive $sum
        where type = 'a set * ('a -> integer) -> integer,
        occurrences = 1;
    instrumental primitives *, +, $div
        where type = integer * integer -> integer,
              occurrences = 1;
    instrumental primitives =
        where type = integer * integer -> formula,
              occurrences = 1;

    instrumental pattern sum
        where type = formula,
              holes = ['a set * ('a -> integer)
                           -> integer : 1,
                       'a set : 1,
                       'a -> integer : 1,
                       integer : 1],
              primitives = [\sum, =],
              primitive_registration = 1,
              occurrences = 0;
    instrumental pattern equality_chain
        where type = proof,
              holes = [integer : log(#t)],
              primitives = [=],
              primitive_registration = 1,
              occurrences = 0;

    instrumental tactic induction;

    relevant primitive n
        where type = integer,
              occurrences = 3;
    relevant primitive 1, i
        where type = integer,
              occurrences = 2;
    relevant primitive 2
        where type = integer,
              occurrences = 1;
    relevant primitive 0
        where type = integer,
              occurrences = 0;
    relevant primitives -, ^
        where type = integer * integer -> integer,
              occurrences = 0;

end;
```

Listing 2 A human-written Q-description of the representation in Figure 4.2b.

```
representation IntSum = rep
    (* Sum of integers between 1 and n,
       specialised to n=5,
       in a dot representational system *)
    modes 1;
    essential type dot-arrangement;
    instrumental primitive $dot
        where type = dot-arrangement,
              occurrences = 30;

    instrumental pattern rectangle
        where type = dot-arrangement,
              holes = [dot-arrangement : 1],
              occurrences = 1;
    instrumental pattern triangle
        where type = dot-arrangement,
              holes = [dot-arrangement : 1],
              occurrences = 2;

    instrumental law count_invariant;
    instrumental tactics constructive_omega_rule,
                         stack_vertical,
                    stack_horizontal;
end;
```

This is an automatically generated Q-description of the representation in Figure 4.2b, constructed by inspecting the correspondences from the Q-description in Listing 1.

Listing 3

```
rep
    modes 1;
    essential type dot-arrangement;
    essential tactics stack_horizontal,
                      stack_vertical,
                      cut_diagonal,
                      cut_ell,
                      cut_square,
                      cut_horizontal,
                      cut_vertical;

    instrumental primitive $dot
        where type = dot-arrangement,
              occurrences = 2;
    instrumental patterns rectangle
        where type = dot-arrangement,
              holes = [dot-arrangement : 1],
              occurrences = 1;
    instrumental tactics constructive_omega_rule,
                         count_equal;
end;
```

4.4.2 *Describing transformed representations*

The pseudo-descriptions we construct consist of all the components that are *likely* to be in the description of an actual representation. That representation is the *transformation* of the original representation into a new representational system. We stress transformation here, as we did in Section 4.2.1—there are almost certainly not formal translations between the representational systems. But this lack of formal translations makes pseudo-descriptions more valuable: they give a sketch of what a transformed representation would be, where formal translations discount the transformation completely. We might not know how each tactic or pattern gets used, but we now have an indication that they *will be* used.

Consider the pedagogical value of having a description of potentially useful representations. By guiding the students with hints drawn from the pseudo-descriptions, we can support them in constructing a transformed representation of their problem—without knowing the transformed representation of the problem. By suggesting the new components in an order determined by importance and strength, we may be able to give them enough guidance that they no longer need the full description to construct the new representation. The same benefits might apply to future automated representation transformation work: pseudo-descriptions can generate heuristics that guide the transformation.

SUMMARY OF SECTION 4.4

Pseudo-descriptions exploit correspondences to build descriptions of potential representation transformations. That is, given an R-description of representation `r` in system `R`, we can estimate the R-description of analogous representation `r′` in system `R′`. So while transformation between representations may be infeasible, we can approximate the transformation through descriptions. We have built a tool, `pseudodesc`, that automates the pseudo-description-generation procedure, and can thus automatically construct these transformed descriptions.

SUMMARY OF CHAPTER 4

Correspondences are bridges between representational systems, identifying the pieces that fill the same purpose even when they appear or behave differently. So we answer our second research question, describing how representational systems and their components are similar. We have grounded correspondences in probability theory, which allows us to automatically identify new correspondences with interactive tools. Correspondences are a novel contribution of this dissertation. Now that we have introduced both correspondences and components, we have the building blocks in place to discuss not just what the representations and representational systems are or how they relate to one another, but we also have the groundwork to *compare* and *evaluate* them.

Automated representation recommendation 5

A problem never exists in isolation; it is
surrounded by other problems in space and time.
— *Russell L. Ackoff*

To choose a representation to solve a problem we must first understand the problem, any alternative representational systems, and how the problem and these systems link together. In Chapter 3 we explored how to encode representations and representational systems using descriptions, while in Chapter 4 we introduced correspondences as links between representational systems.

In this chapter we combine these two strands to recommend alternative representational systems for a given problem. Specifically, we examine the entire pipeline from descriptions to recommendations. We first describe the pipeline, including where and how *analysts* are involved before automation takes over. We develop definitions for the *informational suitability* and *cognitive cost* of representational systems and representations, respectively. A definition for minimally redundant and maximally covering correspondence sets completes the theory, before we examine elements of the `robin` implementation of the framework.

The purpose of this chapter is to consider our third research question: how can we algorithmically evaluate and rank representational systems on their ability to be used to solve a particular problem? In Section 5.2 we complete the first objective by defining appropriate measures of representational system suitability for problem solving, and in doing so make our second contribution of this dissertation; in Section 5.4 we implement these measures, along with the rest of the framework—the third contribution. This chapter is joint work with the rep2rep research group: the overall framework is a joint effort, while some of the details and most of the implementation of informational suitability are my own. Thus, the sixth contribution[1] of this dissertation—informational suitability—is shared with the rep2rep research group.

Parts of this chapter have been published at the *Conference on Intelligence Computer Mathematics* (CICM) 2019 [Raggi et al. 19], then iterated upon at the *International Conference on the Theory and Application of Diagrams* (Diagrams) 2020 [Raggi et al. 20-b] and the *International Conference on Tools with Artificial Intelligence* (ICTAI) 2020 [Raggi et al. 20-a]. Sections 5.3 and 5.4 are yet to be published.

[1] Using the numbering scheme from Section 1.3.

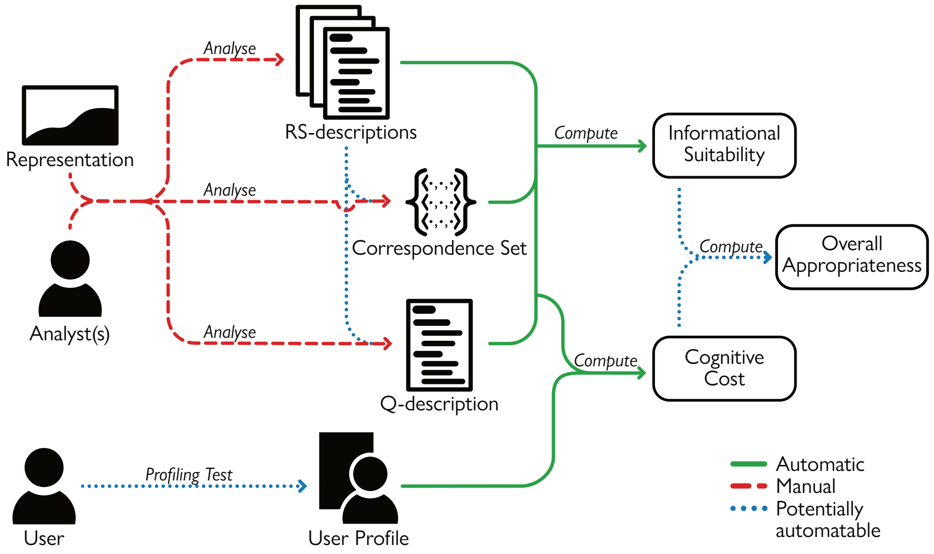


Figure 5.1 The representation recommendation pipeline from beginning to end, including the analysts creating descriptions, user profile generation, computing informational suitability and cognitive cost, and the combination of the two measures. This is a reproduction of Figure 1.2.

5.1 Recommendation pipeline

This dissertation contributes a method for evaluating and recommending representational systems tailored for specific problems and users. This means we have to consider five factors: what is a problem, what are representational systems, how can we encode these, how can we compare them, and how do we use this to order them. People, particularly novices, struggle to change representation [Uesaka et al. 10], so a tool that supports representational system selection has the potential to improve their reasoning and problem solving.

We return to Figure 1.2 (reproduced in Figure 5.1) to contextualise our work so far. Previous chapters on descriptions and correspondence sets described what both are, and partially where each comes from; in this section we consider both the role of analysts, who will eventually be the creators of descriptions, the *procedure* for creating descriptions, how and when correspondences are generated, and the role each piece plays in producing a representational system recommendation.

5.1.1 *Analysts*

Analysts play a central role in the rep2rep process, providing structured, high quality inputs for the framework. In Figure 5.1, their role is indicated by the red dashed lines. Analysts can be categorised into three groups:[2] RS analysts, correspondence analysts, and R/Q analysts. To illustrate the interaction between the analysts and the framework, we draw an analogy to implementing a school curriculum.

RS-analysts and correspondence analysts have the key task of defining

[2] One analyst can belong to several of these groups; this separation allows analysts to *not need* to be capable of doing the work of all three groups.

which representational systems will be included in any implementation of the framework, and work at a higher level of abstraction than the other analysts. They set the direction, and the boundaries of the framework implementation. RS-analysts and correspondence analysts are like the curriculum designers, generating the 'master' resources and guidelines. To design the curriculum is a difficult, slow process with pitfalls that have enduring repercussions; but it only needs to be done infrequently, and is centralised. Similarly, to construct RS-descriptions and correspondences is a difficult, slow process, and the choices made here will impact all future uses of the RS-descriptions and correspondences; representational systems are 'standardised', so this process is done infrequently and can be centralised.

Once an RS-analyst has constructed a set of RS-descriptions, a correspondence analyst sets about linking the representational systems together. In our curriculum example, this role would be filled by 'cross-curriculum' course designers—attempting to integrate learning from several subjects in a cohesive way. This role requires a good understanding of many representational systems, whereas an RS-analyst need only be expert in a single system. However, the correspondence analyst is working with existing descriptions, which in our experience has made the job less difficult.

At a lower level of abstraction, R/Q-analysts focus on concrete instances, on the representations and problems specifically. They are expected to build on the work done by RS-analysts. R/Q-analysts are like teachers, building on resources provided by the curriculum designer and specialising it for their classrooms and students. Again, this is a complex task, but by using the master resources less novel work is necessary to produce derived resources; but the derived resources must be continually created, updated, and improved. Similarly, in our experience we have found R- and Q-descriptions difficult to create, but markedly less so than RS-descriptions; conversely, R- and Q-descriptions must be created for each new problem.

5.1.2 *Descriptions*

Descriptions are a large part of the input to the representation selection framework, so are created early in the pipeline. We identify two phases of description creation: ahead-of-time, and just-in-time description creation. RS-descriptions are created ahead-of-time, meaning before any implementation of the framework is deployed. The representational systems that the implementation will work with are fixed when the RS-descriptions are created; thus these descriptions form a *library*. Conversely, R- and Q-descriptions are largely created just-in-time as the tool is being used. Representations and problems based on the bundled representational systems are too numerous to include in a library distributed with the implementation, and so must be created as needed. This reinforces our analogy with curriculum design: textbooks and such are created ahead-of-time, while practice problems are mostly generated just-in-time.

We begin by considering the ahead-of-time phase, which is the construction of the RS-descriptions which would be in a library distributed with our software. That means the workflow would be completed by an analyst who is expert in both the representational system, and in describing representational systems. One possible workflow of an RS-analyst to generate an RS-description proceeds in seven steps:

1. Collect examples of representations in the representational system. (Optional, but strongly recommended.)

2. (Optionally) generate R-descriptions of the examples.

3. Generalise over the examples if they exist to understand the underlying representational system, otherwise directly reason about the representational system.

4. Identify the types and primitives of the representational system, and write their components in our description language.

5. Identify the patterns of the representational system, converting them to components.

6. Identify the tactics and laws of the representational system, again converting them to components.

7. Create the Bayesian network of component probabilities from a dataset of R-descriptions—ideally the same one that the examples were drawn from—or use expert knowledge to estimate these probabilities.

In our personal experience creating descriptions, steps four through seven occur cyclically, where identifying patterns will influence the types and primitives, while the tactics and laws influence the patterns, and so forth. We emphasise the step of collecting examples because the balance between consistency and diversity is vital: the analyst must have a clear line in their head between what *is* and what is *not* a representation within each system.

For the users of our software, they must describe the problems they are working with—these users are the analysts who are experts in their representations and representational systems, but not in describing representational systems. R/Q-analysts construct the descriptions of specific representations and problems. Their workflow is simpler than that of the RS-analyst, for a given representation or problem:

1. Select the RS-description for the representational system that your representation comes from.

2. Identify the components from the RS-description that belong in your new R-description, based on the representation you are describing.

3. Count the occurrences of each concept in your representation and populate the 'occurrences' attribute of the component associated with that concept.

4. (If producing a Q description) assign an importance to each component.

By the point an R/Q-analyst is producing descriptions, RS-descriptions have already been created and can be referenced by the analyst to help them identify components that are associated with the concepts in their representation. Thus it becomes an identification task, rather than an invention task—easier, but not 'easy.'[3] This workflow, selecting from an RS-description to construct an R-description, reflects the fact that R-descriptions are nearly subsets of RS-descriptions—the only addition is the 'occurrences' attribute. The extension to Q-descriptions also obviously mirrors their definition as an R-description with importance. This relationship between RS-descriptions and R- and Q-descriptions is indicated by a dotted blue line in Figure 5.1.

Shortcomings of RS-descriptions can get identified at this stage: our experience leads us to suspect that communication between RS analysts and R/Q analysts would improve the process for both parties.

We have seen, and will see in the next section, some tooling that exists for correspondence discovery; what about tools for description creation? So far, we have create one small tool (`uniondesc`) while one significantly larger, more sophisticated tool is proposed based on RepNotation [Cheng 20]. Such a tool is future work, and outside the scope of this dissertation.

The `uniondesc` tool is simple: it reads in R- and Q-descriptions and produces RS-descriptions. This tool is designed to support RS analysts at step three, when they have generated R-descriptions and need to turn them into an RS-description. The `uniondesc` tool treats the R-descriptions as sets of components, then it unions them. It does more—stripping occurrences and, for Q-descriptions, importance—but the core is simple. There are currently no sophisticated methods for handling clashes (components with the same kind and value but different attributes); attributes should be verified by the RS-analyst.

5.1.3 *Correspondence discovery*

Correspondence analysts work after the RS-analysts to link together the newly created descriptions. Their task requires an understanding of all the representational systems that they are attempting to associate; as such, it requires knowledge of many representational systems, and considerable expertise in how representational systems are described.

Like descriptions, correspondences form one of the inputs to the representational system recommendation framework; the 'quality' of the correspondence set impacts the potential suitability of the resulting recommendation for the problem and the user solving the problem.[4] The

[3] Similar to how recognition is easier than recall [Lidwell et al. 07].

[4] 'Garbage in, garbage out.'

quality of a correspondence set is a combination of its coverage of all the RS-descriptions, the appropriate use of component formulae to avoid having interrelated correspondences, and the accuracy of the strengths associated with each correspondence. Because correspondences exist between representational systems, they are generated ahead-of-time to be shipped with the framework implementations as part of the representational system library, not generated by the end users.

The workflow for correspondence analysts is similar to that of the other analysts:

1. Collect examples of 'equivalent' representations across representational systems. (Optional, but strongly recommended.)
2. Compare the components across the R-descriptions of the example representations to discover simple correspondences.[5] If the analyst is not using any examples, they must use RS-descriptions as a source of inspiration for correspondences.
3. Consider larger groupings of components to discover correspondences over component formulae.
4. Use the probabilities the RS-analyst computed when constructing the RS-descriptions to calculate the correspondence strengths.
5. Use the `findcorr` tool[6] to extend this base set of correspondences.

Based on our experiences acting as correspondence analysts, we find that—as with RS analysis—these steps are cyclic: steps two through five influence each other, meaning continual rounds of adding correspondences. At this point we recommend constructing pseudo-descriptions using the `pseudodesc` tool[7] and identifying any 'obvious misses': if the analyst would expect particular components to appear in the generated pseudo-descriptions, but they are missing, then this suggests the associated correspondence is missing. The analyst should ask themselves: what components in the original RS-description would correspond to the components that are missing? They should then add a new correspondence between these components and the missing components.

5.1.4 *Analyst bias and limitations*

Analysts are—at least for now—human. This means that the descriptions and correspondence sets, which are the input for our framework and its implementation, are subject to their limitations and biases. While we will explore this again in Section 6.4, here we explore the conceptual implications of relying upon analysts.

We first stress that there is no 'correct' description or correspondence set for any (set of) representational systems and problems. However, we argue that some descriptions/correspondence sets are *less correct* than others, in potentially three ways:

[5] Assuming an R/Q-analyst has already created R-descriptions; if not, the correspondence analyst is strongly advised to create some.

[6] This tool was outlined in Section 4.3.

[7] We outlined this tool in Section 4.4.1.

1. the description/set is missing components/correspondences that capture features of the underlying representation or system(s);

2. the description/set has components/correspondences that do not capture features of the underlying representation or system(s); and

3. the components/correspondences are constructed such that they favour some problems or representational systems 'unfairly.'

Each has a different impact on the inputs to the framework, and thus its recommendations.

In the first instance, missing components and correspondences, we expect to see most often a reduction in overall informational suitability; however, due to the NOT connective, missing components can result in *higher* informational suitability scores. For either case, the informational suitability score can be affected to the detriment of the overall recommendation. If a component is omitted from a Q-description, the effect is restricted to a single problem; if a component is omitted from an RS-description, the effect is felt on all recommendations to and from that representational system; if a correspondence is omitted, then it affects all recommendations within the framework, as that correspondence might have been used to derive new correspondences. This implies a severity scale: the analysts must take the most care with correspondences and RS-descriptions—fortunately, these are the inputs that are least frequently modified.

When a description or correspondence set contains 'extra' elements, we see the opposite effect: generally, informational suitability scores will be inflated due to the framework making spurious connections between representational systems. However, mirroring the previous case, due to the NOT connective we can see the overall informational suitability *decrease* due to extraneous components. The severity parallels the instance of missing components and correspondences: in Q-descriptions the impact is limited, but in RS-descriptions and correspondence sets the impact can be widespread.

Finally, the case of bias by the analysts is the most subtle and difficult to classify. In our experience building descriptions and correspondence sets, analysts can fall into a 'syntax trap': they start describing all aspects of the representation or representational system in terms of the syntax, rather than the underlying ideas. This results in more components and correspondences in representational systems that use a lot of syntax, while systems that use attributes such as position, scale, or rotation are underdeveloped because the syntax is less explicit. Other biases can be more fundamental: an analyst—particularly one describing a problem—might not have a complete grasp of the underlying concepts and how they are used in the representational system more generally. They may produce descriptions that use components incorrectly, or in unintended ways, propagating this to structures such as patterns, and impacting matching with component formulae inside correspondences.

Beyond being careful, there is little we can presently do to mitigate these problems. But there are avenues for future work. Most obvious, but least achievable, would be to remove analysts entirely: rely upon artificial intelligence to interpret the problems and representational systems to produce descriptions and correspondences. However, this is likely nearing human-level intelligence, and is presently infeasible. More realistic would be repurposing user profiling (Figure 5.1, centre bottom) to profile the *analysts*,[8] and so provide a 'reliability' score on the components and correspondences they provide, based on their cognitive strengths and weaknesses. Thus our recommendation would need to consider both the informational suitability, the cognitive cost, and the analyst's reliability. Such a reliability score would help with all three problems, but most directly address the third—analysts bias. Other options include having an analyst *committee*, but this increases the manpower requirements of an already labour-intensive process.

SUMMARY OF SECTION 5.1

The first phase of our pipeline is manual, collecting data from which we will produce a recommendation. Once the manual phase of the pipeline is completed, the system becomes completely automatic: all the descriptions (RS, R, and Q) and the correspondence set are processed, and a recommendation is made without user intervention. We acknowledge that the task of the analyst is significant, but we provide tooling to support this process; we expect more powerful tools and automation to further reduce the burden, and future research will examine how R-descriptions may be automatically constructed from representations.

5.2 Objective function

From provided sets of RS-, R-, and Q-descriptions, along with a set of correspondences, we evaluate the effectiveness of representational systems as a tool for a specific user for a specific problem. Identifying an *effective* representation is a challenging problem: drawing a precise line between effective and ineffective representations is contextual and personal. But the fuzzy boundary has been drawn by previous research,[9] with a common theme of *conceptual clarity* [Cheng 16]. The necessary aspects of the problem must be represented, unnecessary aspects should be avoided, and the presentation should be understandable to the reader.

Effectiveness is defined over two factors: we score each representational system based on its *informational suitability* for the problem; for users, we consider the problem-specific *cognitive cost* of using each alternative representation. Informational suitability focuses on ensuring that as much as possible of what is important to solving the problem is encoded in the new representation accurately. Cognitive cost reflects the human processing cost of the new representation.

[8] We thank examiner Prof. Anthony Cohn for this suggestion.

[9] See Section 2.2.2.

5.2.1 *Informational suitability*

Informational suitability determines whether a representational system can be used to create a representation that has the expressiveness and inferential capabilities to encode and solve the problem as stated. That is, we ask if this representational system is a *viable* system. We would not use our dot representational system if we had to reason about Euler's constant e, for example. We model the potential viability of the alternative representational systems by inspecting the correspondences that link the given problem's Q-description into the alternative RS-descriptions. The more covered the Q-description is by a correspondence set, the more of the problem statement can be accurately captured. Further, if the satisfied correspondences are *strong*, then the new representational system will more faithfully capture the same concepts as the original statement. We prioritise the most *important* components in the Q-description, where importance is with respect to how necessary the information each component encodes is to the underlying problem.

Before we define informational suitability, we must understand how a feature of components, *importance*, can be lifted to correspondences. The lifting process happens through the left formula of a correspondence: that is, for correspondence $\langle a,\ b,\ s \rangle$, we lift component importances through a. When a is a single component, this is simple: we take the importance from the Q-description. But when a is a formula, we compute the importance as the maximum clause importance in any satisfied clause, where the clause importance is the maximum importance of all positive components in the clause. That is,

$$\begin{aligned}\mathrm{importance}_q(a) = \max\{\, & \mathrm{importance}_q(c) \\ & \mid c \in t, t \in \mathrm{clauses}'(a), \mathrm{sat}_q(a)\}\end{aligned}$$

where $\mathrm{clauses}'(a)$ converts a formula to a set of sets of non-negated components,

$$\mathrm{clauses}'(a) = \{\mathrm{positive}(t) \mid t \in \mathrm{clauses}(a)\}$$

and $\mathrm{sat}_q(a)$ asserts that Q-description q satisfies the component formula a. The function $\mathrm{positive}(t)$ extracts the set of components in a term t which are not preceded by a NOT connective; $\mathrm{clauses}'(a)$ is thus a set of sets of components.[10] We define importance on formulae, $\mathrm{importance}_q(a)$, in terms of component importance, $\mathrm{importance}_q(c)$.[11]

We choose to use the maximum importance over all component clauses as the importance of the component formula, but alternative formulations, such as taking the median or mean importance, are also possible. Our motivation for using the maximum over the alternatives stems from how we define importance. A higher component importance means it is more difficult to replace or remove the aspect of the problem that the component encodes; a lower component importance suggests the associated problem aspect is easier to replace or remove. We use the *maximum* component importance: if one aspect encoded by a component is difficult to substitute or remove, then the combination of

[10] We assume, as always, that we are working with formulae in disjunctive normal form.

[11] That is, this function is not recursive—it is a lifting of importance from components to formulae.

aspects captured by the component formula will also be difficult to substitute or remove as a group—even if the remaining aspects are simple to substitute or remove in isolation.

Example 5.1. Given the component formula[12]

$$\alpha = (\text{primitive} +) \ \textsc{or} \ (\text{primitive} \times) \ \textsc{or} \ ((\text{primitive} +) \ \textsc{and} \ (\text{primitive } 1))$$

and a (fragment of a) Q-description

```
rep
    essential primitive ×;
    instrumental primitive +;
end;
```

we can compute $\text{importance}_q(\alpha)$. First, we have

$$\text{clauses}'(\alpha) = \{\{\text{primitive} +\}, \{\text{primitive} \times\}, \{\text{primitive} +, \text{primitive } 1\}\}.$$

We filter this set by sat_q, eliminating the final clause because there is no primitive 1 in our Q-description. Then the series of $\in$ operators effectively flatten the remaining sets, meaning we have

$$\begin{aligned}\text{importance}_q(\alpha) &= \max\{\text{importance}_q(\text{primitive} +), \\ &\qquad\qquad \text{importance}_q(\text{primitive} \times)\} \\ &= \max\{0.6, 1\} = 1.\end{aligned}$$

(The specific values of the importance keywords in the description are implementation-specific, and here are taken from our `robin` codebase. We chose the values empirically.) Thus the importance associated with component formula α in this Q-description is 1 (essential). ▽

Informational suitability, while directly a function of Q-descriptions and RS-descriptions, relies on correspondence sets. But there is no reason to consider *all* correspondences: many will not be relevant. Instead we filter the set down to an approximation of the minimally redundant and maximally covering set of correspondences. At a high level, this selects the correspondences that cover as much of the Q-description as possible, while also avoiding having the correspondences 'overlap' (share components). One might reasonably question why this is appropriate. Each correspondence describes a potential encoding of information from the problem into a potential representation using this particular representational system, and it is reasonable to want more information from the problem to be representable in any new potential representation. But baked into this is that the re-encodings are *compatible*: the correspondences are re-encoding *different* pieces of information. We will dive into this assumption further in Section 5.3.

We now have the pieces required to define the *informational suitability* objective function.

[12] This formula is taken from an algebraic representational system.

Definition 15 (Informational suitability)**.** The informational suitability of a representational system, encoded as an RS-description R, for a problem, encoded as a Q-description q, based on a set of satisfied correspondences C, is

$$\mathrm{IS}_C(q, R) = \sum_{\langle a,\, b,\, s\rangle \in \mathrm{MRMC}_q^R(C)} s \cdot \mathrm{importance}_q(a), \tag{5.1}$$

where $\mathrm{MRMC}_q^R(C)$ is a subset of C such that the correspondences are as minimally redundant and maximally covering as possible of q and R,[13] and $\mathrm{importance}_q(a)$ is the importance of component formula a in the Q-description q.

This is to say, for a 'compatible' set of correspondences, determine how large the set is, weighted by the strength of these correspondences and by the importance of the components they re-encode. At the higher level of representations, we aim to approximate how accurately specific pieces of information from the problem can be preserved and re-encoded in some representation from the target system; we weight this approximation by how important that information is to the problem.

We claim that the MRMC construction—which eliminates correspondences that increase the size of the correspondence set cover—filters the set of correspondences to be 'compatible'. We cannot *guarantee* that no component from the original Q-description is not re-used in incompatible ways, as this semantic knowledge is beyond the capabilities of the current generation of the framework. To counter this, we use component formulae to ensure that correspondences are capturing unrelated concepts: the components between formulae may overlap, only because the representation re-uses the same aspects to encode different information.

We reiterate a point from Section 4.1.4: there is a difference between having two correspondences $\langle a,\ b,\ s\rangle$ and $\langle a,\ b',\ s\rangle$ and having a single correspondence $\langle a,\ b \text{ OR } b',\ s\rangle$. The former is describing two distinct relationships, while the latter is describing a single, non-deterministic relationship. The decision on whether to use two correspondences, or a single correspondence with a component formula using OR, will impact the informational suitability calculation.

One limitation of informational suitability—one shared by humans—is that we must start from a representation in a *sufficiently expressive* representational system.[14] From one perspective, the starting representational system must be sufficiently expressive for any problem stated in a representation using that representational system: indeed, it *is* expressed in that representational system. But this might not be the *intended* problem as expressed: further information might be provided in auxiliary representations, such as natural language captions. Any recommendation made starting from these 'deficient' representations must be treated with caution: it has not taken into consideration all the features of the problem that are not captured in the representation, but which might substantially change the recommendation.

[13] We cannot necessarily guarantee the existence or uniqueness of an MRMC set, so we approach this as an optimisation problem: how close can we get? We go into more depth in Section 5.3.

[14] Humans often do *not* seem bound by this limitation because we bring our own context and background knowledge about the problem. In truly unfamiliar contexts, this limitation would be apparent.

Example 5.2. Suppose a probability problem was stated using Euler diagrams: this accurately conveys the events, which can occur simultaneously and which must be disjoint. Using Euler diagrams as a visual aid in probability is common. *But*, Euler diagrams are *insufficient* to capture *all* of most probability problems: they have no way to express magnitudes, and thus precise probabilities.[15] Thus, attempting to compute the informational suitability of any representational system based on the Q-description for the Euler diagrams representation will miss any restrictions on precise probabilities: the recommendations will be unhelpful at best, and misleading at worst. ▽

5.2.2 *Cognitive cost*

We cannot compute the cognitive cost of a representational system: a representational system is not inherently cognitively 'good' or 'bad', but instead an instance of it can be constructed such that the resulting representation is cognitively costly (or not) relative to the alternatives. So cognitive costs are defined over Q-descriptions only. In the following discussion, the Q-description q is *not* necessarily the given Q-description: it may be that description, but it may also be a description of the alternative representations that are being used to encode the problem.

Cognitive costs, being related to cognition, are relative to a specific *user*—the target user of the representation. What is cognitively appropriate for some may be cognitively inappropriate for others. Thus we must consider *who* we are calculating the cognitive cost *for*. A user is represented inside our framework by their profile, u. This profile is a vector of real values that capture different dimensions of their cognitive abilities, and how they respond to each of the cognitive properties. For example, we have the cognitive dimension of *expertise for a domain*, u_e.[16] Similarly, we might have a dimension for mental visualisation: higher values denote a stronger ability to visualise objects in their mind, without using an external representation. The exact contents of the user profile is ongoing research.

In Section 3.4 we introduced nine cognitive properties. Each cognitive property p has a fundamental cost associated with it, which we define as $\text{cost}_p(q)$ for some Q-description q. The cost of a cognitive property is computed over q, which has importances associated with the components. But there is no reason to believe that the user of the representation understands the importance of components: in particular, novice users struggle to identify what is important and what is not, so *everything* can seem important [Chi et al. 88]. Thus we modify the importances in the description q such that the importance of a component c is

$$\text{importance}^{u}_{q}(c) = 1 + \text{importance}_{q}(c) \cdot u_e - u_e$$

where u_e is the 'expertise' of the user u. An expertise of $u_e = 0$ is completely novice, and thus the importance of any component is 1, meaning everything seems important; an expertise of $u_e = 1$ is perfectly expert,

[15] Two exceptions are 0 and 1: these *can* be expressed with Euler diagrams.

[16] We assume that most deployments of our system will be in one domain—or very few, related, domains—and so the number of u_e's in a profile is very small.

and thus the importance of any component is unchanged, so the user has identified the correct[17] importance. Any value in between encodes degree of expertise, higher being more expert. We define the function $\mathrm{isub}_u(q)$ as a function which modifies the Q-description q to have the modified importance function $\mathrm{importance}^u_q$—effectively updating the Q-description to reflect the importance of components as they would be perceived by user u. That is,

$$\begin{aligned} \mathrm{isub}_u(q) &= \mathrm{isub}_u((r, \mathrm{importance}_q)) \\ &= (r, \mathrm{importance}^u_q) \end{aligned}$$

with $\mathrm{importance}^u_q$ defined as above.

Once the costs of each cognitive property have been computed across all alternative Q-descriptions,[18] we can see the distribution of the individual costs. While from the cost_p function we know the *expected* range of values, in practice the range of values each cognitive property takes can be quite different. To return to the expected range, we introduce the function norm_p. This function is unusual in that it can only be defined after cost_p is computed across all alternative representations, but must be used to define the overall cognitive cost for each alternative Q-description.[19]

At this point, we have the cost of each cognitive property computed against the Q-description with its importances modified. However, this assumes that each cognitive cost will *impact* each user to the same degree. The importance modification is applied uniformly for all cognitive properties: it determines how the user constructs mental models based on the representation, and so all cognitive properties are affected. But the nature of the cognitive property itself also plays a role in how it affects the user: some users will be impacted more for certain properties than others. This is captured by $c_p(u)$, the user-specific cost of cognitive property p, which we use to model the *user sensitivity* of each property.

The user-specific cost of cognitive property p, denoted $c_p(u)$, is an arbitrary function set from either literature or experimental data, depending both on the cognitive property p and the literature and data available. The function then takes the user profile u and interprets the appropriate aspects to return a real value.

Some properties, such as token registration,[20] are nearly independent of expertise;[21] others, such as branching factor of the problem space, are directly related to expertise.[22] We use the user expertise u_e to weight cognitive properties at higher levels of granularity with a greater expertise sensitivity.[23]

Example 5.3. Consider the cognitive property *registration*: the cost of this property is based on attributes of patterns in the Q-description. Assume the importance of the pattern is 0.5: the cost for the expert will be lower, because they are able to recognise they do not need to register tokens via this pattern; the cost for the novice will be higher, because they are *not* able to recognise that the tokens are not important. We model this by raising the importance when modelling a novice user.

[17] 'Correct' meaning 'agrees with the R/Q-analyst-assigned importance'.

[18] Alternative Q-descriptions are provided by either the analysts, or are pseudo-descriptions (Section 4.4).

[19] This is only a theoretical quirk. In practice we compute all the costs in parallel, so the computation works fine, with the caveat that introducing a new alternative representation will affect the scores of the previous alternative representations.

[20] Tokens are closely related to primitives.

[21] Both experts and novices need to read the tokens, even if they will then process them very differently.

[22] Experts can prune the search space towards the solution much more effectively than novices [Chi et al. 88].

[23] The levels of granularity are the columns in Figure 3.3, on page 60.

But consider the user sensitivity of token registration: registration is a fast cognitive process that, even for novices, it is not a significant contributor to the cognitive cost. Thus registration has a low user sensitivity—it does not matter whether the user is novice or expert, the time to register a token will not vary much. Note the distinction between user sensitivity and changing the importance: with user sensitivity, we assert that the cost of performing the registration is similar; changing the importance models whether the registration will be performed at all.

Compare this to a property such as *expression complexity*: familiarity and schema determine the amount of cognitive load induced by expression complexity. Thus, even if the associated components have importance 1, we still wish the cognitive cost to be higher for novices than it is for experts—this property is sensitive to the user. ▽

Taking these points together, we wish to define the *cognitive cost* objective function in such a way that each cognitive property is considered, and the cost associated with that property depends on both the current representation, and the user.

Definition 16 (Cognitive cost)**.** The cognitive cost of a Q-description q for a user u is

$$CC_P(q,u) = \sum_{p \in P} c_p(u) \cdot nci_p^u(q) \tag{5.2}$$

where

$$nci_p^u = norm_p \circ cost_p \circ isub_u,$$

the set P contains the cognitive properties under consideration,[24] $c_p(u)$ is the 'user sensitivity' of p evaluated for the user u, $isub_u(q)$ is the Q-description q with component importances modified by user u's expertise, $cost_p(x)$ is the cost function of p, and $norm_p(y)$ normalises the cognitive costs for comparability across representational systems.

The objective function for cognitive cost in Definition 16 makes extensive use of the user profile u, both in modifying the importance function, and in capturing user sensitivity of the properties. The user profile is developed by observing or testing the user, and captures their cognitive processing potential; this is not a model of analysts. The profile u currently consists only of general *expertise*, which we call u_e, and is modelled as a number from 0 to 1,[25] from novice to expert. We hope to expand this profile to more accurately model the abilities of the user, and develop ways to construct this profile that is minimally obtrusive while being sufficiently accurate and precise.

Example 5.4. Let us compute the value for one cognitive property in our integer sum problem: expression complexity.[26] For simplicity, we work with the Q-description for our algebraic expression

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

[24] We assume this is always the set from Section 3.4.1.

[25] While both are numbers from 0 to 1, we stress that *importance* and *expertise* are independent: the former is with respect to components, while the later is with respect to users.

[26] Expression complexity is an estimate of how 'large' the 'parse trees' in the representation can become.

which contains the following pattern (and pattern-inducing primitive)[27] components:

