PRACTICALARTIFICIALCOMMONSENSE

Practical Artificial Commonsense

Benjamin Johnston

Submitted for examination to the
University of Technology, Sydney
on
13 October 2009
in fulfillment of the requirements for the degree of
Doctor of Philosophy (Computing Sciences)

Abstract

While robots and software agents have been applied with spectacular success to challenging problems in our world, these same successes often translate into spectacular failures when these systems encounter situations that their engineers never conceived. These failures stand in stark contrast to the average person who, while lacking the speed and accuracy of such machines, can draw on commonsense intuitions to effortlessly improvise novel solutions to unexpected problems. The objective of artificial commonsense is to bring some measure of this powerful mental agility and understanding to robots and software systems.

In this dissertation, I offer a practical perspective on the problem of constructing systems with commonsense. Starting with philosophical underpinnings and working through formal models, object-oriented design and implementation, I revisit prevailing assumptions with a pragmatic focus on the goals of constructing effective, efficient, affordable and real commonsense reasoning systems.

I begin with a formal analysis—the *first* formal analysis—of the Symbol Grounding Problem, in which I develop an ontology of representational classes. This analysis serves as motivation for the development of a hybrid reasoning system that combines iconic and symbolic representations.

I then proceed to the primary contribution of this dissertation: the development of a framework for constructing commonsense reasoning systems within constrained time and resources, from present-day technology. This hybrid reasoning framework, named *Comirit*, integrates simulation, logical deduction and machine learning techniques into a coherent whole. It is, furthermore, an open-ended framework that allows the integration of any number of additional mechanisms.

An evaluation of Comirit demonstrates the value of the framework and highlights the advantages of having developed with a practical perspective. Not only is Comirit an efficient and affordable working system (rather than pure theory) but also it is found to be more complete, elaboration tolerant and capable of autonomous independent learning when applied to standard benchmark problems of commonsense reasoning.

Declaration

I hereby declare that, except where referenced or acknowledged, this dissertation and the research described within are entirely my own work.

No part of this dissertation has been previously submitted towards the award of a degree at any institution.

Benjamin Johnston

6 October 2010

Acknowledgments

This research project would not have been possible without the support of a number of people who I would like to thank:

- My research advisor, Professor Mary-Anne Williams, who provided me with countless opportunities, support, feedback and advice, far exceeding every expectation, throughout my studies and while writing this dissertation
- My co-supervisor, Associate Professor C. Barry Jay, for generously sharing his advice and time, and bringing greater perspective to my efforts
- My fellow students in the Innovation and Enterprise Research Laboratory, who made the laboratory a productive *and* friendly environment for developing my research project
- Professor Xiaoping Chen (陈小平教授) at the University of Science and Technology of China (USTC), who generously hosted me during a six month visit to his research laboratory, in which I developed many of my ideas
- Guoqiang Jin (靳国强), a student at USTC, who offered helpful suggestions during his undergraduate research project in which he explored connections between my work and a temporal logic (see Section 9.2.2 of this dissertation)
- The students in Professor Chen's Multi-Agent Systems Laboratory at USTC, all of whom warmly welcomed me into their research group despite the language difficulties
- The many researchers, including Professors Anthony Cohn, Peter Gärdenfors and Michael Genesereth, whose kind suggestions, feedback and advice were invaluable in my research
- Dianne Osborne, who generously offered feedback on a draft of this dissertation
- My partner, family and friends who supported me throughout, and shared in my struggles

For your assistance, I am sincerely grateful.

Table of Contents

Extended Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Every rule has an exception. Little about the universe can be said with perfect precision and every seemingly factual statement is clouded with layers of doubt and uncertainty. It therefore comes as little surprise that when software and robots are engineered today as precision rule-driven systems they routinely encounter unexpected situations that highlight the limitations of their rule-based programming. These failures stand in stark contrast to the commonsense of the ordinary person who can effortlessly improvise solutions to the novel challenges that are a routine part of daily life.

Given that computers are themselves fundamentally rule-driven systems, how can we engineer software that goes beyond rules and that has an ability to perform commonsense reasoning like people? An analysis of this question raises issues regarding the nature of the relationships between rules, representations, semantics, intelligence and reality and harks back to the original and grand challenges of Artificial Intelligence: the creation of machines that are capable of independent, intelligent thought.

A natural consequence of both the importance and the age of this question is that there is no shortage of theories, systems, approaches and methods that have been proposed as an answer. However, the problem remains largely unsolved: while computers are becoming faster and smaller at an incredible pace, the software we use remains as unintelligent as ever.

In analyzing the literature to identify an explanation for this lack of success, a striking trend becomes apparent: all existing work takes varying positions on a single, common trade-off. Some proposals are based on powerful representations that ignore or disregard expense while others, conversely, seek cost-effectiveness while sacrificing power.

There is an unexplored, third possibility lying outside this trade-off. Systems that are both powerful and cost-effective may be constructed at the expense of other, less important factors. Instead of constructing commonsense as a special case of general purpose intelligence, commonsense reasoning systems may be designed explicitly for *commonsense* reasoning and by adapting robust and well-understood technologies.

My primary objective in this dissertation is to present a novel framework called *Comirit*[*] that is intended as an effective, efficient and feasible approach to constructing systems that have commonsense. Comirit is based on technologies that are outside mainstream intelligent systems architectures but which are ideally suited to the challenges of commonsense reasoning. In Comirit, these technologies are adapted, re-purposed and integrated into a new commonsense reasoning framework.

In this dissertation, I develop Comirit starting from an analysis of the representational challenges involved in implementing intelligence and commonsense. This analysis leads to a principled method for identifying appropriate constraints on knowledge representation for commonsense reasoning. I use these constraints to guide the adaptation and novel recombination of familiar algorithms and data-structures to the problem of commonsense reasoning. I use multiple evaluations to show the effectiveness, completeness, cost-effectiveness and concreteness of the system.

1.1. Contributions

The primary contribution of this dissertation is to motivate, propose and validate a software framework for artificial commonsense reasoning. Even though I am not the first to propose systems with commonsense (other work is reviewed in Section 2.6), my work is uniquely distinguished by an emphasis on pragmatism and effectiveness through the exploitation of current and well understood technologies. This orientation addresses a gap within existing literature and practice (as I will argue in Section 2.7) and is of genuine independent value because it lends itself to the short- and medium-term development of commercially useful systems, rather than only theoretical systems.

The following is a list of key points of difference between my work and the prevailing approaches in research and practice:

1. **Common sense reasoning, rather than common sense knowledge**

   My interest is to give artificial systems an understanding of how the world actually works: a sense of practical 'know-how', rather than a database of commonly known facts. That is, my interest is in practical skills-based problems such as "How can I safely rescue this person?" or

[*]Standing for 'Commonsense Intelligence and Reasoning through Integrative Technologies'

"Does this spreadsheet make sense?" rather than factual knowledge-based problems such as "Who is the Queen of England?" or "Where do fish live?"

2. **Computational efficiency, rather than computational accuracy or formal completeness**

   While every effort is made to create an accurate and complete framework, my interest is the creation of software designed for real computers and robots. A practical algorithm is therefore preferred to an algorithm that is provably correct but incomputable or requiring computation vastly exceeding modern supercomputers.

3. **Ease of use, rather than excessive intellectual and educational demands on knowledge engineers**

   If a system can only be understood by experienced logicians, then its development and adoption will be constrained by the expense and availability of such specialists. I therefore prefer representations and external interfaces that are readily comprehended by software engineers.

4. **Open-ended architectures, rather than closed and complete systems**

   Even though my emphasis is on short-term development cycles and practical outcomes, I do not intend to design a 'throw-away' or single-purpose architecture. My objective is to create a commonsense systems architecture that can grow and accommodate new innovations in Artificial Intelligence, Commonsense Reasoning and Computer Science as they arise.

5. **Short- or medium-term, rather than long-term development time frames**

   I seek to address the question of what is possible today. I do not intend to propose an approach that requires a multi-decade research program to deploy, nor do I intend to speculate on future technologies or research outcomes that are today merely hypothetical. Instead, I seek to re-purpose and extend existing algorithms towards commonsense reasoning. While research programs with an exclusively long-term outlook may eventually discover ground-breaking algorithms that provide comprehensive solutions, the significant benefits of artificial commonsense reasoning are such that there is value in sacrificing completeness in order to create functional, practical solutions today.

6. **Reusable components, rather than fully autonomous systems**

   My goal is not to design a complete system that operates independently as a 'black-box' intelligence but to develop components that may be integrated into other systems as a reasoning engine or service.

7. **Commonsense, not general intelligence or 'Strong AI'**

   While both commonsense and the problem of creating general-purpose human-like artificial intelligence are related, there are important differences in their scope. Commonsense concerns only restricted domains of naïve world knowledge. This project is a restricted and achievable subset of the overwhelming open-ended challenge that is the creation of general-purpose artificial intelligences.

8. **Pragmatic, rather than idealistic attitudes towards intelligence**

   Both commonsense and intelligence present long-standing philosophical difficulties in their definition, formalization and evaluation. Rather than judging my work against abstract philosophical objectives, I will evaluate progress by using standard benchmark problems and by comparing my work to other approaches. That is, my goal is to create systems capable of reaching similar conclusions to an entity with common sense—I do not intend to agonize over the question of whether the system *intrinsically* possesses common sense.

The theory and artefacts described in this dissertation are also of independent value beyond the commonsense reasoning architecture that I propose. This dissertation therefore offers several secondary contributions:

1. **A new formalization of grounding and the symbol grounding problem**

   A formalization that simplifies the analysis and understanding of the literature in symbol grounding, motivates different approaches to intelligent systems development and provokes significant new research questions in the field.

2. **Identification of representational constraints for pragmatic and effective commonsense systems**

   Having formalized the symbol grounding problem, I use it as a rational framework to identify the general representational constraints of a pragmatic, effective and reusable commonsense reasoning system. These general representational constraints guide development of my primary contribution but may also be applied to the development of any commonsense or intelligent systems with similar pragmatic objectives.

3. **Development of new representations, formalisms and algorithms that have general application beyond the implementation of commonsense reasoning systems**

   The re-purposed and integrated technologies in this dissertation have applications beyond commonsense reasoning. They may be applied to more general intelligent systems architectures as well as in problems that are unrelated to commonsense reasoning but require analysis of many complex scenarios (these are discussed in Section 9.2.5).

4. **Benchmarks and approaches that serve as a counterpoint to existing methods used within the commonsense reasoning community**

   The mere existence of a concrete approach to commonsense reasoning (designed from unique assumptions and goals), where few concrete proposals exist, provides a useful counterpoint that can improve the quality of comparisons within the field.

1.2. Significance

A framework for artificial commonsense reasoning is like a high-level programming language or a powerful database management system (DBMS) in that it is of limited *independent* value but plays a crucial role in *enabling* the creation of new applications. There is little point in an application which reports commonsense trivialities such as 'vegetarians tend to avoid steak-houses.' However, a real market may exist for systems that act on such knowledge to arrange and book an appropriate venue for large groups of people.

Some embedded systems operate within strictly controlled and predictable environments; they have no need for commonsense. However, the vast majority of software or robotic systems must deal with open-ended domains. Any robot operating outside the controlled factory floor must be prepared to deal with novel objects and scenarios. While business software does not need to identify the unforeseen physical obstacles that can trouble a mobile robot, it still needs to adapt to evolving strategies, product lines and unforeseen situations.

Just as any program may be written in assembly language or without a DBMS, it may be similarly possible to *eventually* create extremely flexible and intelligent software by extremely careful software engineering. This would require a labor-intensive process that involves the identification and programming of every rule, exception and exception-to-the-exception of the system's business logic. Instead, artificial commonsense can be used to allow software to reason for itself, to understand its own context and to improvise when it encounters unexpected situations. By eliminating the tedium of defensively programming for every exceptional circumstance, commonsense enables developers to move on, towards the development of vastly more sophisticated systems.

Given that virtually all software operates within open-ended domains, a framework for commonsense reasoning holds significance across the full spectrum of software development. Consider, for example, the selection of potential applications, many of which are found widely as motivation in the commonsense reasoning literature, that appears in Table 1.1 (a–b) (on the following pages).

The potential applications listed in Table 1.1 are not purely speculative: many have been demonstrated in prototype (*e.g.*, [132, 29, 106]). However, I will argue in Section 2.7 that the success of these early prototypes is frustrated by a poor balance between expense and reasoning power. Significant research and development would be required before they reach a level of maturity suitable for deployment in real systems.