```
instrumental pattern sum
    where type = formula,
          holes = ['a set * ('a -> integer)
                        -> integer : 1,
                   'a set : 1,
                   'a -> integer : 1,
                   integer : 1],
          primitives = [\sum, =],
          primitive_registration = 1,
          occurrences = 1;
instrumental pattern equality_chain
    where type = proof,
          holes = [integer : log(#t)],
          primitives = [=],
          primitive_registration = 1,
          occurrences = 1;
instrumental primitives *, +, $div
    where type = integer * integer -> integer,
          occurrences = 1;
```

We also need to determine the user profile for which we are evaluating the cognitive cost: let us choose a user who is relatively novice ($u_e = 0.2$).

We will work inside out, right to left. That is, first we evaluate $\text{isub}_u(q)$ to adjust the importance values of the components in the Q-description. In the `robin` implementation, an *instrumental* component is given importance 0.6; applying importance_q^u to each pattern gives a new importance of $1 + 0.6 \times 0.2 - 0.2 = 0.92$. That is, the novice is giving these components more weight than an expert would.

To the resulting modified Q-description we apply the cost_p function. This is a specific procedure for each cognitive property, and in the case of expression complexity we examine the types and the holes of the primitives and patterns in a description, and compute (based on the occurrences attribute) how large these values can become. The specific computation is out of scope for this example, but in this case returns a value of 7.144, to three decimal places.

Because we are computing the cognitive cost of just one representation, $\text{norm}_p(x) = x$. All together, we have $\text{nci}_p^u(q) = 7.144$.

The final computation to determine the value of this sum term for computing the cognitive cost is $c_p(u)$, the factor describing how sensitive this cognitive property is to the user. In this case, expression complexity is strongly sensitive to expertise, and we define it to be

$$c_p(u) = 1 - \frac{1}{1 + \exp(10 - 15u_e)},$$

a sigmoid curve that steeply drops down once a specific level of expertise is reached. Given our user profile specifies a user expertise of $u_e = 0.2$, we

[27] Primitive components with composite types induce *associated patterns*; see Section 3.1.2.

have $c_p(u) = 0.999$. Improvements on this function would incorporate the user's ability to successfully map the expressions to their internal *schema*; this level of tailoring remains future work.

Put together, the cognitive cost, considering only the expression complexity for a fairly novice user on the algebraic representation of the sum of integers between 1 and n, is $0.999 \times 7.144 = 7.137$. ▽

5.2.3 *Combining objectives*

Informational suitability and cognitive cost are objective functions that we can optimise independently, but neither completely captures what it means for a representation to be effective at supporting a particular user solving a specific problem. We can suggest the 'informationally optimal' representational system, but it might be worthless if the user cannot understand it. Similarly, we could suggest a 'cognitively optimal' representational system, but it may be totally inappropriate for understanding and solving the problem at hand. We need both to make an appropriate recommendation.

The combined objective function

$$\mathrm{Eff}_{C,P,f}(q, R, u) = f(\mathrm{IS}_C(q, R), \mathrm{CC}_P(T(q, R), u)) \tag{5.3}$$

uses f as a means of combining the informational suitability with the cognitive cost, and $T(q, R)$ is the pseudo-description[28] transformation of q into a Q-description from RS-description R. By defining f such that we weight the recommendation towards favouring informational suitability, we will choose a representation to suit the problem, considering it more valuable than choosing one to suit the user. This would be preferable when the outcome is more important than the user completely understanding the process. But if we set f such that we favour cognitive cost, we might no longer be representing the problem in the most informationally appropriate manner, but we are aiming to ensure that the user is able to make progress towards a solution. This could be preferable in situations where we need to trust or verify solutions, rather than simply get the answer, by ensuring that the user can inspect the solution and understand the approach taken.

In this combined objective function, we set f based on the underlying motivation for solving the problem: do we just want an answer, or do we want an explanation, or are we trying to learn a new skill? This is the idea of *task* suitability, and completes the triad of factors when problem solving. To fully determine an effective representation, you must consider the problem, the user, and the underlying task [Moody 09], whether its to solve a problem, learn a skill, or explain a solution. Currently, the f parameter is the only means by which we consider the task that is driving the problem solving; expanding on this is an important part of the future work improving the rep2rep framework.

Example 5.5. Consider a situation in which we have one problem q, one

[28] Or actual description, if given by an analyst.

user profile u, and two representational systems A and B such that

$$\begin{aligned} \mathrm{IS}_C(q, A) &= 15 & \mathrm{IS}_C(q, B) &= 19 \\ \mathrm{CC}_P(T(q, A), u) &= 4 & \mathrm{CC}_P(T(q, B), u) &= 10 \end{aligned}$$

That is, we have computed the informational suitability and cognitive cost for each representational system for problem q and user u.

In one situation, we might favour a solution that, while not perfectly capturing the information of the original problem is easily understood. In this case, we favour a very low cognitive cost: setting $f_1(i, c) = i/c$, we have

$$\begin{aligned} \mathrm{Eff}_{C,P,f_1}(q, A, u) &= 15/4 = 3.75 \\ \text{and } \mathrm{Eff}_{C,P,f_1}(q, B, u) &= 19/10 = 1.9 \end{aligned}$$

revealing that, although representational system B seems to represent the problem q slightly better, it is more cognitively costly, and so is less favourable than system A.

An alternative situation is one where we wish to emphasise that the new system is capturing everything important about the original problem, even if it is somewhat difficult to understand. By choosing $f_2(i, c) = i/\log_2 c$, we have

$$\begin{aligned} \mathrm{Eff}_{C,P,f_2}(q, A, u) &= 15/\log_2(4) = 3.94 \\ \mathrm{Eff}_{C,P,f_2}(q, B, u) &= 19/\log_2(10) = 5.72 \end{aligned}$$

changing our recommendation. Now, although representation system B is more cognitively costly, it is more informationally suitable and so is selected over system A. ▽

SUMMARY OF SECTION 5.2

To recommend a representational system, we define two measures through which their effectiveness can be algorithmically evaluated: informational suitability, and cognitive cost. Informational suitability considers the problem-specific aspects of a representational system: can it express everything the problem requires? Cognitive cost focuses on the human: is this representational system appropriate for this person to use? We also posit a combined effectiveness score for an overall recommendation. For the remainder of this dissertation, we work with informational suitability only, as it was the primary focus of my research.

5.3 Minimally redundant and maximally covering

In defining informational suitability, we explained that we act over sets of correspondences which are as minimally redundant and maximally covering (MRMC) as possible. That is, we want to cover the most possible components from the Q-description q, while having the components be re-covered by the correspondences as little as possible.

Example 5.6. When dealing with natural numbers, we might like to consider the problem from the perspective of *graphs*: vertices and arcs. But we have a problem: we could encode numbers as the number of vertices, or we could encode numbers as the number of arcs—both are valid, and independent of one another. Thus we have two correspondences:

$$\langle \text{type number},\ \text{type vertex},\ 0.9 \rangle$$

and

$$\langle \text{type number},\ \text{type arc},\ 0.85 \rangle.$$

When computing informational suitability, we do not want to include both correspondences: representing numbers in two different ways in one representation would be confusing. Thus, we want to make sure we take *some* correspondence to cover numbers, while not taking *redundant* correspondences that cover already-covered components.

In this case, one or the other correspondence is sufficient to form an MRMC set of correspondences for the component set {type number}. ▽

But to build up to MRMC, we first need to examine what it means to be maximally covering, and what it means to be minimally redundant.

5.3.1 *Counting correspondences*

The core of the informational suitability objective function in Definition 15 is counting re-encodings of information in a representational system; if more aspects of the problem can be accurately re-encoded in a representation that belongs to the alternative representational system, that system is likely to be more effective for solving the problem. More directly: given a collection of correspondences, count how many there are; more is better. We make this optimisation more sophisticated with strength and importance, but the core remains. But which correspondences do we count, and which correspondences do we *not* count?

For some arbitrary set of correspondences, there is no point counting them all: most will not be related to the problem or representational system being considered. Instead, we begin by filtering the set to relevant correspondences by selecting every correspondence that is satisfied by both the Q-description q and RS-description R. This ensures that everything that is relevant will be considered in the informational suitability computation, but can mean we have correspondences that overlap. Correspondences $\langle a,\ b,\ s \rangle$ and $\langle x,\ y,\ s' \rangle$ 'overlap' if the component formulae a and x share a component, or b and y share a component. But if the correspondences overlap, we have multiple correspondences acting on a single component, which means the same information in the problem that is captured by that component may be re-encoded in different ways. This has the effect of artificially inflating the informational suitability, as the IS score increases but the representational system is not encoding more information. Suppose, with some arbitrary order on the correspondences, we discard all correspondences that cover a component that is covered by an earlier correspondence. Now there is no overlap, but now we might be unable to cover as many components from q as we previously could. The need to reduce overlap but increase coverage are in tension, and the balance dictates the bias of the framework: avoid duplicate information encoding but potentially miss encoding some information at all, or ensure that as much information as possible is encoded, at the risk of encoding the same information in several ways.

We introduce the principle of maximal coverage: if we do not cover a component, there is some feature of the problem statement that we are not considering in the alternative representational system. First, we formally define *covering*, which we met in Section 4.1.3:

Definition 17 (Correspondence covering)**.** Using our definition of clauses′ from Example 5.1, we can define the left cover of a correspondence to be

$$\begin{aligned}\mathrm{leftcover}(\langle\, a,\ b,\ s\,\rangle) &= \bigcup \mathrm{clauses}'(a)\\ &= \{p \mid t \in \mathrm{clauses}(a), p \in \mathrm{positive}(t)\}.\end{aligned}$$

There is an obvious symmetry for the right cover.

Focusing on either the left or right covering exclusively, we have Definition 18.

Definition 18 (Maximally covering sets)**.** A set of correspondences C is maximally left-covering of a description d if there is no correspondence set C' such that

$$d \cap \bigcup \{\mathrm{leftcover}(c') \mid c' \in C'\} \subseteq d \cap \bigcup \{\mathrm{leftcover}(c) \mid c \in C\},$$

where $\mathrm{leftcover}(c)$ is the set of non-negated components left covered component formula c. A set of correspondences can be similarly maximally right-covering.

Because a set of correspondences almost certainly will not cover a Q-description, there may be many maximally covering sets. This is because the subset operation is a partial order, and so all maximally covering sets are ‘equal’ in how much they cover (as measured by set cardinality). We instead must use some other deciding factor than coverage to make a final selection of *which* maximally covering set to use: in our case, the correspondence strengths are one possible discriminator; we will explain the *minimally redundant* condition in the next subsection.

Example 5.7. Take the set of correspondences

$$\begin{aligned}\mathcal{C} = \{&\langle\, a \text{ AND } b,\ x,\ 0.5\,\rangle,\\ &\langle\, a \text{ AND } c,\ y,\ 0.6\,\rangle,\\ &\langle\, b \text{ AND } c,\ z,\ 0.7\,\rangle\}\end{aligned}$$

and the four maximally covering subsets

$$\begin{aligned}
C_1 &= \{\langle a \text{ AND } b,\ x,\ 0.5\rangle, \\
&\qquad \langle a \text{ AND } c,\ y,\ 0.6\rangle\} \\
C_2 &= \{\langle a \text{ AND } b,\ x,\ 0.5\rangle, \\
&\qquad \langle b \text{ AND } c,\ z,\ 0.7\rangle\} \\
C_3 &= \{\langle a \text{ AND } c,\ y,\ 0.6\rangle, \\
&\qquad \langle b \text{ AND } c,\ z,\ 0.7\rangle\} \\
C_4 &= \{\langle a \text{ AND } b,\ x,\ 0.5\rangle, \\
&\qquad \langle a \text{ AND } c,\ y,\ 0.6\rangle, \\
&\qquad \langle b \text{ AND } c,\ z,\ 0.7\rangle\}
\end{aligned}$$

All of these are maximally left-covering of the components a, b, and c. In the following section we eliminate C_4 for not being minimally *redundant*, but still must choose between C_1, C_2, and C_3: we might consider the sum of their strengths to take C_3 as 'best'. ▽

5.3.2 *Minimally redundant sets*

The second condition of MRMC sets, minimal redundancy, is motivated by reducing 'overlap'. Covering the same component over and over may do very little to improve the suitability of the target representational system,[29] but can cause the informational suitability to increase: we have another satisfied correspondence, so we include it in the informational suitability calculation from Definition 15. This conflict—repetition of correspondences is not likely to indicate any benefit for the representational system, but does increase the value of the objective function for that RS-description—demands resolution. We require minimally redundant correspondence sets.

We now define the relative redundancy of correspondence sets.

Definition 19 (Less redundant sets)**.** A set of correspondences C is less redundant than correspondence set C' if the number of redundant elements in the union of the left-cover sets (respectively, right-cover sets) of the correspondences in C is less than the union of the left-cover sets (respectively, right-cover sets) of the correspondences in C'. Given $P = \biguplus\{\text{leftcover}(c) \mid c \in C\}$ and $P' = \biguplus\{\text{leftcover}(c') \mid c' \in C'\}$, if we have that

$$\sum_{p \in P} (\#p - 1) \leq \sum_{p' \in P'} (\#p' - 1)$$

then C is less redundant than C'. The operator $\biguplus S$ is the multiset addition of S, and $\#e$ is the count of element e in the multiset. Multiset addition counts the occurrences of each element in a union of sets.

The 'minus one' in Definition 19 is not strictly necessary, but has an intuitive sense: we are counting the 'extra' occurrences of p and p' in each set: a single occurrence is not redundant, but the repetitions are.

[29] If the representation can produce representations that can encode the information captured by the component, repeatedly stating this is unhelpful.

Example 5.8. Consider the set-of-sets $S = \{\{a,b\},\{a,b,c\},\{b\}\}$. The multiset addition of S is

$$\biguplus S = \{(a,2),(b,3),(c,1)\},$$

where the second value in each pair is the element multiplicity in the original set. Then the redundancy of S is $(2-1)+(3-1)+(1-1) = 3$.

We can reduce the redundancy of S by discarding $\{a,b\}$ and $\{b\}$, leaving $S' = \{\{a,b,c\}\}$. The multiset addition of S' is $\{(a,1),(b,1),(c,1)\}$ with redundancy 0. ▽

A *minimally redundant cover* of a set of components D is a minimally redundant set of correspondences C such that

$$\bigcup\{\text{leftcover}(c) \mid c \in C\} = D.$$

This is different to the *minimum cover* of D, which uses as few correspondences as possible—that is, the size of C is as small as possible.[30] In contrast, a minimally redundant cover might use very many correspondences, yet still be minimally redundant. The obvious case is the set consisting only of distinct correspondences without correspondence formulae: this uses the *most* possible correspondences in a minimum cover, yet has zero redundancy.

Unsurprisingly, an MRMC correspondence set is a set that is both minimally redundant and maximally covering.

Definition 20 (Minimally redundant and maximally covering set)**.** A set $C \subseteq \mathcal{C}$ of sets is minimally redundant and maximally covering if there is no set $C' \subseteq \mathcal{C}$ that is both less redundant and more covering of

$$\bigcup\{\text{leftcover}(c) \mid c \in \mathcal{C}\}.$$

We guarantee neither the existence nor uniqueness of an MRMC set—and practically, we do not need to. Instead, we are interested in getting a set of correspondences that is as MRMC as possible. Viewing Definition 20 as an optimisation problem, we see that there will be a frontier of sets that are all as MRMC as possible, while none are superior or inferior—we saw this in Example 5.7. A secondary condition can be used to select a final candidate, or a weighted cost function on the 'redundant-ness' and 'coverage' used to identify a contextually appropriate set. By setting this condition appropriately, we can direct the MRMC set of correspondences considered by the informational suitability objective function, biasing the results; setting this condition using empirical studies is future work.

5.3.3 *Implications of MRMC reduction*

The above gives us a definition for minimally redundant and maximally covering correspondence sets, but what is the implication of having them,

[30] This is analogous to the *minimum set cover* problem, only using sets of correspondences, not sets of sets.

or not? Suppose we determine a set of correspondences is *not* MRMC: so to address all the components of the source description, there exists at least one correspondence whose components of its left cover are already covered by the remaining correspondences, or are not in the description we are attempting to cover. The latter case is obvious: we discard the correspondence because it is unnecessary. In our example of summing integers, any correspondence covering only the primitive 'log' can safely be discarded.

The more complex case is when the correspondence's left-covered components *are* in the description we are covering, but we still determine the correspondence unnecessary. We identify three possible cases:

The correspondence is always unnecessary. It is possible that in no case would the discarded correspondence ever be included in an MRMC set. Then the correspondence should be removed permanently. Identifying such correspondences would, in general, be very difficult, but some simple cases (such as duplicated correspondences with different strengths) could be trivially resolved.[31]

This correspondence captures an overloaded concept. When the left cover of a correspondence captures an overloaded concept, in the cognitive sense, it is likely that there are many possible sets of components that the correspondence targets. In this case, using an OR connective in the correspondence target is possible, but we do not have a 'good' solution. The ambiguity causing problems for the framework is equally likely to cause problems for humans, too.

There are many possible analogies. A situation that is difficult to distinguish from the previous case, the components in the left cover might have many possible analogies in the target representational system. The components are not overloaded in the source representation, but there is redundancy in the target system. Using a OR connective would capture this choice that must be made when transforming the representation.

5.3.4 *NP-hardness and approximation*

When defining maximally covering sets, we provided a simple and obvious conversion from correspondence sets to sets of sets using the left and right *covers* of correspondences. This essentially converts a component formula into a set of its non-negated components. But because component formulae have the OR connective, this approach is overly conservative: correspondences that *may* overlap but actually *do not* are rejected, because by collapsing all the clauses into one 'super-clause', all clauses are implicitly considered true at the same time. Instead, we convert a set of correspondences into a set of sets, where each *clause* is converted into a set of components, rather than each correspondence.[32] Then, rather than collecting these transformed correspondences into a set, and forming a set of sets of sets, we union all the transformed correspondences, so again

[31] Indeed, we perform this check in the `robin` implementation.

[32] This is the same transformation we made in Example 5.1.

we have a set of sets. Thus a set of n correspondences can produce a set with more than n inner sets.

Example 5.9. Consider the R-description $\{a, b, c\}$, and the correspondence set

$$\{\langle (a \text{ AND } c) \text{ OR } (b \text{ AND } c),\ x,\ 1\rangle, \langle a,\ y,\ 1\rangle, \langle b,\ z,\ 1\rangle\}$$

If we performed the left-cover transformation on this correspondence set, our set-of-sets would be $\{\{a, b, c\}, \{a\}, \{b\}\}$ (focusing exclusively on the left hand side of the correspondences). Thus we would conclude the MRMC set in this case is $\{\{a, b, c\}\}$. But back in the realm of correspondences, this is not accurate: we could transform b AND c to x in the new representation, and transform a to y, completely bypassing the 'conflicting' clause a AND c.

Instead we propose the transformation from correspondence formulae to sets-of-sets which we *union* to produce our final set-of-sets. In this case, our set-of-sets is $\{\{a, c\}, \{b, c\}, \{a\}, \{b\}\}$, and one possible MRMC set is $\{\{b, c\}, \{a\}\}$. This translates back into the realm of correspondences to produce the MRMC set of correspondences that we would expect:

$$\{\langle (a \text{ AND } c) \text{ OR } (b \text{ AND } c),\ x,\ 1\rangle, \langle a,\ y,\ 1\rangle\}. \qquad \triangledown$$

Constructing maximally covering sets is straightforward: simply continue to add sets that contain an element not already in the union. But this trivial greedy algorithm fails to guarantee minimal redundancy. In fact, there is *no known* efficient algorithm that guarantees minimal redundancy, as constructing a minimally redundant set cover is NP-hard.[33]

Theorem 5. *Constructing a minimally redundant set cover is NP-hard.*

Our proof of this theorem relies on techniques that operate directly on sets of sets, rather than sets of correspondences. Thus, we can re-write Definitions 18 and 19 to be in terms of sets of sets, rather than sets of correspondences.[34]

Definition 21 (Maximally covering sets). Let $\mathcal{P}(U)$ denote the powerset of U. Given some universal set U, and a set of sets $\mathcal{S} \subseteq \mathcal{P}(U)$, a set $S \subseteq \mathcal{S}$ of sets is maximally covering of U if there is no set $S' \subseteq \mathcal{S}$ such that $\bigcup S \subsetneq \bigcup S'$.

Definition 22 (Less redundant sets). A set $S \subseteq \mathcal{S}$ of sets is less redundant that a set $S' \subseteq \mathcal{S}$ of sets if

$$\sum_{e \in \biguplus S} (\#e - 1) < \sum_{e' \in \biguplus S'} (\#e' - 1)$$

where $\biguplus S$ is the multiset addition of S.

[33] An NP-hard problem is one that is at least as hard as the hardest NP problems. These are 'intractable': we do not know an efficient way to solve these problems.

[34] Definition 20, of MRMC, will stand as written, assuming these new definitions are taken rather than the previous versions.

We now have all the pieces necessary to prove Theorem 5.

Proof. We construct a reduction from the set cover problem to the minimally redundant set cover problem; constructing a set cover is a known NP-hard problem [Karp 72]. What follows is a sketch of the proof—full details are in Appendix C.

Let $\mathcal{S} \subseteq \mathcal{P}(\mathcal{U})$ be a set of sets such that $\bigcup \mathcal{S} = \mathcal{U}$, where $\mathcal{U}$ is some 'universe' of elements we wish to cover. We wish to find $S \subseteq \mathcal{S}$ such that $\bigcup S = \mathcal{U}$ such that $|S|$ is minimal. This is the minimum set cover.

Take set $\mathcal{D} = d_1, \ldots, d_n$ as a set of dummy elements where $|D| = |U| + 1$ and $\mathcal{D} \cap \mathcal{U} = \varnothing$. Construct the set $\mathcal{S}' = \{s \cup \mathcal{D} \mid s \in \mathcal{S}\}$, that is we add all the dummy elements into every set in $\mathcal{S}$. Further, we define $\mathcal{U}' = \mathcal{U} \cup \mathcal{D}$.

Assume we have S'_*, a minimally redundant set cover from $\mathcal{S}'$ over $\mathcal{U}'$. Because every set contains all dummy elements, this minimally redundant set cover is also a smallest set cover. Then construct

$$S_* = \{s \setminus \mathcal{D} \mid s \in S'_*\}$$

which is a smallest set cover of $\mathcal{U}$ drawn from $\mathcal{S}$. If S'_* could be constructed efficiently, then so could S_*. But constructing is S_* is NP-hard, so constructing S'_* must also be NP-hard. □

Knowing the problem is NP-hard, the 'best' way to construct an MRMC correspondence set is to select every possible subset of the correspondence set, discarding those that do not satisfy the conditions of Definition 20. This is impractical for sets with more than a few dozen correspondences, as it requires checking $O(2^n)$ candidate subsets of a set of n correspondences.[35] We are left with two options: make sure we have very few correspondences, or get an answer that is good enough, quickly. Limiting the number of correspondences goes against the purpose of correspondences: we wish to accurately capture as much information as possible between representational systems. Further, the number of correspondences grows approximately quadratically with the number of representational systems (because ideally every system is linked to every other system by discovering correspondences between their RS-descriptions), meaning even a very small number of representational systems will likely produce more correspondences than can practically be handled.[36]

Instead, we perform a greedy traversal through the space of correspondence subsets, choosing a subset which increases our coverage as much as possible while increasing our redundancy as little as possible. The exact balance of coverage to redundancy is set through the cost function.[37] The greedy approach must check $O(kn) \in O(n^2)$ potential covers, where k is the number of correspondences in the final cover; k is bounded by n because a cover is at most n correspondences, and likely much less. The approach has complexity $O(kn)$ because it must consider $O(n)$ correspondences to add to the cover, k times.

[35] Technically, due to our modified conversion from correspondences to sets of sets, we need $O(2^{f(n)})$, where $f(n)$ is the number of clauses generated from the set of n correspondences.

[36] We have found just *two* representational systems will require discovering dozens of correspondences.

[37] The cost function drives us towards one possible 'solution', but because we proceed greedily we might 'miss' the valid solutions.

5.3.5 *Existing measures similar to MRMC*

The MRMC measure we defined has similarities to the existing maximum coverage and minimum redundant (MCMR) text summarisation problem in natural language processing [Alguliev et al. 11]. Alguliev et al. consider the problem of generating a summary of a set of documents: each document is a set of sentences, and the resulting summary is also a set of sentences, selected from the documents, that should capture as much information as possible while having little repetition. Our goal is analogous: from a set of correspondences, we wish to select those which cover the set of components as much as possible, but having minimal overlap. Alguliev et al. use a similarity metric to determine the similarity between a summary and a set of documents, and use this to set up an integer linear programming problem.

Also similar, but from machine learning more generally, minimum redundancy maximum relevance (mRMR) feature selection attempts to select a subset of features in a dataset such that each feature in the subset has high similarity to the target class, but low similarity to other features in the subset [Peng et al. 05]. They define this similarity, both for the relevance and redundancy of features, using mutual information;[38] again, they rely upon a similarity metric.

In this section, we avoided needing a similarity metric, and instead defined MRMC in the language of sets and covers (Theorem 5, and Appendix C). The result is equivalent: we have an NP-hard problem to solve, regardless of its formalisation.

SUMMARY OF SECTION 5.3

By ensuring the considered set of correspondences is minimally redundant prevents us from artificially *inflating* the informational suitability score; ensuring the set is maximally covering prevents us from artificially *deflating* the informational suitability score. While finding an MRMC set is NP-hard, we use a heuristically guided greedy algorithm to construct an approximate MRMC set efficiently.

5.4 Implementing robin

To better understand the details and behaviour of the framework that we have described, we explore an implementation of the framework. We present here a tool called robin, which reads and processes descriptions and correspondence sets, evaluates representational systems using the informational suitability function described earlier, and produces a ranked list of the representational systems. The entire implementation is available at https://github.com/rep2rep/robin. While this section presents the implementation in pseudocode, our executable implementation is in Standard ML, using the Poly/ML[39] interpreter and compiler.

[38] Mentioned in Section 4.2.3.

[39] polyml.org

5.4.1 *Maximising the objective function*

Our goal is to identify representational systems that might be suitable to solve our problem. We define TopRepresentations, Algorithm 1, to take a Q-description and return a list of RS-descriptions, sorted by their computed informational suitability for the problem specified by the Q-description.

Algorithm 1 Ordering the representational systems based on their informational suitability for a given problem.

```
function TopRepresentations(problem)
    repSystems ← load all RS-descriptions
    Sort(repSystems by InformationalSuitability(problem))
    Reverse(repSystems)
    return repSystems
end function
```

The flow is simple: evaluate all known representational systems, and sort them (descending) by their score. Currently, TopRepresentations only considers the informational suitability of representational systems. This is done in the InformationalSuitability function, Algorithm 2.

Algorithm 2 Computing the informational suitability from Definition 15.

```
function InformationalSuitability(problem, repSystem)
    correspondences ←
        SatisfiedCorrespondences(problem, repSystem)
    mrmc ← MRMC(correspondences, problem)
    score ← $\sum_{c \in mrmc}$ Strength(c) × LiftImportance(c, problem)
    return ⟨ repSystem, score ⟩
end function
```

The code for informational suitability closely resembles Definition 15: we select an MRMC set of correspondences that are satisfied by the relevant Q- and RS-descriptions, lift the appropriate importance from the Q-description to each correspondence, then compute the sum of the correspondence strengths modulated by these importances.

The InformationalSuitability function is defined in terms of many other functions, but we will only cover three in significant detail. The function Strength is an *accessor*, allowing access to one individual part of our custom types without Standard ML pattern matching. Mathematical operators such as summation and max behave as expected. But we will explore how importances are lifted from Q-descriptions to correspondences, how the satisfying set of correspondences is first selected, and how this set is refined to be (approximately) minimally redundant and maximally covering.

5.4.2 *Lifting importances to correspondences*

Importance is a property of the components of Q-descriptions but informational suitability acts over correspondences. So we need to move the relevant importance information from the description to the correspondences; for that we use the function LiftImportance, Algorithm 3. This proceeds by extracting the clauses of the source of the correspondence: the source of correspondence $\langle a, b, s \rangle$ is a, and because the component formula a is represented internally in disjunctive normal form, extracting the clauses is trivial. We then take the highest importance of any component in any satisfying clause, if said component is not preceded by a not connective.[40] The only open question is in determining if the Q-description satisfies the clause.

Transferring importance from components to correspondences.

Algorithm 3

```
function LiftImportance(correspondence, problem)
    clauses ← Clauses(Source(correspondence))
    importance ← max { Importance(c)
        | clause ∈ clauses, Satisfy(problem, clause),
        c ∈ clause, c ∈ problem }
    return importance
end function
```

Algorithm 4 follows Definition 12:[41] a component formula is satisfied if, in any clause in the formula, all the 'positive' components are present in the Q-description, and none of the 'negative' components are in the Q-description. Because we are checking only clauses, this is more general than necessary, but still straightforward.

Determining if a component formula satisfies a description.

Algorithm 4

```
function Satisfy(description, formula)
    for clause ∈ Clauses(formula) do
        b ← true
        for t ∈ clause do
            if IsPositive(t) then
                $b \leftarrow b \wedge t \in description$
            else
                $b \leftarrow b \wedge \text{Unwrap}(t) \notin description$
            end if
        end for
        if b then
            return true
        end if
    end for
    return false
end function
```

The function IsPositive determines if some term t is not preceded by

[40] We are checking clauses, not entire component formulae; thus, components in the clause are in the problem, or must have been preceded by a not connective.

[41] 'Satisfaction', Page 72.

a NOT operator, while UNWRAP unwraps a term of its NOT operator. The use of ∈ hides some complexity around checking whether two components are equivalent: we say the components 'match' (for the purposes of ∈) if they have identical kinds and values, or—if the kind is 'type'—the values unify.

5.4.3 *Finding all satisfied correspondences*

To compute the informational suitability, we need to know the set of correspondences we are summing over: this happens in two stages: first, we filter *all* correspondences down to *satisfied* correspondences; second, we compute an MRMC subset of these correspondences. In this section we consider the first step using SATISFIEDCORRESPONDENCES, Algorithm 5.

Algorithm 5 Select the correspondences which are satisfied by the Q- and RS-descriptions.

```
function SATISFIEDCORRESPONDENCES(problem, repSystem)
    corrs ← load all Correspondences
    corrs ← {CORRSATISFIED(c, problem, repSystem) | c ∈ corrs}
    return corrs
end function
```

Most of the complexity is hidden inside the check of whether a correspondence is satisfied by both the Q-description, and the RS-description, Algorithm 6. Fortunately, using SATISFY from Algorithm 4, this check is also straightforward.

Algorithm 6 Determine if a correspondence is satisfied.

```
function CORRSATISFIED(correspondence, problem, repSystem)
    source ← SOURCE(correspondence)
    target ← TARGET(correspondence)
    return SATISFY(problem, source) ∧ SATISFY(repSystem, target)
end function
```

5.4.4 *Approximating MRMC*

With the relevant set of correspondences selected, we must further refine the selection to a locally optimal approximation of a minimally redundant and maximally covering (MRMC) set.[42] As mentioned earlier, constructing an MRMC set is NP-hard—we would have to exhaustively check all subsets, which is prohibitively time-consuming. Instead, we approximate the solution with hill climbing to reach a locally optimal MRMC set.

The hill climbing algorithm, Algorithm 7, works by moving the current state towards the 'lowest'[43] point by continually stepping to the neighbour that is most steeply down from it.

[42] See Section 5.3.

[43] Traditionally, hill climbing goes up; our problem is better suited to *minimisation*, so our 'hill climbing' algorithm descends valleys.

Optimisation by hill climbing, or valley descending, in this case. Algorithm 7

```
function HillClimbing(Neighbours, Altitude, start)
    state ← start
    alt ← Altitude(start)
    loop
        near ← Neighbours(state)
        if near = ∅ then
            return state
        end if
        next ← $\text{argmin}_{n \in near}(\text{Altitude}(n) - alt)$
        if $alt \leq$ Altitude(next) then
            return state
        else
            state ← next
            alt ← Altitude(next)
        end if
    end loop
end function
```

We specialise the hill climbing algorithm to compute the MRMC set of correspondences, Algorithm 8, by defining the appropriate Neighbours, Altitude, and *state* parameters. The state is a combination of four values:

- the solution set (stored as a list); in the code we call this the *cover*;
- the covered Q-description components *cqs*, stored as a multiset;
- the covered RS-description components *crs*, also a multiset; and
- the list of correspondences yet to be considered for inclusion in the set cover.

The Neighbours of the state are one 'correspondence' away: pick an unused correspondence, find which components it covers, add those to the appropriate multisets, then move the correspondence into the cover. All neighbours are generated by picking all possible correspondences (here denoted NDPick). Our implementation has two 'quirks': first, we do not consider entire correspondences, but instead a single clause from the component formulae on each side; second, we consider how well we cover both the Q-description *and* the RS-description.

As we mentioned in Section 5.3.4, we consider the clauses of component formulae rather than the entire component formulae in correspondences precisely because they are in disjunction—x *or* y can be true, but both need not be. We use this to produce MRMC sets that, while minimally redundant, use *more* correspondences: picking the 'wrong' clauses could cause two correspondences to overlap more than picking the 'right' clauses, so we consider all possible pairs of clauses.

Algorithm 8 Refine the correspondence set to be minimally redundant and maximally covering.

```
function MRMC($correspondences, problem$)
    function Positive($clause$)
        return $\{t \in clause \mid \textsc{IsPositive}(t)\}$
    end function

    function Neighbours($state$)
        $\langle cover, cqs, crs, corrs \rangle \leftarrow state$
        $nbrs \leftarrow \varnothing$
        $picks \leftarrow \textsc{NDPick}(corrs)$
        for $\langle c, corrs' \rangle \in picks$ do
            $ss \leftarrow \textsc{Positive}(\textsc{Clauses}(\textsc{Source}(c)))$
            $ts \leftarrow \textsc{Positive}(\textsc{Clauses}(\textsc{Target}(c)))$
            for $\langle s, t \rangle \in ss \times ts$ do
                $nbrs \leftarrow nbrs \cup \{\langle cover \cup \{c\}, cqs \uplus \{s\}, crs \uplus \{t\}, corrs' \rangle\}$
            end for
        end for
        return $nbrs$
    end function

    function Altitude($state$)
        $\langle \_, cqs, crs, \_ \rangle \leftarrow state$
        $coverage \leftarrow |problem - cqs|$
        $redundancy_q \leftarrow \sum_{p \in cqs} (\#p - 1)$
        $redundancy_r \leftarrow \sum_{p \in crs} (\#p - 1)$
        return $coverage + redundancy_q + 0.5 \times redundancy_r$
    end function

    $init \leftarrow \langle \varnothing, \varnothing, \varnothing, correspondences \rangle$
    $\langle mrmc, \_, \_, \_ \rangle \leftarrow \textsc{HillClimbing}(\textsc{Neighbours}, \textsc{Altitude}, init)$
    return $mrmc$
end function
```

Example 5.10. Consider the component formulae (a AND b) OR c and (a AND d) OR b, for some components a, b, c, and d. These two formulae have, for a transformation based on component covers, redundancy 2: a and b both appear twice. But if the description satisfying these formulae does not contain property b, then the 'clauses of interest' in each formula are c and a AND d: there is no redundancy. ▽

We consider the covering of the RS-description as a secondary objective, as we will see in the ALTITUDE function: covering the RS-description is not strictly necessary, and perhaps not desirable. For example, not using every aspect of algebraic notation is expected: if your problem required *every component* of algebraic notation, this would be surprising.

Turning to the function ALTITUDE, we must consider how to evaluate a state to a real number such that a lower value represents a cover closer to a true MRMC set.[44] In this case, we stay close to Definitions 18 and 19, computing the (missing) coverage as the size of the difference between the set of components from the Q-description and the set of covered components, and the redundancy as the sum of the number of excess times each component occurs in the covering multiset. Because these are two (or three, with the RS-description coverage) real values, we must reduce these to a single value that our hill-climbing algorithm can follow. We choose to combine the values with a weighted sum, as it gives us sufficient freedom to direct the algorithm towards solutions we favour: the weights have been calibrated to produce appropriate MRMC sets, but future empirical testing would better justify the values given here.[45]

SUMMARY OF SECTION 5.4

In this section we outlined the robin implementation of the rep2rep framework. This implementation is fully automated from reading the descriptions through to making a recommendation based on the informational suitability calculation. We will use this implementation in the following chapters to evaluate our framework. The robin codebase is available at https://github.com/rep2rep/robin.

SUMMARY OF CHAPTER 5

This chapter described how we use components, descriptions, and correspondences in a framework designed to maximise the informational suitability and minimise the cognitive cost of an alternative representational system. So, in answer to our third research question, and its subsequent objectives, we have shown how to algorithmically evaluate and rank representational systems based on their potential to support human problem solvers when solving the problem they are faced with. We motivated and defined each objective function, developed what it means for correspondence sets to be minimally redundant and maximally covering, and discussed the implementation of these ideas. This

[44] We could alternatively compute a 'bigger is better' score then take the negative value.

[45] A further improvement would consider the importance and strength of which correspondences are kept, ensuring a 'best case' MRMC set.

chapter thus covers our second and third independent contributions, as well as the sixth contribution alongside the rep2rep research group. In the next chapter we demonstrate the generality and practical utility of our work.

Applying the framework 6

> If it is to be effective as a tool of thought, a notation must allow convenient expression not only of notions arising directly from a problem, but also of those arising in subsequent analysis, generalization, and specialization.
>
> — *Kenneth E. Iverson*

The preceding three chapters have introduced components, descriptions, correspondences, and the algorithm that combines these to produce a representational system recommendation. All this has been done in the context of mathematics—counting problems in this dissertation, and probability problems in the rep2rep project—but the rep2rep framework is not just applicable to mathematics. While Appendix D explores how we can generalise the correspondence framework to an abstract mathematical structure, here we take a more concrete approach.

In this chapter we will demonstrate the practical utility of the framework and its universal nature by applying it to the domain of programming languages: we will see how components, descriptions, and correspondences apply to programming languages, and use the same `robin` implementation already described. Rather than recommend a suitable representational system to solve a problem, we now recommend an appropriate programming language to implement an algorithm. We also use this example to explore how the analysts' descriptions and correspondences sets impact the final recommendation by the framework.

The generalisation to programming languages is my own work specifically for this dissertation, and has not yet been published.

6.1 Programs and programming languages

Stepping away from mathematics to programming languages forces us to make decisions: what is a 'representational system', or a 'problem' in this domain? We take an implementation of an algorithm (a *program*) to be a problem statement, analogous to how a representation of a problem statement is what we have considered a problem. The programming languages are the representational systems: an algorithm is stated within a programming language, just as a problem is stated within a system.

We use three programming languages (C, Standard ML, and Scheme) and three algorithms (merge sort, in-order tree traversal, and finding a longest common subsequence) as a test for the rep2rep framework. As well as outlining the exact bounds of each of these languages and algorithms, this section presents the implementations we will write descriptions for.

6.1.1 *Programming languages as representational systems*

The three programming languages—C, Standard ML, and Scheme—were chosen because they are all well understood languages with small grammars and sufficient standard libraries. They represent imperative, functional, and hybrid paradigms.

The C programming language is *imperative*: a program is a sequence of statements. The language by default has no automatic memory management, so all heap allocations must be requested and released by the programmer. While C has types, they are *static* and *weak*: every piece of memory is assigned a type, but C allows free reinterpretation of the underlying bytes regardless of their type. It makes extensive use of pointers in situations that other languages would consider distinct— arrays, references, higher order functions, and recursive data types are just four examples of all the concepts expressed through pointers. C is expressed at a low level of abstraction: the programmer must consider the machine semantics rather than the problem semantics.

Standard ML is a *functional* programming language: a program is a composition of expressions. It has automatic memory management—no need to explicitly acquire or release blocks of memory—and a *static* and *strong* type system: every expression is given a type, and this type is fixed (to change types, values must be explicitly converted).[1] *Algebraic data types* allow the programmer to quickly and easily create new product or tagged union types. Standard ML eschews loops in favour of recursion, or higher order functions that abstract recursion. While its standard library (the 'Basis Library') is small by modern standards, it is sufficient for the problems we focus on in this chapter. Standard ML is a high level language: the programmer focuses on how data flows through the algorithm, rather than how the machine operates.

Scheme is a dialect of Lisp, meaning it is a hybrid of many programming paradigms (although Scheme follows a more functional approach than other Lisps). It has automatic memory management, like Standard ML, but a *dynamic*, *strong* type system: types are fixed (an integer cannot be added to a character, for example), but only at the last moment (i.e., only checked when the value *must* be a particular type). By default, Scheme has no way to extend the set of types—the programmer is expected to interpret specific structures built from existing types as new types. Scheme, as used in this chapter, is a high level language; were we to make use of macros, the language becomes capable of even more abstraction, able to mutate to suit the specific problem.

6.1.2 *Programs as problem statements*

We chose three algorithms to implement in each programming language: merge sort, in-order tree traversal, and finding the longest common sub-

[1] Famously, Standard ML can *infer* types for the programmer, meaning there is no need for type annotations seen in other static languages.

sequence. Where the algorithm is ambiguous, we favour an approach that is idiomatic to each language. We aimed for concise code.[2] All the implementations are available in Appendix E, but we choose one language for each problem to demonstrate an implementation.

MERGE SORT

Merge sort is an $O(n \log n)$ sorting algorithm, commonly summarised as 'easy split, hard join.' It proceeds by splitting a list in half repeatedly, until no sublist has more than one element. Neighbouring sublists are then merged until they form a single list. The merge procedure looks at the 'front' of each sublist, takes the smaller element, and appends that onto the resulting merged list; this is repeated until one or other list is empty, at which point the remaining elements are appended to the merged list.

Note that splitting a list into equal halves can be done in many ways: in Standard ML and Scheme, we split the list into values that were in even or odd positions; in C, we shifted a pointer to the index half way along the array. Note this means only the C implementation is stable.[3]

Merge sort, implemented in Scheme.

Listing 4

```
(define (split xs)
  (cond ((null? xs) '(() . ()))
        ((eq? 1 (length xs)) `(,xs . ()))
        (else (let* ((x (car xs)) (y (cadr xs))
                     (rest (cddr xs))
                     (splits (split rest))
                     (xs (car splits)) (ys (cdr splits)))
                `(,(cons x xs) . ,(cons y ys))))))

(define (merge xs ys)
  (cond ((null? xs) ys)
        ((null? ys) xs)
        (else (let* ((x (car xs))
                     (y (car ys)))
                (if (<= x y)
                    (cons x (merge (cdr xs) ys))
                    (cons y (merge xs (cdr ys))))))))

(define (mergesort lst)
  (cond ((null? lst) lst)
        ((eq? 1 (length lst)) lst)
        (else (let* ((splits (split lst))
                     (xs (mergesort (car splits)))
                     (ys (mergesort (cdr splits))))
                (merge xs ys)))))
```

[2] Some code has been reformatted to fit this page width.

[3] A *stable* sorting algorithm ensures that if two values compare equal, they are in the same order in the sorted list as the input list. For example, when sorting the list `["z", "a"]` based on string length, a stable sort guarantees the list is unchanged. An *unstable* sort makes no such promise.

in-order tree traversal

An in-order tree traversal walks a binary tree, and returns a list that contains all the values of the tree in the order they would be scanned left-to-right. That is, an in-order tree traversal recursively traverses the left subtree, then yields the value in the current node, then recursively traverses the right subtree. A leaf yields no values and causes no recursion.

In the Standard ML implementation, we define a typical tree datatype. In Scheme, we encode the tree using nested lists of length three, and leaves are `'()`. In C, we define a `struct` (composite data structure) with three slots, then define a tree to be a pointer to this `struct`. The first slot in the `struct` is the value of the tree node, while two of the slots in the `struct` are recursive, expecting trees. Leaves are `NULL`.

Listing 5 In-order tree traversal, implemented in Standard ML.

```
datatype 'a tree = Leaf
                 | Branch of 'a * 'a tree * 'a tree;

fun inorder Leaf = []
  | inorder (Branch (v, l, r)) =
              (inorder l) @ (v::(inorder r));
```

longest common subsequence

A common subsequence of two strings is a sequence of characters that appear in the same order in both strings, but are not necessarily contiguous in either. The longest common subsequence is the longest such sequence of characters, and is not necessarily unique. For example, one possible longest common subsequence of `thisisatest` and `testing123testing` is `tsitest`.

For this algorithm, we used dynamic programming rather than recursion. We did this because the previous two algorithms were inherently recursive, so we chose a solution that was naturally iterative as a point of comparison:[4] we do not want to present three algorithms that are 'easy' to implement in Scheme and Standard ML, disadvantaging C in all cases. This meant using the `Array` and `Array2` modules in Standard ML, and the vector libraries in Scheme.

Listing 6 The longest common subsequence algorithm, implemented in C.

```
int lcs(char* a, int a_len, char* b, int b_len, char** out)
{
  int* grid = calloc((a_len + 1) * (b_len + 1), sizeof(int));
  int ravel = b_len + 1;

  for (int i=1; i < a_len + 1; i++) {
```

[4] It also makes the program run in polynomial time, rather than exponential time.

```
    for (int j=1; j < b_len + 1; j++) {
      if (a[i-1] == b[j-1]) {
        int l = grid[ravel * (i-1) + j-1];
        grid[ravel * i + j] = l + 1;
      } else {
        int l1 = grid[ravel * i + j - 1];
        int l2 = grid[ravel * (i-1) + j];
        if (l1 > l2)
          grid[ravel * i + j] = l1;
        else
          grid[ravel * i + j] = l2;
      }
    }
  }

  int ss_len = grid[ravel * a_len + b_len];
  *out = calloc((ss_len + 1), sizeof(char));
  int k = ss_len - 1;
  int i = a_len;
  int j = b_len;
  while (k >= 0) {
    if (a[i-1] == b[j-1]) {
      (*out)[k] = a[i-1];
      i--; j--; k--;
    } else {
      if (grid[ravel * i + j - 1] > grid[(i-1) * ravel + j])
        j--;
      else
        i--;
    }
  }
  free(grid);
  return ss_len;
}
```

6.2 Descriptions and correspondences

In Section 5.1 we described the workflow of each analyst in the framework. Briefly summarised, the workflow is

For RS-analysts:

1. (Optional) Collect example representations.
2. (Optional) Generate R-descriptions of the examples.
3. Generalise over the examples *or* define the bounds of the system.

4. Identify types and primitives.
5. Identify patterns.
6. Identify tactics and laws.
7. Define the probabilities of the components.

For R/Q-analysts:

1. Choose the appropriate RS-description to reference.
2. Identify the subset of components from the RS-description to describe the representation.
3. Annotate the components with occurrences attributes.
4. (For Q-descriptions) Assign importances to components.

For correspondence analysts:

1. Identify descriptions of 'equivalent' representations.
2. Find one-to-one correspondences between components.
3. Find m-to-n correspondences between components.
4. Using the probabilities from the RS-descriptions, assign strength.
5. Use the `findcorr` tool to add more correspondences.

In this section, we apply this workflow. We begin by describing the programs we are considering, which is slightly more difficult in our case than in the case of the Q-analyst in general because we do not yet have RS-descriptions, even a fragment. We begin by creating the Q-descriptions before the RS-descriptions for two reasons: first, steps 1 and 2 of the RS-analyst workflow are to find examples of, and create descriptions for, specific representations and problems; second, from our experience creating a Q-description without any RS-descriptions is easier than the converse. Once we have the Q-descriptions we exploit the `uniondesc` tool to speed up the creation of RS-descriptions. Missing components are added, Q-descriptions are updated where necessary, and correspondences are generated through a mixture of identifying correspondences by hand, and by using the `findcorr` tool.

6.2.1 *Describing programs (Q-descriptions)*

Using the code listings in Appendix E, we create nine Q-descriptions: one for each program in each programming language. To illustrate the procedure we followed, we construct the description for Listing 4 (merge sort in Scheme). We use the textual format introduced in Section 3.2.3. Getting some boiler-plate out of the way, we begin with the necessary pieces of a description. We also include the `modes 1` component[5] immediately: these are all purely sentential representational systems.

The initial skeleton of our Q-description about merge sort in Scheme. Listing 7

```
representation MergeSort_Scheme = rep
    (* Merge sort, as implemented in Scheme *)
    modes 1;
end;
```

The rest of the description will sit after the `modes` declaration, and before the `end` delimiter. We will no longer show this outer layer; all future Q-description snippets can be assumed to be inserted here.

In Section 5.1.2 we outlined the analyst workflow to generate R- and Q-descriptions, but this workflow depends on a library of RS-descriptions—a library we do not yet have available to us, because we have not made it. Instead, we shall assume an appropriate RS-description exists and assume we are selecting components from it, while in reality we are generating the components as we go. These components will become part of our RS-description. We have thus implicitly completed step 1 of the R/Q-analyst workflow. What follows are sub-steps of step 2: we select the appropriate components first by type, then primitives, then finally patterns. We do not select laws and tactics that a programmer might use when dealing with a programming language: actions such as renaming or refactoring are beyond the scope of this example. For each component we identify, we perform steps 3 and 4: count and assign occurrences, and assign importance.

TYPES

We begin our analysis in step 2 of the workflow with the *types* of the program: what are the grammatical roles? Two immediately obvious types are `number`, and `number list`, as these are fundamental to any numerical list sorting program. Types are ‘uncountable’, and so we do not assign an ‘occurrences’ attribute. As per step 4, we consider these essential. Because Scheme is dynamically typed, the grammar is quite weak. Finally, there is a relevant type `bool`—it’s relevant to the solution, but not integral to the algorithm. At the *algorithmic* level, merge sort makes *decisions*; the Boolean is an implementation artefact.

The types associated with our Scheme merge sort implementation. Listing 8

```
essential types number, number list;
relevant type bool;
```

[5] See the end of Section 3.4.2.

PRIMITIVES

Primitives are simpler to identify than types, because they are 'things you see.' We first consider those which are essential; an essential component cannot be substituted without changing the nature of the underlying concept expressed. The essential primitives are few: mergesort, merge, split, define, and cons. The first three are obvious: the merge sort itself, and its building blocks. The primitive define allows us to create the function mergesort, and so is an essential primitive. The primitive cons is essential to the definition of a *list* in a functional programming language, and a number list is an essential type. Thus we consider cons an essential primitive.

Listing 9 The essential primitives in our Scheme merge sort implementation.

```
essential primitive mergesort
    where type = number list -> number list,
          occurrences = 3;
essential primitive merge
    where type = number list * number list -> number list,
          occurrences = 4;
essential primitive split
    where type = 'a list -> 'a list * 'a list,
          occurrences = 3;
essential primitive define
    where type = identifier list * sexp list -> statement,
          occurrences = 3;
essential primitive cons
    where type = 'a * 'a list -> 'a list,
          occurrences = 4;
```

Moving down the importance scale, we consider the instrumental primitives. Instrumental components cannot be substituted in this particular description, but will not necessarily appear in every program of this algorithm. In this Scheme implementation of merge sort, there are quite a few of these components: decisions I made when writing the code that are not required, but once in place are not trivially changed. Examples include using particular comparison operators,[6] functions such as car and cdr, and constants like 1 and nil.

A design decision made when creating these descriptions was to separate the parameter names and the local variable names across two layers of importances. Parameter names are considered instrumental, while local variable names are only relevant. While parameter names *could* be substituted without changing the meaning of the program, they form part of the interface to the function, so are more important than local variable names.[7] At the same time, they are less important than the names of functions (which we categorised as essential); instrumental is a suitable compromise.

[6] We use zero occurrences to insert components that are *related*, such as the other comparison operators. This provides more opportunities to satisfy correspondences.

[7] A similar argument is often made in the mathematics domain: the letters in $y = mx + c$ *could* be substituted, but this would be surprising.

Instrumental primitives identified in the merge sort Scheme program. Listing 10

```
instrumental primitive <=
    where type = number list -> bool, occurrences = 1;
instrumental primitives <, >=, >
    where type = number list -> bool, occurrences = 0;
instrumental primitive cond
    where type = (bool * 'a) list -> 'a, occurrences = 3;
instrumental primitive let*
    where type = (identifier * sexp) list * 'b -> 'b,
          occurrences = 3;
instrumental primitive null?
    where type = 'a list -> bool, occurrences = 4;
instrumental primitive eq?
    where type = number * number -> bool, occurrences = 2;
instrumental primitive length
    where type = 'a list -> number, occurrences = 2;
instrumental primitive car
    where type = 'a list -> 'a, occurrences = 5;
instrumental primitive cdr
    where type = 'a list -> 'a list, occurrences = 4;
instrumental primitive 1
    where type = number, occurrences = 1;
instrumental primitive nil
    where type = 'a list, occurrences = 3;
instrumental primitive else
    where type = bool, occurrences = 3;
instrumental primitive lst
    where type = number list, occurrences = 6;
instrumental primitive xs
    where type = number list, occurrences = 17;
instrumental primitive ys
    where type = number list, occurrences = 10;
```

Finally, *relevant* is the lowest level of importance we consider in our programming example. A relevant component cannot be removed from this particular description, but they could be substituted with alternative components without changing the expressed underlying concept.[8] The primitives at this layer include the local variable names, as mentioned earlier, and various programming constructs that can be replaced with alternatives. For example, `if` can be replaced with a `cond`, and `cadr` and `cddr` are short-hand for nested `car` and `cdr` function calls. Some primitives here are also necessary as part of the programming language, but do not convey much information. In Scheme, an obvious example are parentheses.

[8] The remaining, unused importance keywords are *circumstantial* (components could be removed safely) and *noise* (components are actively unhelpful).

Listing 11 Finally, the relevant primitives from the merge sort Scheme program.

```
relevant primitives x, y
    where type = number, occurrences = 3;
relevant primitive rest
    where type = number list, occurrences = 2;
relevant primitive splits
    where type = number list * number list, occurrences = 6;
relevant primitive if
    where type = bool * 'a * 'a -> 'a, occurrences = 1;
relevant primitive cadr
    where type = 'a list -> 'a, occurrences = 1;
relevant primitive cddr
    where type = 'a list -> 'a list, occurrences = 1;
relevant primitives $quote, $quasiquote
    where type = sexp -> 'a list, occurrences = 1;
relevant primitive $unquote
    where type = sexp -> 'a, occurrences = 1;
relevant primitive $pair
    where type = 'a * 'a -> ('a * 'a), occurrences = 1;
relevant primitives (, )
    where type = delimiter, occurrences = 73;
```

PATTERNS

We must also describe the patterns in a program. Scheme patterns are interesting: because the syntax is so regular, there are very few patterns that are not captured as the types of primitives.[9] We identify two patterns present in this program, both essential: `eval_sexp`, and `recursion`. The recursion pattern describes a function that operates by calling itself, a common idiom in Scheme.[10] The pattern `eval_sexp` encodes the evaluation of an s-expression to it's result. Note that this is a *pattern*, not a *tactic*: tactics are actions the user takes, but the user is not evaluating the s-expression. As a pattern, `eval_sexp` allows us to define the grammatical constraints of the Scheme programming language—if an s-expression evaluates to a value of type α, then we can use that s-expression wherever we require a value with type α.

Listing 12 All the patterns found in the merge sort Scheme program.

```
essential pattern eval_sexp
    where type = 'a, holes = [sexp: 1], occurrences = 38;
essential pattern recursion
    where type = statement,
          holes = [statement: 1],
          occurrences = 3;
```

[9] Technically, we should list both the primitive with a compound type and the pattern it induces. Fortunately, the `robin` implementation will automatically derive the pattern from the compound type for us.

[10] Interestingly, the Scheme specification *requires* all implementations to optimise tail calls.

By combining the description snippets so far, we end up with a complete Q-description of Listing 4.[11] We perform a similar process for all the programs in all the languages in our example. These descriptions are sufficient, but lack one thing: there is no link between the functions, and what forms their definition. To encode this, we created 'definition patterns'.[12] These patterns group together the primitives that form a function definition. These are unused by the robin implementation—it was not designed for them, and we remove them before producing the RS-descriptions so they do not appear in any correspondences—but they do exemplify the expressive possibilities of the description format. For example, to describe the merge sort function definition, we might add the following pattern given in Listing 13.

Listing 13

A pattern grouping together all the tokens that form the merge sort function definition in Scheme.

```
essential pattern mergesort_def
    where type = statement,
          occurrences = 1,
          primitives = [
              (, ), define, mergesort, lst, cond,
              null?, eq?, else, let*, length, car, cdr,
              split, merge, splits, xs, ys, 1];
```

6.2.2 *Describing languages (RS-descriptions)*

Now that we have analysed all of our examples, we can combine all these Q-descriptions into one RS-description. Steps 1 and 2 of the RS-analyst workflow—collecting examples and creating Q-descriptions—are complete. Thus we proceed to step 3 in the analyst workflow: combining and generalising. The combining can be done by hand, but is largely mechanical: copy the component, and remove the importance keyword and occurrences attribute. As mentioned in Section 5.1.2, we have created the uniondesc tool for exactly this purpose. Applying the uniondesc tool to the Q-descriptions from the Standard ML programming language, we create the RS-description in Listing 14. All the completed RS-descriptions are in Appendix G.

Listing 14

The automatically computed 'RS-description' for Standard ML. Some components have been collapsed where appropriate for space, and blank lines added for conceptual separation. In the holes attribute, a count of #t means the number of holes is proportional to the number of primitives used to instantiate the pattern.

```
representation StandardML = rep
    modes 1;

    types bool, char, int, string,
```

[11] Every Q-description appears in Appendix F.

[12] 'Definition patterns' are unnecessary for the problems we have considered in the rest of this dissertation, and the wider rep2rep project. As such, they are not part of the framework we have described. However, the framework is capable of expressing these patterns, as we demonstrate.

```
      int array, char list, int list, int tree;

primitives |, (, ), [, ], =, =>, then, else, of, in, end
    where type = delimiter;
primitive _ where type = identifier;
primitive if where type = (bool * ('a * 'a)) -> 'a,
                   primitives = [then, else];
primitive let where type = (statements * 'a) -> 'a,
                    primitives = [in, end];
primitive val
    where type = ('identifier * 'a) -> statement,
          primitives = [=];
primitive fn
    where type = (identifier * 'a) -> ('b -> 'a),
          primitives = [=>];

primitives $eq, <> where type = ('a * 'a) -> bool;
primitives <, <=, >, >=
    where type = (int * int) -> bool;
primitives  +, -, Int_max
    where type = (int * int) -> int;

primitive $cons where type = ('a * 'a list) -> 'a list;
primitive @ where type = ('a list * 'a list) -> 'a list;
primitive List_rev where type = 'a list -> 'a list;

primitive String_size where type = string -> int;
primitive String_sub
    where type = (string * int) -> char;
primitive String_implode
    where type = char list -> string;

primitive Array2_RowMajor where type ArrayTraversal;
primitive Array2_sub
    where type ('a array * (int * int)) -> 'a;
primitive Array2_tabulate
    where type = ArrayTraversal
                     -> (int * int * ((int * int) -> 'a))
                         -> 'a array;
primitive Array2_update
    where type = ('a array * int * int * 'a) -> unit;

primitives 0, 1, i, j, x, y, xl, yl where type = int;
primitives xstring, ystring where type = string;
primitive v where type = 'a;
primitive $nil where type = 'a list;
primitive ans where type = char list;
```

```
primitives xs, xs', ys, ys', zs, lst
    where type = int list;
primitive table where type = int array;
primitive tree where type = type -> type;
primitives Leaf, r, l where type = 'a tree;
primitive Branch
    where type = ('a * 'a tree * 'a tree) -> 'a tree;
primitive inorder where type = 'a tree -> 'a list;
primitive mergesort where type = int list -> int list;
primitive lcs where type = string -> string -> string;
primitive split
    where type = int list -> (int list * int list);
primitive merge
    where type = int list -> int list -> int list;
primitive loopx where type = int -> unit;
primitive loopy where type = int -> int -> unit;
primitive reconstruct
    where type = char list -> int -> char list;
primitive getTable where type = (int * int) -> int;
primitive setTable
    where type = (int * int) -> int -> unit;

pattern datatype where type = statement,
                       holes = [guarded_datatype: #t,
                                identifier: #t],
                       primitives = [datatype, =, of];
pattern typeguard where type = guarded_datatype,
                        holes = [identifier: #t],
                        primitives = [|, of];
pattern fun where type = statement,
                  holes = [guarded_declaration: #t,
                           identifier: #t, 'a: 1],
                  primitives = [fun, =];
pattern guard where type = guarded_declaration,
                    holes = [identifier: #t, 'a: 1],
                    primitives = [|, =];
pattern pair where type = ('a * 'b),
                   holes = ['a: 1, 'b: 1],
                   primitives = [,. )];
pattern recursion where type = statement,
                        holes = [statement: 1];

pattern inorder_def
    where type = statement,
          primitives = [fun, inorder, =, |, (, ),
                        Leaf, Branch, $nil, $cons, @,
                        v, l, r];
    pattern lcs_def
        where type = statement,
              primitives = [
                  fun, lcs, =, |, (, ), =>, fn, let, in, end,
                  val, if, then, else, $cons, $nil, 0, 1, _,
                  xstring, ystring, xl, yl, table, i, j, ans,
                  getTable, setTable, loopx, loopy, +, −, <>,
                  reconstruct, Int_max, Array2_tabulate,
                  Array2_RowMajor, Array2_sub, Array2_update,
                  String_size, String_sub, String_implode,
                  List_rev];
    pattern mergesort_def
        where type = statement,
              primitives = [
                  fun, mergesort, =, |, let, in, end, val,
                  (, ), $nil, $openlist, $closelist, split,
                  merge, lst, x, xs, ys, xs', ys'];
    pattern merge_def
        where type = statement,
              primitives = [fun, merge, =, |, (, ), <=, if,
                            then, else, $cons, $nil, x, y,
                            $openlist, $closelist, xs, ys];
    pattern split_def
        where type = statement,
              primitives = [fun, split, =, |, (, ), xs, ys,
                            let, in, end, val, $cons, $nil,
                            $openlist, $closelist, x, y, zs];
    pattern tree_datatype
        where type = statement,
              primitives = [datatype, tree, =, |, of,
                            $type−∗, 'a, Branch, Leaf];
end;
```

This is not yet a suitable RS-description, but is a foundation from which we can create one. In our RS-analyst workflow, step 3 requires the analyst to generalise over the examples to understand the underlying system; we now apply transformations to the automatically generated 'RS-description' to provide a suitably abstract description of the Standard ML programming language. These are steps 4, 5, and 6 of the RS-analyst workflow: finding which components are appropriate. In this case, we do this by filtering and augmenting our foundation generated by `uniondesc`

First, we strip the pattern components for 'definitions': `inorder_def`, etc. These are not a part of the programming language (which is the representational system, and so these patterns are not part of the RS-description), but are part of the program: to say that a 'merge sort definition' is part of Standard ML is the same as requiring that every conceivable function definition must belong in an RS-description. Thus, we remove these from the description immediately. Similarly, we strip out the miscellaneous variable and function names that appeared (i, j, mergesort, getTable): these are not part of the language; similarly numbers are abstracted.[13] But these tokens still 'exist'—they are all strings and numbers, which we import as primitives from other RS-descriptions:

```
import terms as primitives from LatinAlphabet;
import terms as primitives from RealNumerals;
```

The union of the Q-descriptions meant types such as char list and int list are included. In RS-descriptions we can be more general, and so we instead replace all instantiated type constructors with their uninstantiated variants and type variables: 'a list and 'a array. We completely remove the tree type, as well as the tree primitive and tree_datatype pattern associated with it; trees are not part of the Standard ML language.

We do not strip out functions and types that are part of the Basis Library, the collection of functions, types, values, and structures that are distributed with Standard ML. These are an essential part of the language.

Finally, we add in missing components. Components are missing because the union of the programs we described did not contain all primitives, types, and patterns. Conceptually, this must include the entirety of the Basis Library: in this example, we omit these for brevity. However, we have described a sufficient fragment of the standard libraries for all the problems of interest. We also add features that would come up in correspondences with the other programming languages, but do not appear in these programs. References are a good example in Standard ML, sharing many similarities with pointers in C. Thus we add a type 'a ref, and associated primitives ref and !. Option types are similar.

6.2.3 *Correspondences between languages*

Before discussing the correspondences between the languages, we briefly turn our attention to component probabilities. For this example, we set the correspondence strengths (derived from component probabilities) using expert judgement. We have too few examples to accurately calculate component probabilities (either along or in cross-language conjunctions). This is a limitation of this case study: a truly large corpus would more accurately demonstrate how these probabilities come into play. But to avoid bias, we fix the correspondence strengths *before* running robin. This means the strengths are perhaps not as fine-tuned as they should be, but more fairly presents the outcome of the framework without optimisations to get the recommendations we expect.

We now link the programming language RS-descriptions with correspondences. Following the correspondence analyst workflow from Section 5.1.3, we already have our example R-descriptions. For this example,

[13] In Listing 14, this is all the primitives from 0, 1,... to setTable.

we primarily create the correspondences between C and Standard ML, and Scheme and Standard ML, manually; we then use the `findcorr` tool to begin linking Scheme and C, before finishing the correspondence set by hand. We will not outline all the correspondences, only an interesting subset. A full correspondence set appears in Appendix H.

Step two of the correspondence analyst workflow involves finding the 'simple' correspondences. This is a manual process, the results of which are later used to bootstrap the automatic correspondence inference. We begin linking Standard ML and C by linking their types. Most are automatically derived by the `robin` implementation and need not be written down (`int`s are `int`s, for example). One interesting correspondence links many disparate Standard ML constructs with C pointers.[14]

$$\begin{aligned}&\langle\ \text{type}\ \alpha\ \texttt{list}\ \textsc{or}\ \text{type}\ \alpha\ \texttt{ref}\ \textsc{or}\ \text{type}\ \alpha\ \texttt{array}\ \textsc{or}\ \text{type}\ \texttt{string},\\ &\quad \text{type}\ \alpha\ \texttt{pointer},\ 1\ \rangle\end{aligned}$$

That is, seeing any of references, lists, arrays, or strings guarantees that the equivalent C program will use pointers.

Other correspondences are quotidian:

$$\langle\ \text{token}\ \texttt{+}\ \textsc{and}\ \text{token}\ \texttt{1},\ \text{token}\ \texttt{++},\ 0.7\ \rangle$$

encoding the link from a + and a `1` to C's ++. Note we set the strength to 0.7: incrementing is a common operation, and accounts for many uses of + and `1`; there are, of course, many other uses of + and `1`. You would expect the reversed correspondence from C to Standard ML to be slightly stronger, but we argue it is *weaker*: the idiom `i++` (for some loop variable `i`) is common in C but often replaced with other idioms in Standard ML (for example direct recursion on the data structure, or maps, filters, and folds). So observing a ++ in C is less predictive of a + and a `1` in Standard ML than the reverse.[15]

As we move through the components, we observe more sophisticated correspondences are required. The pointer and increment operator correspondences highlight the abstraction disparity—that is, how 'low-level' the language is—between our programming languages, and we see this again with a pattern identified in C: the 'write and increment index' pattern, usually something like `a[i++] = b`.[16] The pattern component is

$$\begin{aligned}\text{pattern}\ \texttt{writeIndexInc} : \{\, &\text{type} \coloneqq \mathit{statement};\\ &\text{holes} \coloneqq [(\text{identifier}, 3)];\\ &\text{primitives} \coloneqq [\texttt{++}, \texttt{=}];\ \}\end{aligned}$$

There are many components in Standard ML that correspond to this pattern; in our language fragment, we identify four. As a correspondence, this is

$$\begin{aligned}&\langle\ \text{primitive}\ \texttt{@}\ \textsc{or}\ \text{primitive}\ \texttt{Array2\_tabulate}\ \textsc{or}\\ &\quad \text{primitive}\ \texttt{List\_rev}\ \textsc{or}\ \text{primitive}\ \texttt{String\_implode},\\ &\qquad \text{pattern}\ \texttt{writeIndexInc},\ 1\ \rangle\end{aligned}$$

[14] We elide attributes in correspondences.

[15] Such analysis is an artefact of limited data; computing precise probabilities from data sets would allow us to set the strength from Definition 13.

[16] Why is there no looping construct? Because it might be `for`, `while`, or `do`; this pattern is about the *content* of the loop.

Seeing any one of concatenation (`@`), tabulation, reversal, or string implosion guarantees the 'write and increment index' pattern will appear in an equivalent C program. Similarly, C's structs find many uses: product types, tagged union types, ad hoc pairs, and records. C uses one implementation for many different concepts; there is a *deficit*, in terms of concept mapping,[17] in the language.

With a largely complete set of correspondences between C and Standard ML, and Scheme and Standard ML, we reach step five of the correspondence analyst process:[18] we use the `findcorr` tool to link Scheme with C. We highlight three derived correspondences of interest. The `findcorr` tool produces this derivation:

$$\frac{\begin{array}{c}\langle\,\text{primitive}\,\texttt{define},\ \text{pattern}\,\texttt{fun},\ 1\,\rangle\\ \langle\,\text{pattern}\,\texttt{fun},\ \text{pattern}\,\texttt{funcdef},\ 1\,\rangle\end{array}}{\langle\,\text{primitive}\,\texttt{define},\ \text{pattern}\,\texttt{funcdef},\ 1\,\rangle}\ [\textsc{cmp}]$$

of a correspondence between the Scheme primitive `define`, and the C pattern `funcdef`, using the intermediate Standard ML pattern `fun`. Similarly, the derivation

$$\frac{\begin{array}{l}\langle\ \text{primitive}\,\texttt{eq?}\ \textsc{and}\ (\text{primitive}\,\texttt{if}\ \textsc{or}\ \text{primitive}\,\texttt{unless}\ \textsc{or}\\ \qquad\qquad\text{primitive}\,\texttt{when}\ \textsc{or}\ \text{primitive}\,\texttt{cond}),\\ \quad\text{pattern}\,\texttt{guard}\ \textsc{or}\ \text{token}\,\texttt{case},\ 1\ \rangle\\ \quad\langle\ \text{pattern}\,\texttt{guard}\ \textsc{or}\ \text{token}\,\texttt{case},\\ \qquad(\text{primitive}\,\texttt{if}\ \textsc{and}\ \text{primitive}\,\texttt{==})\ \textsc{or}\ \text{primitive}\,\texttt{switch},\ 1\ \rangle\end{array}}{\begin{array}{l}\langle\ \text{primitive}\,\texttt{eq?}\ \textsc{and}\ (\text{primitive}\,\texttt{if}\ \textsc{or}\ \text{primitive}\,\texttt{unless}\ \textsc{or}\\ \qquad\qquad\text{primitive}\,\texttt{when}\ \textsc{or}\ \text{primitive}\,\texttt{cond}),\\ \quad(\text{primitive}\,\texttt{if}\ \textsc{and}\ \text{primitive}\,\texttt{==})\ \textsc{or}\ \text{primitive}\,\texttt{switch},\ 1\ \rangle\end{array}}\ [\textsc{cmp}]$$

links together different equality checks in Scheme and C via Standard ML parameter guards or case expressions.[19]

One further correspondence that we *disagree* with, but consider its derivation enlightening, is the correspondence between Scheme's `numbers`, and Standard ML's `ints`. This could be derived many ways, but one interesting route is via string indexing.

$$\frac{\begin{array}{c}\text{primitive}\,\texttt{string-ref} : \{\text{type} \coloneqq \texttt{string} \times \texttt{number} \rightarrow \texttt{char}\}\\ \text{primitive}\,\texttt{String\_sub} : \{\text{type} \coloneqq \texttt{string} \times \texttt{int} \rightarrow \texttt{char}\}\\ \langle\,\text{primitive}\,\texttt{string-ref},\ \text{primitive}\,\texttt{String\_sub},\ 1\,\rangle\end{array}}{\langle\,\text{type}\,\texttt{number},\ \text{type}\,\texttt{int},\ 1\,\rangle}\ [\textsc{rel}]$$

That is, the two corresponding primitives `string-ref` and `String_sub` have types that would unify if `number` and `int` also correspond. The relation R in the definition of the [REL] rule is obscured here, but is the same between `string-ref` and `number`, as it is between `String_sub` and `int`. The correspondence itself is fine, but the derived strength too high: seeing a Scheme `number` does not guarantee using a Standard ML `int`, as

[17] This is one of the cognitive properties from Section 3.4.

[18] In this case, because there were no component probabilities, step four—compute strengths—was done by expert judgement.

[19] This correspondence is 'correct', but in practice we used a different correspondence to capture the same relationship.

Table 6.1 The results of the informational suitability computation over the descriptions of programs and programming languages. The columns are the 'starting' languages that were given as input; the rows are the target programming languages being evaluated. For example, the first column, second row, should be read as the informational suitability of using Standard ML to implement the merge sort algorithm, given that the algorithm was initially presented in C. The scores between columns cannot be directly compared, but the relative scores within each column can be. We typeset the highest score in bold, and the lowest in italic. All scores are rounded to one decimal place. SML is an abbreviation of Standard ML.

	Merge sort			In-order traversal			Longest Common Subsequence		
	C	SML	Scheme	C	SML	Scheme	C	SML	Scheme
C	**26.8**	*13.8*	*19.5*	**24.2**	10.4	12.0	**26.9**	*17.4*	19.4
SML	*15.8*	**17.2**	19.9	*13.7*	**10.8**	**12.6**	16.7	**21.2**	*18.4*
Scheme	16.7	15.2	**20.0**	13.8	*10.1*	*12.0*	*16.5*	17.7	**19.8**

the number might require a `real`.[20] Interestingly, the reverse *would* be fine—and was also derived. Neither correspondence was used, as a more general correspondence links all the Standard ML numerical types with Scheme's `number` type.

A final, manual check on the output of the generated correspondences is strongly recommended. Failing to do this will not break the framework, but you might include lower-quality correspondences. This may result in lower-quality recommendations. Note that there is no requirement to ensure the correspondence set is minimally redundant and maximally covering (MRMC), for two reasons. First, the MRMC operation is formally defined with respect to a specific R- or Q-description, although this could be worked around by defining the R-description to be the union of the left covers of all identified correspondences. Second, it is not *necessary* to perform MRMC refinement at this stage: we will be performing MRMC refinement on the correspondences with respect to a specific problem when making a recommendation.

6.3 Recommending programming languages

With all the descriptions and correspondence sets in place, we can run `robin`. Because the rep2rep framework was sufficiently flexible to encode the programs and programming languages, the tool ran without modification, and produced the informational suitability results shown in Table 6.1. In this section, we will break down how these results are computed, and what this result means in the context of programming languages and algorithms.

6.3.1 *Computing informational suitability*

To illustrate the computation, we will compute one informational suitability score by hand. The informational suitability, as defined in Defini-

[20] Scheme's dynamic typing and numerical tower effectively means all numbers are one 'type'.

tion 15, is[21]

$$\mathrm{IS}_C(q, R) = \sum_{\langle a, b, s\rangle \in \mathrm{MRMC}^R_q(C)} s \cdot \mathrm{importance}_q(a) \qquad (5.1)$$

where q is a Q-description, R is an RS-description, and C is a set of correspondences. We observe there are three key phases in the computation: the MRMC refinement of the correspondence set, lifting importances from components to component formulae, and the final weighted sum. Let us consider each in turn for the longest common subsequence algorithm written in Standard ML, and evaluating the informational suitability of C—that is, $\mathrm{IS}_C(\mathrm{LCS}_{\mathrm{SML}}, C)$.

First, the correspondence set C must be refined from all possible correspondences down to the appropriate subset; this is the $\mathrm{MRMC}^R_q(C)$ set. While this is one 'operation', we do it in two steps: first, we find all correspondences which are satisfied by our problem and representational system, then we select an MRMC subset. From our complete correspondence set, there are 38 that are satisfied. After applying our MRMC approximation algorithm, there are 37—one was eliminated:[22]

$$\begin{aligned}\langle\ & \text{type } \alpha\ \texttt{list} \text{ OR type } \alpha\ \texttt{vector} \text{ OR type } \texttt{string},\\ & \text{type } \alpha\ \texttt{pointer},\ 1\ \rangle\end{aligned}$$

which was matched by the Q-description (which includes `strings`, `char lists`, and `int arrays`) and the RS-description (which includes `pointers`). But this is correspondence was written for *Scheme*, not Standard ML.[23] The equivalent correspondence for Standard ML is

$$\begin{aligned}\langle\ & \text{type } \alpha\ \texttt{list} \text{ OR type } \alpha\ \texttt{ref} \text{ OR type } \alpha\ \texttt{array} \text{ OR type } \texttt{string},\\ & \text{type } \alpha\ \texttt{pointer},\ 1\ \rangle\end{aligned}$$

which correctly includes reference types and arrays, and does not include vectors. The Q-description includes components type `string` and type `char list`, and both correspondences cover these two components; only one is necessary in an MRMC correspondence set. Both have the same strength, so it does not matter which is eliminated; we eliminate the Scheme correspondence.

Now that we have our set of 37 correspondences, we must determine the associated importance; this is the $\mathrm{importance}_q(a)$ value. As stated in Section 5.2.1, the importance of a component formula is the maximum importance of the components that make up the formula. Using again our correspondence from many Standard ML types to C pointers, we have

$$\begin{aligned}\mathrm{importance}_q(a) &= \max\{\mathrm{importance}_q(\text{type } \alpha\ \texttt{list}),\\ &\qquad\quad \mathrm{importance}_q(\text{type } \alpha\ \texttt{array}),\\ &\qquad\quad \mathrm{importance}_q(\text{type } \texttt{string})\}\\ &= \max\{0.6, 0.2, 1\} = 1\end{aligned}$$

[21] This is a copy of the equation from that definition.

[22] Because so few correspondences were eliminated, this suggests we had very little 'overlap' between our correspondences.

[23] This highlights how components and correspondences are not attached to their 'parent' representational system. Naming becomes especially important: conflicts can have unexpected results.

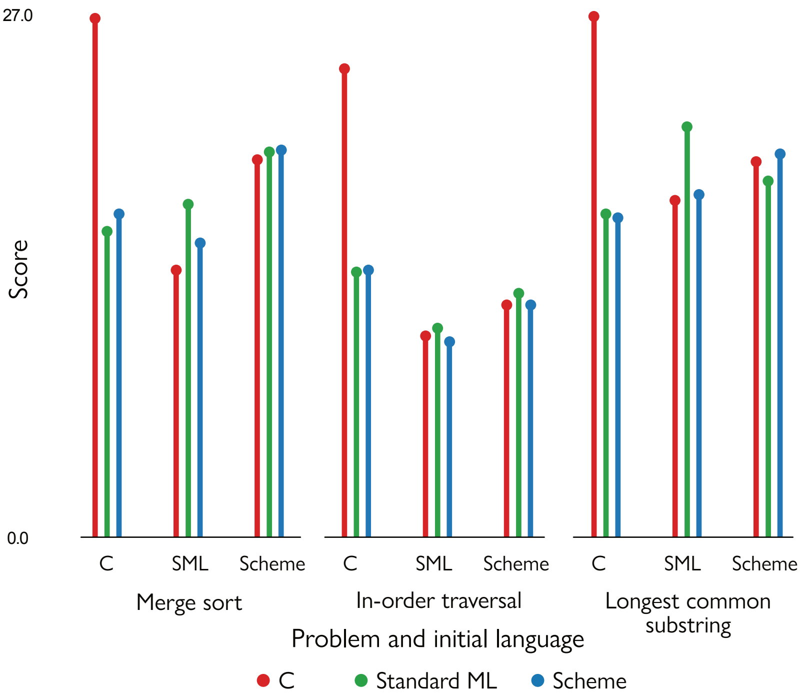


Figure 6.1 The informational suitability scores for each programming language, grouped by the algorithm and initial language. Higher is better.

and so the importance associated with the correspondence is essential. Note that type α `ref` is not included in the max operation: the Q-description does not satisfy that clause, so is not considered.

Finally, we multiply the strength and importance of each correspondence, then sum the result. With our pointer correspondence, we have strength 1 and importance 1, so it contributes 1 to the overall sum. The correspondence from + and `1` to `++` has a strength of 0.7 and medium importance,[24] so contributes $0.7 \times 0.6 = 0.42$ to the overall sum. Over all 37 correspondences, we compute 17.36 as the final informational suitability of the C programming language for programming the longest common subsequence algorithm, based on the algorithm as specified in Standard ML.

6.3.2 *Interpreting the results*

The full results are in Table 6.1, and a graphical presentation is shown in Figure 6.1. Note the results for the in-order traversal algorithm: we see that Standard ML was the preferred language, then C, then Scheme least appropriate, regardless of stating the algorithm initially in Standard ML or Scheme. Scheme broke the trend of each initial language ranking itself highest. In the case of the in-order traversal, we agree, Standard ML *is* a better choice, while Scheme's inability to define new types makes the problem more difficult to express. However, we see the difference is small; certainly not enough to declare one 'better' than the other. This algorithm is sufficiently simple that *all* our languages can implement it.

[24] Both + and `1` have importance medium.

Finally, a fact made more apparent by Figure 6.1 than in Table 6.1, is that the scores for both Standard ML and Scheme are very close. When C is the initial language, it scores itself significantly higher than the other two languages. Two things are happening.

First, Standard ML and Scheme are more closely related to each other than either is to C. When describing a program written in C, we were often describing memory allocations and pointer updates. Conversely, Standard ML and Scheme are memory-managed and use high-level data structures. This conceptual mismatch results in a correspondence mismatch: C is concerned about low-level computer operations, and finds the alternative languages lacking—which, they are! Consider the distinct concepts in Standard ML and Scheme correspond to the same concept in C: lists, references, arrays, and strings are all pointers. Conversely, distinctions made in C—`int`, `unsigned int`, `long int`, etc.—are not made in Standard ML or Scheme. This example focused solely on informational suitability; by considering the cognitive cost of each programming language for different users, we might find different results.

Second, all our programming languages are *Turing complete*. That is, all can express exactly the same concepts. All can be considered equivalently 'informationally suitable', and thus should have similar scores. Indeed, we see this happening: when the languages are similar enough to have strong correspondences to each other, they score similarly.

6.4 Exploring the input space

In Section 6.2, we played the role of analyst for each of correspondences, Q-descriptions, and RS-descriptions. By the nature of this task, we cannot assert that the input we are providing is 'correct', and in Section 5.1.4 we argued that 'correct' is impossible. But we can ask what would happen if we did the analysis differently. Specifically, what would happen if we *ommitted* correspondences or components? The `rep2rep` framework will change its recommendation when its input change. If sufficiently sensitive, it reacts to even subtle changes in the problem and representational systems. This allows for more nuanced recommendations. Conversely, analysts must be careful: small discrepancies between what was *intended*, and what was *actually* described can result in 'incorrect' recommendations.

In this section we remove elements of `robin`'s inputs, and observe the effects this has on the recommendations. We use a modified version of `robin` that does one of the following:

- drop $k \in [1, 50]$ random correspondences;
- drop $k \in [1, 20]$ random components from the Q-description being considered; or
- drop $3 \times k$ random components from the RS-descriptions being considered: $k \in [1, 20]$ from each of the C, Scheme, and Standard ML RS-descriptions.

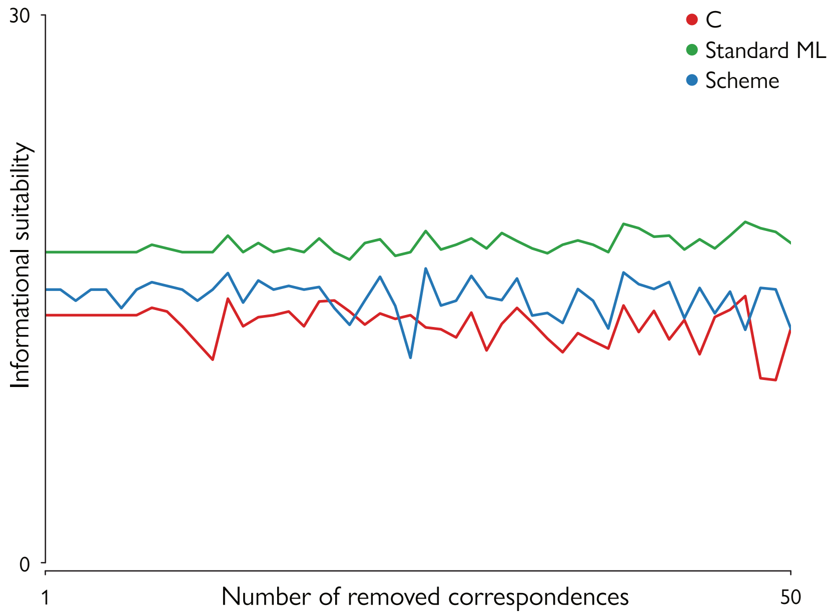


Figure 6.2 The informational suitability scores (vertical axis) assigned by `robin` to each programming language for the problem of implementing merge sort, initially specified in Standard ML. The score varies based on which k correspondences (horizontal axis) have been randomly removed from the correspondence set.

The following sections analyse one representative case for each situation; plots for all cases are included in Appendix I.

6.4.1 *Omitting correspondences*

We first consider the case where correspondences have been omitted. We randomly remove between 1 and 50 correspondences from the correspondence set,[25] then make a programming language recommendation based on a particular algorithm stated in a particular language—exactly as in the previous section. Ordered by the number of correspondences removed, we can plot how the resulting recommendation was affected: in Figure 6.2 we plot the case in which we recommend a programming language to implement merge sort, with the algorithm initially given as a Standard ML program.

Prominent in Figure 6.2 is the increasing variability as the number of omitted correspondences increases.[26] As we remove more correspondences, the chances of removing a correspondence that the framework is using to make its recommendation increases. But missing only a few correspondences is often not damaging: the change in score when missing fewer than ten correspondences is typically under 10%.

Another feature of the plot is that knocking out *more* correspondences can result in a *higher* score. This can occur when we eliminate a correspondence that was previously selected during MRMC refinement: now, the correspondences that were previously discarded can be selected. If multiple correspondences were previously covered by a single

[25] The correspondence set started with 157 correspondences.

[26] Because the correspondences are removed randomly and independently, correspondences omitted for trial k might not be omitted for trial $k + 1$.

correspondence, the combined strengths of the new correspondences can exceed the strength of the previous, more specific correspondence.

Example 6.1. Consider three correspondences,

$$\langle a \text{ AND } b \text{ AND } c,\ d,\ 1.0 \rangle \quad \langle a,\ d,\ 0.8 \rangle \quad \langle b,\ d,\ 0.8 \rangle$$

for the Q-description $\{a, b, c\}$, all with importance 1. During MRMC refinement, we would accept the first correspondence, and discard the other two: the first covers more, and renders the other two redundant. The final informational suitability is 1. However, if the first is not present, MRMC refinement would accept both the remaining correspondences; the final informational suitability is 1.6. ▽

In this case, the order of the recommendation is unaffected until $k = 19$; this is in general *not* true, and the recommendation can be altered for any number of omissions.

6.4.2 *Omitting Q-description components*

We now consider when the analyst has forgotten to include Q-description components for the problem they are attempting to re-represent. We randomly remove between 1 and 20 components from the Q-description,[27] then run `robin` with the updated input. Ordered by the number of components removed, we plot how the score for each language was affected; Figure 6.3, shows the informational suitability scores for each programming language to implement an in-order tree traversal, initially given as a C program.[28]

Figure 6.3 bears little resemblance to Figure 6.2: the variability is reduced, and does not appear to increase as we remove more components. The obvious trend is a score decrease as we remove components. Apparent increases in informational suitability are due to the component removal being independent between trials; some components 'matter more' than others, and may have been one of the k removed components, but not one of the $k + 1$ removed next. Note also that what affects one programming language typically affects all of them: this is again because we are modifying the description of the *problem*.

The overall order of the recommendation is not affected much by the changes, but when the original difference between the informational suitability of the programming languages is small, we see the recommendation order is more liable to change. This is unsurprising: if the two languages are hard to separate, even small changes can tip the balance one way or the other.

6.4.3 *Omitting RS-description components*

Finally, we consider the case where the analyst has failed to include components in the RS-descriptions. We randomly remove between 1 and 20 components from each of the programming languages' RS-descriptions,[29] then re-run `robin`. Again ordering by the number of components

[27] The Q-descriptions had between 27 and 63 components.

[28] This Q-description started with 58 components.

[29] The C, Scheme, and Standard ML RS-descriptions had 57, 53, and 61 components, respectively.

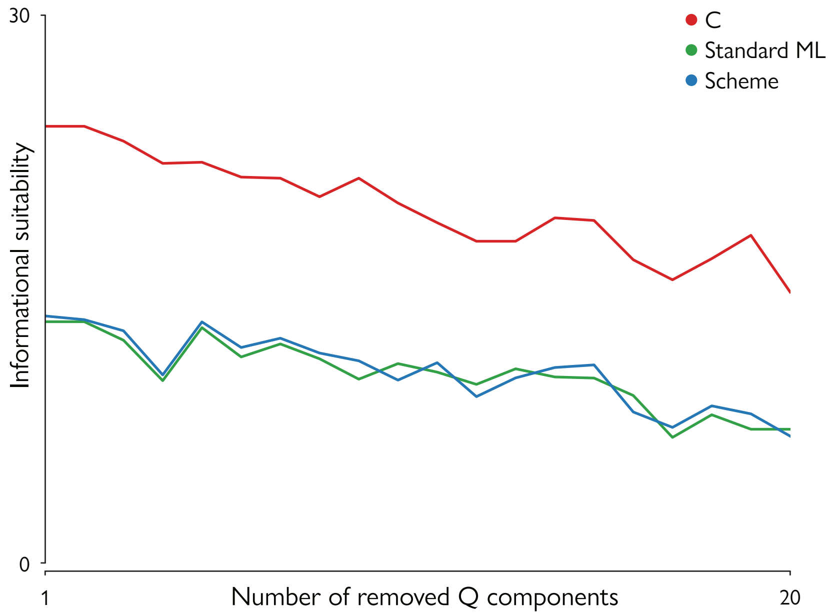


Figure 6.3 The informational suitability scores (vertical axis) assigned by `robin` to each programming language for the problem of implementing an in-order tree traversal, initially specified in C. The score varies based on which k components (horizontal axis) have been randomly removed from the Q-description.

removed, we plot the output; Figure 6.4 shows the informational suitability scores for each programming language to implement the longest common subsequence algorithm, initially stated as a Scheme program.

Figure 6.4 shows a combination of the effects observed when omitting correspondences and Q-description components. Like with correspondences, variability increases as we omit more RS-description components: we are more likely to remove a component on which the framework relies to determine a good informational suitability when we remove more. We also see that informational suitability can again increase, this time due to missing components not precluding particular correspondences. And as with removing Q-description components, we see an overall downward trend: fewer components means fewer overall chances to satisfy a correspondence.

More than in either previous case, we see the overall order of the recommendation be affected by the omission of RS-description components. This is because we are not impacting each representational system equally: a missing component is missing from just one system, which has no bearing on the remaining representational systems. Subtle changes in representational systems are considered by `robin`.

6.4.4 *Discussion*

Across these three situations—omitting correspondences, omitting Q-description components, and omitting RS-description components—we see a trend towards lower, less consistent scores. This is not unexpected:

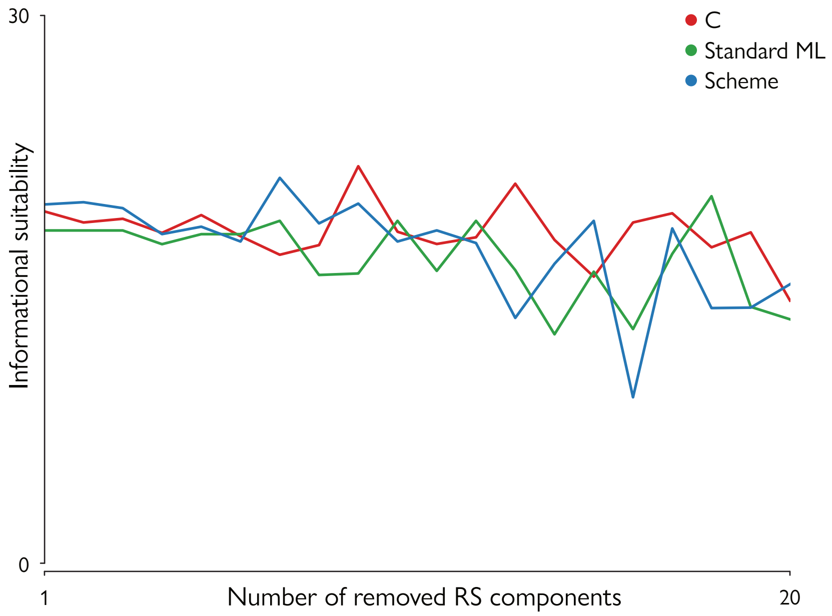


Figure 6.4 The informational suitability scores (vertical axis) assigned by `robin` to each programming language for the problem of implementing the longest common subsequence algorithm, initially specified in Scheme. The score varies based on which k RS-description components (horizontal axis) have been randomly removed from each of the RS-descriptions.

`robin` is sensitive to its inputs, and while small changes have small effects, enough small changes can change the recommendation—sometimes dramatically. Changes to the Q-description are least severe, as the influence is felt across all the candidate representational systems in a proportional way; changes to the RS-descriptions and correspondence set will impact representational systems unevenly, causing more severe deviations.

We would expect similar responses to smaller changes: strengths and importances must also be set by analysts, and will impact the informational suitability of representational systems. As with omitting components versus correspondences, we expect 'incorrect' importances to be more resistant to errors, as they will impact the computation of informational suitability proportionally across all the representational systems. Conversely, 'incorrect' strengths will be unevenly applied across representational systems, potentially leading to larger changes in the scores, and thus the overall recommendation. This reinforces the need for analysts to take care when crafting the inputs.

SUMMARY OF SECTION 6.4

The rep2rep framework, and the `robin` implementation, is sensitive to the inputs provided by the analysts. Even small changes in the descriptions or correspondence sets can impact the resulting recommendation—allowing for subtle distinctions, but requiring precise inputs. We see that the framework is more robust to 'fair' changes, when the change will impact all the representational systems equally. However, when changes impact some representational systems more than others, we see a greater impact on the final recommendation—for better or worse.

SUMMARY OF CHAPTER 6

This chapter demonstrated that the framework described in this dissertation applies not only to mathematics, but also to domains not initially considered. We described how the processes and tools function equally well with probability problems and counting problems, as for our detailed example with programming languages. We note how the framework had the flexibility to encode concepts that were not considered during its development. The `robin` implementation worked without modification, and produced results that were both understandable and insightful. Finally, we explored how changes in the input descriptions and correspondence set can impact the final recommendation.

Evaluating the framework 7

By relieving the brain of all unnecessary work,
a good notation sets it free to concentrate
on more advanced problems.

— *Alfred North Whitehead*

Having described our approach to representational system recommendation, we turn to evaluating the results—are the recommendations *sensible*? In this chapter, we describe two studies: the first study ablates the framework to determine the effect of the constituent factors in producing a recommendation; the second study aims to determine whether experts—of both the subject domain (probability) and representation selection—agree with the output of our framework.

The first study, which we refer to as the 'ablation' study, is an extension of work completed by researchers at the University of Sussex as part of the rep2rep project [Raggi et al. 20-a]: experts evaluated the suitability of representational systems, providing a benchmark for us to compare the framework against. The framework we have presented in this dissertation has many factors in producing a recommendation, even when only considering informational suitability; two are component *importance* and correspondence *strength*. We wish to understand what influence these factors have on the final recommendation. Academic staff and doctoral students were asked to evaluate the suitability of representational systems for a simple probability problem, and their responses were compared to the output of the framework under four conditions: the unmodified informational suitability, informational suitability computed without component importance, informational suitability computed without correspondence strength, and informational suitability computed with neither importance nor strength. We determined that both factors were necessary for our framework to produce recommendations that significantly correlate with the expert recommendation.

The second study is entirely a contribution of this dissertation. To evaluate the results of the framework we have described, we compare its output on a range of problems, representational systems, and user personas to that of human experts in the same conditions. Our experts, in this case, are mathematics teachers: they are competent in the problem domain, and at choosing appropriate representations for problem solvers with a wide range of abilities. The teachers evaluated each representational system, showing a mixture of consistency and inconsistency that illuminates some of their thought processes.

This chapter is divided into three sections. In Section 7.1 we consider the overall motivation of this work, and how we evaluate the success of this dissertation. The ablation study is presented in Section 7.2, and while the extension presented here is a contribution of this dissertation, the underlying data collection and extraction of expert recommendations was performed by rep2rep researchers. The rep2rep portion of this study is detailed in our paper 'How to (Re)represent it?' at the *International Conference on Tools with Artificial Intelligence* (ICTAI) 2020 [Raggi et al. 20-A], while the extension is briefly described in appendices to work presented at the *IEEE Symposium on Visual Languages and Human-Centric Computing* (VL/HCC) 2020 [Stockdill et al. 20-A]. The empirical study involving teachers, Section 7.3, is a contribution of this dissertation: I designed, executed, and analysed the results of this study. This work has not yet been published.

7.1 Success criteria

We begin this chapter with a discussion on what it means for this project to be successful. The goal of this dissertation, and the wider rep2rep project, is conceptually simple: to be able to recommend appropriate representational systems to support any specific person in solving any specific problem. More precisely, we propose three criteria against which we measure the success of this project:

Human-level performance The framework produces a recommendation that is comparable to what human experts would recommend.

No internal redundancy The framework is consistent in its recommendation for fixed inputs, and all the factors contribute to the recommendation.

Effectiveness of recommendations The representational systems recommendation is used by the problem solvers and has a measurable, positive impact on their ability to solve the problem.

Let us consider each of these criteria in turn.

7.1.1 *Human-level performance*

To the best of our knowledge, no existing software attempts to solve the general problem of representation recommendation—we have no systems against which to compare. So we turn to humans who recommend representational systems: teachers. Teachers must analyse and select appropriate representational systems for a diverse set of students attempting to solve problems. The consensus in the literature is that 'novices' are frequently unable to make a good selection, while experts *can* potentially make a good selection [Uesaka et al. 10; Stylianou 02]; hence, for our study we recruit teachers rather than students.

We must consider what it means to meet or exceed this human benchmark. Drawing an analogy to the 'Imitation Game,' commonly referred to as the 'Turing Test' [Turing 50], to be as good as a human is to be indistinguishable from them. Thus, if our framework can produce recommendations that are indistinguishable from that of human experts, we consider this goal achieved. But if the output *is* different, in which direction have we deviated? Better, worse, or equivalent but distinct? This conclusion would better be drawn from a study on the effectiveness of the recommendations.[1] By working with experts we can more easily and rapidly evaluate the effectiveness of our framework, but sacrifice being able to categorise the disagreement.

To determine whether our framework is achieving human-level performance, we present a study involving teachers in Section 7.3.

7.1.2 *No internal redundancy*

If the framework *does* make suitable recommendations based on informational suitability and cognitive cost, then we must ask *why* it works. By the nature of the framework, the recommendation is deterministic: identical inputs will always produce identical outputs. But two points demand inspection: first, we do *not* expect the framework to produce the same result when attempting to re-represent the same problem from different 'initial' representations; second, we wish to determine the influence of the framework's factors on the recommendation.

Expecting the same problem, when posed using a different representation, to produce different results seems at first undesirable: the same problem should have the same 'good' representational systems. But 'the problem' never exists in isolation; every problem is framed within some representational system. If a problem is encoded in a poor system, to the point where key features of the problem are not accurately captured, then the framework is unable to suggest representations that properly account for those key features.[2] Effectively, the framework cannot recover lost information. Only when we start within a sufficiently descriptive representational system with the key features described can we expect the framework to produce meaningful output.

So we must start from a representation in a sufficiently expressive representational system. This raises the question: why would we ever want to recommend a representational system that is *less* expressive? This can occur through a combination of two factors: the original representational system is *excessively* expressive, and the original representational system is more cognitively costly than the alternative representational system. Thus while using informational suitability alone will likely not result in a less expressive representational system being recommended, this can occur when considering both the informational suitability *and* the cognitive costs.

Example 7.1. A more expressive system can sometimes be unnecessary: algebra can express much more than our dot-arrangements, but because our problem of summing integers is expressible with dots, algebra's 'excessive' expressiveness is unnecessary. This will be reflected in the inform-

[1] See the discussion ahead in Section 7.1.3.

[2] We discussed this in Section 5.2.1, and noted that a human would struggle similarly.

ational suitability scores being similar: everything expressed within the original representation can be expressed within both the algebra and dot-arrangement representational systems.

But the cognitive cost of each system could be very different: for lower-ability people in particular, algebraic notation can become overwhelming. In comparison, dot-arrangements are likely to be intuitively understood. In this case, the *less* expressive representational system will be recommended: dot-arrangements would be preferable to algebra for this problem for a lower-ability person. We would not expect to see this occur with a high-ability mathematician: their grasp of algebra would be sufficient that both systems will be near-equally low cost, and so both systems would continue to score similarly. ▽

We present an ablation study in Section 7.2 as a means of untangling the contributing factors to informational suitability.

7.1.3 *Effectiveness of recommendations*

Our final success criterion is that the recommendations we produce are helpful to the user in solving their problem. The literature is mixed on this front, and often depends on *why* the user is solving their problem. Choosing the right representation is worth '10 000 words' [Larkin et al. 87], and can provide solutions 'for free' [Stapleton et al. 17]; conversely, in an education setting, the specific choice of *which* representation can matter less than *diversity* of representations [Ainsworth 08].[3] So to validate our framework's ability to choose effective representational systems we suggest a study that directly measures user performance in problem solving. This study is beyond the scope of this dissertation.

Alternative representations are indisputably beneficial [Ainsworth 08; Cheng 02; Cox 99; Grawemeyer 06], but to deploy our implementation of the framework in a manner which is appropriate for end-user consumption remains future work. In Section 8.2.2 we describe how future versions of the framework's implementation might be deployed in an educational environment, and so allow for us to directly evaluate the effectiveness of its recommendations.

[3] Our framework is trivially adjusted to provide multiple representations—indeed, it could provide many informationally suitable but cognitively diverse representational systems by adjusting how the cognitive cost is computed and used.

7.2 Ablation study

Our framework is novel, and consists of many interconnected factors. In this section, we will focus exclusively on the informational suitability of representational systems; cognitive cost has not been a direct focus of this dissertation. Inspecting the definition of informational suitability, Definition 15 on page 107, we see three factors: component importance, correspondence strength, and MRMC set refinement. We choose to consider component importance and correspondence strength. We do not ablate MRMC set refinement, as this factor was introduced after this study was initially conducted; the correspondence sets were adjusted by hand to be minimally redundant, and so the MRMC set refinement has little to no effect. A repeat of this ablation study including MRMC refinement will be valuable when more correspondences are added without manual refinement.

7.2.1 *Preceding study*

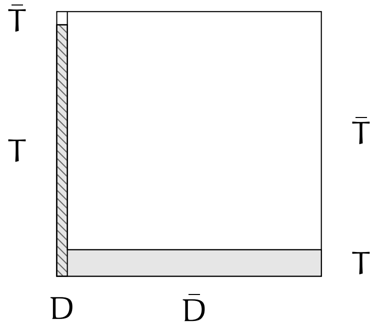
Dr Garcia Garcia, a researcher at the University of Sussex and member of the rep2rep research group, ran a study with one problem—the medical test problem—a set of five representational systems, and asked participants to evaluate the suitability of the systems. This study, conducted via Qualtrics, asked 11 researchers[4] from the University of Cambridge Department of Computer Science and Technology and the University of Sussex School of Engineering and Informatics to evaluate the suitability of five representational systems: area diagrams, Bayesian algebra, contingency tables, Euler diagrams, and natural language (English). The participants were presented with a problem statement in natural language:

> 4% of the population has a disease. For those who have the disease, a test is accurate 95% of the time. For those who do not have the disease, the test is accurate 90% of the time. If you take the test and it comes out positive, what is the probability that you have the disease?

Each representational system was then described briefly, and the participants evaluated its suitability using a seven point Likert scale. The Euler diagrams representational system was seemingly misunderstood by the participants—they assumed it was augmented with real values for computing probabilities, which was not the case—so was discarded. Raggi et al. give more details in 'How to (Re)represent it?' on the rep2rep study that generated the data which we use [Raggi et al. 20-A].

Example 7.2. To demonstrate each representational system considered in this study, we represent the problem statement in each as best we can. The problem was initially given in natural language.

The medical problem in area diagrams might be encoded as follows, where the horizontal dimension is having the disease, while the vertical dimension is the outcome of the test:



The result is the ratio between the hatched area and the grey area.

[4] PhD students, research associates, and academic staff.

In the Bayesian algebra notation, we might state

$$\begin{aligned} \Pr(D) &= 0.04 \\ \Pr(T \mid D) &= 0.95 \\ \Pr(\bar{T} \mid \bar{D}) &= 0.9 \end{aligned}$$

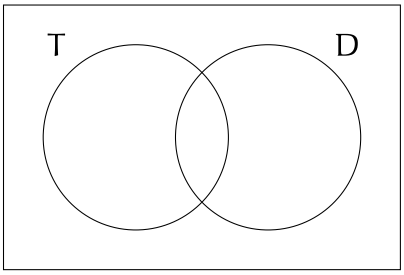
and then calculate $\Pr(D \mid T)$.

Using contingency tables, we could write

	D	$\bar{D}$	
T	a		x
$\bar{T}$		b	
	0.04		1

where we are given information indirectly about a and b, namely that $a/0.04 = 0.95$ and $b/(1 - 0.04) = 0.9$. The goal is to compute the ratio between the cells a and x.

Euler diagrams struggle to encode the problem, as they cannot express most magnitudes. One potential encoding, capturing as much as possible, might be:



That is, a test for the presence of a disease can be positive, and a person can have a disease, and we have any combination of these two events. There is no indication of how large each probability is relative to each other. ▽

From this study, we were able to extract an overall set of informational suitability scores from the participants for each representational system on the medical tests problem. Thus we can ask how well the scores from the participants correlate with the scores from the framework. This correlation, between the framework implementation at the time and the participants' responses, was $r = 0.89$ ($p = 0.053$). We shall do the same with a more recent version of `robin`, and also consider variations without component *importance* and correspondence *strength*.

7.2.2 *Hypotheses*

Based on the preceding study, we have a correlation between the informational suitability scores as given by the framework, and the informational suitability scores from the participants. Going further, we can ablate features from the framework and determine what effect each has on the resulting correlation. We target two features: importance and strength.

When both are disabled, the informational suitability objective function is a count of satisfied correspondences.

We expect the changes we make to `robin` to impact how well the scores it produces correlate with the scores that the expert participants produced. Specifically, we expect that the unablated implementation to be the most strongly correlated:

> **Hypothesis 1.** The unablated `robin` implementation of the rep2rep framework—that is, it considers both strength and importance—will produce scores that correlate more strongly with the scores produced by the experts than any of the ablated implementations.

Further, we expect that the 'fully' ablated version, with neither strength nor importance, will be least strongly correlated:

> **Hypothesis 2.** The `robin` implementation of the rep2rep framework that uses neither importance nor strength will produce scores that correlate less strongly with the scores produced by the experts than any other implementation.

We make no prediction about whether the implementations with exactly one of strength or importance will produce scores that correlate more or less strongly than the other.

7.2.3 *Design, methodology, and experimental setup*

For this study, we ablate the `robin` implementation of the framework of two factors: component importance, and correspondence strength. For the first factor, this is equivalent to all components having importance 1. For the second factor, this is equivalent to all correspondences having strength 1. We thus end up with four conditions: `robin` unchanged, `robin` such that it does not consider component importance, `robin` such that it does not consider correspondence strength, and `robin` such that neither importance nor strength are considered.

Ablating the importance factors from the `robin` implementation of the framework involves changing two lines. To disable importance computation, we make the substitution