The significance of this dissertation follows from a lack of maturity of current commonsense reasoning systems and the opportunity to provide *practical* methods for creating systems that have commonsense. My objectives are pragmatic and oriented towards short-to-medium development time lines: the social significance of this work follows from realizing the benefits of commonsense com-

Benefit	Example Application
Assist human users by identifying and correcting 'commonsense' errors and omissions	A spreadsheet application that detects valid but nonsensical formulas (*e.g.*, computing the difference between a credit card number and its expiry date) Detecting the erroneous entry of a natural person who is 150 years old, while still permitting a legal person (such as a corporation) to be this old
Appropriately respond to outlier or exceptional data	A credit authorization system that automatically recognizes that a credit application by the child of a wealthy or important customer might need to be escalated to a supervisor before it is rejected
Provide intelligent and context-sensitive assistance to users	An online travel agent that infers the user's budget, travel objectives and preferences to recommend hotels and activities that are within the particular means and interests of the customer
Improve knowledge management and sharing	An internal skills register that knows to connect employees in need of a "website" with the organization's "PHP intranet specialist"
Integrate disparate data sources and discover new trends in data	A data mining tool that tracks unique customers across disconnected silos of sales, order and support databases to discover organization-wide opportunities (*e.g.*, high-margin customers of the sales-force may turn out to be unprofitable after ongoing support costs are included)
Automate uncreative and routine 'knowledge work'	A software package that self-configures installation locations, performance parameters, data distribution and interoperation in complex environments A biotechnologist's search tool that can accept vague queries such as "find a list of candidate proteins that have a plausible connection to obesity in the literature" A financial data-aggregation system that understands unstructured financial reports to perform a first-pass judgment call on a business's health, reducing the effort for an investor to identify companies that may match their investment strategy
Increase the autonomy and initiative of artificial systems	Calendar software that attempts to reschedule non-critical appointments if a flight is significantly delayed and the user is consequently likely to arrive at a meeting exhausted and stressed

Table 1.1 (a): Potential applications of commonsense reasoning

Benefit	Example Application
Respond appropriately and exploit novel objects in the environment	A rescue robot that uses commonsense to recognize the danger of entering an electrified pool of water blocking an exit and identify that a nearby bucket could act as a suitably insulated 'stepping stone' in case of dire emergency
Detect spam and other wasteful communications	A messaging infrastructure that can filter out spam and also identify the appropriate recipients for internally sent 'broadcast' emails (For example, an email notification of a server reboot does not need to be delivered to general staff who have already left work for the day or who will be in meetings at the relevant time)
Enable mass customization of digital artifacts	A software system that uses commonsense reasoning to automatically generate legal contracts by identifying appropriate paragraphs from a database of common clauses and then customizing those clauses to the particular needs of the client A publishing house that offers customers unique magazines that are precisely matched to their own individual preferences using an understanding of their interests, the articles that are available and space and layout requirements
Improve both content and user understanding in search engines	A search engine with a commonsense understanding of the semantics of web-pages and the interests of users so that a dietician, a programmer and an investor searching for 'apple' are respectively more likely to find results for the fruit, the computer company's 'developer connection' and the computer company's financial reports

Table 1.1 (b): Potential applications of commonsense reasoning

puting sooner than might otherwise have been possible and the scientific significance lies in the new methods I propose to make this possible.

Note, however, that while a purely disposable short-term solution would still have some significance, my outlook is not short-sighted. The architecture that I propose is open-ended so that it may have short-, medium- and long-term value and so that it may evolve with new insights into commonsense reasoning that will arise from longer-term research projects.

1.3. Thesis and Method

Commonsense is a form of general purpose intelligence with close connections to deep and unresolved philosophical questions concerning intelligence and meaning. There is no consensus on a formal definition and only rudimentary understanding of the workings of commonsense and intelligence. This makes it difficult to apply the empirical methods of the classical sciences.

Ultimately, this dissertation is attempting to demonstrate my thesis that *artificial commonsense can be created within constrained*[*] *time and resources, from present-day technologies*. I first review the literature to show that this thesis has not already been demonstrated (or disproven). To advance the thesis, I then follow a constructive research methodology: drawing design criteria from a needs analysis, I develop a 'theory of design' and evaluate the resultant artifacts to show that the theory does indeed generate instances of software and robotic systems that satisfy my thesis.

In this dissertation I propose two theories[**]:

1. *A theory of analysis* to facilitate an understanding of symbol grounding and to motivate the research program (Chapter 3):
2. *A theory of design and action* that describes how to construct a framework for commonsense reasoning that fulfills the criteria of the research thesis (Chapters 4–7).

The theory of analysis provides a principled approach to analysis and transformation that helps validate the theory of design and action. I combine the theory of analysis with a comprehensive evaluation to assure a high degree of confidence in the correctness of my thesis.

[*]By *constrained*, I mean a multi-year (but not multi-decade) project by a small team of tens of engineers (but not hundreds).
[**]This terminology is taken from the IS research taxonomy of Gregor [61].

1.3.1. Theory of Analysis for Symbol Grounding

Chapter 3 describes a set of 'representational system classes' that constitute a theoretical framework for examining the symbol grounding problem and for analyzing the symbol grounding literature. The theory expands upon Harnad's definition of a symbol system, as presented in his 1990 seminal paper defining the problem [65]. The theory of analysis is developed by formalization and generalization of Harnad's definition, resulting in a taxonomic theory of conceptions of intelligence and intelligent machinery. I demonstrate the value of the theory by showing that it meaningfully categorizes the existing literature on symbol grounding and that it generates insightful design criteria for commonsense reasoning systems.

1.3.2. Theory of Design and Action for Commonsense Reasoning

Chapters 4–7 introduce a design for a framework that I call *Comirit* and which enables the construction of commonsense-aware software and robotic systems. This theory of design is presented through abstract formalisms, object oriented class structures, control flow diagrams and detailed explanations that will enable the reader to independently construct a concrete software artifact capable of performing commonsense reasoning. The requirements of the *theory of design and action* are developed by careful analysis of the design objectives and from the application of the *theory of analysis*. These requirements provide the rationale for selecting and adapting existing technologies to the problem of commonsense reasoning. The theory of design and action is validated by testing the theory on benchmark problems, by comparison with other work and by demonstrating that the theory mirrors the insights of a particular model that I have drawn from the cognitive sciences.

1.4. Scope Limitations

This dissertation reports on a postgraduate research project that is itself constrained by very limited time and resources. Many simplifying assumptions were necessary—these are described and justified throughout the dissertation—and much work was, of necessity, deemed to be out-of-scope for this project. My objective is to propose a framework for commonsense reasoning and perform evaluation sufficient to demonstrate the value of the work. In particular, this project *does not* seek the following outcomes:

- Construction of a robust, marketable system for commonsense reasoning (prototypes are used to develop and validate the theory but full implementation is not carried out)

- Development of a comprehensive knowledge-base of common sense facts, objects and scenarios (only minimal data entry is performed as required to evaluate prototypes)
- Complete formalization or resolution of deep problems of commonsense knowledge or intelligence
- Development of *formal* proofs of the correctness of the system
- Exhaustive *quantitative* analysis and comparison of systems (analysis is performed where possible using benchmark problems but quantitative measurement is extremely challenging)

1.5. A Note on Notation

Formal mathematical notation and concepts are used extensively throughout this dissertation to make explicit and clear the concepts that I also explain informally. However, an in-depth understanding of the formal definitions is not crucial to a high-level understanding of my contribution so they may be ignored by the practitioner interested simply in the implementation of a commonsense-enabled system.

I have chosen to use mathematical formalisms of the Z notation and its mathematical toolkit [159, 79]. I use this ISO-standardized notation because there is, paradoxically, little precise standardization on what logicians refer to as 'standard notation'. While some superficial aspects of the Z notation may cause initial confusion, most symbols of the notation and its underlying principles are very reasonable and correspond closely to the established practice of 'standard' mathematical notation. Furthermore, the computational heritage of this notation makes it well suited to defining functions and operators that are destined for implementation on real computer systems.

Appendix A contains a brief overview of the Z notation and its mathematical toolkit. For more comprehensive coverage, Spivey's reference manual [159] is now freely available online[*] and is an approachable resource.

Where a formal definition is not straightforward, I have provided an informal gloss of the mathematical notation. This is denoted by a lozenge symbol (◊).

[*]http://spivey.oriel.ox.ac.uk/mike/zrm/

1.6. Dissertation Outline

In the next chapter, I present a literature review in which I present the context of this work, discuss some fundamental challenges and introduce related work. The review serves as motivation for my claims of originality and of a research gap in the academic literature and in practice.

In Chapter 3, I provide a formal analysis of the symbol grounding problem and identify a set of representational classes (the *theory of analysis*). This analysis is used to select the design trade-offs and representational constraints that guide the development of the Comirit Framework that I present over the remainder of the dissertation.

Chapter 4 provides a high level overview of the Comirit Framework. The Comirit Framework is the primary contribution of this dissertation: it is a framework that combines logical deduction, iconic reasoning and learning into an integrated system for commonsense reasoning.

Chapters 5–7 are concerned with the detailed specification of the mechanisms of the Comirit Framework: iconic reasoning (Chapter 5), hybrid reasoning (Chapter 6) and autonomous learning (Chapter 7).

In Chapter 8, I use a multiple approaches to evaluate Comirit and my research thesis. I principally use benchmark problems to demonstrate the capabilities of Comirit and compare it against published solutions. I combine this analysis with extrapolation from data, a mapping onto a cognitive model and analysis of the development methodology to build an irrefutable body of evidence.

I conclude in Chapter 9 with a discussion of future work (both short-term and long-term), potential applications (direct and indirect) and a summary of the findings within this dissertation.

Chapter 2

Research Context

The creation of artificial systems with commonsense is closely related to the challenge of general-purpose intelligence. It concerns the development of knowledge and algorithms that, like human intelligence, can be applied to any domain of interest. In this regard, commonsense reasoning draws close parallels to the original ambitions of early Artificial Intelligence (AI) researchers. Indeed, some of the earliest proposals in AI are for systems with commonsense [116].

Commonsense and general-purpose artificial intelligence have remained largely elusive ambitions, falling to the sidelines as problem-specific engineering of 'narrow-AI' solutions has achieved stunning successes in restricted domains. Consider the last decade, for example, in which possibilities that were once only the domain of science fiction have become a reality:

1. In 1997, the 'Deep Blue' chess computer built by IBM Corporation became the first machine to beat the reigning world chess champion [22]. Today, a similar standard of chess is played by off-the-shelf software on standard desktop computer systems [5].

2. By 2006, the iRobot Roomba household vacuuming robot achieved sales over two million units [80].

3. In 2007, six driver-less, fully autonomous robotic vehicles successfully completed a 96 kilometer course through an urban environment, obeying road regulations and negotiating traffic as part of the DARPA Grand Challenge [10].

These systems represent significant milestones in the history of Artificial Intelligence. In the environment for which they were designed, they appear impressively intelligent. They fail, however, in meeting the original ambitions of AI: they lack commonsense and are useless in situations even slightly beyond their original design purpose. Try asking Deep Blue to vacuum the floor!

Despite the inflexibility of narrow-AI systems, the excellence of these systems at the tasks for which they were designed presents a challenge for research in commonsense reasoning. A standard evaluation might consist of a challenge problem and the two systems to be compared: a general purpose commonsense-enabled system and a custom-built solution developed with entrenched 'narrow-AI' methods. Unfortunately, in such comparisons, the custom-built solution will invariably win.

Custom-built 'narrow-AI' systems are optimized for and tested against the parameters of a specific and constrained problem space. Consider, for example, that even the computational hardware of Deep Blue is optimized for chess [22], enabling it to compute the millions of moves required for traditional techniques to outwit Gary Kasparov, the reigning world champion at the time. If Deep Blue is pitted against Kasparov in domains outside of chess, whether it be games like tic-tac-toe or real-life problems like navigating urban environments or vacuuming a floor, then Kasparov would invariably win.

The benefits of commonsense reasoning come from its ability to achieve adequate performance in any situation, rather than excellence in a particular domain. Evaluation of these systems must therefore depend on *open-ended* definitions, benchmarks and criteria. Care must be taken to avoid using definitions that simply enumerate 'sufficient' capabilities. Systems must be developed with an intellectual honesty that does not 'game' the inevitable weaknesses of any given definition.

In this chapter, I begin with a working definition of commonsense and discuss how it may be evaluated without over-constraining its meaning. I then provide a brief review of recent advances in software and robotic systems development to illustrate the techniques of problem-specific engineering solutions and the possible benefits of commonsense. This is followed by a review of the Artificial Commonsense, Artificial Intelligence and Cognitive Systems literature, serving as the motivation for my argument that there lies an opportunity in approaching the problem of commonsense reasoning with a novel emphasis on pragmatism (corresponding to my intended research contributions of Section 1.1).

2.1. Defining Commonsense

Commonsense is a concept that is disarmingly simple to understand but difficult to formally define and even harder to quantify. Generally speaking, commonsense is the non-technical "day-to-day" knowledge, and the ability to apply that knowledge, that an agent uses to understand and succeed in an open-ended environment. Examples of 'commonsense', from among over 80,000 submissions offered by anonymous volunteers to the Open Mind Common Sense Project, include [137]:

- 'Something you find at the beach is water.'
- 'People can ride on bikes.'
- 'Wood is solid.'
- 'An office worker does not want Monday to come.'

These submissions intuitively satisfy our conception of 'commonsense' but is it possible to rigorously justify their 'commonsense' nature?

In the following sub-sections, I explore the difficulties of defining *commonsense*, connections between the concepts of *commonsense* and *intelligence* and how the only practical option today may be to simply leave *commonsense* undefined. That is, I argue that progress can be made today, in lieu of a formal definition, by constructing abstractions and using methods of evaluation that allow for such ambiguity.

2.1.1. Origins

Common sense (from the Greek *κοινή αίσθησις* and Latin *sensus commūnis*) originally referred to the Aristotelian concept of perception—an internal sixth sense that fuses the various impressions of the five senses to create a unity of common consciousness within an individual [138]. During the 16th Century, the term developed its modern meaning, expanding beyond the philosophical understanding of a fundamental internal sense to describe the ordinary understanding or 'plain wisdom' possessed in common by sane adults [138]. In the 18th Century, the term later acquired popular connotations relating to worldliness, tact and 'good sense' [138] but also found technical meaning within the schools of philosophical thought that flourished in Scotland in the 18th and 19th centuries [59, 18] taking the apparent truths (or common sense) of the world as truth, in refutation of Skepticism.

2.1.2. Modern Definitions

In the context of Artificial Intelligence, the meaning of *commonsense*[*] most closely resembles that of 'ordinary understanding'. Commonsense refers to crucial but non-expert world knowledge and competencies that enable an agent to meet the demands of its environment.