```
-  val modulate = fn (i, s) => i * s;
+  val modulate = fn (i, s) => s;
```

meaning we discard the importance value; similarly to disable strength, we make the substitution

```
-  val strength = fn (a, b, s) => s;
+  val strength = fn (a, b, s) => 1.0;
```

and so every correspondence is uniformly strong.

Each variant of the `robin` implementation is capable of producing a recommendation. We consider the *correlation* of each recommendation with the recommendation extracted from the expert participants'

Table 7.1 The informational suitability scores produced by experts and by our framework for the medical tests problem. The four framework variations are: unaltered, without importance, without strength, and without importance or strength. Note that the subscripts of **R**, the rep2rep framework, in the headings denote the *enabled* features: **i** for importance, **s** for strength. So $\mathbf{R}_{\mathbf{is}}$ is the complete framework. All values have been rounded to one decimal place.

	Informational suitability scores				
Rep. Systems	Experts	R_{is}	R_s	R_i	R
Bayesian	6.0	13.7	31.0	13.7	31.0
Area diagrams	4.8	9.5	21.7	9.9	23.0
Contingency tables	4.9	7.6	18.5	8.3	20.0
Natural language	3.5	3.8	14.0	6.8	18.0

responses. A recommendation is a vector of four[5] real numbers, where each number is the informational suitability of the corresponding representational system.

In this ablation study we do not use rank-order statistics because not only is the order important, but the *difference* between each representational system's score is meaningful. If both the experts and framework agree that two representational systems should score similarly, but disagree on the order, that is a minor error; if they agree on the order, but one believes the two representational systems deserve very different scores while the other scores them approximately equally, that is a problem. Using the continuous scores allows us to make this distinction.

Example 7.3. Consider the case where the experts evaluate representational systems A and B such that A had an informational suitability score of 4.2 and B had a score of 4.3. These scores are so close as to be identical. If the rep2rep framework computed scores of 8.9 for A and 8.7 for B, the ranking would be reversed, but the difference relative to the score magnitude remains small—the framework agrees that the representational systems score similarly. Conversely, we might have experts evaluate representational systems X and Y, where X scores 1.3 and Y scores 6.8, while the rep2rep framework computes 7.5 for X and 7.8 for Y. Now the ranking is the same, but the experts produced very different scores (suggesting that Y is much more suitable than X) while the framework could barely distinguish them. ▽

After enabling combinations of importance and strength, we recompile and run the framework over an identical set of correspondences for the medical test problem. As in the preceding study, we compute Pearson's correlation r between the participants' recommendation and those from each of the modified implementations' recommendations, for a total of four correlations.[6]

[5] Remembering that the Euler diagrams representational system was discarded.

[6] The scores reported here differ from previously published work due to updates to the framework and correspondence sets.

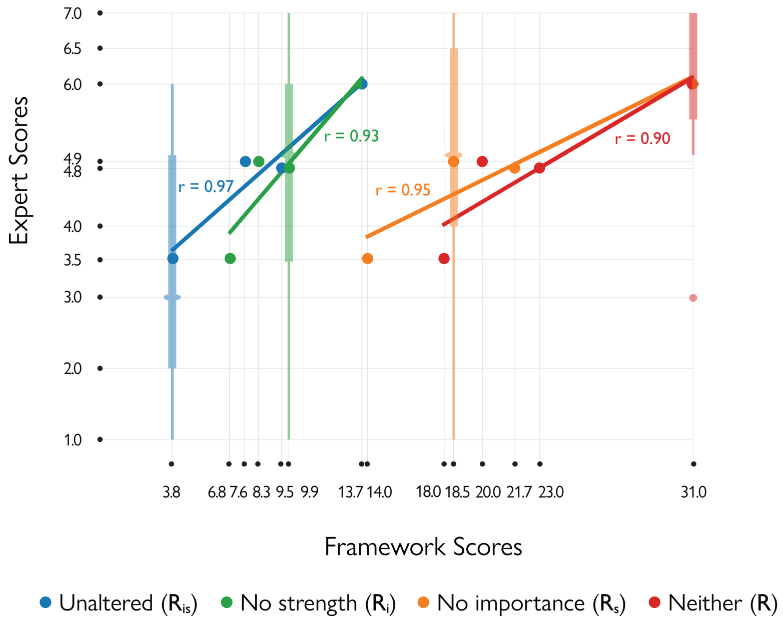


Figure 7.1 The relationship between the scores produced by the framework (x-axis) and the aggregate of experts (y-axis). Each coloured set of points represents the pairing between the mean of the experts' scores and the scores computed by a variant of the framework. Each point within a coloured set is a different representational system. For example, the point $(7.6, 4.9)$ is the score for contingency tables from both the unaltered framework and the aggregated experts' scores. The box-and-whisker plots show the distribution of the participants' responses: blue for natural language, green for area diagrams, orange for contingency tables, and red for Bayesian algebra. The box-and-whisker plots are given just once for each representational system to reduce clutter; all aggregated points on the same y-value share the same distribution (e.g., the four points at $y = 3.5$ all have the same distribution as shown by the blue box-and-whisker plot).

7.2.4 *Results*

The recommendations produced by the experts and framework are shown in Table 7.1: in the first column we have the recommendation from the experts, and the remaining columns contain the recommendations from the `robin` variants.

Figure 7.1 shows the relationships between each framework variant's scores and the mean of the participants' scores: each line is the line of best fit between the recommendation scores of the four framework implementations and the experts' recommendation scores. The overlaid box-and-whisker plots show the distribution of the analysts' scores; these are the same for a representational system in each comparison, so we show just one per ablation to avoid clutter.

7.2.5 *Analysis*

To compare the scores from `robin` to those from the participants, we consider both the participants' scores unaggregated $(n = 11)$, and taking the mean of their responses ('aggregated'). The ablation study indicates that by counting the correspondences (that is, R without strength or importance) satisfied by descriptions of problems and representational systems, we produce informational suitability scores that strongly correlate (unaggregated: $r = 0.41, p = 0.005$; aggregated: $r = 0.90, p = 0.098$)[7] with experts' own scoring. The importance and strength factors of the framework increase the correlation between the framework's scores and those of the experts, both individually (unaggregated: $r = 0.42, p = 0.004$ and $r = 0.43, p = 0.003$, respectively; aggregated: $r = 0.93, p = 0.072$ and $r = 0.95, p = 0.054$, respectively) and together (unaggregated: $r = 0.44$, $p = 0.002$; aggregated: $r = 0.97, p = 0.029 < 0.05$). Thus we have evidence in support of both hypotheses.

But we note that Person's correlation r does not decrease by a large amount when working without importance—in Figure 7.1, shifting from blue to orange. So while importance is a benefit, working without it does not materially worsen the output—although it did tip the result outside statistical significance. Because importance is annotated by hand, it is laborious to include. Thus, while automatically deriving importance is high on our list of research avenues, working without importance is, in the short term, a viable alternative. When we can improve the importance annotation process, we know to expect an improvement in the framework's output.

Based on the results of the ablation study, we can conclude that our framework is meaningfully composed: the correspondences capture necessary links to build an informational suitability score; the strength of correspondence and the importance of components refines this score to improve the representational system recommendation.

7.2.6 *Limitations*

This ablation study is based on a dataset collected from researchers in computer science/informatics departments from two universities in the United Kingdom. Further, we have their recommendations for only a single problem. This limits the potential generality of our results: more diverse participants on a wider set of problems would improve the applicability of our conclusions.

In the rep2rep study that generated the dataset we use here, the results for Euler diagrams were discarded due to inconsistent responses. We conjecture that the experts implicitly 'upgraded' Euler diagrams by assuming they could be augmented with equations when the diagrams themselves were insufficient to solve the problem. While we have no reason to suspect the remaining representational systems were similarly 'upgraded', we cannot rule out that the experts were implicitly evaluating a different set of representational systems than intended.

[7] Pearson's r is stronger in the aggregated cases as there are fewer points to 'deviate' from the line, but the unaggregated r-values are still moderately strong. The opposite occurs with the p-values: more points yields a stronger conclusion, so we see lower p-values in the unaggregated cases than for the mean scores.

SUMMARY OF SECTION 7.2

The ablation study we have presented examined the impact of component importance and correspondence strength on the computation of informational suitability, and how that correlates with expert representational system recommendations. We find that only with the entire framework, including importance and strength, does the framework implementation's recommendation correlate significantly with the expert recommendation. The limitations of this study, particularly around the dataset used, suggest we would benefit from a larger dataset on which to re-run this study; different participants reasoning over more problems, and a greater emphasis on training to prevent misunderstandings.

7.3 Expert study

Building on the previous study, we now seek to extend our evaluation with a larger set of problems. In this study we focus on mathematics teachers, experts in both the domain (probability) and in representation selection. While the motivating goal was to produce a set of recommendations we could compare against the output of the framework, we also found interesting consistencies—and inconsistencies—amongst our participants: recommending representational systems is not simple. This study received ethics approval from the University of Cambridge Department of Computer Science and Technology.

7.3.1 *Hypotheses*

This experiment intends to determine whether experts, specifically secondary school mathematics teachers, are able to produce similar representational system recommendations. These recommendations should not be arbitrary, but consider both the problem being solved and the cognitive profile of the person—in this case, a student—doing the solving. We can then compare the scores produced by `robin`, our framework implementation, with the teachers' responses; they are the benchmark against which we compare our framework.

We break down our high level goals into four hypotheses. Our first hypothesis can be stated as follows:

> **Hypothesis 1.** From the teachers' individual responses it is possible to produce an overall ranking of representational systems for each problem and cognitive context.

That is, their responses should be at least partially consistent with each other—they are all starting from the same problem and cognitive situation (see the following two hypotheses), but they are also working within the same curriculum with a related cohort of students.[8] Thus we would expect that the teachers' responses would be sufficiently similar that we can extract some rank of representational systems.

[8] Most of their students have *also* been educated in the same curriculum.

From the teachers' recommendations, we expect the recommendation to change in response to the situation. Within the rep2rep framework we consider both the problem being solved and the user solving the problem when making a representational system recommendation; we would expect teachers make a similar consideration when asked to evaluate how suitable a representational system would be. That is, their recommendation should vary based on the problem being solved:

> **Hypothesis 2.** The teachers' aggregate representational system recommendations change based on the problem that they are considering.

The recommendation should also vary based on the cognitive abilities of the student that they are making the recommendation for:

> **Hypothesis 3.** The teachers' aggregate representational system recommendations change based on the cognitive context (with informational suitability only, a low-ability student, or a high-ability student) that they are considering.

Finally, we make a direct comparison with our framework implementation. We anticipate that the framework is producing scores comparable to those assigned by experts. Thus, our fourth hypothesis is:

> **Hypothesis 4.** The `robin` implementation of the rep2rep framework produces scores that are correlated with the teachers' responses, when considering the informational suitability of the same problem and representational system.

7.3.2 *Design*

To constrain the scope of the experiment, we designed it in the context of mathematics students aged 15–18 currently learning probability. The original target was England's General Certificate of Secondary Education (GCSE) and A-levels examinations, but due to the COVID-19 pandemic the study was abruptly re-targeted to New Zealand's National Certificate of Educational Achievement (NCEA). We chose the domain of probability as there are a wide variety of potential representational systems, and the problems cover a range of difficulties.

REPRESENTATIONAL SYSTEMS

From the many possible probability representational systems, we selected five for this study: *area diagrams*, *Bayesian algebra*, *contingency tables*, *Euler diagrams*, and *probability trees*.[9] Our motivation for this selection was to capture the diversity of possible representational systems; attributes such as size, absolute or relative position, and order may or may not matter. Each is also obviously distinct from each other—there can be no confusion to which system a particular representation belongs. Examples of each were given in Example 7.2, except for probability trees.

[9] The first four are the same as in the ablation study; we substitute natural language (English) for probability trees.

Area diagrams This geometric representational system uses a unit square which can be partitioned into regions with horizontal and vertical lines, where the area of the region with edged labelled by events X and Y represents the probability of $X \cap Y$; areas of disjoint regions for events A and B can be added together for the probability of $A \cup B$.

Bayesian algebra This is standard algebraic notation, augmented with two probability functions $\Pr(\cdot)$ and $\Pr(\cdot \mid \cdot)$, the laws of conditional probability, and Bayes' Theorem.

Contingency tables This tabular representational system uses a grid of cells where the sum of all the values in the table must be 1. The value in a particular cell in row X and column Y contains the probability of $X \cap Y$. Using these rules, missing values can be computed.

Euler diagrams This spatial system represents events as contours (circles) and the overlapping regions represent their conjunction. This representational system cannot represent the precise magnitude of most probabilities, so is unsuitable for any of our problems. That is, we specifically considered *non-proportional* Euler diagrams.

Probability trees For this topological representational system, events are represented by nodes in a rooted tree, and the (directed) edges are labelled with conditional probabilities. Multiplying along branches computes conjunction, while adding between branches computes disjunction. While the edges between the nodes are meaningful, their length, order, and position are not.

The choice to include area diagrams was motivated because we were aware that this representational system is *not* commonly taught in the United Kingdom.[10] While we do not actively use this fact in our analysis, we wish to see what effect an unfamiliar representational system has on the teachers' responses. We provided training for the teachers in all representational systems, regardless of familiarity.

COGNITIVE CONTEXTS

To evaluate the representational systems, the teachers will need to consider the *cognitive context* that the system will be used in. For this study, we use three contexts: informational suitability (i.e., without any student in mind), a low-ability 'novice' student context, and a high-ability 'expert' student context. We expect the teachers to adjust their responses based on the cognitive contexts, addressing Hypothesis 3.

For the informational suitability, participants were not given any persona to consider when scoring the representational systems. For the contexts involving students, we presented the teachers with *personas*:

[10] Nor, as it turns out, New Zealand; the change in location did not affect this factor.

- Student A is 15 years old, and in Year 11.[11] They are able to add and subtract well but are less confident with multiplication and division. They can perform one or two steps independently if they have seen them done before, but problems that require more steps to solve will leave them unable to start. They cannot use knowledge from other areas of mathematics; they only use skills they have learned in probability to solve probability problems.
- Student B is 17 years old, and in Year 13.[12] They are confident with addition, subtraction, multiplication, and division. They can solve problems that require many steps and are willing to try steps they have not explicitly seen demonstrated before. The student is able to combine knowledge from across mathematics to solve their current problem.

PROBLEMS

Given the context of this study, both in domain and based on the cognitive personas, we selected five typical probability problems for students of this age range. The purpose is to address Hypothesis 2: the teachers change their recommendations based on the problem being solved.

1. 1% of the population has a disease. A test is reliable 98% if you have the disease and 97% if you do not have the disease. Assuming the test comes out positive, what is the probability of having the disease?
2. One quarter of all animals are birds. Two thirds of all birds can fly. Half of all flying animals are birds. Birds have feathers. If X is an animal, what is the probability that it's not a bird and it cannot fly?
3. Let A, B be events, and $\Pr(A) = 0.2$. We also have that $\Pr(B \mid A) = 0.75$ and $\Pr(A \mid B) = 0.5$. Calculate $\Pr(\bar{A} \cap \bar{B})$.
4. There are two lightbulb manufacturers in town. One of them is known to produce defective lightbulbs 30% of the time, whereas for the other one the percentage is 80%. You do not know which one is which. You pick one to buy a lightbulb from, and it turns out to be defective. The same manufacturer gives you a replacement. What is the probability that this one is also defective?
5. Let S, T, U be events. We have that $\Pr(S) = 0.5$. We also have that $\Pr(T \mid S) = \Pr(U \mid S) = 0.1$, and that $\Pr(T \mid \bar{S}) = \Pr(U \mid \bar{S}) = 0.2$. We assume that T and U are independent with respect to S, that is $\Pr(T \cap U \mid S) = \Pr(T \mid S) \times \Pr(U \mid S)$. Calculate $\Pr(U \mid T)$.

The first problem about medical testing was used as a practice task, and always presented first. The responses for this problem were not used in

[11] This is in the New Zealand educational system, approximately equivalent to the final year of GCSEs. Initially this was explicitly labelled as students in the final year of the GCSE studies.

[12] Again, in the New Zealand system, this is A-levels-equivalent. As before, A-level students were initially explicitly mentioned in the persona description.

the analysis; the teachers were *not* made aware that their responses would be discarded for this problem.

Problems 2 and 3 are 'equivalent'—stripping the context of animals, and changing some of the values, these contain the same information and goal. Similarly, stripping problem 4 of its lightbulbs context and changing some numbers yields problem 5. The information content of the problem, and the solution paths in each representational system, would be identical for each pair. The teachers were not informed of this until after completing the study.[13]

TASKS

The core task of the experiment was in two phases: first, the teachers were to assess the 'informational suitability' of the representational systems for each problem; second, they were to assess the suitability of the representational systems for each problem for a specific student persona. For each task, the participants were asked to arrange the representational systems on the online response form shown in Figure 7.2. The participants entered their identification code, the problem, and cognitive context they were considering, then dragged the labels of the systems onto the central scale, 0 to 100. The horizontal position has no meaning, and participants were informed as such. When they were happy with their response, they clicked the 'Save' button, then 'Reset' to return the labels to the top row. Through the relative vertical positioning of the representational systems, we will be able to address all three of our hypotheses.

To consider informational suitability—that is, the representational systems' suitability when considering *only* the problem, and not who might solve it—we presented the teachers with the problem statement, requested that they read the problem (and to *not* solve the problem), asked if they had any questions, then asked:

> Thinking in the general case, how informationally suitable do you think each representation would be? How well it captures the important parts of the problem, and how well it can be used to solve the problem.[14]

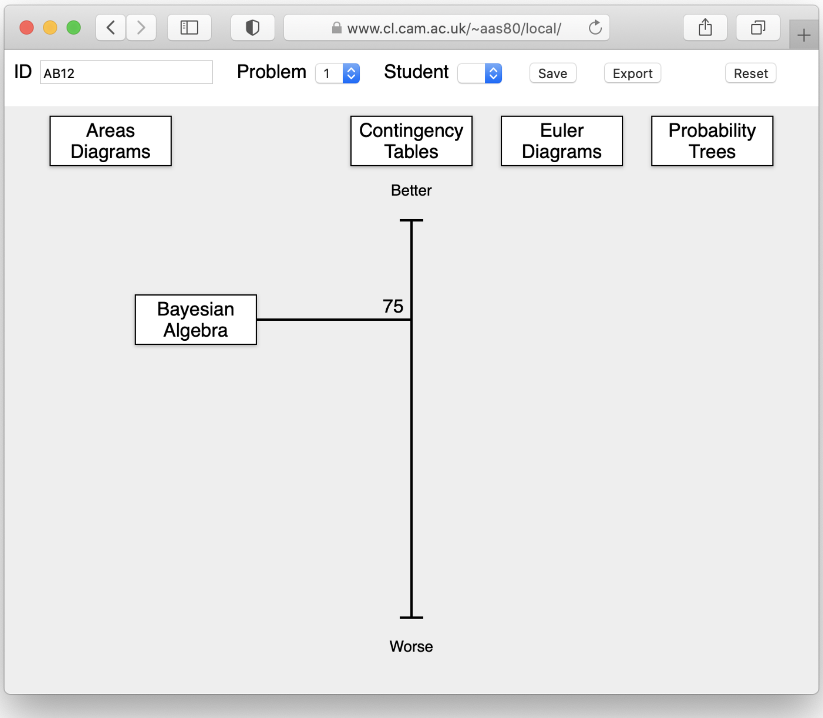
They then proceeded to arrange the representational system labels on the response form. We shall use these responses to address Hypothesis 4.

After all problems had been presented to the teachers, and their responses saved, we moved onto the phase wherein we asked the teachers to consider not just the informational suitability of the problem, but also how appropriate they would be for students via the personas. The teachers then saw the same problems in the same order, but for each they first arranged them based on the following prompt:

> Please arrange the representations based on how suitable each is for Student A to solve the problem.

[13] None of the teachers remarked on this during the experiment, either.

[14] This clarifying fragment acted as a prompt; we explained informational suitability when introducing the participants to the experiment.



ID AB12

Problem 1

Student

Figure 7.2 The response form participants used to arrange representational systems according to the informational and cognitive suitability. Each of the large white boxes begins in the top row, and can be dragged anywhere within the grey region (demonstrated here with the Bayesian Algebra box). Boxes may overlap freely. The line from the white box to the central scale snaps horizontally any distance, and the value attached to it is the user's score for that representational system. The ID box was used to uniquely but anonymously identify respondents; the Problem and Student drop-down selectors identified which problem and persona were currently being considered (a blank Student box indicating no persona). The Save button confirmed the positioning, the Reset button moved all the representational system labels back to the top row, and the Export button was used to return the data. The window here is scaled unusually small to be legible in print; in practice, participants used the form in full-screen on typical laptops. An interactive version of this form is preserved at `https://gistpreview.github.io/?113a52f4bb1d9cdeeab388b8d6d93fea`.

Once their responses had been saved for Student A, we immediately did the same problem for Student B. They completed all problems as before, and then we moved onto the debrief and questioning.

FOLLOW-UP QUESTIONS

Upon completion of all the tasks, the teachers were asked follow-up questions. The nature of this questioning was free-form, but guided by four questions:

1. Did you find this task difficult or easy, and how confident are you in your answers?
2. How familiar were you with each representational system before we started, on a scale from 1 to 10?
3. Which representational systems do you use while teaching, and which are your 'go-to'?
4. When answering our questions, what were the key factors in making your decision?

Other interesting discussion avenues were followed if they came up.

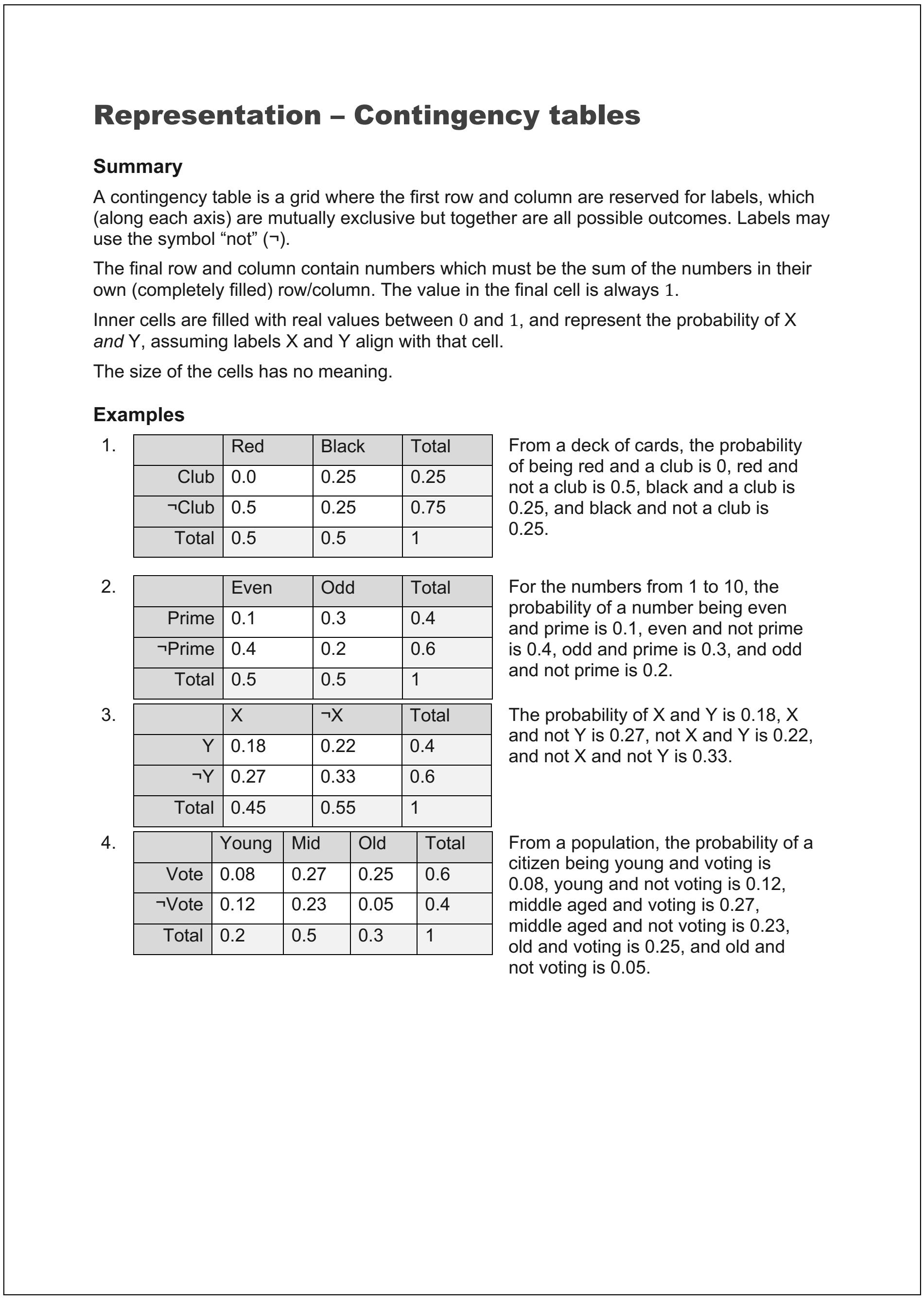
Participants were also sent a short survey to collect demographic information: education, years of teaching experience, recently taught courses, and the school at which they work.

TRAINING

Because the teachers may not all be familiar with the representational systems they will be evaluating—or may have a different understanding of the system to what we intend—we provided training on each. This consisted of going over a single-page PDF document with the teachers; the order that the systems were introduced was counterbalanced. Figure 7.3 shows the training document for contingency tables; all five training documents are in Appendix J.

The training document for each representational system contained a brief description of the system, along with four examples. The training resources were kept uniform in what they described, their length, and their Flesch-Kincaid Reading Grades (mean 5.6, min 3.6 [Euler], max 8.1 [Bayes]).[15] This ensured that no particular resource was more inaccessible than the others for our participants, nor was any representation promoted as 'better' than the others. The examples were a representation belonging to that system, and a short textual description of the representation. Participants were asked to explain how the text described the representation, and then answer some brief questions about extracting information from the representation.

[15] These values are US educational grades; 8th grade students are approximately age 14.



Representation – Contingency tables

Summary

A contingency table is a grid where the first row and column are reserved for labels, which (along each axis) are mutually exclusive but together are all possible outcomes. Labels may use the symbol "not" (¬).

The final row and column contain numbers which must be the sum of the numbers in their own (completely filled) row/column. The value in the final cell is always 1.

Inner cells are filled with real values between 0 and 1, and represent the probability of X *and* Y, assuming labels X and Y align with that cell.

The size of the cells has no meaning.

Examples

1.

	Red	Black	Total
Club	0.0	0.25	0.25
¬Club	0.5	0.25	0.75
Total	0.5	0.5	1

From a deck of cards, the probability of being red and a club is 0, red and not a club is 0.5, black and a club is 0.25, and black and not a club is 0.25.

2.

	Even	Odd	Total
Prime	0.1	0.3	0.4
¬Prime	0.4	0.2	0.6
Total	0.5	0.5	1

For the numbers from 1 to 10, the probability of a number being even and prime is 0.1, even and not prime is 0.4, odd and prime is 0.3, and odd and not prime is 0.2.

3.

	X	¬X	Total
Y	0.18	0.22	0.4
¬Y	0.27	0.33	0.6
Total	0.45	0.55	1

The probability of X and Y is 0.18, X and not Y is 0.27, not X and Y is 0.22, and not X and not Y is 0.33.

4.

	Young	Mid	Old	Total
Vote	0.08	0.27	0.25	0.6
¬Vote	0.12	0.23	0.05	0.4
Total	0.2	0.5	0.3	1

From a population, the probability of a citizen being young and voting is 0.08, young and not voting is 0.12, middle aged and voting is 0.27, middle aged and not voting is 0.23, old and voting is 0.25, and old and not voting is 0.05.

Figure 7.3 The training document for contingency tables. The participants were asked to read the summary at the top, then explain how the textual description was encoded in the representation presented alongside. Participants were then given the correct interpretation.

7.3.3 *Running the experiment*

While the study was designed for the English mathematics curriculum, due to the global covid-19 pandemic, the study was moved to New Zealand at the last moment. As such, we changed the persona of a final-year GCSE student to a Year 11 student (studying for the National Certificate of Educational Achievement [NCEA] Level 1) and the persona of an A-levels student to a Year 13 student (studying NCEA Level 3).

The experiment was run online via videoconferencing software. We first conducted two pilots with high school mathematics teachers from the Canterbury region of New Zealand. The first pilot revealed that the experiment took too long—in excess of 90 minutes—so we removed an 'intermediate' persona of a Year 12 student. The second pilot was run after making this change, and lasted under one hour, without any problems; we thus proceeded with the full experiment.

The participants of this study were high school mathematics teachers, mostly from the Canterbury region of New Zealand. We advertised the study by directly reaching out to the heads of faculty of high schools in Canterbury. We recruited 10 teachers in total (3 male, 7 female) from five separate schools; nine teachers returned usable quantitative data—one teacher returned data that had somehow been corrupted. The participants' teaching experience ranged from two-and-a-half to sixteen years, and all had been mathematics teachers for the entirety of their teaching career. All have a bachelors degree and a postgraduate diploma in teaching; the major of the degree was varied. One participant has a doctoral degree in Statistics, while one has a masters degree in Computer Science. One teacher was studying for a masters degree in Specialist Teaching at the time of the experiment. All had taught courses that included probability content within the past two years.

All the teachers were rewarded with an NZ$20 gift voucher.[16]

The experiment consists of five phases:

1. We introduce and explain the experiment;
2. The teachers are given training for the representational systems;
3. The teachers evaluate each representational system for each problem *without* considering any personas;
4. The teachers evaluate each representational system for each problem *with* consideration to the personas; and
5. We debrief the teachers and ask them the follow-up questions about their experience of completing the above tasks.

Introducing the experiment

After initially getting in contact with the teachers via email, we sent through the consent forms for them to sign and return, and we organised

[16] NZ$20 is approximately £10.

a time to meet via video conference.[17] Upon beginning the video conference and introducing ourselves, we confirmed they had understood the consent form they had signed and returned, and reaffirmed that they were comfortable with the audio recording. During the introduction we motivated the purpose of this experiment to understand how teachers consider solving problems, both in general and for students—they were instructed to *not* solve the problems we gave them. We also explained terms such as 'representational system' and 'informational suitability.'

TRAINING

The teachers were then given training in each of the five representational systems to ensure that every participant was familiar with all of them—and that we agreed upon their definition. Participants consistently made three remarks on the representational systems:

- They were unfamiliar with area diagrams (but one had seen eikosograms before, which are related).
- They were familiar with contingency tables under the name 'two-way tables.'
- They were familiar with Euler diagrams under the name 'Venn diagrams';[18] this is the name used by the NCEA standards specification documents.

This training period lasted about 30 minutes, and the order in which the representation systems were introduced was counterbalanced.

EVALUATING REPRESENTATIONAL SYSTEMS WITHOUT COGNITIVE CONTEXT

Beginning the study tasks, we presented the teacher with each of the five problems. For each problem, we asked them to read and understand the problem but *not to solve it*. The teacher was asked if they understood the problem; every response was affirmative. We then asked them to position the labels of the representational systems in the web interface based on their *informational suitability*. The problems were presented such that the medical testing problem was always given first, as a 'practice' (although this was not disclosed), and then in a counterbalanced manner with the restriction that the pairs of equivalent problems never follow each other.

EVALUATING REPRESENTATIONAL SYSTEMS WITH COGNITIVE CONTEXT

After completing the evaluation task for each problem only for informational suitability, we presented the teachers with a PDF containing the personas of the two students. They were asked to read the personas, and we asked if they had any questions. One participant asked whether either student would be allowed a calculator when solving these problems, and we confirmed that yes, they would have access to a calculator. Another queried how strictly the low-ability student would not use knowledge

[17] The study was originally designed to be run in person, but this was not possible due to national lockdowns during the COVID-19 pandemic.

[18] What they knew as Venn diagrams were in fact Euler diagrams.

from other areas of mathematics, and we confirmed that they had basic knowledge, but they would not use skills beyond basic arithmetic without prompting. Each of the problems were then presented in the same order as the previous phase, but this time the teachers were asked to evaluate the representational systems for first the lower ability student, then immediately after for the higher ability student. Responses were recorded using the same interface.

DEBRIEF AND QUESTIONS

To end the session, we asked the participants our five follow-up questions, as well as any questions that naturally arose from the conversation. We also invited them to complete a demographics survey. Finally, we debriefed the participants on some details of the experiment, notably that the questions came in 'pairs' of the same problem—no participant acknowledged noticing this similarity.

7.3.4 *Quantitative analysis*

To understand the teachers' responses, we break down the data by problem and cognitive context. For each (problem, cognitive context) pair, we consider all of the responses from the participants. We explore two representative examples—the problem that is equivalent to the lightbulbs problem when considering expert student personas, which shows clear groupings in the responses; and the birds problem when considered without any persona, which does not show clear groupings—and include relevant plots for the remaining situations in Appendix K. Tables of all statistical test results are included in Appendix L.

LIGHTBULBS-EQUIVALENT PROBLEM FOR EXPERT STUDENT PERSONA

The lightbulbs-equivalent problem was stated as follows:

> Let $\mathsf{S}, \mathsf{T}, \mathsf{U}$ be events. We have that $\Pr(\mathsf{S}) = 0.5$. We also have that $\Pr(\mathsf{T} \mid \mathsf{S}) = \Pr(\mathsf{U} \mid \mathsf{S}) = 0.1$, and that $\Pr(\mathsf{T} \mid \bar{\mathsf{S}}) = \Pr(\mathsf{U} \mid \bar{\mathsf{S}}) = 0.2$. We assume that T and U are independent with respect to S, that is $\Pr(\mathsf{T} \cap \mathsf{U} \mid \mathsf{S}) = \Pr(\mathsf{T} \mid \mathsf{S}) \times \Pr(\mathsf{U} \mid \mathsf{S})$. Calculate $\Pr(\mathsf{U} \mid \mathsf{T})$.

We asked the teachers to consider each representational system, and in this case they evaluated them based on their suitability to suggest to the high-ability Year 13 student.

To better understand the teachers' responses, we first plot the data in Figure 7.4. We immediately notice two things: the first is that the Bayesian algebra grouping is visually much tighter than the others, and also much higher; the second is that, in particular, area diagrams are spread out.[19] The first observation gives us what appears to be a 'winner': for this problem and this student persona, the teachers would consistently recommend the Bayesian algebra representational system. This

[19] We also notice that one participant—the green dot in the plot—is consistently ranking all non-Bayesian systems at zero.

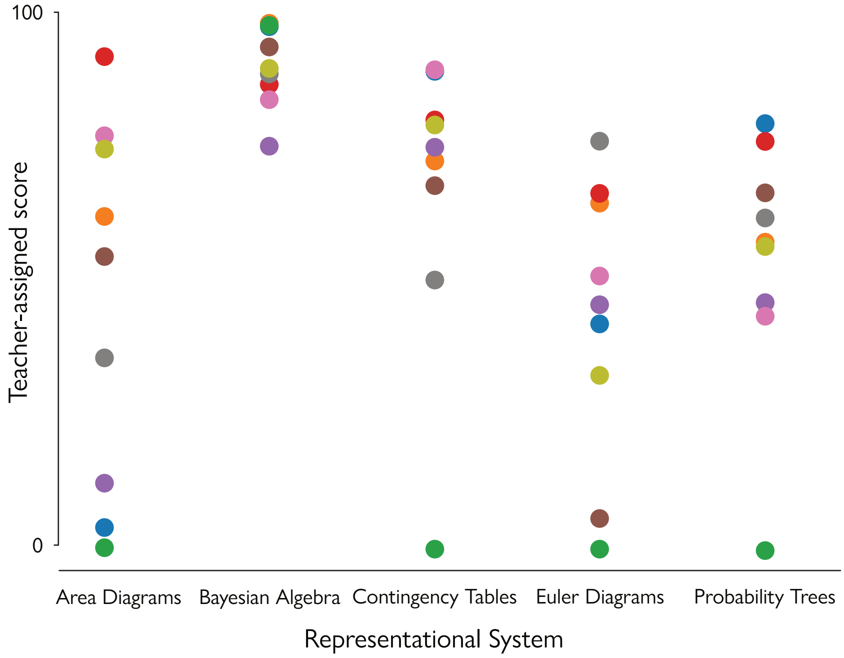
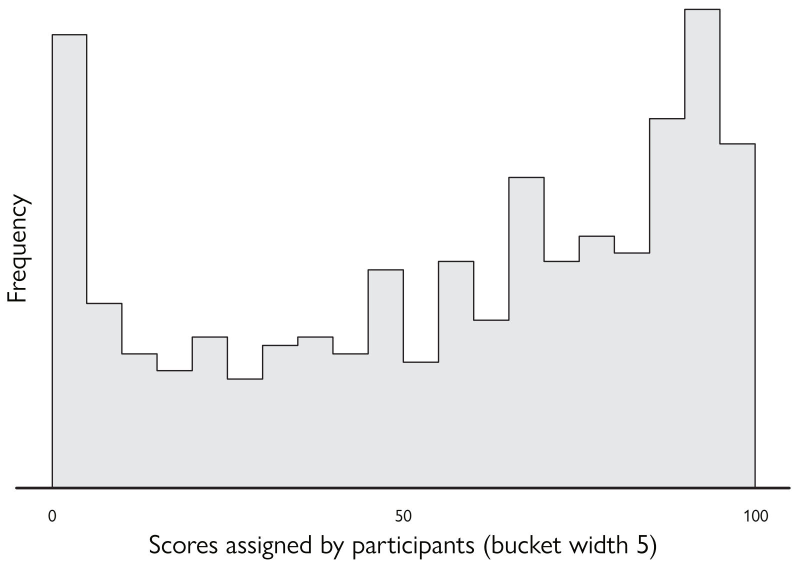


Figure 7.4 The teachers' assigned scores for each representational system, when asked to consider their appropriateness for a high ability Year 13 student when solving a problem that is equivalent to the lightbulbs problem. Each colour represents one participant; the points are not aggregates.

seems to support Hypothesis 1, in that the teachers have consistently identified a representation to recommend. It then appears the teachers would suggest (after Bayesian algebra) to use contingency tables, followed by probability trees, then Euler diagrams. As we mentioned, the responses for area diagrams are too spread to make a clear statement: for this representational system, the teachers were inconsistent with each other. We note that this was the representational system with which our participants were least familiar.

Following this visual inspection, we performed a more formal evaluation. Due to so few scores, and their significant non-normal distribution[20] we used rank-order statistics. The teachers' responses were integers from 0 to 100; we converted these to ranks for each representational system, preserving ties. Using these ranks, we performed a Friedman test between the mean rankings of each representational system. For this problem and cognitive context, we find there is a significant difference between the representational system rankings ($Q = 20.50$, $p = 0.0004 < 0.05$). Post-hoc Wilcoxon signed-rank tests between every pair of representational systems reveal two significant differences after Bonferroni correction (for ten comparisons): between Bayesian algebra and Euler diagrams ($W = 0$, $p = 0.004 < 0.005$) and between Bayesian algebra and probability trees ($W = 0$, $p = 0.004 < 0.005$). Thus we can state that we have evidence that the participants would recommend the Bayesian algebra representational system over the Euler diagrams and probability trees systems.

[20] We plot all the scores in Figure 7.5; formal tests for non-normality also produce significant scores.



A histogram of all the scores provided by the participants, grouped into buckets of width 5 and then counted. We note two peaks, and a left skew; this data is not normally distributed, preventing us from using parametric statistics. Figure 7.5

While not a comprehensive ranking, we have extracted a ranking from the teachers' responses, giving evidence for Hypothesis 1.

BIRDS PROBLEM WITHOUT ANY PERSONA

The birds problem was stated as follows:

> One quarter of all animals are birds. Two thirds of all birds can fly. Half of all flying animals are birds. Birds have feathers. If X is an animal, what is the probability that it's not a bird and it cannot fly?

As before, we plot the teachers' responses in Figure 7.6. This time, any patterns are much less clear. All the representational systems' scores are spread across the scale, with none clearly being better or worse than the others. We might generously state that Euler diagrams has scores that are typically higher than the others, but this is far from conclusive. We also notice a slight separation on contingency tables, but the cause or meaning is difficult to determine. Unlike last time, there is no apparent 'better' representational system.

After performing the same transformation from scores to ranks, we can perform a Friedman test to determine if there exists a significant difference between the rankings of the representational systems. No significant difference was found ($Q = 7.75$, $p = 0.101$), which matches our visual intuition. Thus in this case, we have no evidence to support Hypothesis 1, that the teachers were able to agree on the informational suitability of each representational system for the birds problem. We

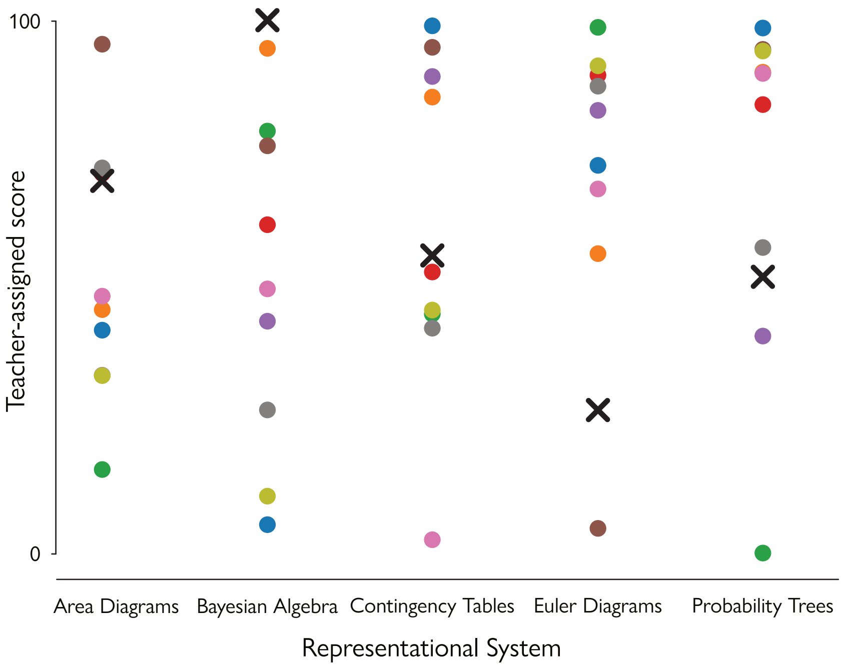


Figure 7.6 The teachers' assigned scores for each representational system, when asked to consider their informational suitability when solving the 'birds' problem. Each colour represents one participant; the points are not aggregates. The black crosses are scores from `robin`; we will discuss this when considering Hypothesis 4.

made the assumption that the teachers are working from a similar situation, knowledge, and experience; there may be individual differences that we have not accounted for.

OTHER COMBINATIONS

These two (problem, cognitive context) pairs are representative of the results from all twelve pairs. We summarise all pairs in Table 7.2, where we list the representational systems that a post-hoc Wilcoxon signed-rank test indicated a significant difference between; the representational system which is ranked higher is listed first in each cell. Tables of statistical test results are in Appendix L.

Based on the summarised results, we see that in one quarter of cases, there is no evidence of a difference between each system. In another quarter, we found evidence that there might be a difference in rankings between the representational systems, but our post-hoc tests were not sensitive enough to determine the difference.[21] But we note that there is no complete row or column in Table 7.2 that is similar: both the problem and the cognitive context seem to be having an influence on the result. With one exception[22] every problem and cognitive context clearly shows a different outcome. Thus, for Hypotheses 2 and 3 we have evidence to suggest that the teachers were considering both the problem and cognitive context in their evaluation of each representational system.

[21] Based on the plots in Appendix K, in some cases we can visually determine which may be different.

[22] The birds problem, and the birds equivalent problem, for informational suitability only. That is, row one, columns one and two.

Table 7.2

Summary of the significant differences found in the rankings of the representational systems for each problem and cognitive context pair. The visually higher scored system is written before the visually lower scored system. An asterisk (∗) means a significant difference ($p < 0.05$) was found, but no pairs of representational systems were found different in post-hoc tests ($p < 0.005$). Entries with a dash (—) were not found to have significant differences between the representational systems. Exact results and p-values are shown in Appendix L.

	Birds	Birds-equivalent	Lightbulbs	Lightbulbs-equivalent
No persona	—	—	∗	Contingency/Areas
Low ability	Trees/Bayes	∗	Trees/Bayes	—
High ability	∗	Bayes/Euler	Trees/Areas, Trees/Euler	Bayes/Trees, Bayes/Euler

COMPARISONS WITH OUR FRAMEWORK

Hypothesis 4 contrasts the results of the implementation, robin, with the responses of the participants. We currently only have an automated implementation of the informational suitability computation, so restrict ourselves to the situation in which we did not ask the teachers to consider a persona. From these four cases—one for each problem—we are able only to make one claim on behalf of the teachers: the contingency tables would be more informationally suitable than area diagrams for the problem equivalent to the lightbulbs problem (read from Table 7.2). Posing the same question to robin,[23] we get the following scores:

	Bayes	Areas	Contingency	Trees	Euler
Score	8.55	6.19	6.10	5.05	2.20

We see that the algorithm was *not* able to clearly separate area diagrams from contingency tables, and considered them approximately equal.

If instead we perform a visual inspection on the teachers' responses, shown in Figure 7.7, we can extract the following ordering,[24] best to worst:

Bayes Contingency Euler Trees Areas

which agrees on the ordering of Bayesian algebra, contingency tables, and probability trees relative to one another, and on the relative ordering of Bayesian algebra, contingency tables, and Euler diagrams. The placement of area diagrams and Euler diagrams differ significantly between the two orderings; it remains inconclusive how correlated the scores from the robin implementation are with the teachers' responses. Formally, we are interested in whether the representational systems are in the same relative position for their rankings, for example that Bayesian algebra ranks higher than Euler diagrams, regardless of their absolute position in the ranking. For this, we can use Kendall's rank correlation coefficient (τ)[25] to determine the correlation of the teachers with our framework [Kendall 38]. Running Kendall's τ test on the two orderings returns a non-significant result ($\tau = 0.40$, $p = 0.48$), failing to find evidence of correlation.

[23] We use the input based on Bayesian algebra, rather than natural language, as we find this retains more information for the algorithm to work with.

[24] While not rigorous, it exemplifies the analysis we are performing. In the next paragraph we work with the teachers' ranks directly.

[25] We use τ-B, which is more suitable for when the rankings are all coming from the same range, allowing ties.

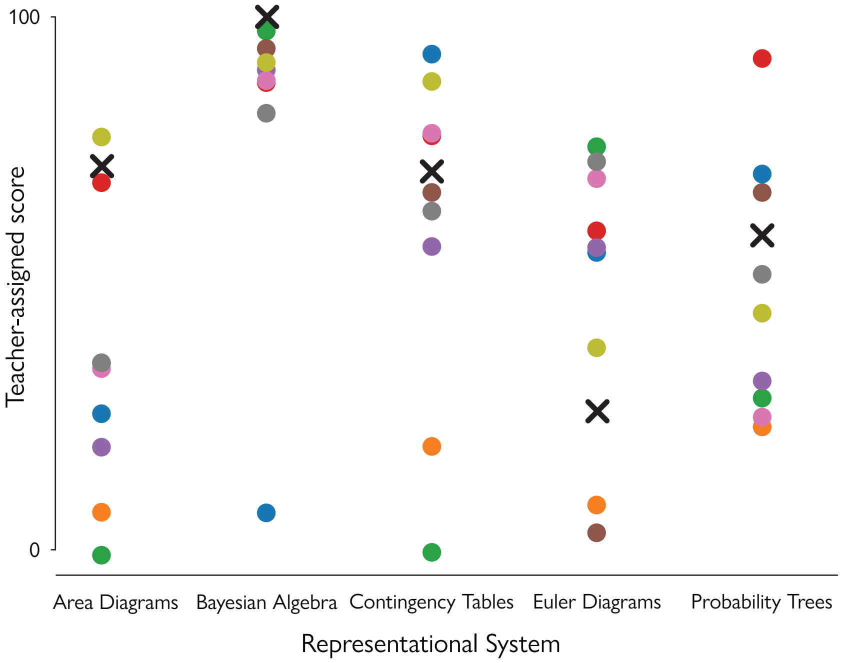


Figure 7.7 The teachers' assigned scores for each representational system, when asked to consider their informational suitability when solving the problem equivalent to the lightbulbs problem. Each colour represents one participant; the points are not aggregates. The black crosses are `robin`'s output, scaled from $[0, 8.55]$ (because 8.55 is the highest score for this problem) to $[0, 100]$.

Performing the same τ analysis between robin and each teacher individually, across all the problems, reveals no instance in which there is significant correlation.[26] So we have failed to find evidence of correlation between robin's scores and the teachers' responses, both individually and in aggregate. This is not surprising, given the teachers do not seem to correlate with each other.[27]

More informally, we can inspect how robin's output compares to the responses of the participants by plotting the scores alongside the participants. We scale the scores from the interval $[0, \max(\text{scores})]$[28] to $[0, 100]$ and include the adjusted scores as black crosses in Figure 7.7. Visually inspecting the placement of the crosses, and the general groupings of the points from the participants' responses, we see that the crosses generally follow the 'mass' of the points.

Of the four situations where we can compare robin's scores against the participants' responses (Figures 7.6, 7.7, and two further plots in Appendix K) we would argue that in three plots robin's scores fall within the distribution of the teachers' scores.[29] While the participants' responses for the birds problem are too spread out and robin's score for Euler diagrams too different, the framework's scores for the remaining three problems are visually following the same groupings. So to address Hypothesis 4, while we cannot state with any statistical confidence that our framework produces scores that agree with the participants' responses, there are indications that the framework is evaluating representational systems in a manner comparable to that of our expert participants. A larger study may allow this analysis to be formalised.

7.3.5 *Qualitative analysis*

The inconsistency of the participants prevented us from making more extensive comparisons between the scores they assign and those computed by the rep2rep framework. We propose two possible causes for this inconsistency: there is an underlying factor we did not control for; or the task is difficult, even for experts. Let us consider each in turn.

Before starting this study, we identified three factors that might influence the participants' responses that we could not control:

- external motivation for using one system over another;
- preference of some representational systems over others; and
- experience as a teacher, and in using particular systems.

During the debriefing interview we asked the participants four questions, which we stated in Section 7.3.2. We asked the participants how familiar they had been with each representational system prior to the training we provided. The participants were universally confident with probability trees, contingency tables, and Euler diagrams;[30] two thirds were comfortable with Bayesian algebra, but the rest had only memories of having learned it before; none had seen area diagrams before the

[26] 32 τ tests, all with $p > 0.05$. In one case for the birds problem we have $p = 0.083$, an *almost* significant correlation, but $\tau = -0.8$ meaning the teacher is *negatively* correlated with robin.

[27] 144 τ tests between all pairs of teachers, all with $p > 0.05$. We have 15 cases (about 10%) where $p < 0.1$, 12 of which are positively correlated, three negatively. This is likely due to chance, and is a Type-I error.

[28] The $\max(\text{scores})$ value is computed per problem, not overall.

[29] The non-normal nature of the data prevents us from verifying this using parametric statistical tests such as t-tests.

[30] Many teachers remarked that they knew Euler diagrams as Venn diagrams; similarly, most referred to contingency tables as 'two-way' tables.

study, but one third still felt they would confidently be able to use the representational system even before our training.[31]

While more than half of teachers initially answered that their responses were primarily based on 'gut instinct' rather than external factors, further discussion revealed influences from the curriculum, including the standard assessments. As stated by one participant:

> The assessments lean pretty heavily into [probability trees and contingency tables]. They'll have a space for you to draw the tree, or they'll give you the table, things like that.

A lack of training and resources was a recurring theme. Only one participant mentioned relevant professional development, and praised its potential:

> We have bugger all PD [professional development]. There is no maths PD organised by the school [...] there is very little maths PD organised by the ministry. Most of the PD is organised by teachers within schools finding stuff themselves online, by maths associations running PD [...] and the amount we run is pathetic really. And yet it's the only stuff going most of the time. [...]
>
> The best PD I've had recently—I had about four years worth—I got onto a programme which was for primary school teachers only. [...] And that was about using rich task problem solving,[32] and that has completely transformed my teaching, and yet almost nobody has had that.

Many of our participants responded similarly on the lack of professional development, and wished for more. This highlights a need for frameworks that allow for using more diverse representational systems: by reducing the barrier to using multiple representational systems, our framework could support improved learning opportunities, for both students and teachers.

The participants were asked if they had any 'go-to' representational systems when teaching probability. All responded with either probability trees or contingency tables, with half pointing out that these are encouraged by the assessment standards, as mentioned above. We observe that these systems (particularly probability trees) are the ones found to be overall favoured by our participants, as seen in Table 7.2. Euler diagrams are only introduced at Year 13—students typically aged 17 to 18—and many teachers acknowledged they were reluctant to use them purely because they felt they were 'too hard' for students.[33]

> The kids do find [Euler diagrams] quite challenging, so you tend to only use those when you really need to, or when they're indicated as a 'good way' to solve that particular problem [by the resource materials]. [...] They're the last resort.

[31] One participant had seen eikosograms, which are related to area diagrams [Oldford et al. 06].

[32] 'Rich task problem solving': using contexts, representations, and discussions to improve mathematics learning [Piggott 08].

[33] As we will note in Section 7.3.6, Euler diagrams (as 'Venn diagrams') are introduced much earlier in the UK.

Based on these responses, we suspect that personal preference and curriculum were factors in our participants' responses. We cannot directly untangle the link between curriculum and preference: the participants all work within the New Zealand mathematics curriculum, so are most familiar with (and have most experience with) the mandated representational systems. In future, we would run a similar experiment on a different cohort of teachers working with a different curriculum, or with teachers from multiple curricula, to identify if this influences the teachers' responses.

We also cannot discount the possibility that this representational system recommendation task was difficult, even for experienced teachers well-versed in the subject matter. As part of the debriefing interview, we asked the participants to self-assess how difficult they found the evaluation task. Responses were split to extremes: just under half responded that it was difficult, with the rest responding that it was easy; there was no obvious relationship between this response and years of experience. Such a binary split on a self-assessment question suggests more work is needed to determine what makes this task simple or difficult; alternatively, we need to find what assumptions some of the participants might be making that cause the task to be easier or more difficult.

Overall, we find that the teachers are only partially able to produce a consistent recommendation of representational systems. There are some general trends, but our participants did not consistently agree with each other—against our initial hypotheses. The inconsistency, and the participants' explicit mention of lack of training in re-representation, indicates that while teachers have an interest in learning about teaching with multiple representations, but this need is not being met; in turn, this means that students may not be exposed to the diversity of representations they could be. This reinforces a need for tools such as ours that are able to support heterogeneous reasoning.

7.3.6 *Limitations and threats to validity*

This study is limited in scope, and thus in its generality. Further, we identify some threats to the validity of this study, which we were unable to address in the study design. Five key limitations and threats are:

- number of participants;
- scope of population;
- potential learning effects;
- limited alignment of tasks for the population; and
- concerns over study materials.

The primary limitation of this study is the low number of participants: while nine is viable, the power of the study is limited. The responses from this study provide interesting preliminary data, but the fields of information representation and education would both benefit from a larger version of this study: does the disagreement we have found continue to be present over larger groups of participants? Are there regularities in the disagreements that we were unable to discover? Can a more complete recommendation be extracted from a larger pool of participants?

We recruited our participants from a limited set of schools in a geographically restricted area. Participants were self-selected, likely knew each other professionally, and shared an interest in representations in education. This potentially reduces the diversity of our participants, and their responses. The scope of this study was also restricted to probability problems. While these restrictions reduce the variability in our sample, they restrict the generality of our results.

A design limitation of this study is that the initial problems and representational system selection was based on the mathematics curriculum in England. While some changes could be made quickly—such as translating the English GCSE/A-levels student personas the NZQA framework—we could not make others because we did not identify them ahead of time. One notable change is the order and age at which different representational systems are introduced by each curriculum. For example, in England, Venn diagrams[34] are introduced at 'Key Stages' 3 and 4,[35] aimed at students aged 11 to 14; in New Zealand, Venn diagrams are introduced at NCEA Level 3,[36] aimed at students aged 17 to 18. Such a large difference is surprising, and may account for why our New Zealand-based participants were reluctant to recommend Euler diagrams: they associate them with advanced mathematics content.

Finally, in the experiment resources, we identified concerns after the experiment had been run. Of small consequence are 'typos' in the training documentation: participants noticed some mistakes, but inferred the intended meaning. A larger consideration is the nature of the 'isomorphic' problems: we had 'contextual' problems (birds or lightbulbs) and 'context-less' problems using letters as variables and a probability function. These 'context-less' variants might have encouraged participants to favour the Bayesian algebra system, which also uses letters as variables and a probability function. Indeed, we see this in Table 7.2: every situation where Bayesian algebra is preferred is a 'context-less' problem.[37] In future studies, we suggest using 'equivalent' problems that retain a context to avoid the Bayesian algebra bias,[38] but to use a *different* context.

We took steps to mitigate other potential threats to validity, which we briefly summarise here.

Learning effects Because the participants did not see the problems or representational systems in the same order due to counterbalan-

[34] Neither curriculum mentions Euler diagrams.

[35] https://www.gov.uk/government/publications/national-curriculum-in-england-mathematics-programmes-of-study/national-curriculum-in-england-mathematics-programmes-of-study

[36] https://www.nzqa.govt.nz/nqfdocs/ncea-resource/achievements/2019/as91585.pdf

[37] They are also exclusively for the 'expert' student persona.

[38] Interestingly, this bias is similar to 'identity' correspondences within our framework.

cing, no one representational system or problem would produce higher or lower scores based on its position in the order of presentation and the participants becoming more experienced.

Carry-over effects Participants might have been answering questions by remembering their previous answers and submitting them again. We counteract this by requiring them to reset the positions of the labels in their response between each problem and persona; to submit the same answer twice they must manually recreate it.

Task practising So that participants understood the task they were asked to perform, we discard their responses to the first problem considered. In each case, this was the 'medical' problem. This means that if the participant was unsure of the task based only on our description of the task, they could experiment and ask clarifying questions on this problem without affecting the results of the remaining problems.

Fatigue We limit the experiment duration for each participant to under and hour. This prevents the participant becoming unnecessarily fatigued, which could influence their responses.

Avoiding priming In the *debriefing* interview we asked participants to rate how familiar they were with each representational system *before* the experiment began. While we could have asked this question before training the participants, and likely get a more honest response, doing so might have primed them to favour which representational systems they already knew.

Familiarity bias We cannot discount familiarity with each representational system influencing the participants' responses; area diagrams being scored inconsistently may be an artefact of unfamiliarity. We provided training in each representational system which lasted about 30 minutes—thus using about half of the experiment time.

7.3.7 *Conclusions*

This study, while not able to directly evaluate the framework as intended, has provided valuable information about how teachers evaluate representational systems. We have found, contrary to Hypothesis 1, they are not as consistent as we expect: they often fail to agree with each other on the suitability of a particular representational system. But they *are* reacting to the situation in which they are making a recommendation: the teachers' responses do indicate that Hypothesis 2 (that the problem is a factor in their evaluation) and Hypothesis 3 (that the cognitive context is a factor in their evaluation) may be correct. Further studies are needed to determine the influence of these factors—and potentially others—on the final recommendation. Finally, Hypothesis 4 is weakly supported by the apparent grouping of our framework's scores and the teachers' responses; we are capturing at least some of what influences the teachers' representational system recommendation. Together these four hypotheses emphasise that our framework is grounded and useful: much like the experts, we are considering the problem *and* the cognitive context to make a suitable representational system recommendation.

SUMMARY OF SECTION 7.3

This section detailed a study used to empirically evaluate the framework presented in this dissertation. We found that our expert participants—high school mathematics teachers—did not typically produce consistent recommendations for specific problems and student personas. Nonetheless, where viable to check, we find the framework produces similar scores to our participants. It also revealed some interesting insights into how the participants understood representations. We note the effects of which problems are given and the cognitive context on the representational system recommendation.

SUMMARY OF CHAPTER 7

In this chapter we described an ablation study and a user study as means to evaluate the rep2rep framework as presented in this dissertation. The ablation study demonstrated that each of component importance and correspondence strength contribute to the final representational system recommendation in a way that improves the correlation with experts' recommendations. We also performed a study with teachers, asking them to evaluate the suitability of representational systems for a set of problems with student personas. The teachers provided insights into their approach to re-representation, but also revealed the need for tools which are able to consider the problem and problem solver.