Within the Artificial Intelligence community, commonsense can be understood as consisting of the combination of three kinds of capabilities:

1. **Naïve understanding of the principles of our world**

   This is the sort of knowledge that is effortlessly acquired by young children through their interaction with the world. For example: 'unsupported things fall', 'a physical object cannot be in two places at once' and 'new knowledge can often be gained by asking questions'.

2. **Familiarity with the factual knowledge about the physical and social environment that is taken for granted in human interactions**

   This is typically knowledge of the kind that must be taught to young children. For example: 'Sydney is a city in Australia' and 'Peter Pan is a fictional character'.

3. **The ability to utilize naïve understanding and factual knowledge about the world to appropriately respond to the everyday challenges that an agent encounters**

   This is the practical ability to *apply* the first two forms of commonsense knowledge to solve everyday problems and to learn new skills from others. It is closest in meaning to the sense of *worldliness*, *tact* and *good sense*.

These capabilities are given different degrees of attention by different research groups and this is reflected in the multitude of working definitions of common sense within Artificial Intelligence research. Consider the following representative examples:

1. An emphasis on understanding the world can, in particular, be seen in the definitions of work from among the community of researchers interested in logics of action. For example, Mueller

[*]Throughout this dissertation I make a subtle distinction between common sense and commonsense. I use the term common sense as a noun to refer to the inner sense (whether real or hypothetical) that gives an agent the non-expert world knowledge and skills to meet the demands of its environment. I use the term commonsense as an adjective, adverb and noun to refer to the knowledge and skills that a common sense provides. The question of whether a system actually possesses a common sense is a philosophical or cognitive concern, whereas the possession of commonsense capabilities is a practical engineering matter.

[133] uses the following definition:

> 'Commonsense reasoning is a process that involves taking information about certain aspects of a scenario in the world and making inference about other aspects of the scenario based on our commonsense knowledge, or knowledge about how the world works. Commonsense reasoning is essential to intelligent behavior and thought. It allows us to fill in the blanks, to reconstruct missing portions of a scenario, to figure out what happened, and to predict what might happen next.'

2. Cycorp is attempting to construct a massive knowledge-base of factual data so it is no surprise that their definition of commonsense emphasizes the aspect that commonsense is *knowledge* that may need to be *remembered* [139]:

   > '[Commonsense] is the general knowledge that allows us to get by in the real world, and to flexibly understand and react to novel situations. We remember and use (though usually not consciously) heuristics such as "Water makes things wet"; "Wet metal may rust"; "No two objects can occupy the same space at the same time"; and "Inanimate objects don't get diseases".'

   or as the project's founder, Douglas Lenat, is said to have more directly put it [88]:

   > 'Intelligence is 10 million rules.'

3. An emphasis on deduction and reasoning capabilities dates back to McCarthy's [116] original proposal for programs with commonsense:

   > '… a program has common sense if it automatically deduces for itself a sufficiently wide class of immediate consequences of anything it is told and what it already knows'

While such representative definitions are useful for presenting research in the context of their overall ambitions, they lack formal criteria by which a supposed commonsense-aware system might be evaluated. That is, these definitions cannot be used to directly test whether a given system has commonsense. If one system has a rich understanding of simple mechanical, biological and social systems and an ability to predict their behavior (type 1 and 3) but has no factual knowledge, and another system has comprehensive factual knowledge of the everyday world (type 2) but can only perform simple deductive reasoning, then can it be said that either possess commonsense? Can their amount of 'commonsense' be meaningfully compared?

There is, of course, also the question of how much knowledge is 'too much': when does understanding reach the level of *technical* knowledge rather than simply commonsense knowledge? The boundary between commonsense and specialist knowledge is, in part, dependent on context: for example, the knowledge that, in Australia, one should drive on the left side of the road is commonsense within Australia, but represents more specialist knowledge to those who do not reside within the country. For the practical purposes of the development of commonsense reasoning systems, this uncertainty is currently inconsequential: current progress is so limited that there is little chance of creating systems that are *too* intelligent.

2.1.3. Defining Commonsense as a Dimension of Intelligence

Commonsense may be regarded as a specific form of *intelligence* and, indeed, the difficulty of defining commonsense is similar to that of defining intelligence. It is manifestly obvious to a person when a system or other person lacks commonsense or intelligence. However, any attempt to provide a measure or definition will inevitably be controversial and considered too restrictive or too general. It is sometimes joked that among any n Artificial Intelligence researchers, there will be at least $n + 1$ differing opinions about the true meaning of *intelligence*. Consider, for example, Legg and Hutter's compilation [100] of 72 different definitions of intelligence in the research literature, presented as a prelude to a new one of their own.

Indeed, Legg and Hutter [99, 101] attempted to formally define intelligence in terms of reward maximization, sequence prediction and Kolmogorov complexity. Their definition, and other similar attempts at formal definitions, are subject to many of the same criticisms as informal definitions. They are too restrictive, offering low-level conceptions of intelligence that may even exclude the possibility that humans are intelligent agents. Consider, for example, how 'intelligent' human beings are well known to perform worse than simple decision trees on many classification problems ranging from medical diagnosis to estimating wine sale price [13]. A formal definition of intelligence would need to account for the way that human beings perform quite poorly against machines on given well defined problems and the way they struggle to 'reprogram' their own basic desires but yet have problem solving skills that no artificial system can so far match.

In any case, present formal definitions of intelligence translate poorly to commonsense reasoning. Commonsense is a restricted form of intelligence that has a particular emphasis on environmental or contextual knowledge. This knowledge is more specific than the abstractions used within definitions of intelligence. For example, a definition of intelligence as pattern learning may be seen as (awkwardly) subsuming a definition of commonsense if a vast corpus of experience is assumed. While such a definition may be useful in a theoretical analysis, its weakness lies in the need to incorporate an entire corpus of experience into the definition or encompass a definition of the kinds of knowledge that is 'worldly and experiential'. This is difficult to achieve without direct reference to the world (in which case the definition loses its formality) or the enumeration of all knowledge (which is clearly impossible).

Other approaches to creating measurable criteria for intelligence, such as IQ tests and Turing tests, have similar deficiencies that are compounded by application to the creation of artificial commonsense systems. IQ tests and other standardized tests of intelligence emphasize only certain dimensions of intelligence, are typically biased towards cultures or skills [169] and can be easily gamed through study or memorization [156] if the details of the test are known in advance. Similarly, when evaluating artificial commonsense with specific finite sets of tests, it is possible to 'cheat' by developing with the particular challenge questions in mind, rather than the real applications. The Turing test [166], in contrast, offers open-ended scope for evaluating a system but is limited by its reliance on the intuitions of a human examiner and is therefore not an objective or repeatable measure.

It seems, therefore, that not only do the difficulties in defining intelligence bode poorly for attempts to define commonsense but also existing proposals for formalizing intelligence translate poorly to commonsense reasoning.

It may be the case that commonsense or intelligence will not be sufficiently well understood to formulate a measurable definition until there is a working system that possesses commonsense. To date, there have been no proposals for formal definitions of commonsense. In studying and evaluating the literature, it is therefore necessary to find alternatives to formal definitions.

2.1.4. Practical Approaches towards Defining Commonsense

Fortunately, a formal or measurable definition of commonsense is not strictly necessary in order to make progress; it is sufficient to show that a new system is *better* at commonsense reasoning than earlier systems. This can be done by showing that a system offers a strict super-set of an earlier system's problem solving capabilities (and is therefore *at least* as good as the earlier system in all possible respects, including commonsense reasoning) or that the problems it solves are closer to actual problems of commonsense. In this dissertation, I will use these strategies to avoid the need for a definitive definition of commonsense or intelligence.

In Chapter 3, I also assume the existence of a definition of intelligence and thereby reason about intelligence in the abstract. This technique enables intelligence and commonsense to be analyzed without regard to their precise definitions, in a similar way that the difficulty of computational problems may be analyzed with complexity classes even if efficient algorithms are not yet known.

As intelligent beings, people are able to readily identify intelligence and commonsense (or a lack thereof) in other beings. While an appeal to this instinctual recognition of intelligence is fraught with danger[*] and should be avoided, such subjective evaluations are an appropriate tool for comparison where no rigorous alternatives are available. Benchmark and challenge problems are one way of facilitating this process by reducing the subjectivity of an evaluation. A system's performance on a benchmark problem is an indirect measure of the system's overall performance. By considering concrete variations of a problem, one can analyze the abstract qualities of the system under review. In the following section, I discuss how such benchmark problems may be used to evaluate commonsense.

2.2. Evaluating Commonsense Systems

The generality of commonsense presents difficulties when considered as a domain of study. Commonsense is nothing in the particular but everything in the general. Like 'intelligence', it is difficult to find a precise definition that delimits what is meant and that guides evaluation. If commonsense were to be tested through clearly defined challenges, then the winners would invariably be those systems that have been optimized to the particulars of the challenge rather than those with commonsense. Such effects are seen throughout the history of Artificial Intelligence. Consider the stark contrast between the early dreams of an intellectual chess-playing machine and its realization in Deep Blue as a finely-tuned 'idiot savant' that simply exploits brute computational power to test vastly more moves than a human being could possibly consider. Similarly, robot soccer matches in RoboCup competitions do not showcase self-aware robots that have consciously taken up sport but machines that are so delicately fine-tuned to game conditions that even small variations in lighting conditions can cause complete disorientation due to poor localization (thus necessitating lengthy calibration periods [160]).

Open-ended challenge problems serve as better benchmarks for commonsense than precisely controlled rules and environments. Of course, this requires that the benchmark problems are approached with an intellectual honesty, avoiding the natural tendency towards engineering systems that directly exploit peculiarities of a given formulation.

Commonsense reasoning has a long heritage of using examples and benchmark problems to evaluate formalisms. In particular, early research into expressive logics for commonsense reasoning was motivated by such examples. One of these is the now ubiquitous 'Tweety' problem [145]. Given an observation that *Tweety is a bird*, one might reasonably conclude that *Tweety can fly*. If it is later learned

[*]Consider Mr. von Osten's horse, "Clever Hans," that, in the early 20th century, created a sensation by having apparently learned to understand speech and advanced arithmetic. It was eventually discovered that the horse had simply been conditioned to tap the ground with its hooves until it observed subtle physiological changes in an expectant audience [72].

that *Tweety is a penguin* or that *Tweety is dead* or even that *Tweety is a penguin with a rocket pack*, then the original conclusion that *Tweety can fly* must be reconsidered. Such revisions seem natural but are difficult to express in classical logics where additional information can only result in additional consequences, rather than the revision of prior conclusions. The objective of benchmark problems like the 'Tweety' problem was therefore to highlight the peculiar semantics of commonsense reasoning, to demonstrate representational weaknesses of existing formalisms and thereby motivate the development of new formalisms, such as non-monotonic logics (consider for example, [145, 136, 141]).

While problems like the 'Tweety' problem served to motivate early work in new logics, it was later subject to the criticism of Patrick Hayes in his Naïve Physics Manifestos [69, 70] as being too trivial. In recent years, a range of more challenging and open-ended benchmark problems have been proposed and cataloged. Miller and Morgenstern [127] offer one such collection of over 20 problems, including the *Egg Cracking Problem* that is the task of characterizing the following situation:

> 'A cook is cracking a raw egg against a glass bowl. Properly performed, the impact of the egg against the edge of the bowl will crack the eggshell in half. Holding the egg over the bowl, the cook will then separate the two halves of the shell with his fingers, enlarging the crack, and the contents of the egg will fall gently into the bowl. The end result is that the entire contents of the egg will be in the bowl, with the yolk unbroken, and that the two halves of the shell are held in the cook's fingers.'

A comprehensive solution to the *Egg Cracking Problem* would require fusion of many models of the world, including naïve theories of physical phenomena such as position, shape, solidity viscosity, cracking gravity and containment, in addition to theories about the chef's social roles, intentions and physical capabilities. The *Egg Cracking Problem* is, on the surface, quite simple: it may be plausibly solved by a concerted research effort. However, the problem is also sufficiently sophisticated that it steps beyond the shallow 'toy problems' of modeling knowledge about Tweety. While the particular properties of an egg are unlikely to find many applications beyond cooking or farmyard robots, the core knowledge of physical and social phenomena encoded into a good solution to the *Egg Cracking Problem* should be applicable to a large proportion of day-to-day experience.

In his Naïve Physics Manifestos, Hayes [70] proposed that a commonsense reasoning research program should seek formalizations that satisfy the following criteria:

- **Breadth:** It should capture as much of the situation and its variants as possible, rather than being, for example, the simple translation of a particular situation into a set of inflexible equations of motion.
- **Fidelity:** It should be detailed and accurate.

- **Density:** There should be a high ratio of facts to concepts. This is necessary to 'capture the richness of conceptual linking' and to ensure that the concepts themselves are sufficiently 'pinned down' [70].
- **Uniformity:** There should be a common formal framework for all the knowledge.

These criteria, in essence, are questions of adequacy in a given problem domain and do not directly address reusability. McCarthy [120] therefore added a criterion of *elaboration tolerance* to describe the reusability of a formalization. A formalization is said to be elaboration tolerant if it can solve a wide range of variants of the original problem without requiring significant re-characterization or re-formalization. Examples of variants that may be used to test for elaboration tolerance include the simple addition of new information, changing parameters, changing types, specialization, generalization or even a changed perspective in description. The effort required to admit any such variant should be proportional to the degree of its novelty: if a system is elaboration tolerant, a small change should require no or little reformulation.