8 Conclusions & future work

> No book can ever be finished. While working on it we learn just enough to find it immature the moment we turn away from it.
>
> — *Karl Popper*

Representation choice can have a dramatic effect on a problem solver's ability to solve a problem. By representing a problem effectively, the solver could reach a solution in a manner more suited to them, and their problem. To support the solver, we aimed to recommend effective representational systems, where effectiveness is a function of both the informational suitability of the overarching representational system, and the cognitive costs of a particular representation. This dissertation contributes a novel method based on correspondences to recommend representational systems tailored to both the problem and the solver.

This project has been driven by three research questions, each leading to a set of objectives. Our first question grounds our research.

> **Question 1**. What constitutes a problem, representation, and representational system, and can we describe each of these in a way that is equally suited to many varieties of representations?

This lead to three objectives:

- distinguish between problems, representations, and representational systems;
- identify the fundamental components of a representation, applicable to all representational modalities; and
- arrange these components into descriptions of problems, representations, and representational systems.

The second research question pointed more directly at evaluating the 'similarity' of representational systems, and understanding how to interpret this as analogy.

> **Question 2**. How are representational systems—and their components—similar, and can we state which components are similar across systems?

The subsequent objectives were to:

- define a similarity relation on components and descriptions;

- interpret this relation with respect to the underlying representational systems; and
- determine this relationship between two arbitrary representational systems' descriptions, potentially automatically.

Finally, the third question brought together these ideas to produce a representational system recommendation framework, and an implementation.

> **Question 3**. How can we algorithmically evaluate and rank representational systems based on their ability to be used to solve a particular problem?

So our final addition to the set of objectives was to:

- define a measure of 'suitability' for a representational system with respect to a problem and user;
- implement this suitability function as practical validation; and
- evaluate the 'correctness' of (our implementation of) this suitability function.

8.1 Contributions

By completing the objectives stated above, we have made theoretical and practical contributions to the field of artificial intelligence, namely in information representation for collaborative human-computer systems. Because the work presented in this dissertation was done alongside researchers from the rep2rep project, we split the contributions in two: four novel contributions of this dissertation, and three contributions in collaboration with the rep2rep research group.

8.1.1 *Contributions of this dissertation*

The primary contribution of this dissertation, introduced in Chapter 4 and built upon in Chapter 5, is the theory of *correspondences*. A correspondence is a means of linking components from different RS-descriptions based on the probability of the component being present in R-descriptions of representations. This allows us to understand how information gets re-represented, and understand how effectively the informational content is preserved; this is our answer to the second research question. We proved that correspondences are contrapositive, and that new correspondences can be derived via *reversal* and *composition*. We developed a *relation* heuristic to further extend sets of correspondences automatically. Correspondences induce *pseudo-descriptions*, which describe potentially analogous representations in different representational systems. Finally, we generalise correspondences to work outside the rep2rep framework.

In Chapter 5, specifically in Section 5.3, we considered how sets of correspondences behave when used to determine the *informational suitability* of representational systems. Considering correspondences that are not applicable to the given descriptions is not useful, but if we discard too many we fail to cover the given Q-description, and thus fail to understand how to re-represent some of the information in the problem. Conversely, considering sets of correspondences that cover the same components multiple times is also not useful. If the component has been covered, we can already re-represent the information it encodes in the potentially analogous representation; being able to re-encode the same information in several—possibly incompatible—ways does not improve the 'suitability' of the alternative representational system. We defined the concept of minimally redundant and maximally covering (MRMC) sets, then gave a practical implementation to efficiently find correspondence sets that approximate an MRMC set. This helps us to justify the effectiveness of informational suitability as an objective function for recommending representational systems for problem solving.

Continuing in Chapter 5, in Section 5.4, we described the `robin` implementation of our framework. The implementation consumes descriptions and correspondence sets, and then estimates the informational suitability of representational systems by applying the definition of informational suitability[1] to the Q-description of the problem and RS-description of the alternative representational systems. We presented the implementation of the informational suitability computation, how we lift importance from components to correspondences, and how we approximate an MRMC set. Thus we contribute a practical method for algorithmic representational system recommendation, addressing our third research question.

The final contribution solely of this dissertation is an empirical evaluation of the framework. Section 7.3 describes our evaluation, which involved presenting mathematics teachers with (informal, human-readable) descriptions of representational systems and student personas, and asking them to evaluate each system on its suitability to solve each of five problems in the cases of no particular student, a novice student persona, and an expert student persona. We established that our participating teachers do not have a strongly consistent view of representational system suitability when controlling for problem and student profile, but our framework produces recommendations that are in line with the teachers.

8.1.2 *Contributions as part of the* rep2rep *project*

The rep2rep project contributes a framework to describe representations and representational systems in terms of their *components*. In Chapter 3 we introduced components, and how they can be composed into *descriptions*, and so addressed our first research question. While the ideas of components and descriptions in this dissertation are a joint contribution,

[1] Definition 15, page 107.

they have been continually refined and updated in conjunction with this work. The practical aspects of components and descriptions, such as a textual format and a data structure in Section 3.2, were contributed as part of this dissertation.

In Chapter 5, we presented informational suitability—which was developed jointly as part of the rep2rep project—as a means to answer our third research question on algorithmic representation recommendation. Definition 15 was influenced by both the rep2rep project and by the work in this dissertation, and this mix of influences results in a combined contribution with no lead contributor. The definition of informational suitability relies on correspondences, a contribution of this dissertation.

The final shared contribution of this dissertation and the rep2rep research project was a second evaluation, which we called the ablation study; we presented this in Section 7.2. This study pulled apart the factors influencing the informational suitability score, and determined the contribution of each. We concluded that component *importance* and correspondence *strength* are both important contributors to the final score, and both are required to produce a result that correlates significantly with human experts. This evaluation was performed as part of this dissertation, but the data was largely generated by rep2rep researchers.

8.2 Future research avenues

This dissertation provides a foundation for automated representation recommendation, and there are many open threads of research to extend this work. In this section, we highlight two categories of these research opportunities: improvements to the framework, and applications of the framework.

8.2.1 *Improvements to the framework*

Most directly related to the work in this dissertation, we found there were certain situations where design decisions impacted our ability to incorporate later ideas. With future iterations of the rep2rep framework, we might be able to make refinements in components, correspondences, and the objective functions.

A limitation noted in Section 3.1.3 is that tactics are parameterised over the *number* of laws they must be instantiated with. For example, rewriting is parameterised over one law and one pattern, because you can rewrite the pattern according to the law. Rewriting $a + b$ to $b + a$ requires the pattern of addition, along with the law of commutativity of addition. But there is nothing in the tactic that ensures the tactic, law, and pattern are compatible with each other. For example, we could try and apply the law of commutativity of addition to matrix multiplication, with nonsensical results. Having some way to assign 'types' to the laws and tactics would improve this situation.

In Section 5.1.4 we discussed the impact of analysts' bias on the inputs, and subsequent outputs, of the `rep2rep` framework and `robin`. Any 'mistakes'[2] that have been made can cause problems beyond the initial components or correspondences: we discussed how missing or extraneous correspondences can affect *all* recommendations by `robin`, including recommendations not directly using those correspondences. While removing analysts from the pipeline entirely is not yet feasible, we might be able to limit the impact of the 'mistakes.' We noted, as suggested by reviewer Prof. Anthony Cohn, that we might repurpose the user profiling to profile the analysts, and so moderate the impact of the descriptions and correspondence sets they provide. Thus alongside strength and importance, we can module informational suitability and cognitive costs by the *reliability* of the analyst who provided the component/correspondence.

In Section 5.3 we defined a *minimally redundant and maximally covering* set of correspondences. Correspondence sets are made 'consistent' by applying the MRMC refinement operation; that is, we keep what we need and no more. But there are other ways to ensure consistency: one potential avenue is correspondence 'bundling.' Using predefined sets of correspondences that are known to be consistent would mean that only correspondences from a single bundle could be used in an informational suitability computation—consistency is guaranteed, assuming the bundles are consistent. This approach has two drawbacks: first, it requires significant manual effort to bundle the correspondences and ensure the set is consistent; second, it requires either considerable duplication of correspondences which would exist in many bundles at the same time, or it requires some sort of inheritance relation on correspondence bundles. We decided to proceed with the MRMC approach, but are eager to explore correspondence bundling.

One recurring idea in the literature on analogy is that higher-level, abstract relationships between the representations produce more effective analogies than lower-level, concrete relationships [Gentner 83; Thagard 92]. Correspondences allow relationships at any level of abstraction, but do not distinguish whether the relationship is more abstract or more concrete. If we were able to capture this 'abstractness,' we would consider weighting the informational suitability calculation by component importance, correspondence strength, *and* correspondence abstraction. Gentner's structural similarity would provide a strong base for this work to build on when combined with correspondences between representational systems.

Finally, one of the largest gaps in this work, which we are actively working to fill, is that we consider only two of the three factors of *cognitive fit* [Vessey 91; Moody 09]: the problem, and the user. We do not consider the task in which the user encountered the problem: are they trying to reach a solution, understand the problem more deeply, learn a new skill, or something else? We briefly touched on this in Section 5.2.3, but a more detailed treatment is needed.

[2] Reiterating a point from Section 5.1.4, there is no true 'correct' set of components or correspondences, but some sets are less correct.

8.2.2 *Applications in tutoring systems*

During this dissertation we focused on using representations in problem solving, but occasionally hinted at applications in intelligent tutoring systems (ITSs). We believe this would be an excellent application of our work, and are actively working on a proof-of-concept ITS. In addition to the significant engineering challenges this poses, there are two primary research tasks that need to be completed: how can we convert a *recommendation* to an *explanation*, and *when* do we make a *recommendation*?

At present, the only information that comes with our recommendation is a vector of scores: each representational system is assigned a single number that captures how 'effective' we have computed it to be. But we have intermediate results that are potentially much more informative: the MRMC correspondence sets. By selecting a few strong, important correspondences from these sets we have an encoding of what is most influential in the suggested analogy. We propose converting these correspondences into a natural language explanation that can be presented to students using our ITS. Consider the educational benefit of presenting a student, stuck on our sum of natural numbers problem, with a description such as the following:

> To represent this problem using dot diagrams, consider representing the numbers as dots, and the summation as stacking the dots.

This is likely not a difficult natural language generation task, with potentially huge didactic benefits.

Finally, one problem we have not considered during this dissertation, or in the rep2rep project, is *when* to make a recommendation. In the context of an ITS, we must consider when a student is stuck, when to *allow* them to be stuck, and when to prevent them getting stuck at all. This goes hand-in-hand with the question of how much support to give: suggesting a representation change, suggesting which representation to change to, and describing how the transformation to the new representation might occur. These are interesting considerations specific to the ITS application, but with close links to our existing work.

CLOSING REMARKS

Representing a problem well can make it trivial to solve; represent it poorly, and it becomes impossible. Choosing an appropriate representation is a difficult, long-standing problem in artificial intelligence; this dissertation contributes a novel approach for the identification of alternative representations of problems through *correspondences*. Exploiting correspondences, we demonstrated how to compute the *informational suitability* of alternative representational systems; this theory and our implementation were demonstrated by applying both to the problem of programming language selection. We also performed an empirical study in which we asked teachers to evaluate representational system suitability, discovering agreement between our framework and our teachers, but highlighting a need for tools that support using multiple representational systems. This dissertation creates possibilities for such tools that react to the problem and user—ones that consider the representational needs of the *human*, not the computer.

Bibliography

[Ainsworth 08]
S. Ainsworth. 'The Educational Value of Multiple-representations when Learning Complex Scientific Concepts.' In: *Visualization: Theory and Practice in Science Education*. Ed. by J. K. Gilbert, M. Reiner and M. Nakhleh. Vol. 3. Springer, 2008. Chap. 9, pp. 191–208. doi: 10.1007/978-1-4020-5267-5_9.

[Alexander et al. 12]
R. G. Alexander and G. J. Zelinsky. 'Effects of part-based similarity on visual search: The Frankenbear experiment.' In: *Vision Research* 54 (2012), pp. 20–30. doi: 10.1016/j.visres.2011.12.004.

[Alguliev et al. 11]
R. M. Alguliev, R. M. Aliguliyev, M. S. Hajirahimova and C. A. Mehdiyev. 'MCMR: Maximum coverage and minimum redundant text summarization model.' In: *Expert Systems with Applications* 38.12 (2011), pp. 14514–14522. doi: 10.1016/j.eswa.2011.05.033.

[Anderson 02]
J. R. Anderson. 'Spanning seven orders of magnitude: a challenge for cognitive modeling.' In: *Cognitive Science* 26.1 (2002), pp. 85–112. doi: 10.1207/s15516709cog2601_3.

[Baker et al. 92]
S. Baker, A. Ireland and A. Smaill. 'On the use of the constructive omega-rule within automated deduction.' In: *Logic Programming and Automated Reasoning*. Ed. by A. Voronkov. Vol. 624. Lecture Notes in Computer Science. Springer, 1992, pp. 214–225. doi: 10.1007/BFb0013063.

[Barker-Plummer et al. 08]
D. Barker-Plummer, J. Etchemendy, A. Liu, M. Murray and N. Swoboda. 'Openproof - A Flexible Framework for Heterogeneous Reasoning.' In: *Diagrammatic Representation and Inference, Diagrams 2008*. Ed. by G. Stapleton, J. Howse and J. Lee. Vol. 5223. Lecture Notes in Computer Science. Springer, 2008, pp. 347–349. doi: 10.1007/978-3-540-87730-1_32.

[Barwise et al. 96-a]
J. Barwise and J. Etchemendy. 'Heterogeneous Logic.' In: *Logical*

Reasoning with Diagrams. Ed. by G. Allwein and J. Barwise. Oxford University Press, 1996. Chap. 8, pp. 179–200.

[Barwise et al. 96-b]
J. Barwise and J. Etchemendy. 'Visual Information and Valid Reasoning.' In: *Logical Reasoning with Diagrams*. Ed. by G. Allwein and J. Barwise. Oxford University Press, 1996. Chap. 1, pp. 3–25. doi: 10.5555/275772.275773.

[Blackwell et al. 01]
A. F. Blackwell, K. N. Whitley, J. Good and M. Petre. 'Cognitive Factors in Programming with Diagrams.' In: *Artificial Intelligence Review* 15.1 (2001), pp. 95–114. doi: 10.1023/A:1006689708296.

[Blanchette et al. 16]
J. C. Blanchette, C. Kaliszyk, L. C. Paulson and J. Urban. 'Hammering towards QED.' In: *Journal of Formalized Reasoning* 9.1 (2016), pp. 101–148. doi: 10.6092/issn.1972-5787/4593.

[Boy de la Tour et al. 14]
T. Boy de la Tour and N. Peltier. 'Analogy in Automated Deduction: A Survey.' In: *Computational Approaches to Analogical Reasoning: Current Trends*. Ed. by H. Prade and G. Richard. Springer, 2014, pp. 103–130. doi: 10.1007/978-3-642-54516-0_5.

[Bundy et al. 90]
A. Bundy, F. van Harmelen, C. Horn and A. Smaill. 'The OYSTER-CLAM system.' In: *10th International Conference on Automated Deduction*. Ed. by M. E. Stickel. Springer, 1990, pp. 647–648. doi: 10.1007/3-540-52885-7_123.

[Carnie 07]
A. Carnie. *Syntax: A Generative Introduction*. 2nd ed. Blackwell Publishing Ltd, 2007.

[Cheng 02]
P. C.-H. Cheng. 'Electrifying diagrams for learning: principles for complex representational systems.' In: *Cognitive Science* 26.6 (2002), pp. 685–736. doi: 10.1016/S0364-0213(02)00086-1.

[Cheng 04]
P. C.-H. Cheng. 'Why Diagrams Are (Sometimes) Six Times Easier than Words: Benefits beyond Locational Indexing.' In: *Diagrammatic Representation and Inference, Diagrams 2004*. Ed. by A. F. Blackwell, K. Marriott and A. Shimojima. Vol. 2980. Lecture Notes in Computer Science. Springer, 2004, pp. 242–254. doi: 10.1007/978-3-540-25931-2_25.

[Cheng 16]
P. C.-H. Cheng. 'What Constitutes an Effective Representation?' In: *Diagrammatic Representation and Inference, Diagrams 2016*. Ed. by M. Jamnik, Y. Uesaka and S. Elzer Schwartz. Vol. 9781. Lecture Notes in Computer Science. Springer, 2016, pp. 17–31. doi: `10.1007/978-3-319-42333-3_2`.

[Cheng 20]
P. C.-H. Cheng. 'A Sketch of a Theory and Modelling Notation for Elucidating the Structure of Representations.' In: *Diagrammatic Representation and Inference, Diagrams 2020*. Ed. by A.-V. Pietarinen, P. Chapman, L. Bosveld-de Smet, V. Giardino, J. Corter and S. Linker. Lecture Notes in Computer Science. Springer, 2020, pp. 93–109. doi: `10.1007/978-3-030-54249-8_8`.

[Cheng et al. 21]
P. C.-H. Cheng, G. Garcia Garcia, D. Raggi, A. Stockdill and M. Jamnik. 'Cognitive Properties of Representations: A Framework.' In: *Diagrammatic Representation and Inference, Diagrams 2021*. Ed. by A. Basu, G. Stapleton, S. Linker, C. Legg, E. Manalo and P. Viana. Vol. 12909. Lecture Notes in Computer Science. Springer, 2021, pp. 1–16. doi: `10.1007/978-3-030-86062-2_43`.

[Chi et al. 88]
M. T. H. Chi, R. Glaser and M. J. Farr. *The nature of expertise.* Hillsdale, NJ, US: Lawrence Erlbaum Associates, Inc, 1988.

[Chomsky 56]
N. Chomsky. 'Three models for the description of language.' In: *IRE Transactions on Information Theory* 2.3 (1956), pp. 113–124. doi: `10.1109/TIT.1956.1056813`.

[Condell et al. 10]
J. Condell, J. Wade, L. Galway, M. McBride, P. Gormley, J. Brennan and T. Somasundram. 'Problem solving techniques in cognitive science.' In: *Artificial Intelligence Review* 34.3 (2010), pp. 221–234. doi: `10.1007/s10462-010-9171-0`.

[Constable et al. 86]
R. L. Constable, S. F. Allen, H. M. Bromley, W. R. Cleaveland, J. F. Cremer, R. W. Harper, D. J. Howe, T. B. Knoblock, N. P. Mendler, P. Panangaden, J. T. Sasaki and S. F. Smith. *Implementing Mathematics with the Nuprl Proof Development System*. Prentice-Hall, Inc., 1986.

[Coquand 06]
T. Coquand. *Type Theory*. Ed. by E. N. Zalta. `https://plato.stanford.edu/archives/fall2018/entries/type-theory/`. 2006.

[Cover et al. 05]
T. M. Cover and J. A. Thomas. *Elements of Information Theory*. John Wiley & Sons, Ltd, 2005. doi: 10.1002/047174882X.

[Cox 99]
R. Cox. 'Representation construction, externalised cognition and individual differences.' In: *Learning and Instruction* 9.4 (1999), pp. 343–363. doi: 10.1016/S0959-4752(98)00051-6.

[Dietterich et al. 95]
T. Dietterich and E. B. Kong. *Machine Learning Bias, Statistical Bias, and Statistical Variance of Decision Tree Algorithms*. Tech. rep. Oregon State University, 1995.

[Dolnick 89]
E. Dolnick. 'Panda paradox.' In: *Discover Magazine* (Sept. 1989), pp. 71–72.

[Falkenhainer et al. 89]
B. Falkenhainer, K. D. Forbus and D. Gentner. 'The structure-mapping engine: Algorithm and examples.' In: *Artificial Intelligence* 41.1 (1989), pp. 1–63. doi: 10.1016/0004-3702(89)90077-5.

[Gentner 83]
D. Gentner. 'Structure-mapping: A theoretical framework for analogy.' In: *Cognitive Science* 7.2 (1983), pp. 155–170. doi: 10.1016/S0364-0213(83)80009-3.

[Gentner 02]
D. Gentner. 'Analogy in Scientific Discovery: The Case of Johannes Kepler.' In: *Model-Based Reasoning: Science, Technology, Values*. Ed. by L. Magnani and N. J. Nersessian. Springer, 2002, pp. 21–39. doi: 10.1007/978-1-4615-0605-8_2.

[Giardino 13]
V. Giardino. 'Towards a diagrammatic classification.' In: *The Knowledge Engineering Review* 28.3 (2013), pp. 237–248. doi: 10.1017/S0269888913000222.

[Gordon et al. 79]
M. J. Gordon, A. J. Milner and C. P. Wadsworth. *Edinburgh LCF: A Mechanised Logic of Computation*. 1st ed. Vol. 78. Lecture Notes in Computer Science. Springer, 1979. doi: 10.1007/3-540-09724-4.

[Grawemeyer 06]
B. Grawemeyer. 'Evaluation of ERST – An External Representation Selection Tutor.' In: *Diagrammatic Representation and Inference, Diagrams 2006*. Ed. by D. Barker-Plummer, R. Cox and N. Swoboda.

Lecture Notes in Computer Science. Springer, 2006, pp. 154–167. DOI: 10.1007/11783183_21.

[Green et al. 98]
T. Green and A. Blackwell. 'Cognitive dimensions of information artefacts: a tutorial.' In: *BCS HCI Conference*. Vol. 98. 1998.

[Harrison 09]
J. Harrison. 'HOL Light: An Overview.' In: *Proceedings of the 22nd International Conference on Theorem Proving in Higher Order Logics, TPHOLs 2009*. Ed. by S. Berghofer, T. Nipkow, C. Urban and M. Wenzel. Vol. 5674. Lecture Notes in Computer Science. Springer, 2009, pp. 60–66. DOI: 10.1007/978-3-642-03359-9_4.

[Huet et al. 04]
G. Huet, G. Kahn and C. Paulin-Mohring. 'The Coq Proof Assistant: A Tutorial.' In: *The Coq Project* (2004).

[Inglis et al. 09]
M. Inglis and J. P. Meji□a-Ramos. 'On the Persuasiveness of Visual Arguments in Mathematics.' In: *Foundations of Science* 14.1 (Mar. 2009), pp. 97–110. DOI: 10.1007/s10699-008-9149-4.

[Jamnik et al. 99]
M. Jamnik, A. Bundy and I. Green. 'On Automating Diagrammatic Proofs of Arithmetic Arguments.' In: *Journal of Logic, Language and Information* 8.3 (1999), pp. 297–321. DOI: 10.1023/A:1008323427489.

[John et al. 96]
B. E. John and D. E. Kieras. 'The GOMS Family of User Interface Analysis Techniques: Comparison and Contrast.' In: *ACM Trans. Comput.-Hum. Interact.* 3.4 (1996), pp. 320–351. DOI: 10.1145/235833.236054.

[Johnson-Laird 06]
P. N. Johnson-Laird. 'Models and heterogeneous reasoning.' In: *Journal of Experimental & Theoretical Artificial Intelligence* 18.2 (2006), pp. 121–148.

[Karp 72]
R. M. Karp. 'Reducibility among Combinatorial Problems.' In: *Complexity of Computer Computations*. Ed. by R. E. Miller, J. W. Thatcher and J. D. Bohlinger. Springer, 1972, pp. 85–103. DOI: 10.1007/978-1-4684-2001-2_9.

[Kendall 38]
M. G. Kendall. 'A new measure of rank correlation.' In: *Biometrika* 30.1-2 (1938), pp. 81–93. DOI: 10.1093/biomet/30.1-2.81.

[Koedinger et al. 90]
K. R. Koedinger and J. R. Anderson. 'Abstract planning and perceptual chunks: Elements of expertise in geometry.' In: *Cognitive Science* 14.4 (1990), pp. 511–550. doi: 10.1016/0364-0213(90)90008-K.

[Kotovsky et al. 85]
K. Kotovsky, J. Hayes and H. Simon. 'Why are some problems hard? Evidence from Tower of Hanoi.' In: *Cognitive Psychology* 17.2 (1985), pp. 248–294. doi: 10.1016/0010-0285(85)90009-X.

[Kuhn et al. 05]
F. Kuhn, P. von Rickenbach, R. Wattenhofer, E. Welzl and A. Zollinger. 'Interference in Cellular Networks: The Minimum Membership Set Cover Problem.' In: *Computing and Combinatorics*. Ed. by L. Wang. Springer, 2005, pp. 188–198. doi: 10.1007/11533719_21.

[Kullback et al. 51]
S. Kullback and R. A. Leibler. 'On Information and Sufficiency.' In: *The Annals of Mathematical Statistics* 22.1 (1951), pp. 79–86. doi: 10.1214/aoms/1177729694.

[Larkin et al. 80]
J. H. Larkin, J. McDermott, D. P. Simon and H. A. Simon. 'Models of competence in solving physics problems.' In: *Cognitive Science* 4.4 (1980), pp. 317–345. doi: 10.1016/S0364-0213(80)80008-5.

[Larkin et al. 87]
J. H. Larkin and H. A. Simon. 'Why a Diagram is (Sometimes) Worth Ten Thousand Words.' In: *Cognitive Science* 11.1 (1987), pp. 65–100. doi: 10.1111/j.1551-6708.1987.tb00863.x.

[Lidwell et al. 07]
W. Lidwell, K. Holden and J. Butler. *Universal Principles of Design: 100 Ways to Enhance Usability, Influence Perception, Increase Appeal, Make Better Design Decisions, and Teach Through Design*. 1st ed. Rockport Publishers Inc, Mar. 2007.

[Melis et al. 99]
E. Melis and J. Whittle. 'Analogy in Inductive Theorem Proving.' In: *Journal of Automated Reasoning* 22.2 (1999), pp. 117–147. doi: 10.1023/A:1005936130801.

[Miller 56]
G. A. Miller. 'The magical number seven, plus or minus two: some limits on our capacity for processing information.' In: *Psychological Review* 63.2 (1956), pp. 81–97. doi: 10.1037/h0043158.

[Minsky 11]
Y. Minsky. *Effective ML revisited*. Blog Post (accessed 28 Nov. 2020). `https://blog.janestreet.com/effective-ml-revisited/`. 2011.

[Moody 09]
D. Moody. 'The "Physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering.' In: *IEEE Transactions on Software Engineering* 35.6 (2009), pp. 756–779. doi: `10.1109/TSE.2009.67`.

[Mossakowski et al. 07]
T. Mossakowski, C. Maeder and K. Lüttich. 'The Heterogeneous Tool Set, Hets.' In: *Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2007*. Ed. by O. Grumberg and M. Huth. Vol. 4424. Lecture Notes in Computer Science. Springer, 2007, pp. 519–522. doi: `10.1007/978-3-540-71209-1_40`.

[Newell et al. 59]
A. Newell, J. Shaw and H. A. Simon. 'Report on a General Problem-Solving Program.' In: *Proceedings of the International Conference on Information Processing*. 1959, pp. 256–264.

[Oldford et al. 06]
R. W. Oldford and W. H. Cherry. *Picturing Probability: the poverty of Venn diagrams, the richness of Eikosograms*. Retrieved from `http://www.stats.uwaterloo.ca/~rwoldfor/papers/venn/eikosograms/paperpdf.pdf`. 2006.

[Paulson 89]
L. C. Paulson. 'The foundation of a generic theorem prover.' In: *Journal of Automated Reasoning* 5.3 (1989), pp. 363–397. doi: `10.1007/BF00248324`.

[Paulson et al. 10]
L. C. Paulson and C. Blanchette. 'Three Years of Experience with Sledgehammer, a Practical Link between Automatic and Interactive Theorem Provers.' In: *The 8th International Workshop on the Implementation of Logics, IWIL 2010*. Ed. by G. Sutcliffe, S. Schulz and E. Ternovska. Vol. 2. 2010, pp. 1–11. doi: `10.29007/36dt`.

[Peng et al. 05]
H. Peng, F. Long and C. Ding. 'Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy.' In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 27.8 (2005), pp. 1226–1238. doi: `10.1109/TPAMI.2005.159`.

[Piggott 08]
J. Piggott. *Rich Tasks and Contexts*. Online article (accessed 6 Dec. 2020). `https://nrich.maths.org/5662`. 2008.

[Pólya 57]
G. Pólya. *How to Solve It: A New Aspect of Mathematical Method*. Princeton Science Library. Princeton University Press, 1957.

[Raggi et al. 16]
D. Raggi, A. Bundy, G. Grov and A. Pease. 'Automating Change of Representation for Proofs in Discrete Mathematics (Extended Version)'. In: *Mathematics in Computer Science* 10.4 (2016), pp. 429–457. doi: 10.1007/s11786-016-0275-z.

[Raggi et al. 20-a]
D. Raggi, G. Stapleton, A. Stockdill, M. Jamnik, G. Garcia Garcia and P. C.-H. Cheng. 'How to (Re)represent it?' In: *IEEE 32nd International Conference on Tools with Artificial Intelligence, ICTAI 2020*. Ed. by M. Alamaniotis and S. Pan. 2020, pp. 1224–1232. doi: 10.1109/ICTAI50040.2020.00185.

[Raggi et al. 19]
D. Raggi, A. Stockdill, M. Jamnik, G. Garcia Garcia, H. E. A. Sutherland and P. C.-H. Cheng. 'Inspection and Selection of Representations'. In: *Intelligent Computer Mathematics, CICM 2019*. Ed. by C. Kaliszyk, E. Brady, A. Kohlhase and C. Sacerdoti Coen. Springer, 2019, pp. 227–242. doi: 10.1007/978-3-030-23250-4_16.

[Raggi et al. 20-b]