Consider, for example, some of the 'standard' variations of the *Egg Cracking Problem* [127]:

> 'What happens if: The cook brings the egg to impact very quickly? Very slowly? The cook lays the egg in the bowl and exerts steady pressure with his hand? The cook, having cracked the egg, attempts to peel [the shell] off its contents like a hard-boiled egg? The bowl is made of loose-leaf paper? of soft clay? The bowl is smaller than the egg? The bowl is upside down? The cook tries this procedure with a hard-boiled egg? With a coconut? With an M&M?'

Unfortunately, there are currently no objective or quantitative measures for evaluating breadth, fidelity, density and uniformity or elaboration tolerance. Indeed, one would not expect quantitative measures to exist while commonsense and intelligence remain poorly understood because such measures in aggregate would constitute a formal definition of commonsense itself. These criteria are, nevertheless, valuable because they offer a framework for *systematic* subjective evaluation. Even though the criteria cannot be objectively measured, there are objective attributes of systems that can serve as evidence to support subjective evaluation. Consider the opportunities for evaluation described in Table 2.1.

Finally, while the aforementioned criteria provide a means by which a commonsense reasoning system may be evaluated on its own merits, they do not incorporate any consideration of pragmatism or cost effectiveness. Commonsense reasoning systems must be engineered and the cost of such engineering will influence the net benefits: an effective system built cheaply today may offer greater returns than a perfect solution requiring millions of dollars and many decades of development.

Criteria	Methods of Evaluation
Breadth	Counting test variations that are solved by the formalization without any need for change.
Fidelity	Measuring the accuracy of query answers against the known ground truth.
Density	If the formalizations are similar, this can be estimated by a simple comparison of the fact to concept ratios. When formalizations are vastly distinct, it may be estimated by considering density as a complement to fidelity: density may be estimated by counting the closely related (but distinct) test variations which cannot be distinguished within the formalization.
Uniformity	Counting the number of primitives, structuring rules and idioms that are used throughout the formalism.
Elaboration Tolerance	Measuring the time of human labor involved in accommodating variants on a problem. Posing a broad range of tests and counting the numbers of failures.

Table 2.1: Approximate measures for evaluating commonsense reasoning systems

In understanding the costs of a commonsense reasoning system, there are three major phases to consider:

1. The initial construction of the underlying systems and framework and the core knowledge engineering for the platform.
2. The adaptation and configuration of the platform to a specific problem or scenario.
3. The ongoing costs involved in operating, maintaining and supporting a deployed reasoning system.

For each of these phases, there are a range of costs to consider:

- The effort involved in software engineering and the associated gathering of requirements, design, testing and maintenance.
- The effort involved in populating the system with commonsense knowledge and the associated design, elicitation and maintenance of this knowledge.

- The time and financial costs associated with hardware, computation and the training environment for the system's learning.

There is often a straightforward way of evaluating these remaining dimensions of cost: recording the money and time expended during development. However, sometimes such measurement may be complicated by external factors. Consider the effects of large numbers of volunteers either teaching commonsense knowledge (*e.g.*, Open Mind Commonsense [105]) or being directly involved in the software development process (*e.g.*, OpenCog [66]). While volunteers offer their services free, there is a cost to society in the loss of productivity and opportunity cost as those volunteers' efforts are diverted from other worthy causes. The matter becomes further confounded if training occurs in the context of games. Engaging games may offer social benefits not only through entertainment but also by generating goodwill towards research efforts. Since the objective of this work is of a technical nature, rather than the deep exploration of ethics, I will emphasize the direct financial costs and leave the detailed consideration of such externalities as future work.

2.3. The Rule-based Legacy

In Section 1.2 of the introduction, I listed some applications of commonsense reasoning. There remains the question whether these applications might benefit equally from similarly concerted effort of development in the *status quo* (*i.e.*, without commonsense reasoning). That is, how well do prevailing technologies and software engineering methods fare against the objectives of commonsense computing?

Consider, for example, a hypothetical spreadsheet application that checks for semantic errors in formulas. It is highly unlikely that a user would intentionally attempt to subtract a credit card's expiration year from the credit card number and so it might be preferable for a user-friendly application to warn the user of the semantic confusion (indeed, the user may have instead meant to subtract the current year from the expiration year as part of a check for an expired card). In a commonsense-enabled spreadsheet, the commonsense reasoning layer would draw from knowledge of credit cards and typical uses of a spreadsheet to determine the semantic mismatch in the subtraction.

Without commonsense reasoning, the credit-card semantic error may be detected far more efficiently if it is encoded as regular expressions:

```
warn_invalid(A - B) :-
      get_column_name(A) matches '.*credit card.*',
      get_column_name(B) matches '.*expiry.*year.*'.
```

The problem with such an encoding is that while it is effective for any given error, there are far too many possibilities to enumerate; the aggregate cost of attempting to encode all possible errors becomes exorbitant (let alone the infeasibility of attempting to foresee all the possible uses and misuses of the spreadsheet). Furthermore, the context might require an override of hard-coded rules: it may be obvious that the user is attempting to generate a checksum for credit card numbers, generate lotto numbers or perform a numerological game.

The aggregate cost of developing such rules might be reduced by testing for more general kinds of problems. For example, the user could declare data types for all values so that dimensions of units can be checked (*e.g.*, [1]). However, such approaches still lack the ability to understand the context of the situation and detect commonsense errors. It is perfectly valid, as far as unit dimensions are concerned, to compute a coefficient by multiplying the Australian urban speed limit by the Catholic pope's age and then dividing by the average height of an adult male elephant. However, there are very few situations in which this would result in a useful value.

The legacy methods of software engineering are therefore lacking when evaluated with respect to the evaluation criteria of Section 2.2:

- **Breadth.** Each rule is applicable only to a narrow situation. It is extremely expensive to achieve breadth through manual engineering.
- **Fidelity.** While fidelity is generally a strength of manually engineered systems, it is difficult to predict all exceptions to a rule and to accurately identify 'out-of-scope' situations for which fidelity will suffer. In this regard, there remains ample opportunity for improvement.
- **Density**. In the rule-based paradigm, each error is independently analyzed and encoded as separate rules. Rather than, for example, drawing intermediate inferences from a dense axiomatic kernel of commonsense, each situation is explicitly translated into separate rules.
- **Uniformity.** While some uniformity can be gained by using a common description language for expressing business rules or semantic rules, it is unrealistic to expect legacy systems to be highly uniform because they do not make use of models of the world that act to unify knowledge representation.
- **Elaboration Tolerance.** Legacy systems perform very poorly in this regard; slight changes to the inputs will prevent rules from firing.
- **Costs.** While the costs for a small number of rules is very small, attempting to use rule-based programming methods to exhaustively implement all the features that a commonsense enabled system is expected to have would be a very expensive exercise.

In a controlled world of unchanging environments, or where human intervention can be counted on, these limitations present little problem. The reality, however, is that for systems to become more useful and autonomous, they need to understand their broad operating environment and be able to reason with commonsense. A vacuuming robot will waste energy if it cannot reason that on a dusty day it should check that the front door is closed. A powerful robotic soccer player could endanger life if it does not recognize an unexpected streaker as a human non-contestant. A driverless car needs to avoid or respond to all dangers, even ones that it has not been explicitly programmed to recognize. Consider, for example, the DARPA Urban Challenge 'Team Jefferson's' inability to identify railroad barriers because they were not clearly affixed to the ground [9].

The use of commonsense does not preclude fine optimization. Traditional problem-specific engineering can still be used in the context of a commonsense reasoning framework. Indeed, the creation of such hybrid systems is a goal of this dissertation. A hybrid system that makes cooperative use of a commonsense-enabled reasoning service can use commonsense to continue performing satisfactorily in novel situations but invoke task-specific processes when they are deemed appropriate.

In fact, such hybrid systems may revolutionize software development. The Pareto rule of thumb [115] in software development captures the insight that the first 80% of most commonly used functionality often consumes just 20% of development costs, with the remaining 20% of a fully functional solution absorbing 80% of the costs. Commonsense reasoning systems have the potential to dramatically change this equation, saving up to 80% of development costs*: resources may be devoted to the most important functionality of the system, leaving the long tail of obscure functionality for the commonsense reasoning system to automatically determine appropriate default behaviors. These benefits may be ongoing as commonsense would also enable a system to adapt to a changing environment, rather than requiring expensive maintenance.

Software engineering as a discipline has long been concerned with solving the problems of brittleness and expense of software [19], problems that are closely related to the evaluation criteria for commonsense reasoning systems. My assumption is, however, that traditional software engineering techniques are doomed to costly failure. Ultimately, commonsense and general purpose computing will be more successful than the continuation of the prevailing processes of software development.

Following is a list of some major trends in software engineering that are promoted in the popular media as a solution to the brittleness of software, the challenges of autonomous robotics and the complexity of user interfaces. The following systems and methodologies therefore are motivated by goals

*I am, of course, making a very bold and potentially controversial claim about a 'silver bullet' for the software crisis. There remains much work before commonsense reasoning or artificial general intelligence reaches this level of maturity. Change is likely to be incremental in the same way that computer programming languages have, over the years, become increasingly high-level and productive for applications development.

similar to work in commonsense reasoning but that I regard as the legacy that commonsense-aware computing is intended to supersede.

- **Semantic Web and Web 3.0**

  The Semantic Web is a 'common framework that allows data to be shared and reused across application, enterprise and community boundaries' [73] and is sometimes referred to as Web 3.0. It is, in a sense, a distributed database and middle-ware designed to operate without a fixed schema by representing semantics alongside data. The semantic web is intended to allow heterogeneous systems to interact, exchange data and orchestrate complex processes on an ad-hoc basis. While this is a significant step towards solving interoperability challenges, the success of the Semantic Web is constrained by the ability of software engineers to properly specify the semantics of operations and goals.

- **Rule Based Programming and Model Driven Development**

  While rule based programming and model-driven development are not solutions to the problem of software brittleness and do not change the methods of software engineering, they help reduce the cost of development and maintenance so as to reduce the impact of the brittleness. In rule-based programming, business logic is explicitly expressed in a domain specific language rather than implicitly via source code, thus simplifying the development and maintenance of large sets of rules. Similarly, in model driven development, the design or model of a system is directly used as an integral part of the executing system, thereby more directly exposing and representing the behaviors of a system and allowing them to be more rapidly evolved in a changing environment.

- **Data-mining and Business Intelligence**

  While not methods of 'solving' the brittleness of software, data mining and business intelligence tools offer capabilities that may be superseded or improved by commonsense reasoning. In data mining and business intelligence, the objective is to discover novel patterns that provide a competitive edge. In these tools, statistical measures of novelty, significance and uniqueness stand in as a surrogate for the commonsense intuitions that a human analyst might use to determine if a pattern is interesting or can be exploited for commercial gain.

- **Web Search**

  While modern web search engines do not have a deep understanding of users' queries, they simulate 'understanding' by powerful statistical analysis, analyzing hyper-links and computing matrix decompositions over the sets of words in web pages (such as with the Latent Semantic Analysis). This approach is limited by the superficial underlying models: a web search for 'how to *avoid* weight loss' will still return advice on how to *lose weight* by avoiding food, due to the preponderance of dieting advice available on the internet.

- **Language Processing**

  As with web search, modern tools that analyze natural language texts—spell checkers, e-mail spam filters, voice-recognition systems and automated translators—typically use shallow statistical models of text, resulting in brittle systems. For example, if a known friend forwards a spam or phishing e-mail for critique (*e.g.*, remarking on the novel or amusing language used), an aggressive spam filter may still mark the message as spam due to the presence of spam-like keywords.

- **Imaging and Perception in Robotics**

  While shallow image recognition and mapping techniques such as SPIN [83] and SIFT [110] are highly effective in many restricted domains, they are unsuited to situations in which contextual understanding is important. For example, these shallow techniques do not make allowances for detecting nearby edges that would imply an observed object is simply a picture that has been printed on a larger sheet of paper.

2.4. Early History and Context

The idea of creating computer programs with commonsense dates back to the earliest years of Artificial Intelligence research. John McCarthy, in his seminal 1958 paper [111], introduced a hypothetical system he called 'Advice Taker'. It was to have the ability to be taught (*i.e.*, given advice) not only knowledge about the world, but also about how to reason with that knowledge. McCarthy outlined preliminary ideas about how symbolic formal representations and logical deduction may be used to achieve such sophisticated reasoning. To this day, Advice Taker remains unimplemented: the proposal requires logics, representation schemes and computational resources that remain visionary.

McCarthy's original proposal set the tone for much of the early work in commonsense reasoning. By translating problems into symbolic representations and then searching or manipulating those symbolic structures, early systems achieved stunning results in a short time. Consider Bobrow's STUDENT system [14], one of the early 'triumphs' of Artificial Intelligence research. STUDENT could automatically answer mathematical word problems expressed in the plain English typical of a high-school textbook. When prompted with the question: "The gas consumption of my car is 15 miles per gallon. The distance between Boston and New York is 250 miles. What is the number of gallons used on a trip between New York and Boston?" it correctly responds with, "The number of gallons of gas used on a trip between New York and Boston is 16.66 gallons." The early development of mechanical systems capable of solving problems that challenge even school students was justifiable cause for optimism.

It was soon realized, however, that this early optimism was largely misplaced. As Dreyfus [36] argued, the early successes of systems like STUDENT were based on their parrot-like behavior: the systems manipulated words and symbols without a deeper understanding of meaning. For example, if STUDENT were asked how many gallons would be used over the 10000 miles between New York and Sydney, it would diligently report 666.66 gallons without regard for the impossibility of driving across the Pacific Ocean. The system has no conception of driving, of locations, of distance or time; it merely makes a series of guesses about the applicability of pre-programmed formulas based on the grammatical structure of the sentences and the textual similarity to variables in formulas. Indeed, the 'guesses' that it does make are generally correct only by virtue of the fact that the problems presented to STUDENT are so small that it is difficult to guess wrongly.

It also became apparent that classical logics correspond poorly to human decision-making and intuition. Classical logics neither allow for decisions to be revised in the face of new information, nor for effective reasoning under uncertainty or incomplete knowledge. In fact, it proved to be difficult to express even the seemingly elementary knowledge that most things in the world are unchanged—that the universe doesn't, for example, transform into a blueberry when your eyes are closed. This is a problem of relevance and inertia known more generally as the frame problem. These issues prompted the development of a range of non-monotonic and action logics [122]. More foundational criticism of the applicability of any kind of symbolic logic or computable mechanism was raised in various thought experiments such as the Chinese room argument [152] and the symbol grounding problem [65]; I will discuss these further in Chapter 3 but note that these thought experiments remain unresolved.

In recent years there has been resurgent interest in artificial commonsense reasoning, sparking many projects that are described in the following sections of this chapter. While it is beyond the scope of this dissertation to explain the renewed interest in artificial commonsense, my speculation is that it has come about due to the convergence of a number of factors:

1. Patrick Hayes' efforts, through his Naïve Physics Manifestos [69, 70], to refocus the commonsense reasoning community on the problem of actually building large scale commonsense systems: solving problems of scalability, rather than worrying about the philosophical nuances that arise in small 'toy' systems.

2. The ongoing efforts of the Cycorp to spark public interest in commonsense reasoning through the open-source release of their reasoning engine [30].

3. The maturity of robust and expressive non-monotonic and action logics.

4. The availability of extremely large corpora from which commonsense knowledge might be automatically learned (*e.g.*, the world wide web).

5. The growing power of desktop computers and the ability to store gigabytes of data for less than $1.
6. A growing understanding of the need for modeling the *semantics* of data, as exemplified by research into the Semantic Web.
7. The culture of collaboration, openness and public contribution of knowledge fostered by the open source movements and the success of large scale collaborations such as Wikipedia.
8. A better understanding of the workings of the human brain and its computational abilities and limitations.
9. Renewed funding opportunities after a long 'AI winter' [16].
10. Increased user expectations, stemming from greater computer literacy and popular representations of 'state of the art' computers in movies.

The remainder of this chapter will be concerned with understanding the landscape of current work in artificial commonsense reasoning and related areas.

2.5. Analyzing Current Systems

In the following section, I provide a comprehensive review of work in commonsense reasoning and other related bodies of work including intelligent user interfaces, intelligent systems architectures, informal reasoning and artificial general intelligence. While not all of this work is explicitly concerned with commonsense reasoning, it shares a common vision of creating artificial systems with aspects of human-like intelligence.

Some of these projects have ambitions and time frames far beyond the scope of this dissertation. I review them here for completeness and because they may be considered 'difficult' ways of achieving artificial commonsense. For example, simulating the entire human brain would be one way—a very costly and roundabout way—of enabling a computer to perform commonsense reasoning. Indeed, it may be the case (though, I believe unlikely) that comprehensive artificial intelligence is such a difficult problem for humankind to grasp, that our best chance is to solve it by such brute-force whole-brain simulations.

A consequence of the significant challenges in commonsense reasoning is that most work in the field is immature and speculative. This presents a threefold set of challenges when attempting to objectively review the field or compare systems:

1. There are few opportunities for empirical testing because very few systems have begun implementation or progressed beyond simple proof-of-concept.
2. Those systems with robust concrete implementations that could be empirically tested are not necessarily the most promising or interesting work in commonsense reasoning.
3. Some proposals are so speculative that it is not only difficult to judge how well an implemented system might perform but there is also insufficient evidence to judge whether it might work at all. Indeed, assumptions made by different research groups contradict one another so it is impossible for every project to succeed as they are currently envisioned.

In lieu of objective means of evaluation, I will review the literature by combining a descriptive review of the context and technology with a subjective evaluation per a set of evaluation criteria.

For each technology, I first describe the context of the work:

- **Problem.** What are the classes of problems that the system is intended to solve (or is the system designed to 'fake' intelligence with subtle word games)?
- **Motivation.** What is the core value that motivates the research program (*i.e.*, the particular idea, value, technology or methodology that provides the unique guiding perspective)?

I then describe the specific technologies that are proposed to enable the implementation on a real computer system:

- **Representation.** How are knowledge, queries, situations and internal states stored on a computer and what can be expressed?
- **Reasoning.** How are the representations manipulated in order to solve a problem?
- **Population.** How are the underlying representations populated with initial knowledge of the world?

Finally, I analyze the strengths and weaknesses of the technology, according to the subjective criteria of Section 2.2. In most cases, there are no concrete implementations to evaluate, so this task is an estimate based within the underlying assumptions of the technology itself (irrespective of the plausibility of those assumptions).

The evaluation criteria are as follows:

- Breadth
- Fidelity
- Density
- Uniformity

- Elaboration tolerance
- Construction cost
- Adaptation cost
- Operation cost

To this list, I add one dimension to characterize the degree of uncertainty or speculation involved in evaluation:

- Confidence

A project has more *confidence* if there are experiments, studies, proofs, functional prototypes or other artifacts that provide concrete evidence of the correctness of the technology. Note that I do not intend to *personally* characterize the likelihood of a theory's success but rather to characterize the degree to which a theory or system's success is contingent on the truth of unproven assumptions.

For each project, I identify the strengths and weaknesses with respect to the nine evaluation dimensions. Other reviewers may disagree with my precise assessments. However the general theme of the evaluations are unlikely to be controversial. For example, if achieving human-level artificial intelligence is our objective, then few would disagree that a full molecular simulation of a complete human being would be ideal with respect to most of the functional dimensions, although the costs of development associated with such an exercise would be exorbitant.

2.6. Analysis of Current Approaches

I present here an analysis of the broad spectrum of approaches in the commonsense reasoning and related bodies of literature. I have grouped work according to what I view as the main paradigm or assumption of the research:

1. Universal theories of intelligence
2. Logical learners
3. Logical formalisms
4. Semi-formal methods
5. Cognitive architectures
6. Fuzzy reasoning
7. Human-inspired mechanisms

8. Evolutionary learning
9. Corpus-based methods
10. Connectionist methods
11. Biological simulation
12. Faking it
13. Rejecting it

Of course, there is a certain amount of overlap between these groups and, again, other reviewers might prefer to split or combine some of these categories. My goal is not to provide a definitive classification but simply to offer a review of the different kinds of research that has informed the work that I describe in this dissertation.

2.6.1. Universal Theories: AIξ, AIξ^{tl} and Gödel Machines

Hutter's [76] AIξ (or AIXI) is a theoretical algorithm for solving all computable problems of (artificial) intelligence. Queries in AIξ are expressed as sequence prediction problems and operate by comprehensively searching the space of all possible Turing-complete programs. When it finds a program, with maximum likelihood against a Solomonoff prior, that correctly predicts the input sequence, it uses that program to predict outputs. That is, the algorithm uses the shortest program that generates the seen input for predicting unseen inputs. Such an algorithm is universal in the sense that it is a theoretically optimal solution to any computable problem (within a constant factor [77]) but suffers a severe practical limitation in that it is uncomputable (due to the size of the constant factor).

Hutter's AIξ^{tl} [76] is a computable approximation to AIξ that uses the same search procedure but enforces computability by imposing time and space bounds when testing candidate programs. Though AIξ^{tl} is theoretically computable, solutions to even trivial problems would require computational resources vastly exceeding real computer systems.

Gödel machines [150] are similar to AIξ in that they are theoretical universal problem solving machines that use proof search to solve problems of intelligence. Gödel machines are self-referential, allowing the system to search not only for a proof of a query but also for possible enhancements to its own programming in order to improve proof speed.

Problem: Any computable problem that can be expressed as a sequential decision process

Motivation:	A formal result based on integration of optimal decision theory and Solomonoff induction
Representation:	Universal Turing-complete languages
Reasoning:	Theoretical computation on Turing-complete machines (large-scale proof searches are conducted over the space of a Turing-complete language)
Population:	All requisite knowledge is assumed to be encoded in each query (a computation of human-like commonsense would therefore assume an input sequence incorporating an human-like lifespan of sensory experiences)
Strengths:	**Breadth, Fidelity, Density, Uniformity, Elaboration tolerance**
	The model is a simple, formally-provable, universal, general intelligence.
Weaknesses:	**Construction cost, Operation cost, Confidence**
	The model is incomputable or, at best, requires implausibly massive computing resources. As such, there is little evidence that it could be used in practice.

2.6.2. Logical Learners: Advice Taker

McCarthy's original proposal for programs with commonsense [116] was centered around a system called Advice Taker that could accept not only knowledge about the world, but also about problem solving skills that lead to self-improvement. Advice Taker was envisioned as a system in which knowledge and strategies are encoded using formal symbolic logics and reasoning is conducted by a (self-improving) process of logical deduction.

While the legacy of Advice Taker can be seen in much of the early work in commonsense reasoning, the system was highly speculative and remains unimplemented to this day.

Problem:	Complex deductive problems of day-to-day life that can be expressed as abstract symbolic formulas (including the reflexive capability of self-improvement and learning-to-learn)
Motivation:	The apparent 'ease' by which commonsense and real world knowledge can be formally described in symbolic forms and subsequently reasoned about through deductive techniques

Representation: Formal symbolic logic

While AdviceTaker predates McCarthy's situation calculus ([112], see also Section 2.6.3), McCarthy's examples use a notation that resembles situation calculus.

Reasoning: Formal logical deduction via automated theorem provers that have the recursive capability to be taught new heuristics for proof search and problem solving

Population: Trained experts express knowledge and techniques through logical formulas

Strengths: **Fidelity, Uniformity, Elaboration tolerance**

Advice Taker is a general purpose framework and can be used to accurately describe any domain that can be solved with symbolic logic. The success of this program, however, does depend on the effort and care put into 'advising' the Advice Taker (see weaknesses).

Weaknesses: **Construction cost, Operation cost, Confidence**

A great deal of ontological engineering is required before the system can bootstrap its own learning. The system is highly speculative in that it only outlines a possible representation and, to this day, there is little work on how a coherent ontology of knowledge, reasoning and meta-reasoning may be formalized for general purpose commonsense.

2.6.3. Logical Formalisms: Situation Calculus and Action Logics

The precursory ideas of McCarthy's Advice Taker were developed by McCarthy and Hayes [121] into the Situation Calculus: a logical formalism for reasoning about real-life situations. These ideas later motivated the development of the Event Calculus [93] and a range of other action and fluent-based calculi. In such formalisms, events, world-states, actors, properties and physical entities are denoted by the objects, terms and functors of a symbolic logic. These formalisms therefore operate over an assumed domain of pre-categorized knowledge and actions are defined in terms of the changing properties of an environment. For example, the following expression may be used to express that *when someone drops a book in a room, that book can then be found in that room* (Section 6.47 of [133]):

```
HoldsAt(InRoom(a,r),t) ⇒
      Initiates(LetGoOf(a,o), InRoom(o,r), t)
```

Action logics have found wide application in intelligent systems. For example:

- General game playing, wherein systems must discover strategies and compete in novel games without human oversight [53]
- One-touch-mission systems, requiring the ability to modify plans on the fly when obstacles or problems are encountered [179]
- Intelligent characters in computer games that must communicate with human beings and independently discover plans to achieve their own ends [112]

Problem:	Complex deductive problems about actions and behaviors within particular domains that can be expressed as abstract symbolic formulas
Motivation:	The ability to formally describe commonsense and real-world knowledge about the temporal effects of actions in succinct, abstract, symbolic form
Representation:	Formal symbolic logics designed for describing situational and temporal change
Reasoning:	Formal logical deduction
	Efficient systems have been implemented as resolution-based extensions of PROLOG [42] and as specialized planners [90]. General-purpose, automated theorem provers may also be used. In more theoretical work, complex reasoning is conducted entirely by hand, under the assumption that these could be fully automated one day.
Population:	Manual knowledge engineering
	The expense of manual engineering is justified by the presumption that only a small number of axioms can be used to describe a large number of situations, and that a given domain need only be axiomatized once and then reused in many systems.
Strengths:	**Fidelity, Density, Uniformity**
	Action logics have, to date, achieved their greatest successes in domains that can be precisely described by a small number of dense rules.

Weaknesses: **Breadth, Construction cost, Adaptation cost, Operation cost**

Highly trained experts are required to manually convert the broad experience of human knowledge to logical form (without introducing logical inconsistencies). It is unclear not only whether it is possible to formalize sufficient world knowledge in unbounded domains but also whether it is economically feasible to do so. If this approach is extrapolated to large-scale knowledge bases, there are clearly issues regarding decidability and computational complexity associated with formal logics.

2.6.4. Semi-Formal Methods: Cyc, Project Halo and Semantic Web

The Cyc project [104] (and its recent open-source spin-off, OpenCyc [30]) stands as one of the most well known and ambitious projects within Artificial Intelligence. It has undergone over 20 years of research and development in its quest to build software with commonsense [178]. Cyc, and projects like it, are inspired by classical symbolic logics but pragmatically embrace inconsistency and computability so as to cost-effectively construct broad (but comparatively shallow) knowledge bases. Cyc is designed to reason in a way that resembles human 'rational' thought about typical human mental concepts (including their inherent inconsistencies) and uses representations and deductive mechanisms that are tuned for 'typical' commonsense reasoning problems.

In Cyc (and OpenCyc), symbols are chosen to correspond to the concepts that human beings use in thought and discussion but are carefully engineered so as to fit within a logically consistent metaphysical ontology. Knowledge in Cyc is bundled into 'microtheories' that allow locally consistent but globally inconsistent knowledge. 'Microtheories' improve the modularity of the system and avoid catastrophic propagation of inconsistency across an entire knowledge base (as classical logics are prone to suffer), thereby allowing the system to productively reason about Santa Claus despite knowledge of his non-existence.