D. Raggi, A. Stockdill, M. Jamnik, G. Garcia Garcia, H. E. A. Sutherland and P. C.-H. Cheng. 'Dissecting Representations'. In: *Diagrammatic Representation and Inference, Diagrams 2020*. Ed. by A.-V. Pietarinen, P. Chapman, L. Bosveld-de Smet, V. Giardino, J. Corter and S. Linker. Vol. 12169. Lecture Notes in Computer Science. Springer, 2020, pp. 144–152. doi: 10.1007/978-3-030-54249-8_11.

[Sampson 85]
G. Sampson. *Writing Systems: a Linguistic Introduction*. Stanford University Press, 1985.

[Scaife et al. 96]
M. Scaife and Y. Rogers. 'External cognition: how do graphical representations work?' In: *International Journal of Human-Computer Studies* 45.2 (1996), pp. 185–213. doi: 10.1006/ijhc.1996.0048.

[Shannon 48]
C. E. Shannon. 'A mathematical theory of communication'. In: *The Bell System Technical Journal* 27.3 (1948), pp. 379–423. doi: 10.1002/j.1538-7305.1948.tb01338.x.

[Shimojima 96]
A. Shimojima. 'Operational Constraints in Diagrammatic Reason-

ing'. In: *Logical Reasoning with Diagrams*. Ed. by G. Allwein and J. Barwise. Oxford University Press, 1996. Chap. 2, pp. 27–48.

[Shimojima 01]
A. Shimojima. 'The Graphic-Linguistic Distinction'. In: *Thinking with Diagrams*. Ed. by A. F. Blackwell. Springer, 2001, pp. 5–27. doi: 10.1007/978-94-017-3524-7_2.

[Simon et al. 78]
D. P. Simon and H. A. Simon. 'Individual differences in solving physics problems'. In: *Children's thinking: What develops?* Lawrence Erlbaum Associates, Inc, 1978, pp. 325–348.

[Simon et al. 71]
H. A. Simon and A. Newell. 'Human problem solving: The state of the theory in 1970'. In: *American Psychologist* 26.2 (1971), pp. 145–159.

[Slind et al. 08]
K. Slind and M. Norrish. 'A Brief Overview of HOL4'. In: *Proceedings of the 21st International Conference on Theorem Proving in Higher Order Logics, TPHOLs '08*. TPHOLs '08. Montreal, P.Q., Canada: Springer, 2008, pp. 28–32. doi: 10.1007/978-3-540-71067-7_6.

[Sloman 75]
A. Sloman. 'Afterthoughts on Analogical Representations'. In: *Workshop on Theoretical Issues in Natural Language Processing, TINLAP 1975*. Association for Computational Linguistics, 1975, pp. 164–168. doi: 10.3115/980190.980235.

[Someren et al. 98]
M. W. van Someren, P. Reimann, H. P. A. Boshuizen and T. de Jong, eds. *Learning with Multiple Representations*. Advances in Learning and Instruction Series. ERIC, 1998.

[Stapleton et al. 04]
G. Stapleton, J. Howse, J. Taylor and S. Thompson. 'What Can Spider Diagrams Say?' In: *Diagrammatic Representation and Inference, Diagrams 2004*. Ed. by A. F. Blackwell, K. Marriott and A. Shimojima. Lecture Notes in Computer Science. Springer, 2004, pp. 112–127. doi: 10.1007/978-3-540-25931-2_12.

[Stapleton et al. 17]
G. Stapleton, M. Jamnik and A. Shimojima. 'What Makes an Effective Representation of Information: A Formal Account of Observational Advantages'. In: *Journal of Logic, Language and Information* 26.2 (2017), pp. 143–177. doi: 10.1007/s10849-017-9250-6.

[Stapleton et al. 07]
G. Stapleton, J. Masthoff, J. Flower, A. Fish and J. South-

ERN. 'Automated Theorem Proving in Euler Diagram Systems.' In: *Journal of Automated Reasoning* 39.4 (2007), pp. 431–470. DOI: `10.1007/s10817-007-9069-y`.

[Stenning et al. 01]
K. STENNING and O. LEMON. 'Aligning Logical and Psychological Perspectives on Diagrammatic Reasoning.' In: *Artificial Intelligence Review* 15.1 (2001), pp. 29–62. DOI: `10.1023/A:1006617525134`.

[Stenning et al. 95]
K. STENNING and J. OBERLANDER. 'A Cognitive Theory of Graphical and Linguistic Reasoning: Logic and Implementation.' In: *Cognitive Science* 19.1 (1995), pp. 97–140. DOI: `10.1207/s15516709cog1901_3`.

[Stockdill et al. 21]
A. STOCKDILL, D. RAGGI, M. JAMNIK, G. GARCIA GARCIA and P. C.-H. CHENG. 'Considerations in Representation Selection for Problem Solving: A Review.' In: *Diagrammatic Representation and Inference, Diagrams 2021*. Ed. by A. BASU, G. STAPLETON, S. LINKER, C. LEGG, E. MANALO and P. VIANA. Vol. 12909. Lecture Notes in Computer Science. Springer, 2021, pp. 1–17. DOI: `10.1007/978-3-030-86062-2_4`.

[Stockdill et al. 20-A]
A. STOCKDILL, D. RAGGI, M. JAMNIK, G. GARCIA GARCIA, H. E. A. SUTHERLAND, P. C.-H. CHENG and A. SARKAR. 'Correspondence-based analogies for choosing problem representations.' In: *IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2020*. Ed. by M. HOMER, F. HERMANS, S. TANIMOTO and C. ANSLOW. 2020, pp. 1–5. DOI: `10.1109/VL/HCC50065.2020.9127258`.

[Stockdill et al. 20-B]
A. STOCKDILL, D. RAGGI, M. JAMNIK, G. GARCIA GARCIA, H. E. A. SUTHERLAND, P. C.-H. CHENG and A. SARKAR. 'Cross-domain correspondences for explainable recommendations.' In: *Proceedings of the Workshop on Explainable Smart Systems for Algorithmic Transparency in Emerging Technologies (ExSS-ATEC)*. Ed. by A. SMITH-RENNER, S. KLEANTHOUS, B. LIM, T. KUFLIK, S. STUMPF, J. OTTERBACHER, A. SARKAR, C. DUGAN and A. SHULNER. Vol. 2582. CEUR Workshop Proceedings. CEUR-WS.org, 2020.

[Stylianou 02]
D. A. STYLIANOU. 'On the interaction of visualization and analysis: the negotiation of a visual representation in expert problem solving.' In: *The Journal of Mathematical Behavior* 21.3 (2002), pp. 303–317. DOI: `10.1016/S0732-3123(02)00131-1`.

[Superfine et al. 09]
A. C. SUPERFINE, R. S. CANTY and A. M. MARSHALL. 'Translation

between external representation systems in mathematics: All-or-none or skill conglomerate?' In: *The Journal of Mathematical Behavior* 28.4 (2009), pp. 217–236. DOI: 10.1016/j.jmathb.2009.10.002.

[Sweller 88]
J. SWELLER. 'Cognitive Load During Problem Solving: Effects on Learning.' In: *Cognitive Science* 12.2 (1988), pp. 257–285. DOI: 10.1207/s15516709cog1202_4.

[Thagard 92]
P. THAGARD. 'Analogy, explanation, and education.' In: *Journal of Research in Science Teaching* 29.6 (1992), pp. 537–544. DOI: 10.1002/tea.3660290603.

[Turing 50]
A. M. TURING. 'Computing Machinery and Intelligence.' In: *Mind* LIX.236 (Oct. 1950), pp. 433–460. DOI: 10.1093/mind/LIX.236.433.

[Tversky 11]
B. TVERSKY. 'Visualizing Thought.' In: *topiCS* 3.3 (2011), pp. 499–535. DOI: 10.1111/j.1756-8765.2010.01113.x.

[Uesaka et al. 10]
Y. UESAKA, E. MANALO and S. ICHIKAWA. 'The Effects of Perception of Efficacy and Diagram Construction Skills on Students' Spontaneous Use of Diagrams When Solving Math Word Problems.' In: *Diagrammatic Representation and Inference, Diagrams 2010*. Ed. by A. K. GOEL, M. JAMNIK and N. H. NARAYANAN. Lecture Notes in Computer Science. Springer, 2010, pp. 197–211. DOI: 10.1007/978-3-642-14600-8_19.

[Urbas et al. 14]
M. URBAS and M. JAMNIK. 'A Framework for Heterogeneous Reasoning in Formal and Informal Domains.' In: *Diagrammatic Representation and Inference, Diagrams 2014*. Ed. by T. DWYER, H. PURCHASE and A. DELANEY. Vol. 8578. Lecture Notes in Computer Science. Springer, 2014, pp. 277–292. DOI: 10.1007/978-3-662-44043-8_28.

[Urbas et al. 12]
M. URBAS, M. JAMNIK, G. STAPLETON and J. FLOWER. 'Speedith: A Diagrammatic Reasoner for Spider Diagrams.' In: *Diagrammatic Representation and Inference, Diagrams 2012*. Ed. by P. COX, B. PLIMMER and P. RODGERS. Lecture Notes in Computer Science. Springer, 2012, pp. 163–177. DOI: 10.1007/978-3-642-31223-6_19.

[Vessey 91]
I. VESSEY. 'Cognitive Fit: A Theory-Based Analysis of the Graphs Versus Tables Literature.' In: *Decision Sciences* 22.2 (1991), pp. 219–240. DOI: 10.1111/j.1540-5915.1991.tb00344.x.

[Weisberg et al. 73]
R. Weisberg and J. M. Suls. ‘An information-processing model of Duncker’s Candle Problem.’ In: *Cognitive Psychology* 4.2 (1973), pp. 255–276. doi: `10.1016/0010-0285(73)90014-5`.

[Zazkis et al. 16]
D. Zazkis, K. Weber and J. P. Meji⍰a-Ramos. ‘Bridging the gap between graphical arguments and verbal-symbolic proofs in a real analysis context.’ In: *Educational Studies in Mathematics* 93.2 (2016), pp. 155–173. doi: `10.1007/s10649-016-9698-3`.

[Zhang 97]
J. Zhang. ‘The Nature of External Representations in Problem Solving.’ In: *Cognitive Science* 21.2 (Apr. 1997), pp. 179–217. doi: `10.1207/s15516709cog2102_3`.

[Zhang et al. 95]
J. Zhang and D. A. Norman. ‘A representational analysis of numeration systems.’ In: *Cognition* 57.3 (1995), pp. 271–295. doi: `10.1016/0010-0277(95)00674-3`.

Appendices

Component formula probability A

The work throughout this dissertation assumes the semantics of event operations ($\cap$, $\cup$, and complement) align with the semantics of component operations (AND, OR, and NOT). The purpose of this appendix is to demonstrate the validity of this assumption.

We assume that an RS-description R is a finite set of components, and that an R-description r is a finite set that contains some of those components in the RS-description. That is, $r \subseteq R$.[1] We can model r as a tuple t of length $|R|$ consisting of Booleans such that the ith Boolean is `true` if the ith component in R (for some arbitrary but fixed ordering of R) is an element of the R-description r; otherwise it is `false`. Thus the probability of some jth component $c_j \in R$ occurring in some R-description r is equivalent to the probability of the jth Boolean in t being `true`. Each of these Booleans is modelled by a Bernoulli random variable, and so $\Pr(t_j = \mathtt{true}) = p$ and $\Pr(t_j = \mathtt{false}) = 1 - p$.

Example A.1. Consider some RS-description $R = \{x, y, z\}$. A specific R-description $r = \{x, z\}$ can be modelled by the tuple

$$t = (\mathtt{true}, \mathtt{false}, \mathtt{true}).$$

Each index in the tuple has a specific probability of being `true` or `false`, which we model as the probability of the associated component occurring in some R-description. For example, $\Pr(x) = \Pr(t_1 = \mathtt{true})$. $\triangledown$

This works at the level of a single R-description, but we are defining probabilities at the level of RS-descriptions. So we lift our model from a single tuple t of an R-description r to become a set of tuples T to model the RS-description R, meaning that an R-description is part of an RS-description if $t \in T$. The probability of a component is thus defined over the set of tuples that mark that component as `true`:

$$\Pr(c_i) = \frac{|\{t \in T \mid t_i = \mathtt{true}\}|}{|T|}.$$

This also means the conditional probability of components across representational systems is defined quite simply:

$$\Pr(c_j^R \mid c_k^S) = \frac{\sum_q \left|\left\{t^R \in T_q^R \mid t_j^R = \mathtt{true} \land t^S \in T_q^S \land t_k^S = \mathtt{true}\right\}\right|}{\sum_q \left|\left\{t^S \in T_q^S \mid t_k^S = \mathtt{true}\right\}\right|}$$

where T_q^R is the subset of T^R (the set of tuples modelling RS-description R) that are descriptions of some problem q. This definition has clear similarities to Equation 4.1, page 75.

[1] We are ignoring the attribute 'occurrences', assuming that it is not 0, as it has no bearing on the presence or absence of a component.

Now consider how the operations AND, OR, and NOT behave. Let us begin with NOT, because it is simplest. The semantics of NOT are such that for NOT x, if the component x is satisfied by R-description r, then the formula is *not* satisfied; conversely, if the component x is *not* satisfied by r, then the component formula *is* satisfied. In our tuple model, we are validating that the tuple has a `false` in the relevant index. Lifting this to RS-descriptions, where before we focused on the set of tuples where t_x was `true`, we are now interested in the set of tuples where t_x is `false`. That is,

$$\{t \in T \mid t_x = \mathtt{false}\} = T \setminus \{t \in T \mid t_x = \mathtt{true}\},$$

which is the set complement under universe T. Thus, the NOT operator is modelled as set complement.

The semantics of AND are such that for x AND y, if both of component x and component y are satisfied by R-description r, then the formula is satisfied. In our tuple model, we are checking that both of the Booleans associated with x and y are `true`. Lifting this to RS-descriptions, we need a set of tuples where all of the Booleans at the relevant indices are `true`: this is the *intersection* of the sets of tuples where in one the x-index Boolean is `true` and in the other the y-index Boolean is `true`. Thus, when we write x AND y, we are referring to the intersection

$$\begin{aligned} &\{t \in T \mid t_x = \mathtt{true} \wedge t_y = \mathtt{true}\} \\ =&\{t \in T \mid t_x = \mathtt{true}\} \cap \{t \in T \mid t_y = \mathtt{true}\} \end{aligned}$$

and so we can say that AND is modelled by $\cap$.

Finally, the semantics of OR are such that for x OR y, if either of x or y are satisfied by R-description r, then the formula is satisfied. Using our tuples, we are checking that at least one of the Booleans in the x index or y index is `true`. Lifting this to the level of RS-descriptions, we are interested in the set of tuples where one or both of the relevant indices are `true`: this is the *union* of the set of tuples where the x index is `true` and the set of tuples where the y index is `true`. That is,

$$\begin{aligned} &\{t \in T \mid t_x = \mathtt{true} \vee t_y = \mathtt{true}\} \\ =&\{t \in T \mid t_x = \mathtt{true}\} \cup \{t \in T \mid t_y = \mathtt{true}\} \end{aligned}$$

and so we can say that OR is modelled by $\cup$.

Example A.2. Let $R = \{x, y, z\}$, and T^R be the set

$$\begin{aligned} T^R = \{&(\mathtt{false}, \mathtt{false}, \mathtt{true}) \\ &(\mathtt{false}, \mathtt{true}, \mathtt{false}) \\ &(\mathtt{false}, \mathtt{true}, \mathtt{true}) \\ &(\mathtt{true}, \mathtt{false}, \mathtt{false}) \\ &(\mathtt{true}, \mathtt{false}, \mathtt{true}) \\ &(\mathtt{true}, \mathtt{true}, \mathtt{true})\} \end{aligned}$$

that models our RS-description R.[2] The 'arbitrary order' that the tuples use to index is that given in our declaration of R. The model of x is the set of all tuples from T^R where the first Boolean is `true`:

$$\begin{aligned} x \equiv \{&(\texttt{true}, \texttt{false}, \texttt{false}) \\ &(\texttt{true}, \texttt{false}, \texttt{true}) \\ &(\texttt{true}, \texttt{true}, \texttt{true})\} \end{aligned}$$

And so $\Pr(x) = 3/6 = 0.5$.

If we consider $\textsc{not}\, z$, this is the set of all tuples from T^R where the third Boolean is `false`:

$$\begin{aligned} \textsc{not}\, z \equiv \{&(\texttt{false}, \texttt{true}, \texttt{false}) \\ &(\texttt{true}, \texttt{false}, \texttt{false})\} \end{aligned}$$

and so $\Pr(\textsc{not}\, z) = 2/6 = 0.\dot{3} = 1 - \Pr(z)$.

Finally, a more complex component formula $x\,\textsc{and}\,(y\,\textsc{or}\,z)$. Modelled as tuples, we first find the set modelling y or z. This is the union of the sets for y and z, or equivalently the set where at least one of the second or third Boolean in the tuple is `true`:

$$\begin{aligned} y \;\textsc{or}\; z \equiv \{&(\texttt{false}, \texttt{false}, \texttt{true}) \\ &(\texttt{false}, \texttt{true}, \texttt{false}) \\ &(\texttt{false}, \texttt{true}, \texttt{true}) \\ &(\texttt{true}, \texttt{false}, \texttt{true}) \\ &(\texttt{true}, \texttt{true}, \texttt{true})\} \end{aligned}$$

This is then intersected with the set for a found earlier:

$$\begin{aligned} x \;\textsc{and}\; (y \;\textsc{or}\; z) \equiv \{&(\texttt{true}, \texttt{false}, \texttt{true}) \\ &(\texttt{true}, \texttt{true}, \texttt{true})\} \end{aligned}$$

Thus $\Pr(x \;\textsc{and}\; (y \;\textsc{or}\; z)) = 2/6 = 0.\dot{3}$. ▽

Much of this complexity is not necessary in practice, as AND, OR, and NOT behave as expected.

[2] This is *not* all possible Boolean tuples; some R-descriptions do not describe any representation, so are not included.

B Composition strength

Here we provide a complete proof of Lemma 3, Section 4.3.3, page 87: the strength of composed correspondence $\langle\, a,\ c,\ s_{a\to c}\,\rangle$ is the product of the strengths of $\langle\, a,\ b,\ s_{a\to b}\,\rangle$ and $\langle\, b,\ c,\ s_{b\to c}\,\rangle$. That is, $s_{a\to c} = s_{a\to b} \cdot s_{b\to c}$. We use the notation $s_{x\to y}$ to denote the strength of the correspondence from x to y.

Before the main proof, we prove a small lemma.

Lemma 6.

$$\Pr(x \mid \textsc{not}\, y) = \frac{\Pr(x) - \Pr(x \mid y) \cdot \Pr(y)}{1 - \Pr(y)}$$

Proof. Using the definition of conditional probability, and Bayes' theorem for the final equality:

$$\begin{aligned}
\Pr(x \mid \textsc{not}\, y) &= \frac{\Pr(x \textsc{ and not}\, y)}{\Pr(\textsc{not}\, y)} \\
&= \frac{\Pr(\textsc{not}\, y \textsc{ and } x)}{1 - \Pr(y)} \\
&= \frac{\Pr(\textsc{not}\, y \mid x) \cdot \Pr(x)}{1 - \Pr(y)} \\
&= \frac{(1 - \Pr(y \mid x)) \cdot \Pr(x)}{1 - \Pr(y)} \\
&= \frac{\Pr(x) - \Pr(y \mid x) \cdot \Pr(x)}{1 - \Pr(y)} \\
&= \frac{\Pr(x) - \Pr(x \mid y) \cdot \Pr(y)}{1 - \Pr(y)}
\end{aligned}$$

as required. □

With that out of the way, we move on to the main proof.

Proof. In Lemma 3, we assume that the probability that a and c are satisfied in the respective descriptions is independent given that b is satisfied in its description. That is,

$$\Pr(a \textsc{ and } c \mid b) = \Pr(a \mid b) \cdot \Pr(c \mid b).$$

By the definition of correspondence strength (Definition 13, Section

4.2.2, page 76) we have

$$\begin{aligned} s_{a\to b} &= \frac{\Pr(b \mid a) - \Pr(b)}{1 - \Pr(b)}, \\ s_{b\to c} &= \frac{\Pr(c \mid b) - \Pr(c)}{1 - \Pr(c)}, \\ \text{and } s_{a\to c} &= \frac{\Pr(c \mid a) - \Pr(c)}{1 - \Pr(c)}. \end{aligned}$$

We need to show that the product of the first two is equal to the third.

The third equation, the definition of $s_{a\to c}$, is defined using $\Pr(c \mid a)$, which we rewrite as $\Pr(c \text{ AND } a) / \Pr(a)$. The conjunction, c AND a, can be rewritten as the disjunction of disjoint component formulae

$$(a \text{ AND } b \text{ AND } c) \text{ OR } (a \text{ AND NOT } b \text{ AND } c).$$

Because they are disjoint,

$$\begin{aligned} &\Pr((a \text{ AND } b \text{ AND } c) \text{ OR } (a \text{ AND NOT } b \text{ AND } c)) \\ &= \Pr(a \text{ AND } b \text{ AND } c) + \Pr(a \text{ AND NOT } b \text{ AND } c). \end{aligned}$$

This first term can be rewritten (using the result of our independence assumption, and Bayes' Theorem) as

$$\begin{aligned} \Pr(a \text{ AND } b \text{ AND } c) &= \Pr(a \text{ AND } c \mid b) \cdot \Pr(b) \\ &= \Pr(a \mid b) \cdot \Pr(c \mid b) \cdot \Pr(b) \\ &= \Pr(c \mid b) \cdot \Pr(b \mid a) \cdot \Pr(a). \end{aligned}$$

By symmetry, we also have

$$\begin{aligned} &\Pr(a \text{ AND NOT } b \text{ AND } c) \\ &= \Pr(c \mid \text{NOT } b) \cdot \Pr(\text{NOT } b \mid a) \cdot \Pr(a) \\ &= \Pr(c \mid \text{NOT } b) \cdot (1 - \Pr(b \mid a)) \cdot \Pr(a) \\ &= \frac{\Pr(c) - \Pr(c \mid b) \cdot \Pr(b)}{1 - \Pr(b)} \cdot (1 - \Pr(b \mid a)) \cdot \Pr(a) \end{aligned}$$

using Lemma 6 for the third equality.

We now combine these results to compute $\Pr(c \mid a)$:

$$\begin{aligned}
&\Pr(c \mid a) \\
&= \frac{1}{\Pr(a)} \cdot \Pr(c \text{ AND } a) \\
&= \frac{1}{\Pr(a)} \cdot (\Pr(a \text{ AND } b \text{ AND } c) + \Pr(a \text{ AND NOT } b \text{ AND } c)) \\
&= \Pr(c \mid b) \cdot \Pr(b \mid a) + \frac{\Pr(c) - \Pr(c \mid b) \cdot \Pr(b)}{1 - \Pr(b)} \cdot (1 - \Pr(b \mid a)) \\
&= \frac{\Pr(c \mid b) \cdot \Pr(b \mid a) - \Pr(c \mid b) \cdot \Pr(b) \cdot \Pr(b \mid a)}{1 - \Pr(b)} \\
&\quad + \frac{\Pr(c) - \Pr(b \mid a) \cdot \Pr(c) - \Pr(c \mid b) \cdot \Pr(b)}{1 - \Pr(b)} \\
&\quad + \frac{\Pr(c \mid b) \cdot \Pr(b) \cdot \Pr(b \mid a)}{1 - \Pr(b)} \\
&= \frac{\Pr(c) - \Pr(b \mid a) \cdot \Pr(c) - \Pr(c \mid b) \cdot \Pr(b) + \Pr(c \mid b) \cdot \Pr(b \mid a)}{1 - \Pr(b)} \\
&= \frac{(\Pr(c \mid b) - \Pr(c))(\Pr(b \mid a) - \Pr(b)) - \Pr(c) \cdot \Pr(b) + \Pr(c)}{1 - \Pr(b)} \\
&= \frac{\Pr(b \mid a) - \Pr(b)}{1 - \Pr(b)} \cdot (\Pr(c \mid b) - \Pr(c)) + \frac{\Pr(c) - \Pr(c) \cdot \Pr(b)}{1 - \Pr(b)} \\
&= s_{a \to b} \cdot (\Pr(c \mid b) - \Pr(c)) + \Pr(c)
\end{aligned}$$

using the definition of $s_{a \to b}$.

We substitute this result into our definition of $s_{a \to c}$, yielding

$$\begin{aligned}
s_{a \to c} &= \frac{\Pr(c \mid a) - \Pr(c)}{1 - \Pr(c)} \\
&= \frac{s_{a \to b} \cdot (\Pr(c \mid b) - \Pr(c)) + \Pr(c) - \Pr(c)}{1 - \Pr(c)} \\
&= s_{a \to b} \cdot \frac{\Pr(c \mid b) - \Pr(c)}{1 - \Pr(c)} \\
&= s_{a \to b} \cdot s_{b \to c}
\end{aligned}$$

using the definition of $s_{b \to c}$, and thus completing our proof. □

C Minimally redundant set cover problem is NP hard

This appendix provides a complete proof of Theorem 5, Section 5.3.4, page 119: that constructing a minimally redundant covering set is NP hard. The proof is a (Turing) reduction from the 'minimum set cover' problem to minimally redundant set covering.

Before we proceed with the proof of Theorem 5, we need a lemma about preserving minimum set covers.

Lemma 7. *Let $\mathcal{U}$ be a set of elements to cover, and $\mathcal{D}$ be a set of 'dummy' elements disjoint from $\mathcal{U}$. Further, let $\mathcal{S}' \subseteq \mathcal{P}(\mathcal{U} \cup \mathcal{D})$ be the set of sets from which we can draw a cover, obeying the property that every $s \in \mathcal{S}'$ contains every element of $\mathcal{D}$. If $S' \subseteq \mathcal{S}'$ is a minimum set cover of $\mathcal{U} \cup \mathcal{D}$, then*

$$S = \{s \setminus \mathcal{D} \mid s \in S'\}$$

is a minimum set cover of $\mathcal{U}$ using sets drawn from $\mathcal{S} = \{s \setminus \mathcal{D} \mid s \in \mathcal{S}'\}$.

Proof. We proceed with proof by contrapositive. That is, assume that S is *not* a minimum set cover of $\mathcal{U}$. There are two reasons why S might not be a minimum set cover: first, it might not cover $\mathcal{U}$; second, it might not be a minimum cover.

In the first case, if S does not cover $\mathcal{U}$, then S' did not cover $\mathcal{U} \cup \mathcal{D}$, violating the assumption of the lemma.

In the second case, there must exist some S_1 that covers $\mathcal{U}$ but $|S_1| < |S|$. But then from S_1 we could construct

$$S'_1 = \{s \cup \mathcal{D} \mid s \in S_1\}$$

that covers $\mathcal{U} \cup \mathcal{D}$, but also $|S'_1| < |S'|$, meaning that S' was not a minimum set cover of $\mathcal{U} \cup \mathcal{D}$, violating the assumption of the lemma.

In either case, we contradict the assumptions of the lemma; by contraposition, we have proved the lemma. □

We may now proceed with the proof that constructing a minimally redundant set cover is NP hard. The proof technique was inspired by [Kuhn et al. 05].

Proof. Let $\mathcal{S} \subseteq \mathcal{P}(\mathcal{U})$ be a set of sets such that $\bigcup \mathcal{S} = \mathcal{U}$, where $\mathcal{U}$ is some 'universe' of elements we are covering. We wish to find $S \subseteq \mathcal{S}$, and that $\bigcup S = \mathcal{U}$ such that $|S|$ is minimum. This is a minimum set cover of $\mathcal{U}$, and we will show how to construct this cover.

Take set $\mathcal{D} = d_1, \ldots, d_n$ as a set of dummy elements where $|D| = |U| + 1$ and $\mathcal{D} \cap \mathcal{U} = \varnothing$. Construct the set $\mathcal{S}' = \{s \cup \mathcal{D} \mid s \in \mathcal{S}\}$, that is, we add all the dummy elements into every set in $\mathcal{S}$. Further, we define $\mathcal{U}' = \mathcal{U} \cup \mathcal{D}$. We shall use these dummy elements to count how many sets form the minimally redundant cover.

Assume we have S'_*, a minimally redundant cover from $\mathcal{S}'$ over $\mathcal{U}'$. That is, there is no cover S'_1 such that

$$\sum_{e \in \biguplus S'_1} (\#e - 1) < \sum_{e \in \biguplus S'_*} (\#e - 1).$$

We must now recover S_*, the minimum set cover of $\mathcal{U}$, from S'_*.

By the construction of $\mathcal{S}'$, we know that it is possible to cover $\mathcal{U}'$, so S'_* covers $\mathcal{U}'$. We also know that every $s \in S'_*$ contains every element in $\mathcal{D}$; that is, $\mathcal{D} \subseteq s$. Then the minimum redundancy of any possible cover is *at least* $|\mathcal{D}| \times (|S'_*| - 1)$. We now demonstrate that this implies that

$$S_* = \{s \setminus \mathcal{D} \mid s \in S'_*\}$$

is a minimum set cover of $\mathcal{U}$.

The redundancy of the elements in $\mathcal{D}$ (the 'dummy redundancy') dominates the redundancy of the elements in $\mathcal{U}$. The *most* redundant possible cover of $\mathcal{U}$ would be

$$S = \{\mathcal{U} \setminus \{u\} \mid u \in \mathcal{U}\}$$

with a redundancy of

$$\sum_{u \in \mathcal{U}} (|\mathcal{S}| - 1) - 1 = |\mathcal{U}|^2 - 2|\mathcal{U}|. \qquad (*)$$

The dummy redundancy of the same set would be

$$\begin{aligned} |\mathcal{D}| \cdot (|S| - 1) &= (|\mathcal{U}| + 1) \times (|\mathcal{U}| - 1) \\ &= |\mathcal{U}|^2 - 1 \end{aligned}$$

which is always greater than (*) for nonempty $\mathcal{U}$. Thus, for *any* cover of $\mathcal{U}'$, the dummy redundancy will be greater than the redundancy of elements in $\mathcal{U}$. So the least redundant cover will have the least dummy redundancy.

Every set in $\mathcal{S}'$ increases the dummy redundancy by the same amount. Thus, for S'_* to have the least redundancy, and so the least dummy redundancy, it must also have the least *number* of sets. That is, S'_* is not only a minimally redundant set cover, it is also a minimum set cover of $\mathcal{U}'$. From each $s \in S'_*$ we discard the elements of $\mathcal{D}$ to construct S_* as above; by Lemma 7, this new S_* is a minimum set cover of $\mathcal{U}$.

So we have constructed a minimum set cover using a minimally redundant set cover, and thus shown that minimally redundant set cover construction is at least as hard as constructing a minimum set cover. As the latter is NP hard, so must the former be NP hard. The construction of S_* from S'_* operates within polynomial time, $O(|S'_*| \times |\mathcal{U}|)$, assuming $O(1)$ addition and deletion on a set. □

Generalising correspondences D

We defined the correspondence framework from this dissertation in the context of components and supporting mathematical problem solving, but there is no inherent limitation on correspondences that restrict them to this context. In this appendix, we consider what it would mean to apply correspondences outside the rep2rep component framework.

The generalisation we present here relies on two definitions: an inner *system* structure, and an outer *cross-system* structure.[1] The system is the structure from which *instances* emerge: in our case, this is R-description instances from an RS-description system. Each system must have atoms, relations, and a probability function on the atoms; the cross-system structure must consist of systems, two equivalence relations (one between the atoms, and the other on the relations of the systems), and a conditional probability function across the systems. Most of these requirements are visualised in Figure D.1. Let us see how these requirements came to be, and why these requirements exist.

D.1 System structure

Let us examine an RS-description, which is the 'original' system which correspondences are defined between. We see they have components, and these components consist of kinds, values, and attributes. The attributes can link components together. Further, we have a probability function on the components, which returns the probability of a component appearing in a representation. At such an abstract level, RS-descriptions are simple. But from the perspective of correspondences, we can abstract this further.

Components are not directly used by correspondences: the fact they are kind-value-attribute triples is not considered. So the first abstraction is to replace components with *atoms*.[2] Thus the set of components within an RS-description is a set A of atoms in the system. However, by discarding components we did lose attributes—specifically, the relations induced by attributes. Thus we reclaim this inner structure of RS-descriptions by defining a set R of relations (of any arity) on the atoms A. Finally, we carry the probability function over in the expected way. We assume that 'component formulae' are also transferred to the atoms, such that $\mathcal{F}(A)$ is the set of formulae over A using the connectives AND, OR, and NOT.[3]

Using these abstractions, we can define a *system*.

Definition 23 (Correspondence systems)**.** A *system* is a triple $(A, \mathrm{R}, \mathrm{Pr})$

[1] We use structure with its most general meaning: a collection of objects that have particular properties.

[2] Atoms are indivisible things we can distinguish.

[3] See the end of Section 4.2.2 for details.

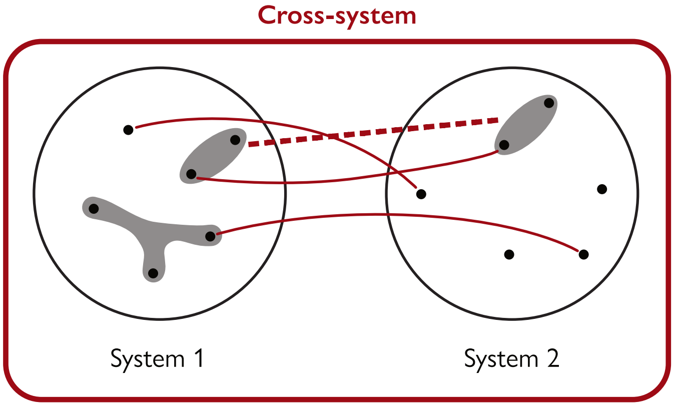


Figure D.1 The aspects of a correspondence structure, with system structures in black and the cross-system structure in red. The smallest black dots are the atoms, the grey regions are the relations, and the surrounding black circles denote the systems. The red region denotes the cross-system structure, the solid red lines denote the equivalence relation between atoms, and the dashed red line denotes an equivalence between the system relations. The probability functions are not depicted.

where A is a set of atoms, $R \subseteq \bigcup_{i=2}^{\infty} A^i$ is the union of relations (of any finite arity[4]) on the atoms, and $\Pr : \mathcal{F}(A) \to [0,1]$ is a probability function on combinations of atoms using AND, OR, and NOT connectives. These connectives are equivalent to those in Definition 11.

The atoms of A and relations of R are visible in Figure D.1 as black points and grey regions, respectively. Closely related, and for completeness, we define an instance as well.[5]

Definition 24 (System instance)**.** An *instance* I of system $s = (A, R, \Pr)$ is some subset of the atoms of s, that is $I \subseteq A$.

The structure of a system gives the necessary components for correspondences to act over. But in practice we find one extra 'soft' requirement produces better results: the system must be *composable*. By this we mean that a small number of atoms is sufficient to generate a large number of instances. In our representational system context, an instance is an R-description. We favour a small number of atoms, because in practice we find that the closer the number of atoms is to the number of instances, the less able correspondences are to exploit the structure of a system. This occurs because the number of special cases quickly dominates the number of generic rules that correspondences capture: if the number of atoms is similar to the number of valid instances, then many atoms only occur in a single instance, and so the correspondence required to cover that atom is only used for that instance.

Example D.1. Consider two possible knowledge bases: one of films (that is, a system whose instances describe films), and the other of recipes (a system whose instances describe recipes). If we consider the atoms that might be in the films knowledge base, we might find actors, titles, genres, and more. But the number of titles would be close to the number of

[4] Note that, for example, $A^3 = A \times A \times A$.

[5] Instances are not depicted in Figure D.1.

films—that is, every instance would have a semi-unique atom for the title. Conversely, the recipes knowledge base might have atoms such as the ingredients, or the equipment required. The set of ingredients is much smaller than the set of recipes,[6] with significant re-use between the recipes. ▽

There is no strict requirement for systems to be composable, but a smaller atom to instance ratio is better.

D.2 Cross-system structure

Now we have the system structure, we collect the systems into a set S and explore the structure we need *between* each system. In the context of representational systems, we need to be able to compute the 'cross-representational conditional probability' of components—that is, $\Pr(b \mid a)$ for components a and b—as part of the definition of correspondence strength. We also require a way to check if components occurred in more than one system for the [IDY] rule.

For the cross-system structure we thus need the set of systems S, as well as a relation $\equiv$, which is a relation on $A_i \times A_j$ for equivalent atoms across systems.[7] We also require $\Pr(\cdot \mid \cdot) : A_i \times A_j \to [0, 1]$ to be the cross-system conditional probability function. This is enough for correspondences generally, as well as the [IDY], [REV], and [CMP] rules. But we do not yet have enough structure for the [REL] rule: the contextual relations within each system are currently incompatible. We have two remedies: either we assume all R_i are the same R set of relations (as we do in the case of attributes on components); or we introduce *another* relation $\sim$ on $R_i \times R_j$ for equivalent *relations* between systems.

We can now define cross-system structures.

Definition 25 (Cross-system structure)**.** The cross-system structure of a set of systems S is the four-tuple $(S, \equiv, \sim, \Pr(\cdot \mid \cdot))$ where $\equiv$ is the relation of equivalent atoms between systems, $\sim$ is the relation of equivalent relations between systems, and $\Pr(\cdot \mid \cdot)$ is the cross-system conditional probability of atom formulae.

We see S, $\equiv$, and $\sim$ in Figure D.1 represented as the black enclosing curves inside the red curve, the solid red lines, and the dashed red line, respectively. This fully generalises the structures which correspondences can operate over.

[6] Assuming a particular level of abstraction! If we differentiate every variety of apple, we might have a problem.

[7] This can equally be a set of equivalence relations for all pairs of systems. For simplicity we assume they are all one overloaded relation. We make this simplification for all cross-system functions.

E

All code listings from the programming languages example

This appendix contains the code listings that are part of the extended example in Chapter 6. It is available online at `https://github.com/aaronstockdill/thesis-appendices/tree/master/code-listings`.

F

Program Q-descriptions

This appendix contains the Q-descriptions that are part of the extended example in Chapter 6. It is available online at `https://github.com/aaronstockdill/thesis-appendices/tree/master/program-q-descriptions`.

G

Programming language RS-descriptions

This appendix contains the RS-descriptions that are part of the extended example in Chapter 6. It is available online at `https://github.com/aaronstockdill/thesis-appendices/tree/master/language-rs-descriptions`.

H

Correspondences between programming languages

This appendix lists all the correspondences used to calculate the informational suitability of each programming language across all the algorithms, as part of the extended example in Chapter 6. Note that the scores in Table 6.1 are not directly calculable due to an undocumented feature in the code called *type correspondences*. This automatically deduces that there must be *something* in a representation that produces an expression of the right type, but does not assert what it must be. The feature is experimental, but useful, and is found here: `https://github.com/rep2rep/robin/blob/master/src/strategies/correspondences/list.sml#L47`.

This list also does not include the so-called 'import correspondences': a hack which associates 'import components' with components that will occur in the Q-descriptions. We do this because imports are still being implemented. An example 'import correspondence' would be

$$\langle\, \text{primitive}\ \texttt{0},\ \text{import}\ \texttt{RealNumerals},\ 1\,\rangle$$

which means that whenever a Q-description contains a `0` primitive component, an RS-description which 'imports' real numerals will correctly be awarded the correspondence. We are working to properly manage imports in a future version.

$$\langle\, \text{type}\ \alpha\ \texttt{pointer},\ \text{type}\ \alpha\ \texttt{vector}\ \textsc{or}\ \text{type}\ \texttt{string}\ \textsc{or}\ \text{type}\ \alpha\ \texttt{list},\ 0.7\,\rangle$$

$$\langle\, \text{type}\ \texttt{int}\ \textsc{or}\ \text{type}\ \texttt{float},\ \text{type}\ \texttt{number},\ 1\,\rangle$$

$$\langle\, \text{primitive}\ \texttt{NULL},\ \text{primitive}\ \texttt{\$nil},\ 0.5\,\rangle$$

$$\langle\, \text{pattern}\ \texttt{declareStruct},\ \text{primitive}\ \texttt{\$pair},\ 0.5\,\rangle$$

$$\begin{aligned}\langle\ & \text{primitive}\ \texttt{malloc}\ \textsc{or}\ \text{primitive}\ \texttt{calloc},\\ & \quad \text{primitive}\ \texttt{\$quote}\ \textsc{or}\ \text{primitive}\ \texttt{\$quasiquote},\ 0.2\ \rangle\end{aligned}$$

$$\langle\, \text{primitive}\ \texttt{++},\ \text{primitive}\ \texttt{+}\ \textsc{and}\ \text{primitive}\ \texttt{1},\ 0.6\,\rangle$$

$$\langle\, \text{primitive}\ \texttt{--},\ \text{primitive}\ \texttt{-}\ \textsc{and}\ \text{primitive}\ \texttt{1},\ 0.6\,\rangle$$

$$\begin{aligned}\langle\ & \text{pattern}\ \texttt{writeIndexInc},\\ & \quad \text{primitive}\ \texttt{append}\ \textsc{or}\ \text{primitive}\ \texttt{reverse}\ \textsc{or}\\ & \quad \text{primitive}\ \texttt{make-vector}\ \textsc{or}\ \text{primitive}\ \texttt{vector-unfold},\ 0.1\ \rangle\end{aligned}$$

$$\begin{aligned}\langle\ & \text{pattern}\ \texttt{index},\\ & \quad \text{primitive}\ \texttt{car}\ \textsc{or}\ \text{primitive}\ \texttt{cadr}\ \textsc{or}\ \text{primitive}\ \texttt{caddr}\ \textsc{or}\\ & \quad \text{primitive}\ \texttt{string-ref}\ \textsc{or}\\ & \quad \text{primitive}\ \texttt{vector-ref}\ \textsc{or}\ \text{primitive}\ \texttt{vector-set},\ 0.1\ \rangle\end{aligned}$$

⟨ pattern `index` AND primitive `$ptr-&`,
primitive `cdr` OR primitive `cddr`, 0.2 ⟩

⟨ pattern `funcdef`, primitive `define` OR primitive `lambda`, 1 ⟩

⟨ pattern `if`,
primitive `if` OR primitive `cond` OR
primitive `unless` OR primitive `when`, 1 ⟩

⟨ pattern `switch`, primitive `cond` AND primitive `eq?`, 1 ⟩

⟨ primitive `==`, primitive `eq?`, 1 ⟩

⟨ pattern `iddef`, primitive `let*`, 0.95 ⟩

⟨ primitive `==` AND primitive `NULL`, primitive `null?`, 0.5 ⟩

⟨ pattern `for` OR pattern `while`, pattern `recursion`, 1 ⟩

⟨ type α `pointer`,
type α `ref` OR type α `list` OR type α `array` OR type `string`, 0.7 ⟩

⟨ primitive `==`, primitive `$eq`, 1 ⟩

⟨ primitive `NULL`, primitive `$nil` OR primitive `NONE`, 0.5 ⟩

⟨ primitive `!=`, primitive `<>`, 1 ⟩

⟨ primitive `++`, primitive `+` AND primitive `1`, 0.6 ⟩

⟨ primitive `--`, primitive `-` AND primitive `1`, 0.6 ⟩

⟨ primitive `$ptr-&`, primitive `ref`, 0.1 ⟩

⟨ primitive `malloc` OR primitive `calloc`, type `array`, 0.2 ⟩

⟨ pattern `writeIndexInc`,
primitive `@` OR primitive `Array2_tabulate` OR
primitive `List_rev` OR primitive `String_implode`, 0.1 ⟩

⟨ pattern `index`,
primitive `Array2_sub` OR primitive `Array2_update` OR
primitive `String_sub`, 0.2 ⟩

⟨ pattern `if` AND (primitive `>` OR primitive `>=` OR
primitive `<` OR primitive `<=`),
primitive `Int_max`, 0.4 ⟩

⟨ pattern `iddef`, primitive `let` AND primitive `val`, 1 ⟩

⟨ pattern `declareStruct`, pattern `datatype` OR pattern `pair`, 0.7 ⟩

⟨ pattern `funcdef`, pattern `fun` OR primitive `fn`, 1 ⟩

⟨ pattern `for` OR pattern `while`, pattern `recursion`, 1 ⟩

⟨ (pattern `if` AND primitive `==`) OR pattern `switch`,
pattern `guard` OR primitive `case`, 1 ⟩

⟨ type α `vector` OR type `string` OR type α `list`, type α `pointer`, 1 ⟩

⟨ type `number`, type `int` OR type `float`, 0.9 ⟩

⟨primitive `$nil`, primitive `NULL`, 1⟩

⟨primitive `$pair`, pattern `declareStruct`, 0.8⟩

⟨ primitive `$quote` OR primitive `$quasiquote`,
primitive `malloc` OR primitive `calloc`, 0.3 ⟩

⟨primitive + AND primitive `1`, primitive ++, 0.7⟩

⟨primitive – AND primitive `1`, primitive ––, 0.7⟩

⟨ primitive `append` OR primitive `reverse` OR
primitive `make-vector` OR primitive `vector-unfold`,
pattern `writeIndexInc`, 1 ⟩

⟨ primitive `car` OR primitive `cadr` OR
primitive `caddr` OR primitive `string-ref` OR
primitive `vector-ref` OR primitive `vector-set`,
pattern `index`, 1 ⟩

⟨ primitive `cdr` OR primitive `cddr`,
pattern `index` AND primitive `$ptr-&`, 0.8 ⟩

⟨primitive `define` OR primitive `lambda`, pattern `funcdef`, 0.9⟩

⟨ primitive `if` OR primitive `cond` OR
primitive `unless` OR primitive `when`,
pattern `if`, 1 ⟩

⟨primitive `cond`, pattern `switch`, 0.7⟩

⟨primitive `cond` AND primitive `eq?`, pattern `switch`, 1⟩

⟨primitive `eq?`, primitive ==, 1⟩

⟨ primitive `length` OR primitive `string-length`,
pattern `while` AND pattern `index` AND primitive ++, 0.9 ⟩

⟨primitive `let*`, pattern `iddef`, 0.95⟩

⟨ primitive `max`,
pattern `if` AND (primitive > OR primitive >= OR
primitive < OR primitive <=), 1 ⟩

⟨primitive `null?`, primitive == AND primitive `NULL`, 0.5⟩

⟨pattern `recursion`, pattern `for` OR pattern `while`, 1⟩

⟨type `number`, type `int` OR type `real`, 0.9⟩

⟨type α `vector`, type α `array`, 1⟩

⟨primitive `$pair`, pattern `pair`, 0.5⟩

⟨ primitive `$quote` OR primitive `$quasiquote`,
primitive `[` AND primitive `]`, 1 ⟩

⟨primitive `append`, primitive `@`, 1⟩

⟨ primitive `car` OR primitive `cdr` OR primitive `cadr` OR
primitive `cddr` OR primitive `caddr`,
primitive `$cons`, 0.4 ⟩

⟨primitive cons, primitive $cons, 1⟩

⟨primitive define, pattern fun, 1⟩

⟨primitive eq?, primitive $eq, 1⟩

⟨ (primitive if OR primitive unless OR
primitive when OR primitive cond) AND primitive eq?,
pattern guard OR primitive case, 1 ⟩

⟨ primitive if OR primitive unless OR
primitive when OR primitive cond,
primitive if, 1 ⟩

⟨primitive lambda, primitive fn, 1⟩

⟨primitive length, primitive List_length, 0.6⟩

⟨primitive let*, primitive let, 1⟩

⟨primitive list->string, primitive String_implode, 1⟩

⟨primitive make-vector, primitive Array_array, 0.8⟩

⟨primitive max, primitive Int_max, 0.9⟩

⟨primitive null?, primitive $nil, 0.8⟩

⟨primitive reverse, primitive List_rev, 1⟩

⟨primitive string-length, primitive String_size, 1⟩

⟨primitive string-ref, primitive String_sub, 1⟩

⟨primitive vector-ref, primitive Array2_sub, 0.9⟩

⟨primitive vector-set, primitive Array2_update, 0.9⟩

⟨primitive vector-unfold, primitive Array2_tabulate, 0.9⟩

⟨ type α ref OR type α list OR type α array OR type string,
type α pointer, 1 ⟩

⟨primitive $eq, primitive ==, 1⟩

⟨primitive $nil OR primitive NONE, primitive NULL, 1⟩

⟨primitive <>, primitive !=, 1⟩

⟨primitive + AND primitive 1, primitive ++, 0.7⟩

⟨primitive – AND primitive 1, primitive ––, 0.7⟩

⟨ primitive @ OR primitive Array2_tabulate OR
primitive List_rev OR primitive String_implode,
pattern writeIndexInc, 1 ⟩

⟨ primitive Array2_sub OR primitive Array2_update OR
primitive String_sub,
pattern index, 1 ⟩

⟨ primitive Int_max,
pattern if AND (primitive > OR primitive >= OR
primitive < OR primitive <=), 0.9 ⟩

⟨ primitive `String_size` OR primitive `List_length`,
pattern `while` AND pattern `index` AND primitive `++`, 0.9 ⟩

⟨primitive `let` AND primitive `val`, pattern `iddef`, 1⟩

⟨pattern `datatype` OR pattern `pair`, pattern `declareStruct`, 0.9⟩

⟨pattern `fun` OR primitive `fn`, pattern `funcdef`, 0.9⟩

⟨pattern `recursion`, pattern `for` OR pattern `while`, 1⟩

⟨ pattern `guard` OR primitive `case`,
(pattern `if` AND primitive `==`) OR pattern `switch`, 1 ⟩

⟨type `int` OR type `real`, type `number`, 1⟩

⟨type α `array`, type α `vector`, 1⟩

⟨primitive `$cons`, primitive `cons`, 1⟩

⟨primitive `$eq`, primitive `eq?`, 1⟩

⟨primitive `$nil`, primitive `null?`, 0.8⟩

⟨primitive `@`, primitive `append`, 1⟩

⟨primitive `Array2_sub`, primitive `vector-ref`, 0.9⟩

⟨primitive `Array2_tabulate`, primitive `vector-unfold`, 0.9⟩

⟨primitive `Array2_update`, primitive `vector-set`, 0.9⟩

⟨primitive `Int_max`, primitive `max`, 0.9⟩

⟨primitive `List_rev`, primitive `reverse`, 1⟩

⟨primitive `List_length`, primitive `length`, 1⟩

⟨primitive `String_implode`, primitive `list->string`, 1⟩

⟨primitive `String_size`, primitive `string-length`, 1⟩

⟨primitive `String_sub`, primitive `string-ref`, 1⟩

⟨primitive `fn`, primitive `lambda`, 1⟩

⟨ primitive `if`,
primitive `if` OR primitive `unless` OR
primitive `when` OR primitive `cond`, 1 ⟩

⟨ pattern `guard` OR primitive `case`,
(primitive `if` OR primitive `unless` OR
primitive `when` OR primitive `cond`) AND primitive `eq?`, 1 ⟩

⟨primitive `let`, primitive `let*`, 1⟩

⟨ primitive `[` AND primitive `]`,
primitive `$quote` OR primitive `$quasiquote`, 1 ⟩

⟨pattern `fun`, primitive `define`, 1⟩

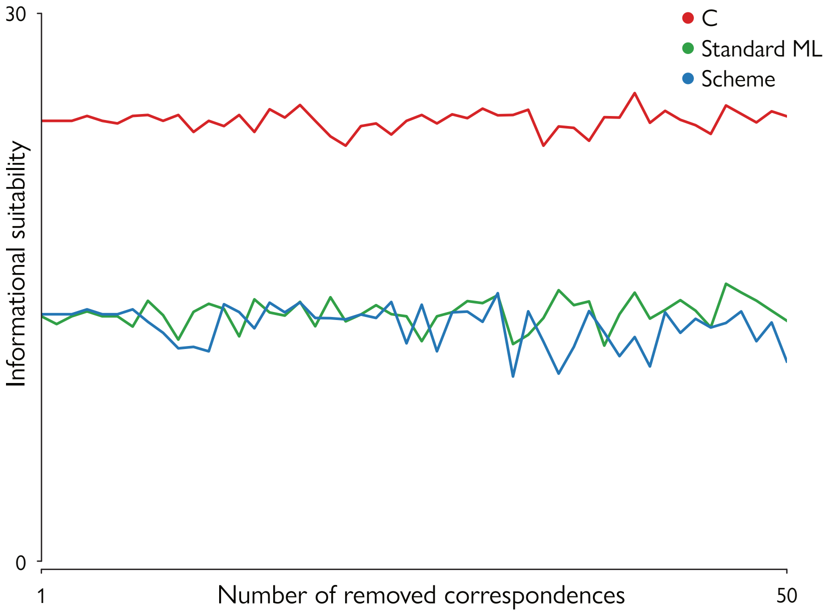
⟨pattern `pair`, primitive `$pair`, 0.5⟩

I Plots of scores after omitting correspondences and components

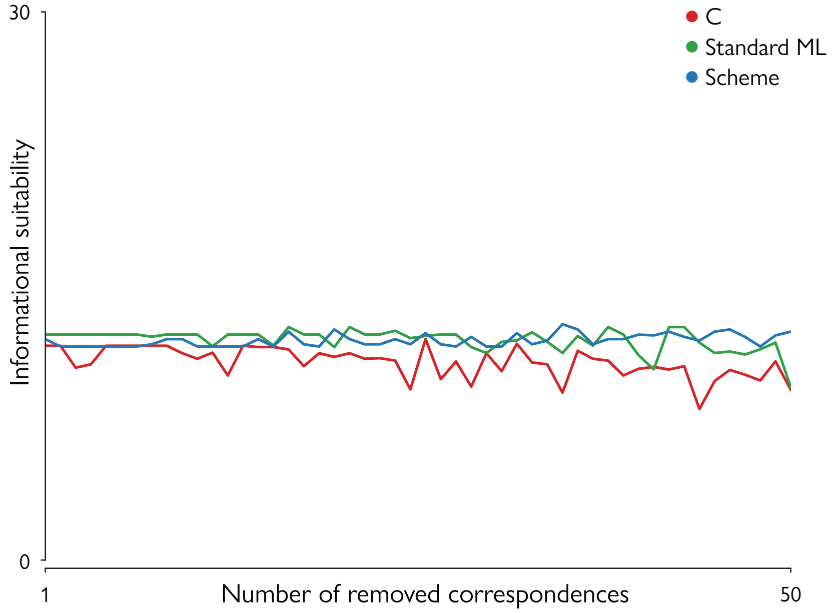
The following plots summarise the outputs from the framework after providing inputs with 'missing' correspondences and components, as described in Section 6.4. These plots form part of the analysis in Section 6.4, page 149.

As explained in Section 6.4, the vertical axis is the informational suitability computing by `robin`, while the horizontal axis is the number of correspondences or components randomly removed from the correspondence set or descriptions, respectively. In the case of RS-descriptions, the same number of components was removed from each description.

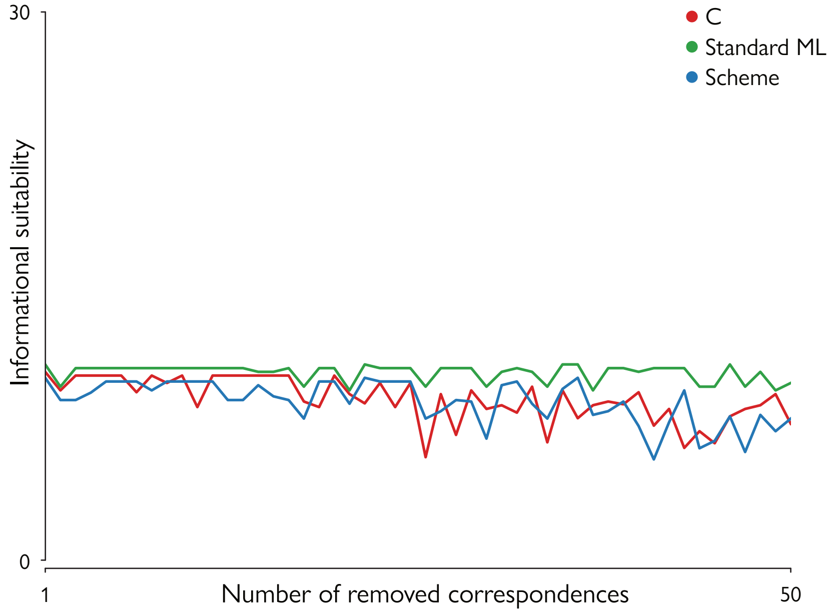
Removing correspondences for in-order tree traversal in C



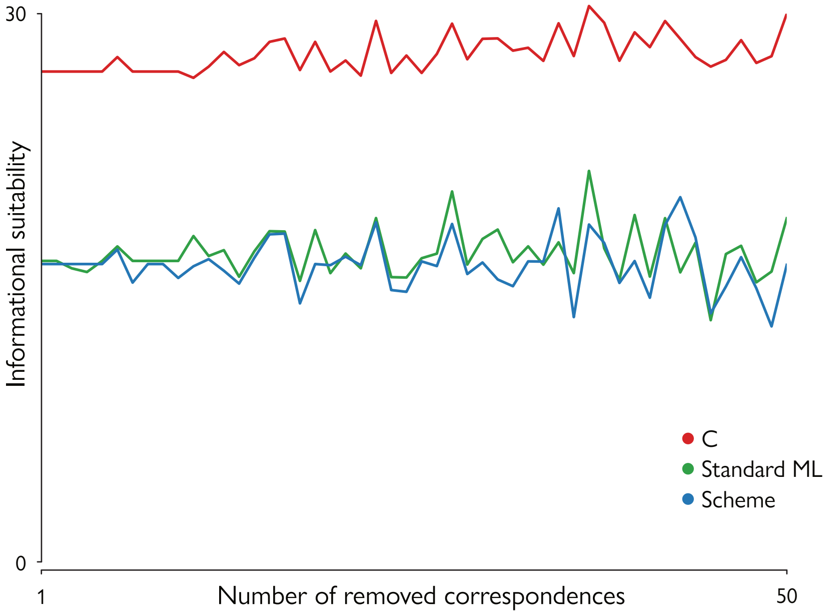
Removing correspondences for in-order tree traversal in Scheme



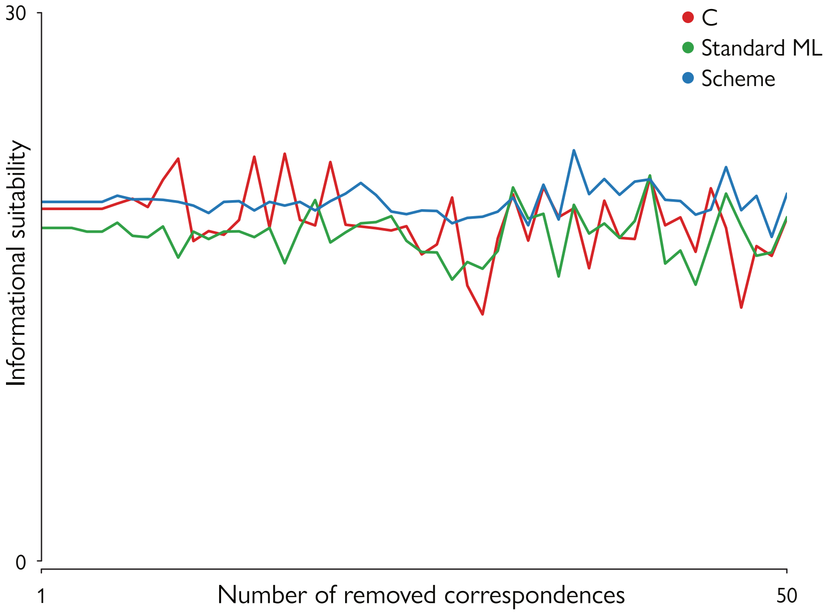
Removing correspondences for in-order tree traversal in Standard ML



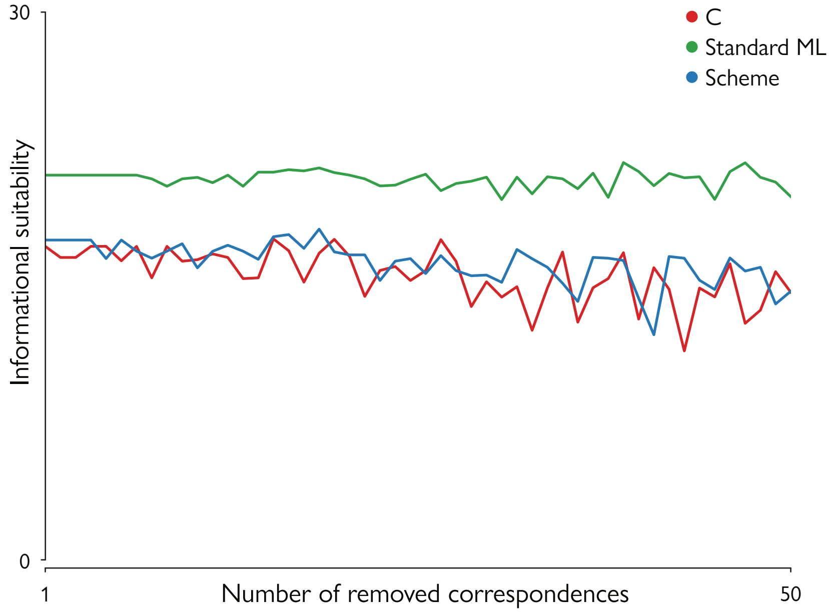
Removing correspondences for the longest common subsequence in C



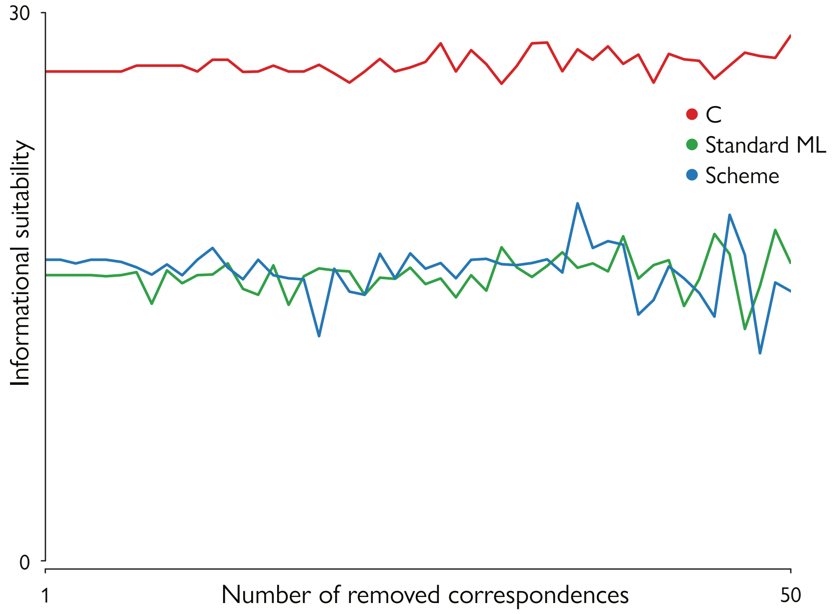
Removing correspondences for the longest common subsequence in Scheme



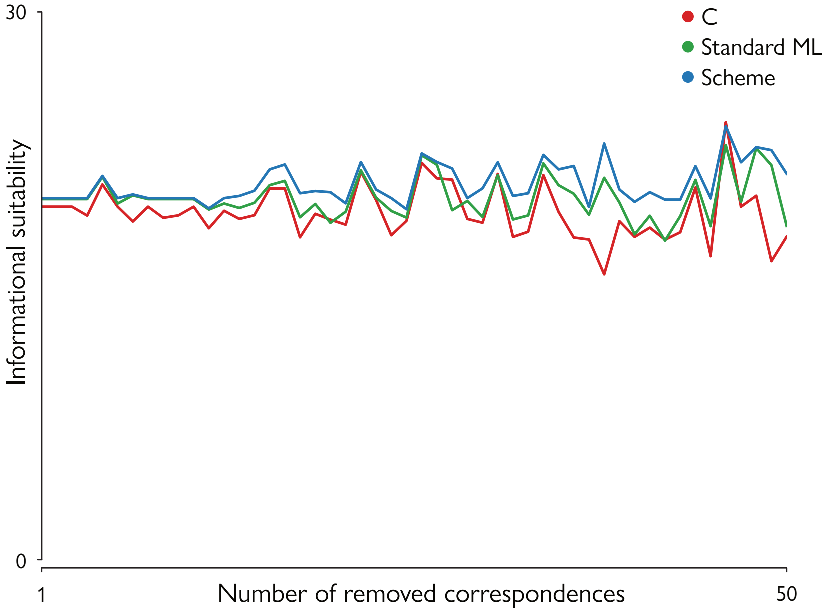
Removing correspondences for the longest common subsequence in Standard ML



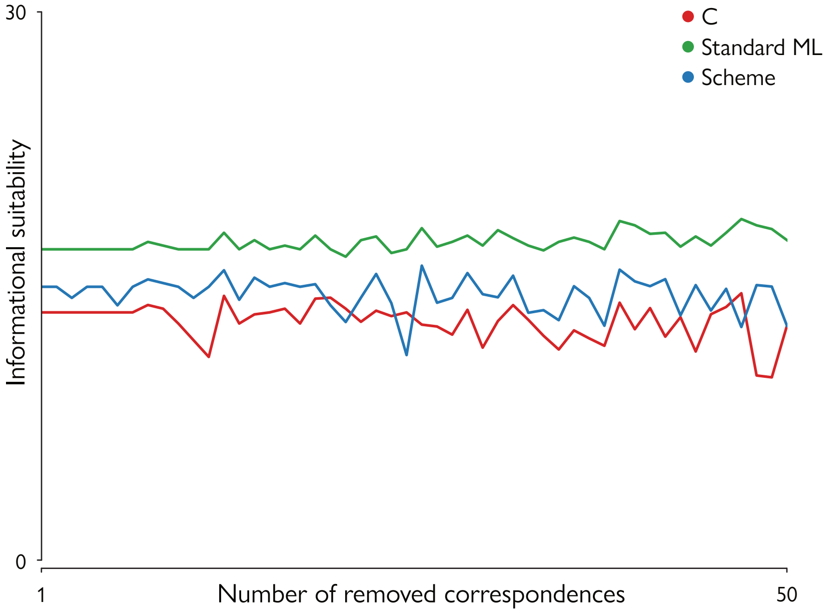
Removing correspondences for merge sort in C



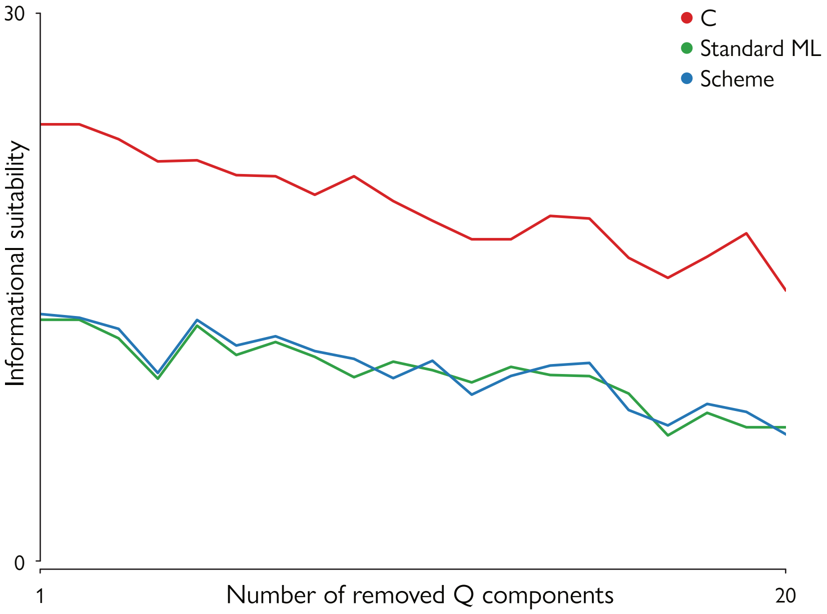
Removing correspondences for merge sort in Scheme



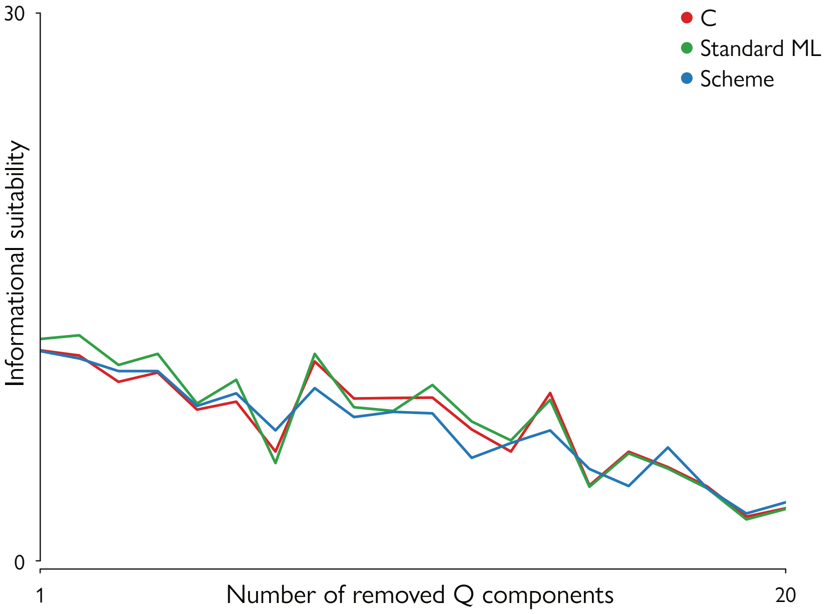
Removing correspondences for merge sort in Standard ML



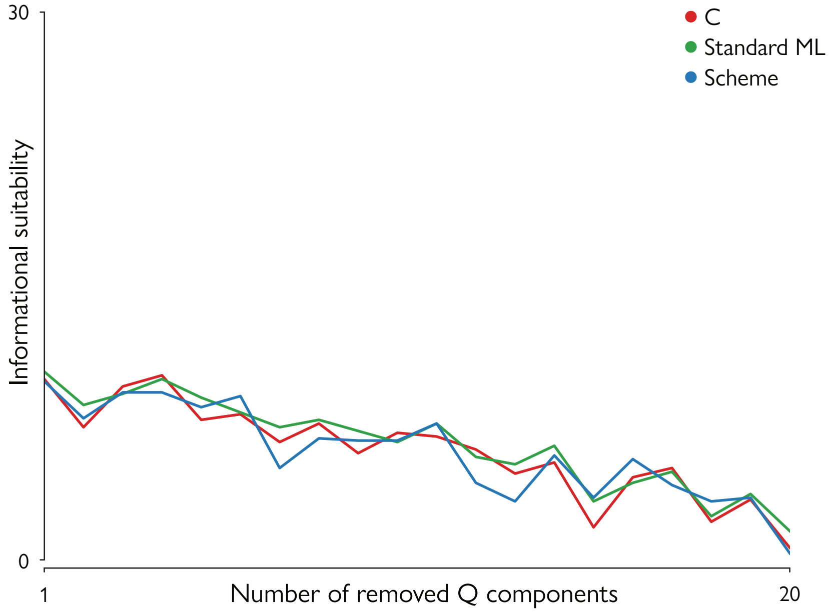
Removing Q-components for in-order tree traversal in C



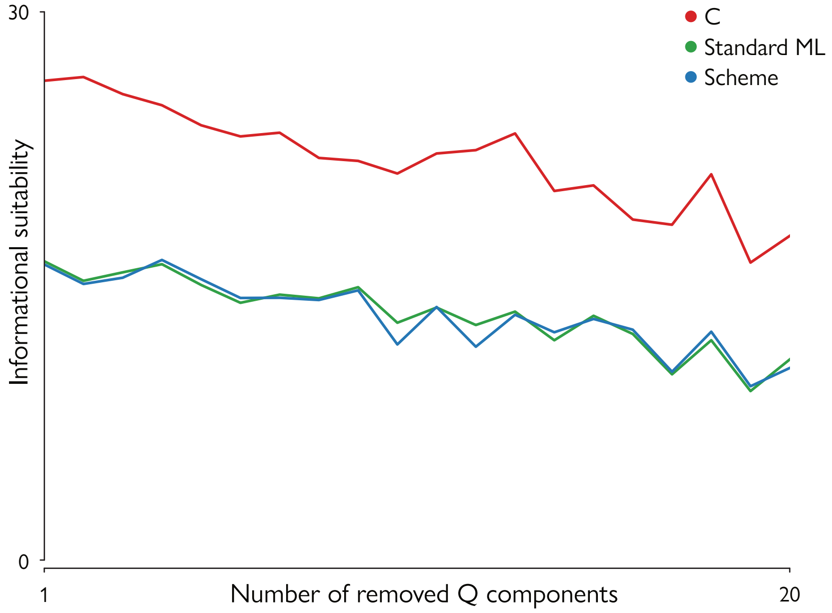
Removing Q-components for in-order tree traversal in Scheme



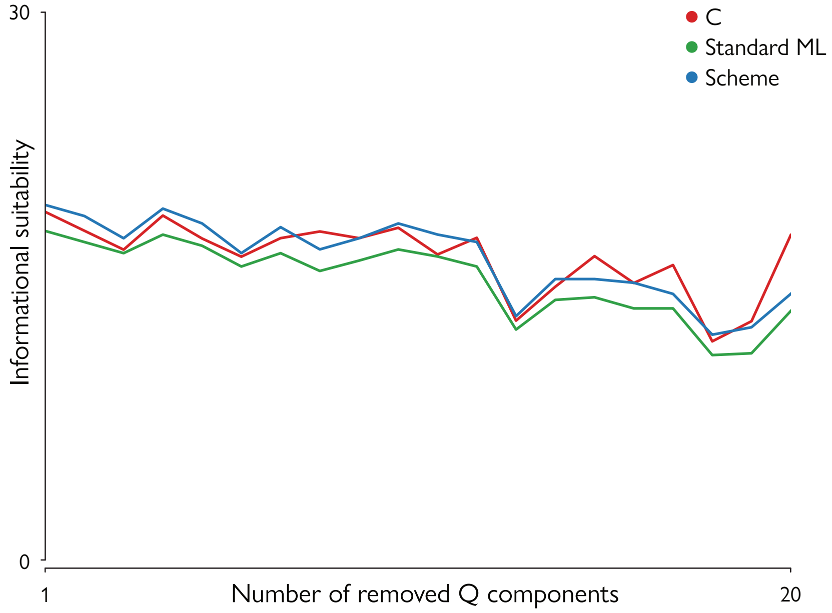
Removing Q-components for in-order tree traversal in Standard ML



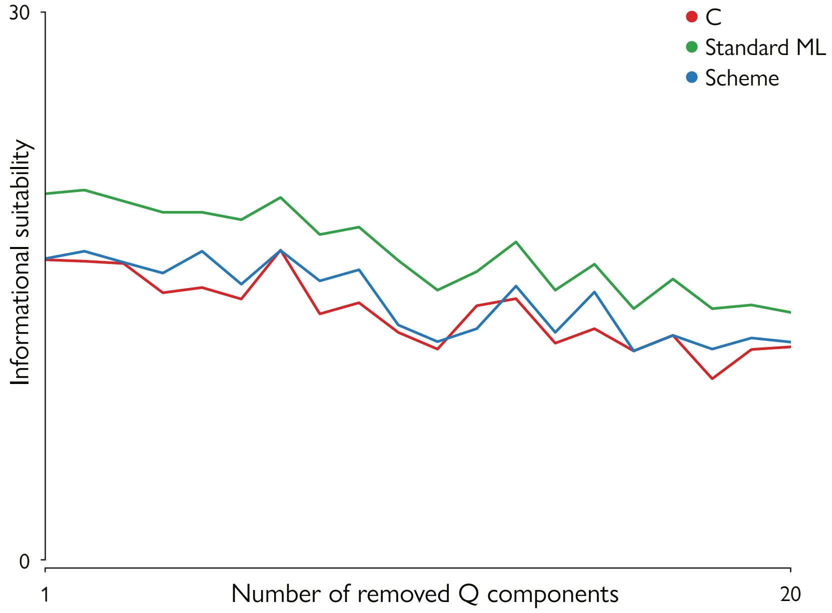
Removing Q-components for the longest common subsequence in C



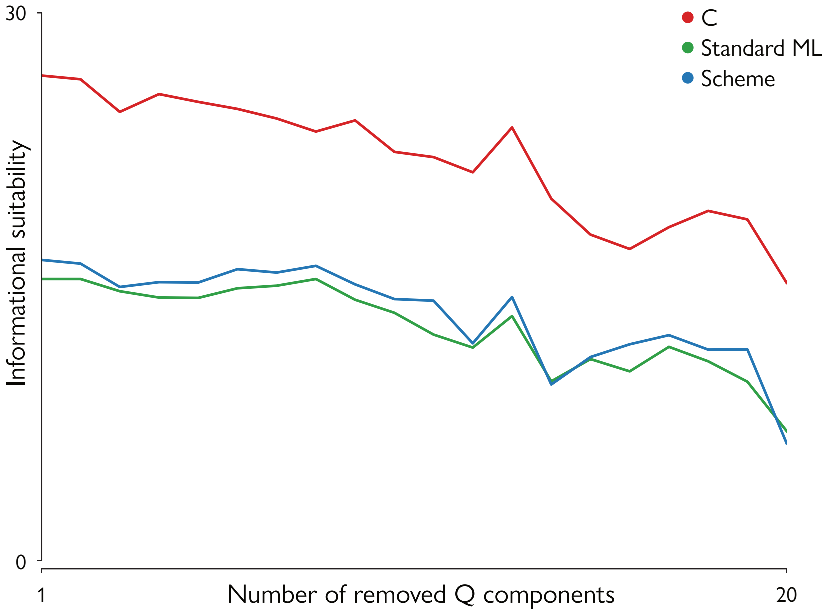
Removing Q-components for the longest common subsequence in Scheme



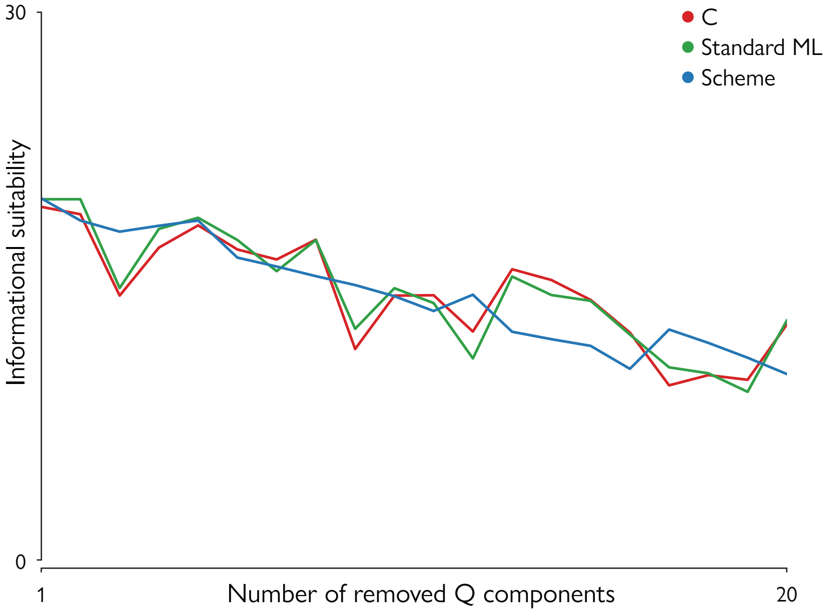
Removing Q-components for the longest common subsequence in Standard ML



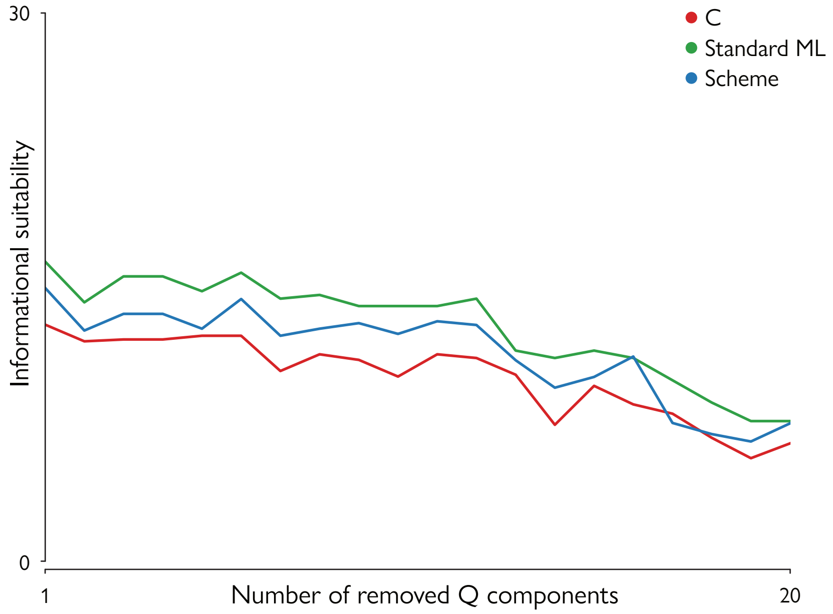
Removing Q-components for merge sort in C



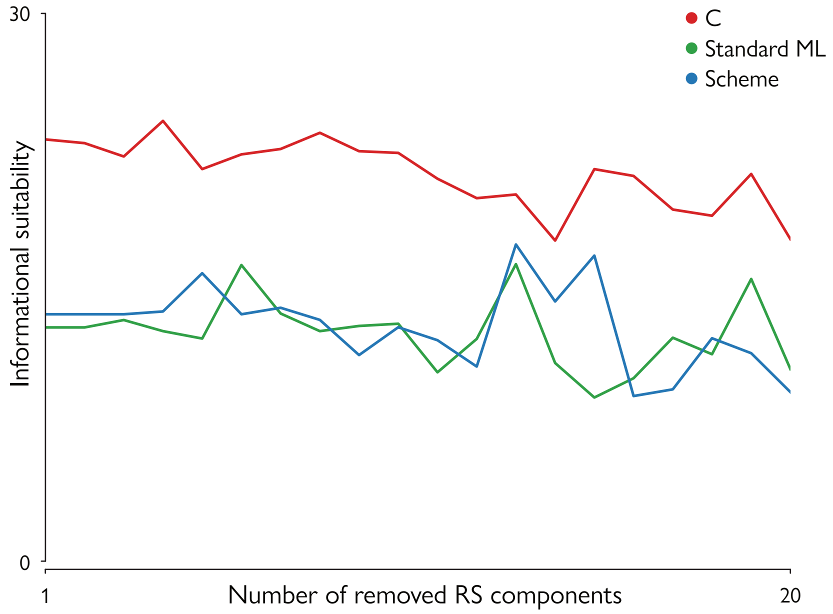
Removing Q-components for merge sort in Scheme



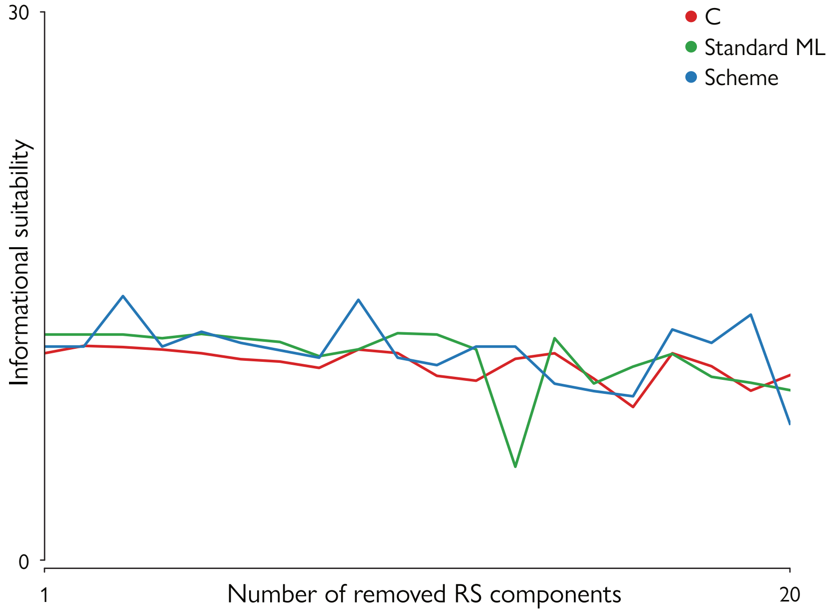
Removing Q-components for merge sort in Standard ML



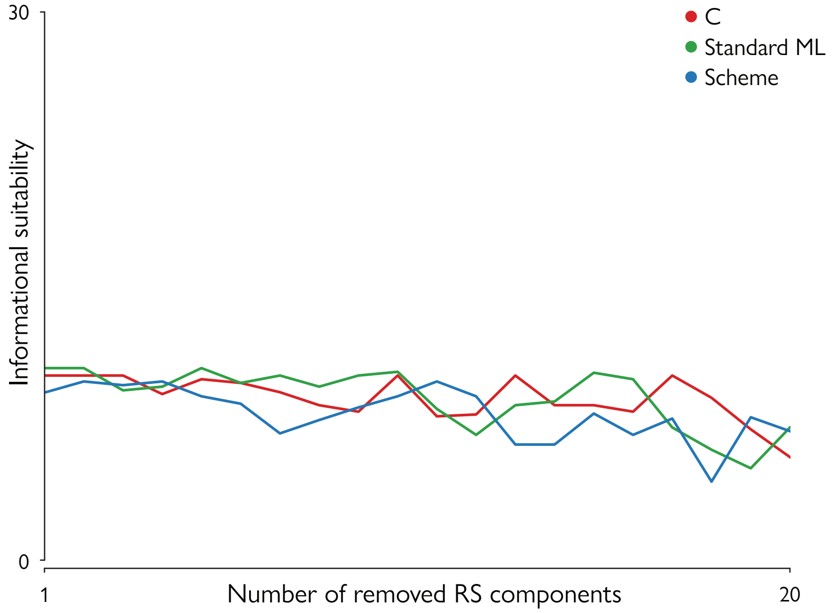
Removing RS-components for in-order tree traversal in C



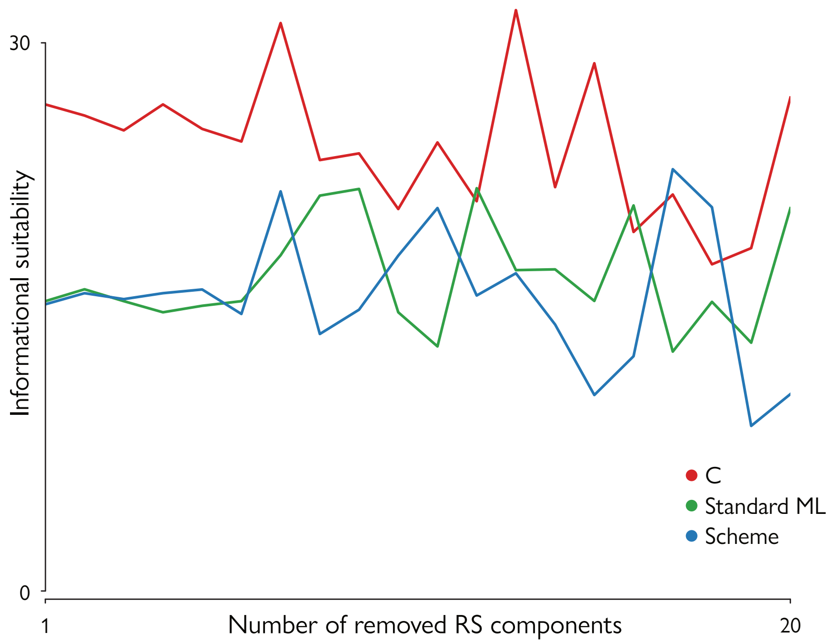
Removing RS-components for in-order tree traversal in Scheme



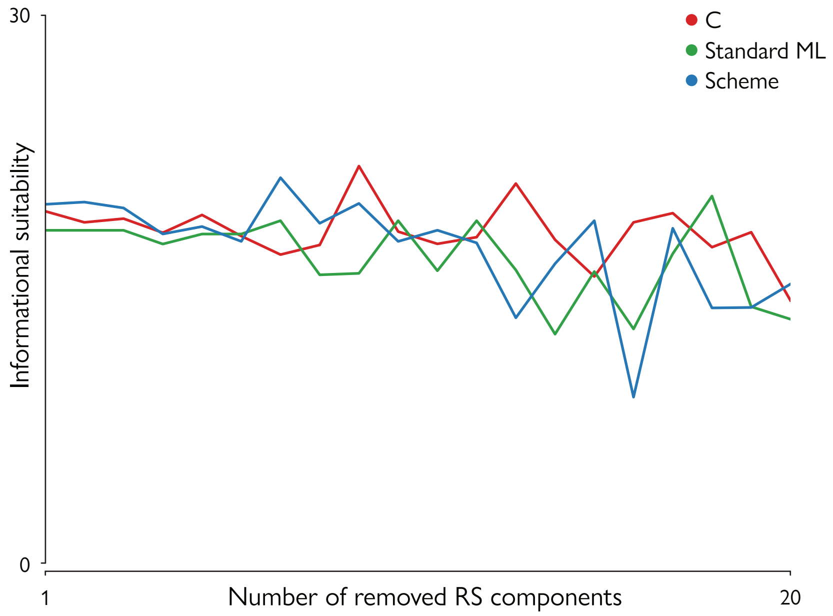
Removing RS-components for in-order tree traversal in Standard ML



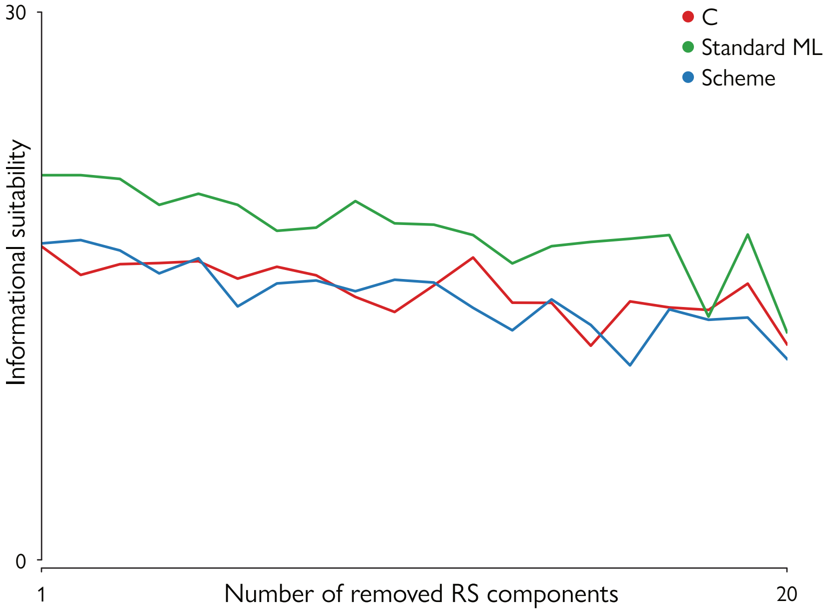
Removing RS-components for
the longest common subsequence in C



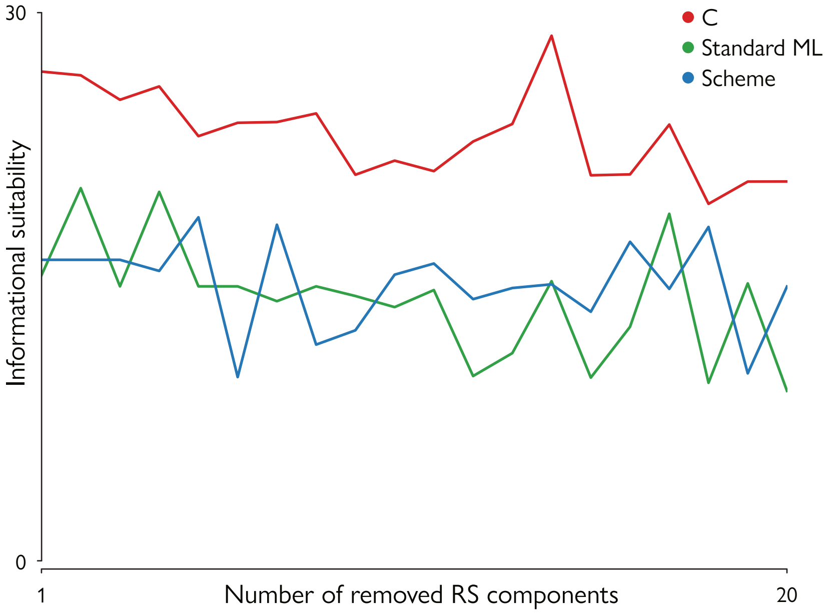
Removing RS-components for
the longest common subsequence in Scheme



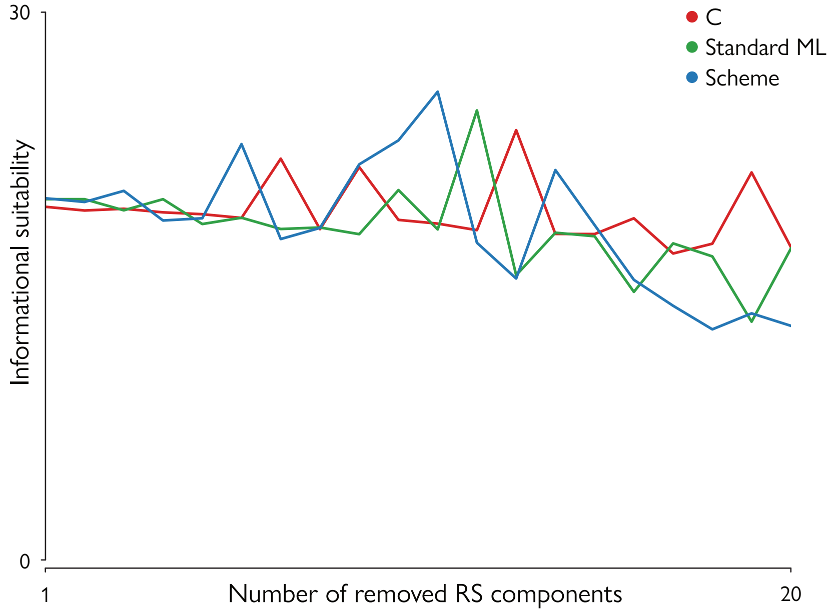
Removing RS-components for the longest common subsequence in Standard ML



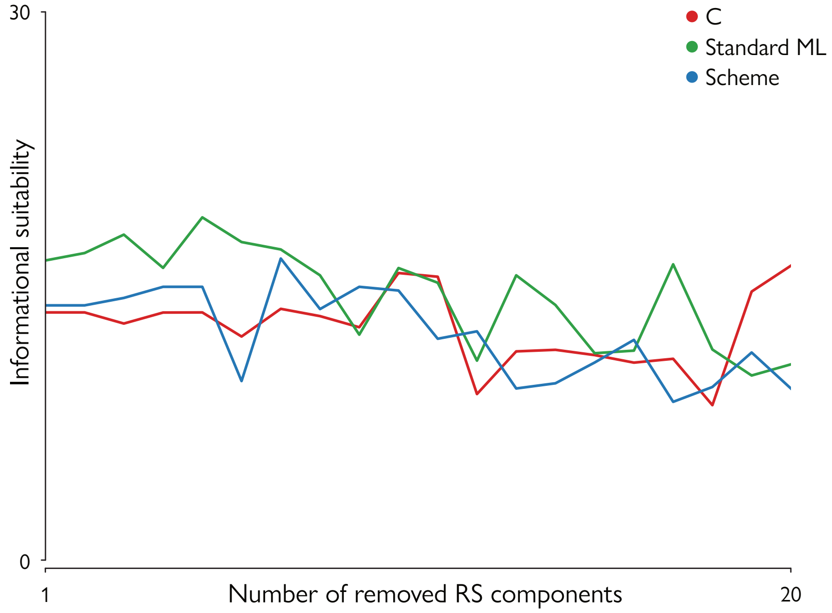
Removing RS-components for merge sort in C



Removing RS-components for merge sort in Scheme



Removing RS-components for merge sort in Standard ML



J Representational system training resources

The following pages are direct copies of the training material given to participants during our experiment described in Section 7.3, page 165.

The documents are included verbatim from the study; errors present here were also present in versions shown to participants. In particular, the Area Diagrams information sheet incorrectly states in the second example that 'three of the five even numbers are prime'—three of the five *odd* numbers are prime, not even. A few participants did pick up on this, and correctly inferred the mistake. Many did not pick up on our error: we believe they implicitly understood the intended meaning.

The example also required the participants to have general knowledge about integers and playing cards; they all had no problem understanding the examples as given.

Representation – Area diagrams

Summary

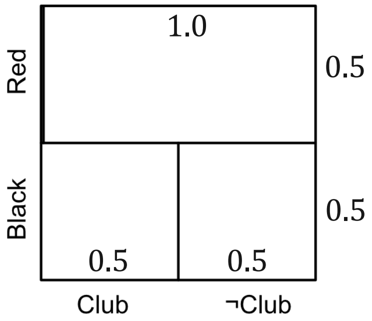
An area diagram is a unit square representing all possible outcomes, with labels for events, their split length representing the probability of each event. Labels might use “not” (¬)

The area enclosed by lines represents the probability X *and* Y together, where X and Y are the edge labels. Areas can be added together to find A *or* B, where A and B are areas.

The order of the events and factors is not meaningful.

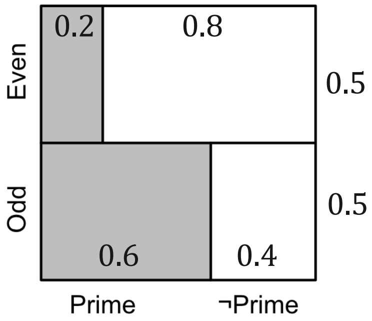
Examples

1.



In a deck of cards, half are red, and half are black. No red cards are clubs. Half the black cards are clubs.

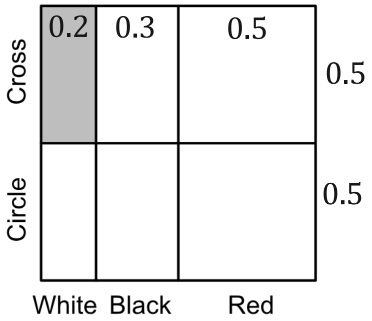
2.



Of the numbers between 1 and 10, half are even, and half are odd. One of the five even numbers is prime. Three of the five even numbers are prime.

Thus 40% numbers are prime.

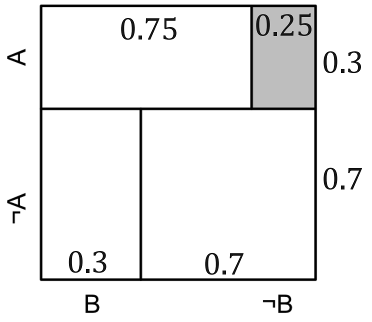
3.



Counters are 20% white, 30% black, and 50% red. On one side they have a cross, and the other they have a circle, with an even chance of being either side.

The probability of a white counter showing a cross is 10%.

4.



The probability of A is 30%. The probability of B given A is 75%, but only 30% given not A.

Thus, the probability of A and not B is 7.5%.

Representation – Bayesian algebra

Summary

Bayesian algebra consists of numbers, letters, and words, which are combined using standard mathematical operations ($+$, $-$, $\times$, $\div$) and probability functions $\mathrm{P}(x)$ and $\mathrm{P}(x|y)$ which map events numbers between 0 and 1. Symbols or “and”, “or”, and “not” (∩, ∪,¬) are used to combine events.

Progress is made by rewriting equations through applying operations, simplifying equations, and rearranging terms.

The size and absolute position of equations have no meaning.

Examples

1.

$$\begin{aligned} \mathrm{P}(\text{red}) &= 0.5 \\ \mathrm{P}(\text{club}) &= 0.25 \\ \mathrm{P}(\text{club} \mid \text{red}) &= 0 \end{aligned}$$

In a deck of cards, half are red, and one quarter are clubs. If the card is red then it cannot be a club.

2.

$$\begin{aligned} \mathrm{P}(E) &= 0.5 \\ \mathrm{P}(P \cap E) &= 0.1 \\ \mathrm{P}(P \mid E) &= \frac{\mathrm{P}(P \cap E)}{\mathrm{P}(E)} = 0.2 \end{aligned}$$

Let U be the set of integers from 1 to 10. Let E be the event that a number from U is even and let P be the event that a number from U is prime. The probability that a number from U is both prime and even is 0.1. Then the probability that a number in U is prime given that it is even is 0.2.

3.

$$\begin{aligned} \mathrm{P}(M) &= 0.92 \\ \mathrm{P}(N) &= 0.24 \\ \mathrm{P}(M \mid N) &= 0.75 \\ \mathrm{P}(M \cap N) &= \mathrm{P}(M \mid N) \cdot \mathrm{P}(N) \\ &= 0.75 \times 0.24 = 0.18 \end{aligned}$$

The probability of M is 0.92, and N is 0.24. Given N, the probability of M becomes 0.75. Thus the probability of both M and N is 0.18.

4.

$$\begin{aligned} \mathrm{P}(\text{meow} \mid \text{hungry}) &= 90\% \\ \mathrm{P}(\text{hungry}) &= 10\% \\ \mathrm{P}(\text{meow}) &= 15\% \\ \mathrm{P}(\text{hungry} \mid \text{meow}) &= \mathrm{P}(\text{meow} \mid \text{hungry}) \cdot \frac{\mathrm{P}(\text{hungry})}{\mathrm{P}(\text{meow})} \\ &= 60\% \end{aligned}$$

The cat will meow if it is hungry 90% of the time. The cat is hungry 10% of the time, and the cat meows 15% of the time. Thus, the probability that the cat is hungry given that it is meowing is 60%.

Representation – Contingency tables

Summary

A contingency table is a grid where the first row and column are reserved for labels, which (along each axis) are mutually exclusive but together are all possible outcomes. Labels may use the symbol “not” (¬).

The final row and column contain numbers which must be the sum of the numbers in their own (completely filled) row/column. The value in the final cell is always 1.

Inner cells are filled with real values between 0 and 1, and represent the probability of X *and* Y, assuming labels X and Y align with that cell.

The size of the cells has no meaning.

Examples

1.

	Red	Black	Total
Club	0.0	0.25	0.25
¬Club	0.5	0.25	0.75
Total	0.5	0.5	1

From a deck of cards, the probability of being red and a club is 0, red and not a club is 0.5, black and a club is 0.25, and black and not a club is 0.25.

2.

	Even	Odd	Total
Prime	0.1	0.3	0.4
¬Prime	0.4	0.2	0.6
Total	0.5	0.5	1

For the numbers from 1 to 10, the probability of a number being even and prime is 0.1, even and not prime is 0.4, odd and prime is 0.3, and odd and not prime is 0.2.

3.

	X	¬X	Total
Y	0.18	0.22	0.4
¬Y	0.27	0.33	0.6
Total	0.45	0.55	1

The probability of X and Y is 0.18, X and not Y is 0.27, not X and Y is 0.22, and not X and not Y is 0.33.

4.

	Young	Mid	Old	Total
Vote	0.08	0.27	0.25	0.6
¬Vote	0.12	0.23	0.05	0.4
Total	0.2	0.5	0.3	1

From a population, the probability of a citizen being young and voting is 0.08, young and not voting is 0.12, middle aged and voting is 0.27, middle aged and not voting is 0.23, old and voting is 0.25, and old and not voting is 0.05.

Representation – Euler diagrams

Summary

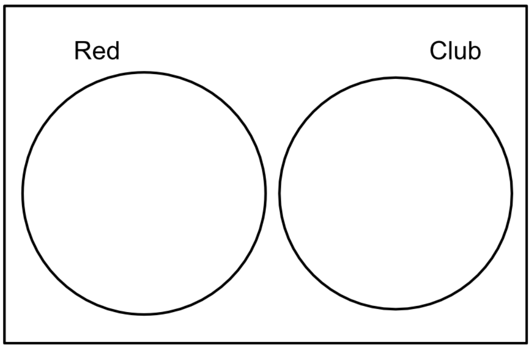
Euler diagrams consist of a “universe” denoted by a rectangle, and ellipses representing events. Events are named with letters or words.

The region inside the curve represents events occuring. Regions inside two curves represent X and Y occuring simultaneously. Regions that do not overlap are disjoint.

The size or shape of the curves are not meaningful.

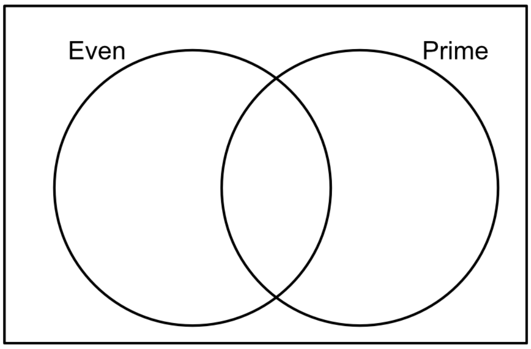
Examples

1.



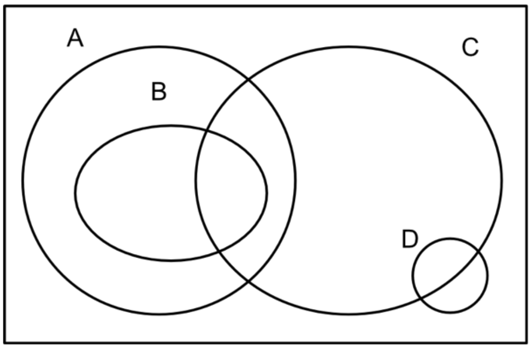
Some cards are red. Some cards are clubs. No card is a red club.

2.



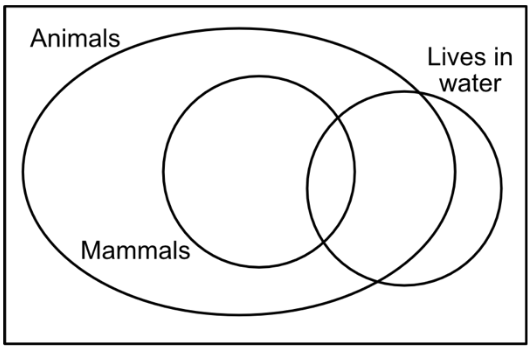
There are even numbers. There are prime numbers. There are even and prime numbers.

3.



Some (but not all) As are Cs, and some (but not all) Cs are As. All Bs are As, and some (but not all) Bs are also Cs. Some (but not all) Ds are Cs, but no D is also an A.

4.



All mammals are animals, but not all animals are mammals. Some mammals live in water, but some do not; some animals live in water, but some do not. Some things that live in water are not animals.

Representation – Probability trees

Summary

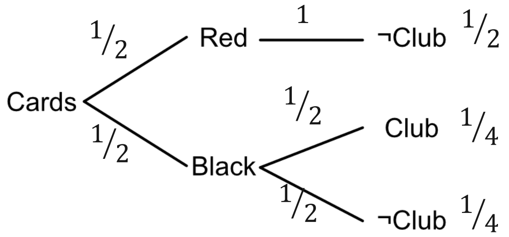
Probability trees consists of events and branches. Events sometimes use a "not" symbol (¬). Each event has exactly one "prevous" event, except for the first event which has no previous. Branches are labelled with the probability of the next event occuring given that the previous event has occurred. The sum of adjacent branches must be 1.

X *and* Y is computed by multiplying along branches; X *or* Y by adding between branches.

Neither the length of branches nor the order of adjacent events is meaningful.

Examples

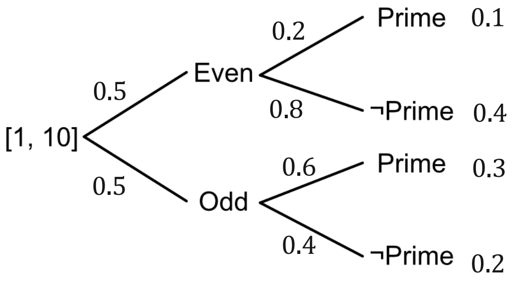
1.



$P(club) = {}^1\!/_4$

Half of the cards in a deck are red, the other half are black. No red card is a club, but half the black cards are a club. The total probability of getting a club is ¼.

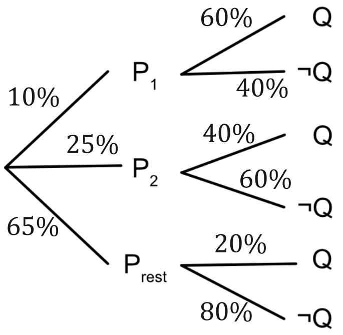
2.



$P(prime) = 0.1 + 0.3 = 0.4$

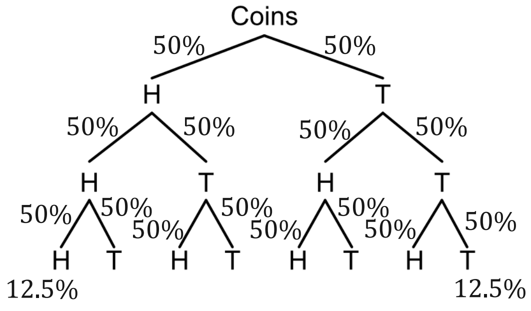
For the numbers from 1 to 10, half of the numbers are even. One of the five even numbers is prime. Three of the five even numbers are prime. The total probability of a number between 1 and 10 being prime is 0.4.

3.



The probability of P_1 is 10%, P_2 is 25%, and the remaining Ps together have probability 65%. If P_1 is true, then Q has probability 60%, whereas given P_2 Q has probability 40%. Otherwise, Q has probability 20%.

4.



$P(HHH \cup TTT) = 12.5\% + 12.5\% = 25\%$

Toss three coins, each with a 50% chance of begin heads or tails. The probability of getting all heads or all tails is 25%.

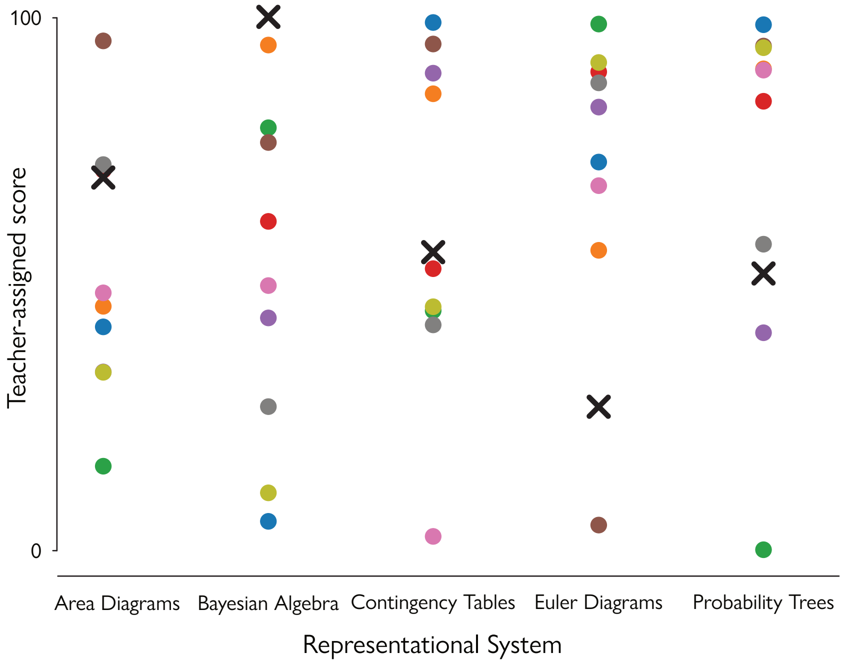
Plots of teachers' scores

K

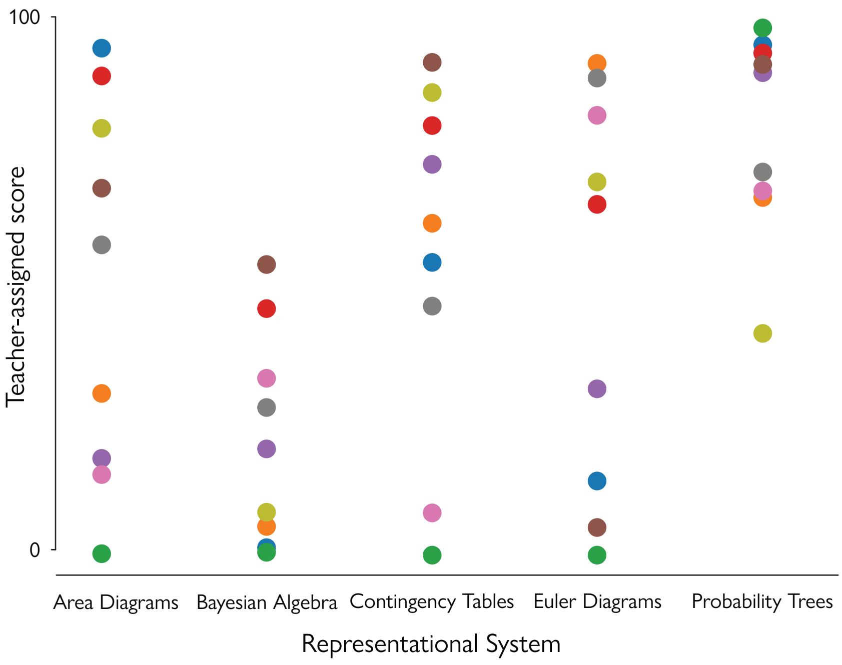
The following plots summarise the responses from the teachers who participated in the evaluation described in Section 7.3. These plots form part of the analysis in Section 7.3.4, page 175.

As explained in Section 7.3.4, the points are coloured by the participant identifier: every blue dot (one in each column) is a response by one unique participant, for example. The colours are consistent between plots—for example, the 'blue' participant's responses are *always* blue. The crosses are not participant responses, but the output of the `robin` implementation of the rep2rep framework after adjusting the scores to the same [0, 100] range.

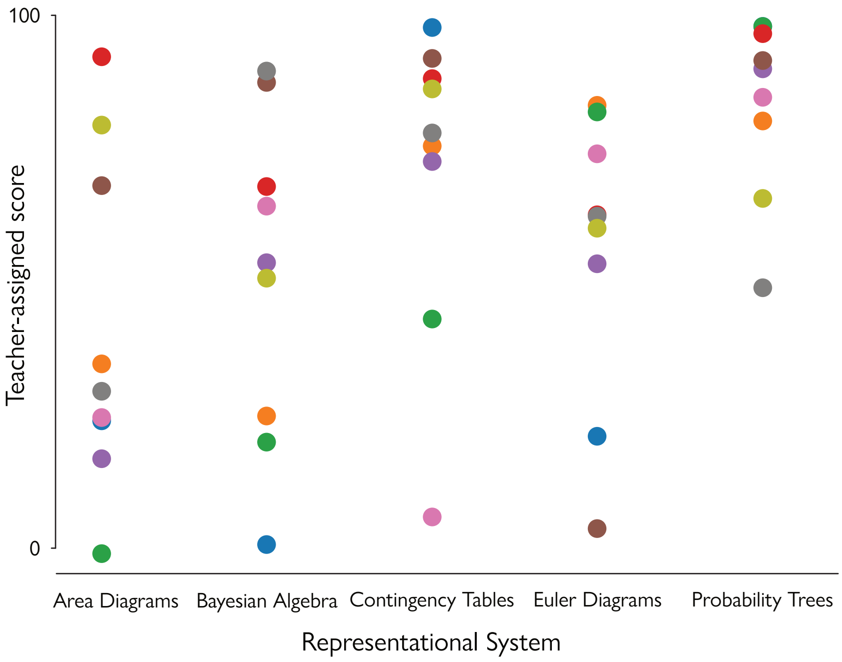
Birds problem, informational suitability



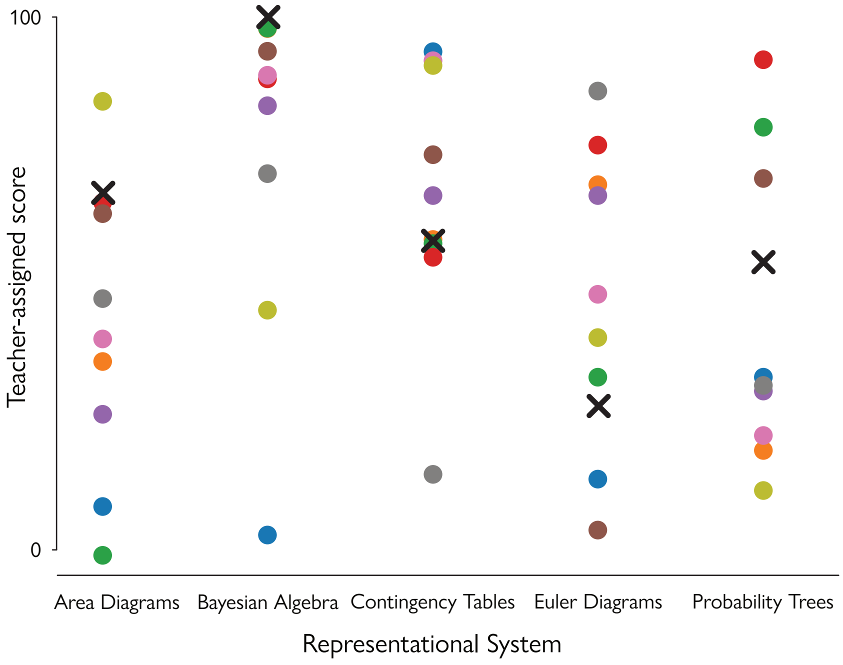
Birds problem, low ability



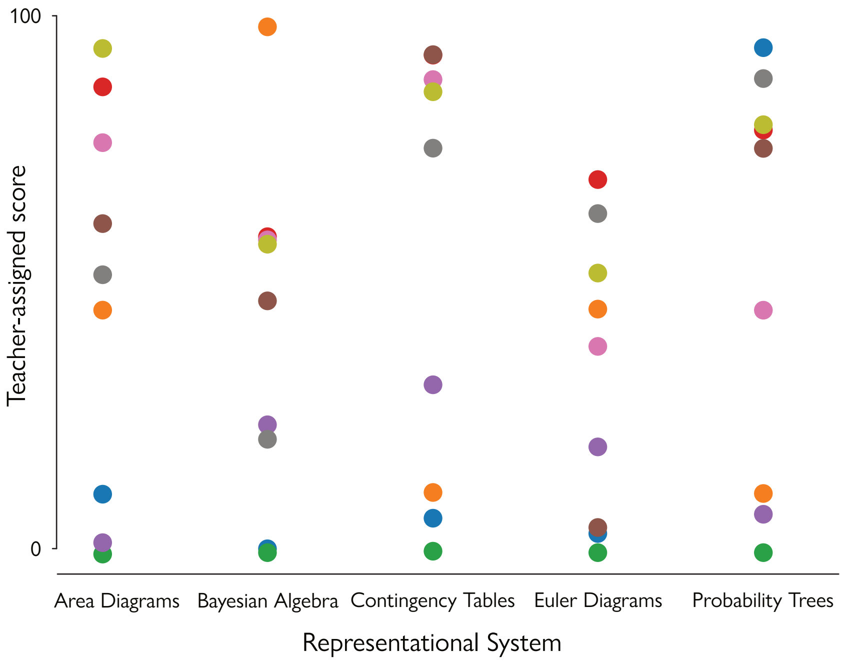
Birds problem, high ability



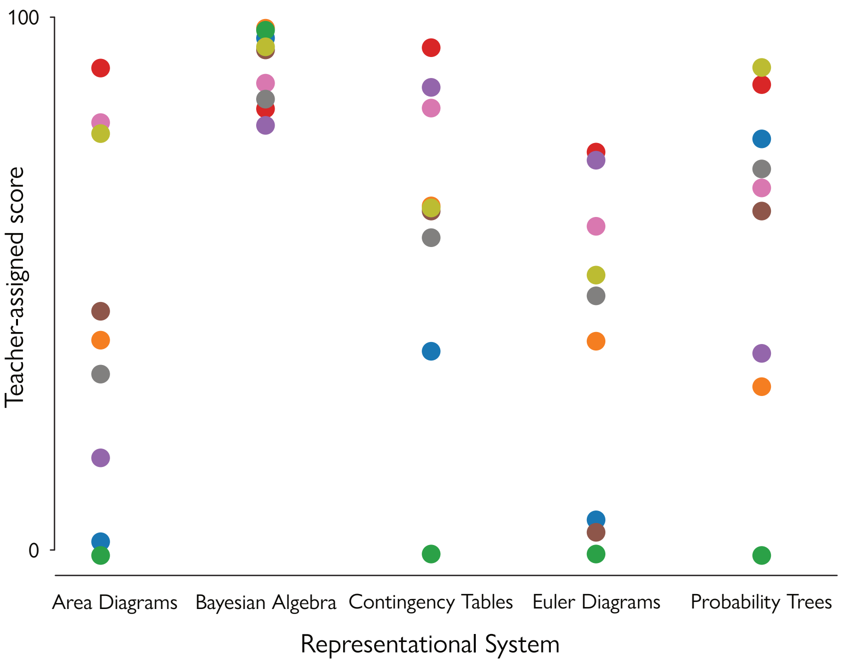
Birds-equivalent problem, informational suitability



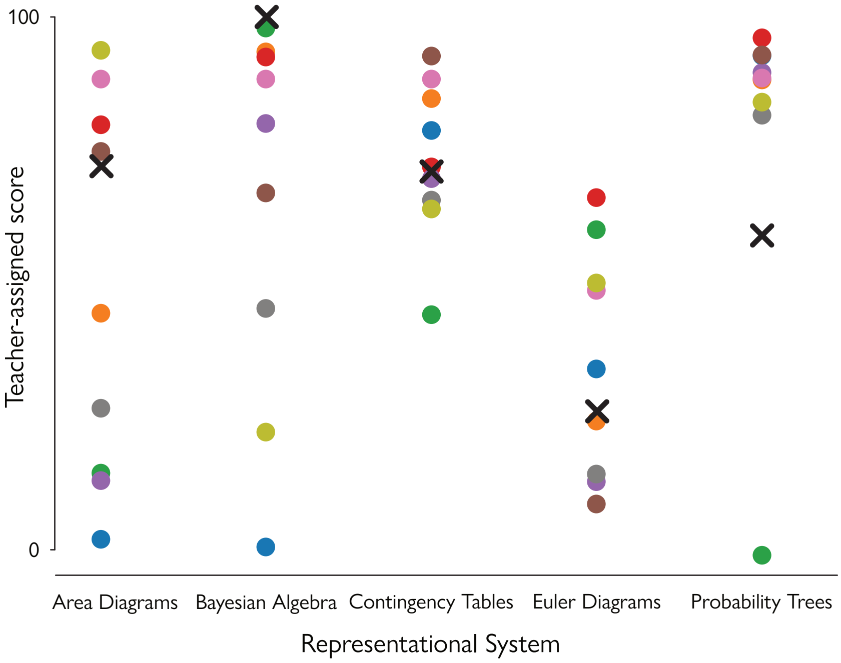
Birds-equivalent problem, low ability



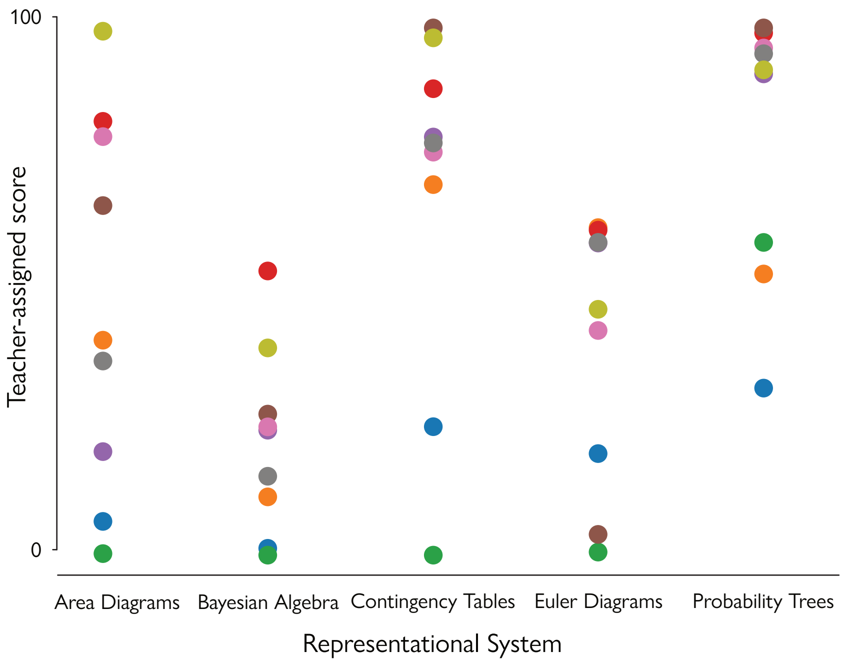
Birds-equivalent problem, high ability



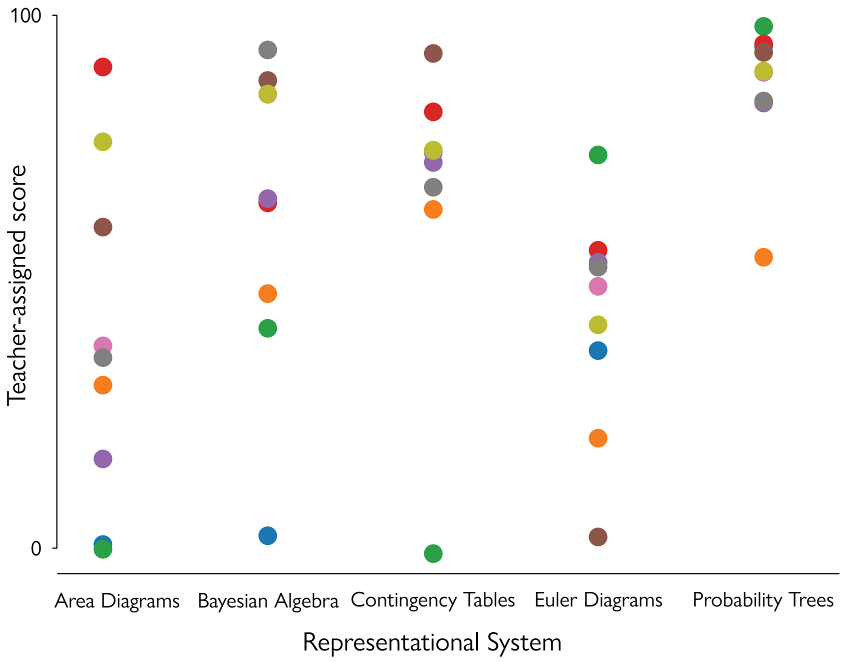
Lightbulbs problem, informational suitability



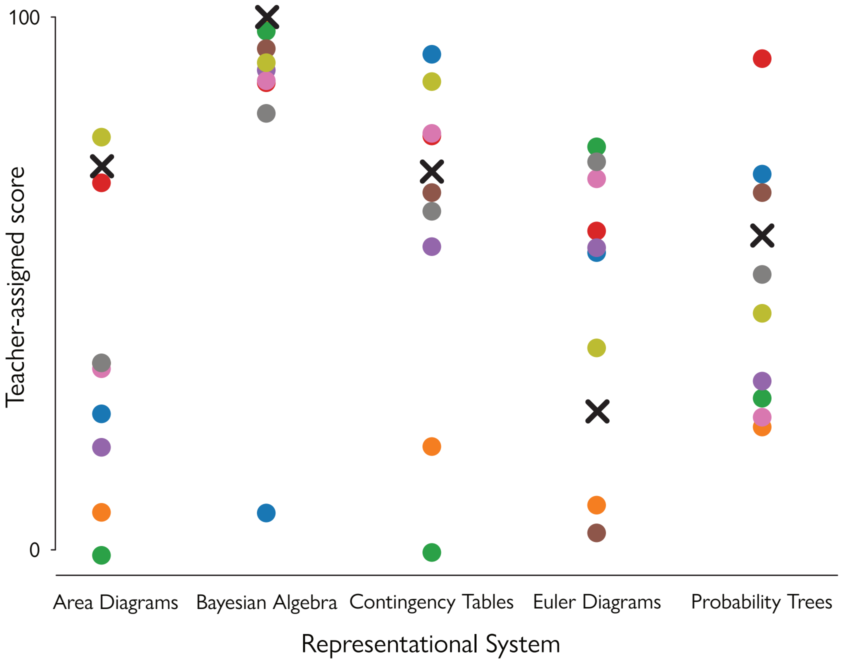
Lightbulbs problem, low ability



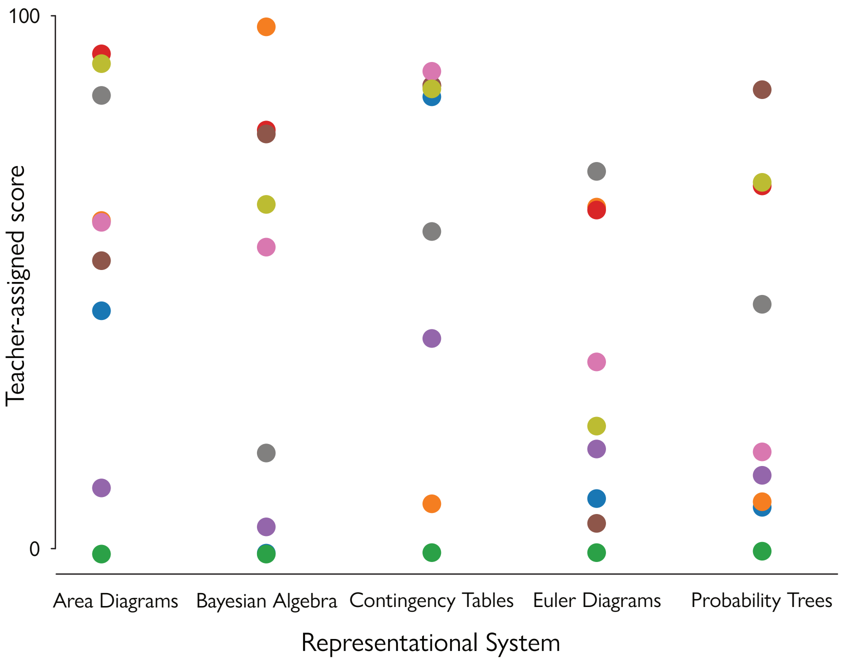
Lightbulbs problem, high ability



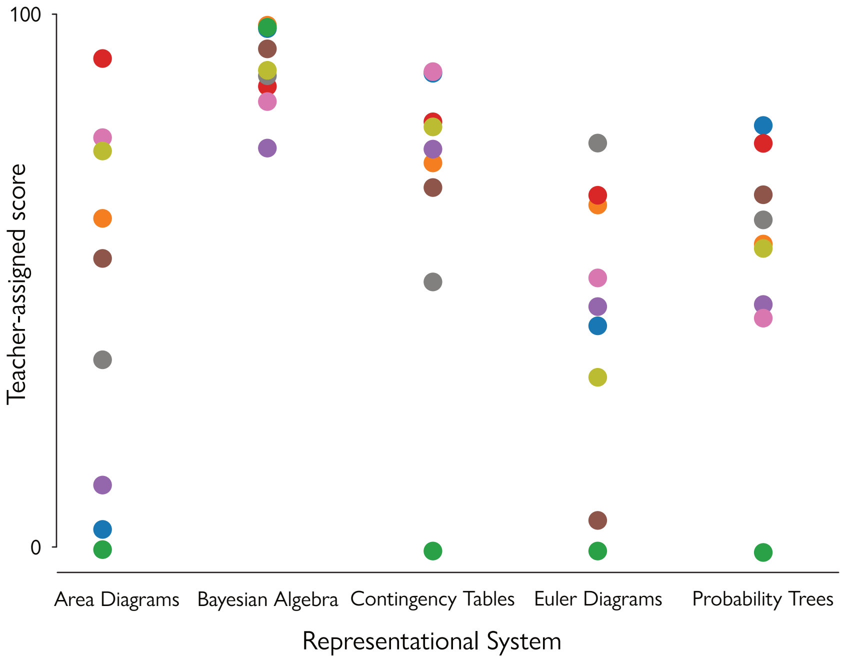
Lightbulbs-equivalent problem, informational suitability



Lightbulbs-equivalent problem, low ability



Lightbulbs-equivalent problem, high ability



L Summary tables of teacher's responses statistics

The following tables summarise the analysis of the responses from the teachers who participated in the evaluation described in Section 7.3. These tables form part of the analysis in Section 7.3.4, page 175.

In the Friedman test tables, the final column contains an asterisk if the p-value is below 0.05, indicating we should pursue post-hoc tests. In the Wilcoxon test tables, the final column contains an asterisk if the p-value is below 0.005, which is the significance threshold after Bonferroni correction.

L.1 Birds problem

Friedman tests

Context	Friedman Q	p	
No persona	5.03	0.284	
Low ability	11.29	0.024	*
High ability	11.01	0.027	*

Wilcoxon tests, low ability

Representational Systems		Wilcoxon W	p	
Areas	Bayes	10.5	0.164	
Areas	Contingency	15.5	0.719	
Areas	Euler	14.0	0.570	
Areas	Trees	5.0	0.067	
Bayes	Contingency	9.0	0.129	
Bayes	Euler	8.5	0.129	
Bayes	Trees	0.0	0.004	*
Contingency	Euler	17.5	0.944	
Contingency	Trees	7.0	0.121	
Euler	Trees	10.0	0.164	

Wilcoxon tests, high ability

Representational Systems		Wilcoxon *W*	p
Areas	Bayes	21.5	0.910
Areas	Contingency	5.0	0.039
Areas	Euler	16.0	0.496
Areas	Trees	2.0	0.012
Bayes	Contingency	8.5	0.129
Bayes	Euler	14.5	0.618
Bayes	Trees	6.5	0.074
Contingency	Euler	13.5	0.301
Contingency	Trees	9.0	0.389
Euler	Trees	6.0	0.055

L.2 Birds-equivalent problem

Friedman tests

Context	Friedman Q	p	
No persona	7.75	0.101	
Low ability	9.98	0.041	*
High ability	18.18	0.001	*

Wilcoxon tests, low ability

Representational Systems		Wilcoxon *W*	p
Areas	Bayes	10.5	0.285
Areas	Contingency	11.0	0.203
Areas	Euler	4.5	0.102
Areas	Trees	15.5	0.722
Bayes	Contingency	8.0	0.098
Bayes	Euler	13.5	0.518
Bayes	Trees	11.5	0.359
Contingency	Euler	3.5	0.020
Contingency	Trees	6.5	0.105
Euler	Trees	7.0	0.119

Wilcoxon tests, high ability

Representational Systems		Wilcoxon *W*	p	
Areas	Bayes	2.0	0.012	
Areas	Contingency	2.0	0.012	
Areas	Euler	15.0	0.669	
Areas	Trees	10.5	0.286	
Bayes	Contingency	10.5	0.164	
Bayes	Euler	0.0	0.004	*
Bayes	Trees	2.5	0.012	
Contingency	Euler	0.0	0.011	
Contingency	Trees	7.5	0.136	
Euler	Trees	10.0	0.164	

L.3 Lightbulbs problem

Friedman tests

Context	Friedman Q	p	
No persona	12.02	0.017	*
Low ability	22.01	0.000	*
High ability	20.83	0.000	*

Wilcoxon tests, no persona

Representational Systems		Wilcoxon *W*	p
Areas	Bayes	13.0	0.478
Areas	Contingency	9.0	0.203
Areas	Euler	7.0	0.120
Areas	Trees	3.0	0.035
Bayes	Contingency	15.0	0.670
Bayes	Euler	6.5	0.055
Bayes	Trees	9.5	0.231
Contingency	Euler	2.5	0.012
Contingency	Trees	5.5	0.143
Euler	Trees	4.5	0.039

Wilcoxon tests, low ability

Representational Systems		Wilcoxon *W*	p	
Areas	Bayes	3.0	0.031	
Areas	Contingency	4.0	0.048	
Areas	Euler	22.0	1.00	
Areas	Trees	4.5	0.039	
Bayes	Contingency	0.0	0.011	
Bayes	Euler	3.0	0.020	
Bayes	Trees	0.0	0.004	*
Contingency	Euler	7.0	0.074	
Contingency	Trees	9.5	0.222	
Euler	Trees	2.0	0.012	

Wilcoxon tests, high ability

Representational Systems		Wilcoxon *W*	p	
Areas	Bayes	6.5	0.074	
Areas	Contingency	3.0	0.034	
Areas	Euler	21.5	1.00	
Areas	Trees	0.0	0.004	*
Bayes	Contingency	21.5	1.00	
Bayes	Euler	5.0	0.039	
Bayes	Trees	2.5	0.020	
Contingency	Euler	7.0	0.074	
Contingency	Trees	2.5	0.028	
Euler	Trees	0.0	0.004	*

L.4 Lightbulbs-equivalent problem

Friedman tests

Context	Friedman Q	p	
No persona	16.02	0.003	*
Low ability	7.43	0.115	
High ability	20.50	0.000	*

Wilcoxon tests, no persona

Representational Systems		Wilcoxon *W*	p	
Areas	Bayes	1.0	0.008	
Areas	Contingency	0.0	0.004	*
Areas	Euler	12.5	0.301	
Areas	Trees	5.0	0.039	
Bayes	Contingency	9.0	0.129	
Bayes	Euler	4.0	0.027	
Bayes	Trees	8.0	0.098	
Contingency	Euler	8.5	0.098	
Contingency	Trees	11.0	0.319	
Euler	Trees	17.0	0.570	

Wilcoxon tests, high ability

Representational Systems		Wilcoxon *W*	p	
Areas	Bayes	1.5	0.012	
Areas	Contingency	4.5	0.055	
Areas	Euler	16.0	0.774	
Areas	Trees	17.5	0.943	
Bayes	Contingency	3.5	0.034	
Bayes	Euler	0.0	0.004	*
Bayes	Trees	0.0	0.004	*
Contingency	Euler	3.0	0.034	
Contingency	Trees	2.5	0.028	
Euler	Trees	14.5	0.608	

Colophon

The typeface in this dissertation is Sabon Next,
with body copy set at 12pt with 15pt leading.
Headings use the display weight.

Captions are set using Gill Sans Nova,
sized to 11pt with 13.6pt leading.

Monospaced type (including code listings)
is set with Menlo reduced to 82%.

This dissertation was prepared with LuaLaTeX
using a custom document class
inspired by the `cam-thesis` class,
The Elements of Typographic Style
by Robert Bringhurst, and
The Visual Display of Quantitative Information
by Edward Tufte.