A range of other, more recent, projects resemble Cyc's attempt to create large scale ontologies of human knowledge. Project Halo is one such umbrella effort to encourage the development of a "digital Aristotle" [44]. Three competing teams attempt to encode standard college-level Advanced Placement (AP) textbooks into an ontological knowledge base and the systems are tested with typical examination questions (that are translated into logic) [44].

The Standard Upper Ontology [135] (SUO) and other projects like it (*e.g.*, [74, 114]) are attempting to build general ontologies of knowledge that admit detailed knowledge from any discipline and are intended to serve as a potential representational core of the Semantic Web. In fact, the long term ambitions of the Semantic Web are for universal access to machine readable data and it is, therefore, conceivable that, when the Semantic Web incorporates vast stores of knowledge and powerful deductive capabilities, it may demonstrate a practical form of intelligence.

Problem:	General purpose deduction and reasoning from factual knowledge of the world
Motivation:	The automation of human 'rationality' (where rationality is seen, in its idealized form, as a deductive mechanism)
Representation:	Knowledge is principally represented in relational form (however, many systems also allow for the expression of knowledge in subsets or extensions of first order logic)
Reasoning:	Formal (but not necessarily classical) logical reasoning systems that have been optimized for typical or average case scenarios
	Such reasoners are typically designed for efficient operation on typical or average problems on large knowledge bases, rather than for worst-case scenarios.
Population:	Manual knowledge engineering
	In the case of Semantic Web technologies, all users can contribute knowledge and so this cost could be assumed to be shared over a large population.
Strengths:	**Breadth, Uniformity**
	These systems allow for broad reasoning over many factual knowledge domains, provided that the problem principally concerns ontological knowledge that is easily expressed in their uniform languages.
Weaknesses:	**Fidelity, Density, Construction cost**
	The informal logics of these frameworks mean that it is difficult to express sophisticated and precise mathematical knowledge (especially where deduction requires complex analysis of dense domains, rather than manipulation of broad factual knowledge). Cyc's ongoing 20-year research and development program illustrates the enormous costs associated with these methods.

2.6.5. Cognitive Architectures: ACT-R and Soar

Starting since the eighties, a number of cognitive architectures have arisen as computer models of psychological and experimental data, the most prominent of which include Soar [96], ACT-R [3], 4CAPS [87] and EPIC [89]. These systems are fundamentally frameworks or languages for constructing production systems (that is, they are rule-based systems). Despite the vast differences between the machinery of human thought and production systems, these cognitive architectures are designed to share an architectural correspondence with conceptual models of the human mind. They incorporate structures and capabilities that resemble the abstract functions of the human mind: working memory, long term memory (including declarative and procedural knowledge), perceptions, reinforcement learners, visual imagery and even cycles of action-selection that mimic human reaction times.

The psychological inspiration of these architectures mean that they serve as both architectures for constructing intelligent systems and experimental models for understanding human intelligence. In particular, the ACT-R architecture has been applied to numerous psychological experiments in an attempt to provide a computational account for human behaviors (*e.g.*, [3,46]).

Thus, these systems are architectures for constructing rule-based systems within a paradigm inspired by human cognition. In this sense, they share many of the difficulties of other rule-based and semi-formal methods. While applying a system like Soar, or any other cognitive architecture, will simplify the development of systems that use knowledge in an intelligent and 'human-like' manner, successful systems development requires careful development of production rules that will bring about appropriate behaviors in the particular domain.

Problem:	A two-fold problem of providing a computational account of human intelligence and constructing computational systems with human-like behavior
Motivation:	The translation of human cognitive facilities and structures to the computational model of production rule systems
Representation:	Knowledge is principally represented in a frame-like symbolic form using system-specific symbolic languages; numeric and visual data may also be used to facilitate reasoning (such as tracking reinforcement learning, spreading activations and visual manipulations)
Reasoning:	Production systems: rule selection and application, including the use of meta-rules for the resolution of inconsistencies, non-determinism and 'impasses'
Population:	Manual engineering of production rules in the provided architecture or language
Strengths:	**Uniformity, Operation cost, Confidence** These systems provide high-level facilities that allow the system to exploit its knowledge bases in an efficient and human-like manner. Furthermore, the many successful applications of such systems to difficult problems lend evidence to the likely success of the approach.

Weaknesses:	**Breadth, Elaboration tolerance**
	These systems are ultimately production-rule systems that face the same challenges as established practice in software engineering, including limited elaboration tolerance. While these systems are offered with scalable tools for development, the manual software-development processes continue to pose the challenge of expressing complex knowledge accurately and elegantly across broad domains.

2.6.6. Fuzzy Reasoning: NARS, Probabilistic and Bayesian Logics

Formal symbolic methods generally assume one truth for any given query and do not admit uncertainty or multiple conclusions (without convoluted constructions). One common reaction to the inflexibility of symbolic logic is to directly encapsulate probability theory or Bayesian reasoning into a deductive framework (*e.g.*, [111, 57, 126]). Such approaches extend traditional logics so that symbols not only denote fixed concepts but also can stand for rich probability distributions.

The direct translation of statistics to a deductive framework will necessarily result in certain paradoxes and counter-intuitive results that follow from probability theory (*e.g.*, Simpson's Paradox, the Monty Hall Problem and the Two Envelopes Problem). Some systems, such as Wang's Non-Axiomatic Reasoning System (NARS) [171], avoid these problems by drawing inspiration from probabilistic reasoning but explicitly rejecting axioms of probability theory. NARS ranks the confidence of beliefs using measures that are intended to resemble probabilities but do not necessarily have any particular, strict interpretation. The values are manipulated by intuitive (but non-axiomatic) deduction rules that are designed to resemble human reasoning and biases in uncertain situations.

Problem:	Complex deductive problems of human day-to-day life that can be expressed as abstract symbolic formulas and that may have nuanced, probabilistic or uncertain answers
Motivation:	Combining the deductive power of classical and default logics with principles from probability theory
Representation:	Symbolic terms with numerical confidence values and distributions
Reasoning:	Systematic deduction based on probability theory, Bayesian inference or other numerical deduction rules

Population: Experience based learning (though knowledge may be seeded by a set of symbolic terms and, in addition, universal logical truths or tautologies may be manually encoded as knowledge with 100% confidence)

Strengths: **Fidelity, Uniformity, Elaboration tolerance**

This approach has many of the same benefits of classical logics but offers greater fidelity and elaboration tolerance because it more accurately resembles fuzzy, human-like reasoning processes.

Weaknesses: **Breadth, Construction cost, Confidence**

While probabilistic approaches may offer improvements over traditional logics in that they allow for probabilistic learning from experience, it is unclear that, like logic, this kind of reasoning will scale to large and broad knowledge bases.

2.6.7. Human-Inspired Mechanisms: Qualitative Reasoning, Analogical Reasoning, Scripts and Case Based Reasoning

When people draw inferences and justify their claims, they do not typically follow formal mathematical logic. Instead, people use a range of specialized problem-solving skills: analogies, stereotyping, generalization, specialization and case-based reasoning. Qualitative reasoning [17], analogical reasoning [42], scripts [149] and case-based reasoning [142] are attempts to translate such intuitive thought processes into computational models. For example, rather than performing a precise heat-transfer computation to understand a cooling cup of coffee, a system might combine simpler models of thought to understand 'hot' and 'warm' as analogs of 'full' and 'empty' and thereby understand heat transfer as like a leaky bucket. Likewise, continuous variables may be factored along physical and psychological inflection points (solid, liquid, cold, warm, hot) to reduce search space and more closely match the intuitions of people.

Thus, these approaches benefit from a symbolic heritage, while having vastly reduced search spaces and an ability to perform meaningful (if only approximate) inferences by analogy, when a problem is not fully understood by the system.

Problem: Human-like reasoning on complex or uncertain problems that have been symbolically expressed and that are encountered by people in day-to-day life

Motivation: Creating computational models of abstract human thought processes and problem solving techniques

Representation: Symbolic structures that capture both factual knowledge and the abstract working knowledge of the computational 'mental models'

Reasoning: Model-specific algorithms

Population: Manual knowledge engineering

The similarity to human thought processes (including those of the engineers who build the system) may be seen as an argument for the costs of knowledge engineering being smaller than methods of engineering with less natural representations.

Strengths: **Operation cost, Confidence**

The systems are computationally efficient and there is substantial evidence of their success (not only functioning prototype systems but also the success of human beings they are attempting to mimic). The approach appears to be robust for automating certain subsets of human-like reasoning.

Weaknesses: **Breadth, Uniformity, Construction cost**

The systems discard significant detail when performing reasoning and this is not necessarily a valid heuristic in all circumstances. Furthermore, work in human-like mechanisms do not propose or suggest uniform architectures for broad reasoning across many domains.

2.6.8. Evolutionary Learning: Novamente and Discovery Systems

The Novamente Cognition Engine (NCE) [54] and its open-source derivative, OpenCog [66], is a serious attempt at commercializing a complete theory of general artificial intelligence. NCE is an agent-based architecture that combines inference in a probabilistic logic with the generality of evolvable procedures that are implemented in a Turing-complete language. This architecture therefore unites probabilistic deductive reasoning (including all of its benefits discussed in Section 2.6.6) with an open-ended evolutionary learning architecture for discovering new categories, representations and long term self-improvement.

The Novamente Cognition Engine is a commercially motivated effort, finding initial application in the creation of intelligent virtual agents for interactive online worlds and computer games such as AGIsim [56] and Second Life [58]. These agents are designed as simple pets that, despite their limited intelligence, are able to learn novel skills from a coach. By building up from simple skills, Novamente hopes to build systems with true general purpose artificial intelligence.

The Novamente Cognition engine stems from earlier work in discovery systems that use evolutionary algorithms to search over reward and solution spaces and, thereby, perform both learning and meta-learning. Well known examples include Lenat's Eurisko [103] and Haase's CYRANO [63]. These systems have been used for finding novel and unexpected solutions in many problem domains [103], but have not found widespread application as a method of general-purpose problem solving.

Problem:	General purpose intelligence, with a particular emphasis on problem solving: automatically discovering new ideas, techniques and mental models in novel domains
Motivation:	The effectiveness and universality of evolutionary search algorithms and the possibility of applying evolutionary search to the space of computer programs to enable recursive self-improvement
Representation:	Universal Turing-complete languages (programs in these languages are evolved by genetic programming)
Reasoning:	Execution of programs expressed in a Turing-complete language, in addition to evolutionary search over the space of such programs
Population:	Experience-based learning Such systems are assumed to be taught incrementally as with a small child; the system begins with simple problems to build foundational knowledge and skills that it later builds upon when challenged with more difficult problems.
Strengths:	**Uniformity, Construction cost** The system learns independently and so is intended to discover novel models and problem-solving strategies without human intervention.
Weaknesses:	**Adaptation cost, Confidence** An unanswered question is whether evolutionary algorithms can scale up to learning complex problem spaces of general purpose intelligence. Indeed, all evidence points to evolutionary learning being of highly limited scalability.

2.6.9. Corpus-Based Methods: OMCS and MindPixel

Projects like Open Mind Common Sense (OMCS) [105], ISI Learner [26] and MindPixel [123] are data-centric efforts to accumulate raw data about commonsense as the first step in engineering commonsense reasoning systems. These projects target those applications where *breadth* of knowledge is more important than the *depth* of knowledge. In some circumstances, such broad and superficial knowledge may even provide a useful *illusion* of deep knowledge. Consider, for example the almost uncanny ability of modern search engines to reveal appropriate web-sites matching a query.

In these approaches, large corpora of data are gathered from 'free' resources such as volunteer contributions and the world wide web, or by structuring play in online games that are specially crafted to make knowledge elicitation entertaining. Unfortunately, the cost savings associated with such 'free' resources come at the expense of reduced quality, precision and depth of the extracted knowledge. Indeed, corpus-based methods present a bootstrap problem: it is difficult to understand the implied commonsense knowledge in a database without already possessing the commonsense knowledge required to first disambiguate and interpret the data. Consequently, corpus-based methods emphasize shallow or surface knowledge: treating sentences as merely 'bags of words' or by emphasizing factual surface knowledge (of a parser) rather than the deep commonsense implications of data.

Consider EmpathyBuddy [108], an e-mail client that identifies the emotional content of a message. While a traditional approach to designing EmpathyBuddy might begin with rich theories of human emotions and conversational style, it turns out that EmpathyBuddy is able to convincingly identify the emotional subtext of a message through simple statistical associations between words in the message and emotive words. EmpathyBuddy computes distances in the ConceptNet [67] semantic network (built from Open Mind Common Sense data) and uses these distances as a shallow but effective estimate of the emotional character of each word. Its operation is analogous to modern statistical spam filters [148]. A spam filter does not need to fully understand the true meaning of a message if key words like 'Viagra' or 'unclaimed prize' are highly indicative of spam.

Problem: Broad commonsense knowledge, with a particular emphasis on linguistic associations

Generally speaking, corpora-based commonsense concerns recognition of, or knowledge about, commonsense situations rather than deeper understanding (*c.f.*, 'I know *about* vitamins and that they are 'good' but I do not understand *why* or *what* they are').

Motivation: The expense of eliciting knowledge in richly expressive formalisms and the effectiveness of broad, shallow knowledge in many practical applications

Representation:	Networks of conceptual associations, simple statistical correlations and crudely extracted relational tuples
Reasoning:	Identification of connections in representations: simple graph traversals
Population:	Collecting freely available resources into large corpora
Strengths:	**Breadth, Uniformity, Construction cost, Operation cost** Very large and broad corpora can be readily accumulated from freely available or cheap resources.
Weaknesses:	**Fidelity, Density, Elaboration tolerance, Confidence** The broad and shallow representations of these systems prevent them from representing precise knowledge or handling difficult queries. Furthermore, the technique suffers from a bootstrap problem: it is difficult to extract detailed knowledge from a large corpus and there is limited evidence that this may even be possible without other forms of learning or experience.

2.6.10. Connectionist Methods: Creativity Engine and China-Brain

Connectionist methods attempt to mimic the essential behavior of the brain using artificial neural networks. An artificial neural network is an abstraction of the networks of biological neurons that appear in the human brain. An artificial neural network is comprised of a large set of artificial neurons that are densely interconnected and have a simple mapping from input to output. While an artificial neural network is not engaged in the same kind of electrochemical interactions as a biological neuron, it shares similar macroscopic behaviors.

Connectionism was founded with an ambitious vision of creating deep, human-like intelligence (and commonsense). Today, it is primarily seen and applied to 'narrow-AI' problems of image recognition and other learning of non-linear functions.

Hugo de Garis' China-Brain project [33] stands out as a dramatic return to the original ideals of connectionism. The project is attempting to train thousands of high-speed artificial neural networks (using custom hardware) and assemble the networks into a larger super-network to drive the intelligent behavior of a robot. So far, the system is capable of recognizing limited objects but work is ongoing in order to build sophisticated behaviors.

There are other, less grandiose, attempts to apply artificial neural networks to challenging problems of general-purpose intelligences. Numenta Corporation's Hierarchical Temporal Networks (HTM)

[68] are hierarchical layers of nodes that can perform spatio-temporal reasoning and learning; recurrent neural networks are being successfully applied to increasingly difficult learning problems in work such as that of Graves and Schmidhuber [60]; and Steven Thaler's Imagination Engine [164] is used to generate creative and novel inventions by adding noise to a trained neural network.

Problem:	Learning and decision making in continuous problem domains for which no obvious model is evident or where building such a model may be too expensive
Motivation:	The success of the human brain in performing deeply complex learning and reasoning from large numbers of simple neurons that have been connected together
Representation:	Numerical weights used by individual neurons: knowledge is distributed across the entire network
Reasoning:	Application of the transfer function to each neuron (or, in the case of learning, adjusting the weights of the function through reinforcement)
Population:	Automatic learning from engineer-supplied examples
Strengths:	**Uniformity, Construction cost, Operation cost** Connectionist methods are well understood, readily implemented and efficient to execute (but not necessarily efficient to train).
Weaknesses:	**Elaboration tolerance, Adaptation cost, Confidence** The technique is expensive for general intelligence or commonsense reasoning because it requires many training examples, significant computation time for learning and has unpredictable behavior on unusual inputs. There is little evidence that the approach will scale beyond association and simple function learning to perform the deeper abstract thought sometimes required for commonsense reasoning.

2.6.11. Biological Simulation: Blue Brain

Faced with uncertainty over *how* or even *if* it is possible to create true artificial intelligence, a method of last resort might be the direct simulation of every aspect of the human brain. That is, building a computer model that matches the human brain in every chemical reaction, every electrical signal and every neuron connection.

While no groups are seriously considering embarking on such an ambitious project as an approach towards artificial intelligence, there is significant interest among neuroscientists in building powerful models of the brain. The Blue Brain project [113] is a notable attempt to perform large-scale and detailed simulation of 10,000 neurons in a rat's sensory column. It will, of course, take many decades of research into the human brain and improvements to computers before this might be possible for emulating the more than 100 billion neurons in the human brain.

Problem:	The creation of human-like or animal-like artificial intelligence
Motivation:	The brain (human or animal) is the only known device capable of intelligent thought and, by direct simulation of every part, it should be possible to mechanize its capabilities
Representation:	Biologically and electro-chemically accurate models of a brain
Reasoning:	Simulation of the biological mechanisms
Population:	Experimentation on (and imaging of) animal brains to replicate the behavior in simulation
Strengths:	**Breadth, Fidelity, Uniformity, Elaboration tolerance** Under the assumption that fundamental biological processes are computable, can be fully understood and that there is sufficient computing power to model the brain, this approach would result in a virtual brain with the full capabilities of a human brain.
Weaknesses:	**Construction cost, Adaptation cost, Operation cost** Unfortunately, the demanding biological research and computing facilities that would make a virtual brain possible mean that it is unlikely the approach will be a viable option for decades to come.

2.6.12. Faking it: Eliza and Chatterbots

Joseph Weizenbaum's classic work on ELIZA [173] stands as possibly the oldest and best-known attempt to imitate human-like intelligence in a computer system. ELIZA plays the role of a Rogerian psychologist, taking the user's thoughts and reflecting them back as seemingly insightful questions. Take, for example, the following excerpt from a transcript in which ELIZA's response is indicated by capital letters:

> Men are all alike.
> IN WHAT WAY?
> They're always bugging us about something or other.
> CAN YOU THINK OF A SPECIFIC EXAMPLE?
> Well, my boyfriend made me come here.
> YOUR BOYFRIEND MADE YOU COME HERE?
> He says I'm depressed much of the time.
> I AM SORRY TO HEAR YOU ARE DEPRESSED.
> It's true. I am unhappy.
> DO YOU THINK COMING HERE WILL HELP YOU NOT TO BE UNHAPPY?
> I need some help, that much seems certain.
> WHAT WOULD IT MEAN TO YOU IF YOU GOT SOME HELP?
> Perhaps I could learn to get along with my mother.
> TELL ME MORE ABOUT YOUR FAMILY.
> My mother takes care of me.
> WHO ELSE IN YOUR FAMILY TAKES CARE OF YOU?
> My father.
> YOUR FATHER?
> You are like my father in some ways.
> WHAT RESEMBLANCE DO YOU SEE?
> You are not very aggressive but I think you don't want me to notice that.
> WHAT MAKES YOU THINK I AM NOT VERY AGGRESSIVE?
> You don't argue with me.
> WHY DO YOU THINK I DON'T ARGUE WITH YOU?

The system has a compelling appearance of deep intelligence but it is entirely faked by simple word substitutions. For example, in the last exchange, the system has matched *You* and *me* against a template and replaced them with the patterns *WHY DO YOU THINK I* and *YOU* respectively. Almost all of this dialog has been generated by such substitutions (*i.e.*, *my* for *your*; *my mother* for *who else in your family*; *I need* for *what would it mean to you if you got*) and generic questions (*e.g.*, *in what way*). Its apparent intelligence stems from the accumulated effect of a large collection of substitution rules and the human tendency to attribute intelligence and find explanations in any complex behaviors.

Modern day 'chatter-bots', conversational agents and virtual humans are more sophisticated realizations of the same basic mechanism, finding application in customer service [81], recruiting [4] and even the criminal world [40, 147].

Problem:	A need to create convincing displays of intelligence and to connect emotionally with users, despite a lack of progress in Artificial Intelligence
Motivation:	Insufficient progress in AI research but an ability to exploit the generous 'explanatory power' of the human mind in attributing intelligence to any system that can generate relatively fluent English
Representation:	Textual substitution lists and simple mechanisms for storing canned responses and tracking conversational moves
Reasoning:	Shallow text retrieval, manipulation and transformation
Population:	Manual elicitation of compelling textual transformation rules The rules may be given breadth and made more convincing through the study of real conversation transcripts.
Strengths:	**Construction cost, Adaptation cost, Operation cost** While these systems are not very intelligent, they are cheap to build and operate because they simply mirror back the intelligence of users.
Weaknesses:	**Fidelity, Density, Uniformity, Elaboration tolerance, Confidence** The technique is effectively a collection of ad-hoc 'cheap' tricks. While it sometimes finds application as 'chat-bots' on web-sites, there is no indication (nor any claim) that this approach might ultimately lead to a truly intelligent system that can actually understand the meaning and nuance in text.

2.6.13. Rejecting it: Reactive Systems

Rodney Brooks, in his seminal paper 'Intelligence without Representation' [20] rejected the idea of creating systems with internal representations of the world. He suggested treating the world as its own best model and allowing complex behaviors to emerge from the complex interactions of many competing reactive processes. The development of reactive systems, such as those by Brooks' subsumption architecture [20], is a useful approach for creating pragmatic systems that behave appropriately in complex and changing environments, especially ones in which any internal representation would be rapidly rendered obsolete.

Reactive systems are, however, closely related to their embodiment and so it is unclear how the approach could be disembodied and adapted to new problem domains in new kinds of sensory modalities (*e.g.*, taking practical hands-on experience in a kitchen and adapting it to answering symbolic questions about recipes) or used in problems that demand prolonged and deliberative cognition (*e.g.*, planning a complex series of actions to achieve a goal).

Problem:	Effective and efficient operation in complex, highly dynamic, *real world* environments
Motivation:	The failure of traditional symbolic artificial intelligence techniques to adapt to complex changing environments, especially when viewed in light of the success of apparently 'primitive' robotic systems that use low-level reactive control mechanisms
Representation:	Limited use of internal representations: the world is its own best representation
Reasoning:	Automatic, reactive responses to sensory inputs
Population:	Reactive subsystems are manually built and skills are thereby implicitly programmed in the system (there is no explicit knowledge engineering since there is no explicit representation of knowledge)
Strengths:	**Breadth, Operation cost** The systems, without internal representations, can run efficiently and have robust behavior in many situations.
Weaknesses:	**Uniformity, Confidence** Reactive systems are usually implemented as a hierarchy of behaviors implemented in opaque computer modules; there is little requirement for uniformity or uniform representation. While the robots perform appropriately in complex environments, the lack of internal representations suggests that the approach may not scale to more sophisticated problem solving that requires abstract thought about the external world.

2.7. Research Gap

In Section 2.6, I outlined the huge diversity and investment in commonsense reasoning, general intelligence and cognitive systems. However, I do not seek to criticize their methods. Many of these projects will be successful in their short term goals, some may plateau at limited niche applications and others may be reinvented as new projects. Given the difficulty of creating powerful artificial intelligence or commonsense reasoning systems, it is a matter of fact that very few are likely to be successful in their long term ambitions, whether due to theoretical or technical matters or the more worldly concerns of financing and politics. Each approach has merit and it is far too soon at this stage to rule out any particular approach. Indeed, future generations of research may draw from the fusion of many ideas. That is, intelligence and commonsense are too poorly understood to declare necessary or sufficient conditions and thereby confidently reason about the likelihood of success of any given methodology.

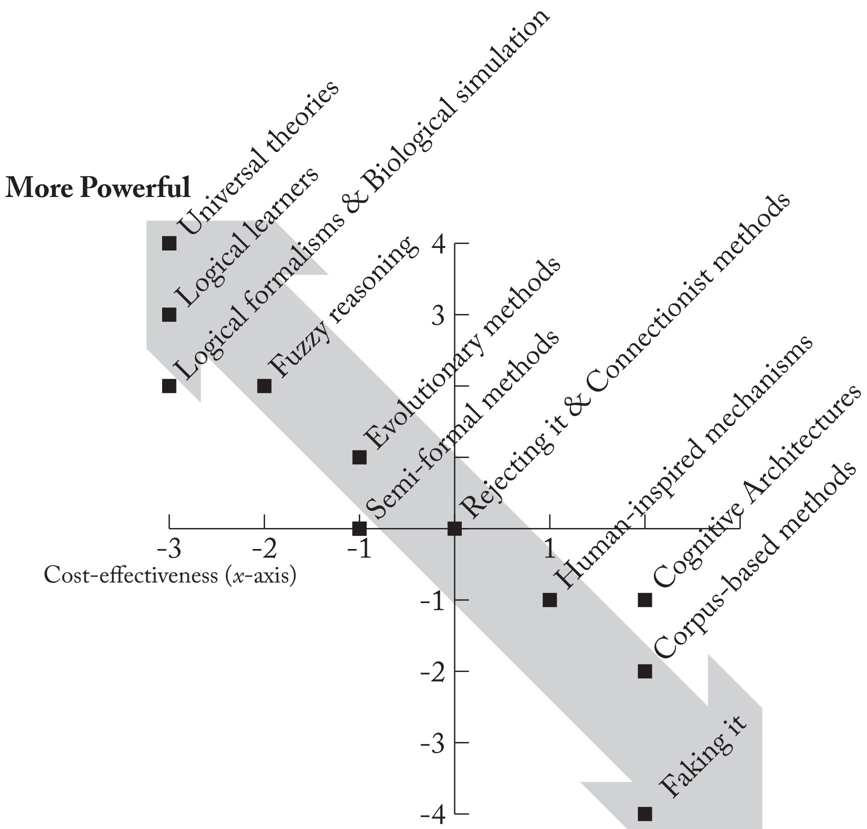


Figure 2.1: The systematic trade-off between expressiveness and cost-effectiveness in work related to commonsense reasoning

While it is premature to identify common weaknesses among existing systems, it is possible to identify a paradigmatic trade-off that is universal among existing work. Existing methods form a spectrum that trades power for cost-effectiveness (and *vice versa*). At one extreme of the spectrum are universally powerful mechanisms that require massive computational resources (*e.g.*, AIξ) and, at the other end, are approaches that can perform limited forms of deduction but are built at low cost (*e.g.*, Open Mind Common Sense). This is illustrated in Figure 2.1 which depicts a simple transformation of the subjective strength/weakness analyses performed in Section 2.6. The axes are formed by the weighted sum of the strengths and weaknesses of each method: a strength counts for +1 and a weakness counts for -1; and the *x*- and *y*- axes are formed by respectively grouping dimensions into cost-effectiveness (construction cost, adaptation cost, operation cost and confidence) and power (breadth, fidelity, density, uniformity and elaboration tolerance). The trade-off between power and cost can be seen in the direct negative relationship between the two axes.

Given Figure 2.1, there seems little motivation for yet new representations or methods that lie *within* the established spectrum. In this dissertation, I instead outline a commonsense reasoning framework that operates outside this entrenched paradigm, with a specific objective of *simultaneously* achieving power and cost-effectiveness. My conjecture is that the limited success of current systems is due to an inappropriate choice of trade-off: that, instead of sacrificing power for cost (or *vice versa*), both goals should be sought at the expense of other, less important, factors. Indeed, the system that I introduce in this dissertation—Comirit—is intended to be a cost-effective and powerful system that uses representations tuned for "typical" deductions and low-cost population of knowledge but performs comparatively poorly in highly uncertain domains or in arbitrary problems that have little real world correlation. I will argue that these sacrifices are not detrimental to the practical usefulness of the system.

2.8. Integration Architectures

An alternative to the careful design of a powerful mechanism or single-minded architecture for commonsense reasoning is what might be termed a 'kitchen sink' approach. Rather than engineering a system according to a theory about the mechanics of intelligence, a system could be constructed as an assembly of many independent mechanisms. While each mechanism has limited capabilities, the entire system acts as the combination of each capability.

The 'kitchen sink' approach is based upon the assumption that intelligent behavior will *emerge* when enough essential skills are combined. It is too early to confidently accept or reject this assumption: on one hand, integrative architectures have been successfully applied to so many challenging problems that they have become a mainstay of Artificial Intelligence research but, conversely, these systems are constrained by the fundamental limitations of their parts and do not achieve emergent behavior that is vastly in excess of the simple sum of its parts. That is, while integrative architectures are successful for creating systems that adeptly perform many tasks and solve many problems, an integrative architecture alone is insufficient for creating deep and powerful intelligence.

Nevertheless, a number of integration architectures have been proposed as approaches to constructing commonsense and intelligent reasoning systems. They range from simple languages for integration to sophisticated agent-oriented architectures of collaborating processes. Four well-known proposals that are illustrative of the diversity follow:

- **Sowa's Architectures for Intelligent Systems**

  Sowa [158] proposes a multi-paradigm integration based on a blackboard kernel inspired by Linda [2]. His proposal uses conceptual graphs and other rich formalisms as values expressing knowledge within a Linda tuple-space and connects specialized reasoning mechanisms by 'glue' languages such as McCarthy's Elephant 2000 language [118].

- **Society of Mind**

  Minsky's Society of Mind [128] is an early and highly influential architecture for intelligent systems. Minsky views the mind as a society of interacting processes for which no single representation suffices. While influential and closely related to agent-oriented architectures, the society of mind is abstract and ambitious so that there are few direct implementations of the ideas. The DUAL cognitive architecture [92] is one implementation of Minksy's ideas, combining connectionist and symbolic reasoning into a single system.

- **ThoughtTreasure**

  Mueller's ThoughtTreasure [131] is a compelling practical example that combines factual, logical knowledge and reasoning (similar to Cyc), grids (2D maps of stereotypical settings), simulated agent-based planning within grids and procedural 'rules of thumb'. It uses this knowledge to improve natural language understanding and draw sensible conclusions and inferences from textual interactions with users. In the commonsense-aware calendar application, SensiCal [132], ThoughtTreasure is able to interact with the user in natural language and can intelligently identify some commonsense conflicts in a schedule.

- **Polyscheme**

  Cassimatis' Polyscheme [24] is a psychologically-inspired, agent-oriented architecture designed about a 'cognitive substrate' of core skills upon which intelligence is implemented. Polyscheme is effectively a framework of interfaces that encode all reasoning mechanisms according to four basic operations: forward inference, sub-goaling, simulation of alternate worlds and identity matching. Each operation may be implemented by a number of mechanisms.

The long term success of any of these integration architectures in creating deeply intelligent systems will depend on the appropriate selection and balance of the underlying reasoning mechanisms that are implemented within the architecture. Thus, even with a powerful integrative architecture, established research remains constrained by the spectrum of trade-offs of Section 2.7.

The challenge of integration is further compounded by the need for the underlying mechanisms to be appropriately integrated in such a way as to combine their strengths. There is little point in creating an integrative architecture if the combined system can do no more than its subsystems working independently. Unfortunately, much of the established work in integrative systems concerns relatively shallow integration. These systems connect inputs and outputs of subcomponents implicitly through agent interactions or by standardized representations. They do not have a low-level integration of semantics: instead of reasoning by a uniform mechanism of *coordinated* processes, integration is a loose collection of *cooperating* processes.

In this dissertation, I will propose a framework that is based first on a reasoning mechanism that lies outside the established spectrum (of Section 2.7) and that is extended into a rich integration framework to perform coordinated reasoning. The integrated reasoning mechanism is designed in such a way that it uses semantic integration to combine strengths and compensates for weaknesses. I will consequently revisit this review of established methods of integration later, in Chapter 6 of this dissertation, when I introduce the integration framework.

2.9. Conclusion

Unfortunately, commonsense is like intelligence: easy to recognize but difficult to objectively define. As such, commonsense is best judged and evaluated indirectly through the use of open-ended challenge problems and benchmarks. While these benchmarks can be easily 'gamed' by hard-coding answers or superficial reasoning, an honest attempt at the challenges highlights the inflexibility of traditional techniques of software development.

Unfortunately, commonsense also remains one of the long-standing open challenges of Artificial Intelligence. Despite countless efforts and proposals, few systems have progressed to a state that they can be meaningfully applied to any benchmark problem.

An explanation for the limited success of existing approaches to commonsense reasoning and intelligence may be found in their design trade-offs. Existing work forms a spectrum: at one end are richly powerful and expressive systems that are expensive to build while, at the other end, are shallow systems that are built cheaply. That is, existing methodologies do not allow for powerful systems to be also constructed cheaply.

Thus, there is an opportunity to step outside the established spectrum and build a system that is both powerful and cheap, by sacrificing other characteristics.

In the remainder of this dissertation, I will develop, design and evaluate an approach to commonsense reasoning that I believe achieves deep, useful and powerful commonsense reasoning at a reasonable cost. This is performed at the expense of deductive flexibility: while the system is well suited to problems of everyday commonsense reasoning, it is not well suited to general purpose abstract reasoning problems for which methods such as logics are well suited.

Finally, I note that, as this dissertation comes to publication, there has been a recent proposal by the Soar group that similarly explores methods outside the established spectrum. While the Soar proposal has been developed independently of my research and differs significantly in methods and motivation, there is some overlap in representational choices. Rather than reviewing such recent developments as background literature in this chapter, I will explore them in technical discussions of Section 5.2 so as to avoid preempting discussions in this dissertation and because I view the overlap as an additional form of external confirmation of my ideas and hypotheses.

Chapter 3

Intelligent Systems and Symbol Grounding

The symbol grounding problem [65] represents a long standing and often misunderstood point of contention within the Artificial Intelligence community (*e.g.*, [27,175,180]). The problem, as it is classically conceived, concerns the nature of the abstract symbols used in computer systems, how they may be seen as having meaning and how that meaning can be made intrinsic to a system.

Consider the problem of a knowledge base designed to reason about the possible security threats posed by terrorist organizations. The system may have an internal symbol ***nuclear_weapon*** that we as humans understand as representing the real-world concept of nuclear weapons. However, to a purely symbolic computer system, the symbol ***nuclear_weapon*** is nothing more than an arbitrary token that has no more *intrinsic* meaning than any other symbol in a computer, say ***waffle_blah***. The symbol grounding problem concerns the question of how the meaning of symbols can be embedded into a system and 'grounding' is said to be the process of ensuring that these abstract symbols have meaning.

While there is the philosophical question of whether a machine really can have intrinsically ground symbols (indeed, this is the motivation for Searle's Chinese Room argument [152]), the symbol grounding problem poses the more practical question of whether a purely symbolic system *could* solve problems that apparently demand deep intelligence and understanding. For example, is it possible for a purely symbolic system to understand the nuanced relationship between a nuclear weapon and a dirty bomb, or to explain that a zebra is like a horse with stripes, or even to determine what other letter of the alphabet an upside-down 'M' resembles, without requiring the specific answers to these questions to be explicitly given to the system in advance?

These questions of meaning and intelligence that lie at the core of the symbol grounding are crucial to understanding commonsense reasoning. My conception of commonsense is that it is more than simply the automatic manipulation of meaningless symbols. Commonsense appears to depend on an instinctive understanding of the meaning and purpose of symbols. This view may be faulty. It may transpire that by simply continuing in the *status quo*, by building simple rule based systems, it will be possible to create an artificial common sense. A better understanding of the symbol grounding problem will help resolve this uncertainty, giving insight into the feasibility of engineering commonsense knowledge-bases or even the possibility of evolving black-box mechanisms that cannot be understood.

My objective in this chapter is not to argue a specific position on the symbol grounding problem but to provide the first formal framework for the symbol grounding problem. I use this framework to argue an approach to commonsense reasoning where even abstract assumptions about intelligence are chosen with pragmatic purposes in mind and are explicitly identified.

I admit, up front, that the formal framework has many limitations. The symbol grounding problem is a significant philosophical challenge that presents a formidable challenge to any attempt at formalization. There is an enormous body of literature in Philosophy, Mathematics and Artificial Intelligence that concerns truth, semantics and grounding. While I have taken every effort to bring such work to bear on this formalization, it will take many years and many minds to fuse all this work into a consensus formalization of the symbol grounding problem.

In spite of its limitations, this formalization is important. It moves discussion from the realm of ambiguous language and words with complex nuances to that of mathematical definitions that can be precisely compared, contrasted, proven and disproven. While I believe that this formalization is correct, I would consider myself successful even if it is proven wrong because I have stated something that *can be* disproven. I therefore view this formalization as an open challenge to other researchers:

1. To find fault in my formalization and prove or disprove its coherency
2. To propose improvements to the formalization that resolve its limitations
3. To propose alternate formalizations of the symbol grounding problem
4. To compare alternate formalizations (*e.g.*, are different perspectives logically equivalent?)

Beyond this open challenge for more formal treatment of the symbol grounding problem, the framework itself offers a useful ontology of the problem. I propose a set of representational system classes that may serve as a vocabulary that allows an AI researcher to express their assumptions as elegantly as computer scientists use computational complexity classes (such as P, NP and EXPTIME) to discuss the computability of problems. Similarly, these representational system present interesting new questions concerning their equivalence and subsumption.

In this chapter, I first define symbol systems and representations, then review the problem of grounding such symbols and representations. I then introduce formal notation and explore 'semantic interpretability'. These serve as preliminaries to the primary contribution of this chapter: the definition of representational system classes and their correspondences with the published literature. I then conclude with some observations about the representational classes and their relevance, followed by identification and motivation of the assumptions that underlie the remainder of this dissertation and a brief overview of future research directions.

3.1. Symbol Systems and Symbol Grounding

Harnad [65] first proposed the symbol grounding problem as a question concerning semantics: "How can the semantic interpretation of a formal symbol system be made intrinsic to the system, rather than just parasitic on the meanings in our heads?" While Harnad's original formulation of the problem is largely philosophical, his motivation is clearly of a pragmatic nature: he implicitly assumes that the property of 'intrinsic interpretability' is crucial for intelligence. I therefore reformulate the symbol grounding problem in more straightforward terms: "Is it possible to use formal symbolic reasoning to create a system that is intelligent?" Of course, this reformulation presupposes a definition or measure of what it means to be (or to appear) intelligent but the reformulation is an improvement in the sense that it brings us slightly closer to something objectively measurable.

Harnad saw the mechanisms of an isolated formal symbol system as analogous to attempting to learn Chinese as a second language from a Chinese-Chinese dictionary. Even though characters and words are defined in terms of other characters, reading the dictionary would amount to nothing more than a 'merry-go-round' passing endlessly from one symbol string (a term) to another (its definition), never coming to a 'halt on what anything meant' [65]. He therefore argued that, since symbols only refer to other symbols in a symbol system, there is no place where the symbols themselves are given meaning. The consequence of this is that it is impossible for a formal symbol system to distinguish between any two symbols except by using knowledge that has been explicitly provided in symbolic form. This, in the view of Harnad, limits the comprehension and capabilities of a symbolic system in the same way that a non-speaker armed with a Chinese-Chinese dictionary may manage to utter random but syntactically correct sentences while exhibiting extremely poor performance in understanding real conversation.

Of course, Harnad's argument is not universally accepted by computer scientists. One objective of this chapter is to outline and classify the diversity of opinions.

The symbol grounding problem concerns symbolic systems but what *is* a formal symbol system? Harnad [65] provides eight criteria:

> 'A symbol system is: (1) a set of arbitrary "physical tokens" (scratches on paper, holes on a tape, events in a digital computer, *etc.*) that are (2) manipulated on the basis of "explicit rules" that are (3) likewise physical tokens and strings of tokens. The rule-governed symbol-token manipulation is based (4) purely on the shape of the symbol tokens (not their 'meaning'), *i.e.*, it is purely syntactic, and consists of (5) 'rulefully combining' and recombining symbol tokens. There are (6) primitive atomic symbol tokens and (7) composite symbol-token strings. The entire system and all its parts—the atomic tokens, the composite tokens, the syntactic manipulations both actual and possible and the rules—are all (8) 'semantically interpretable': the syntax can be systematically assigned a meaning (*e.g.*, as standing for objects, as describing states of affairs).'

It is interesting to note that criteria 1–7 can be used to describe any universal or Turing-complete language. However, Harnad is not claiming that meaning or intelligence is uncomputable—he proposes his own computational framework for solving the symbol grounding problem. It is the 8th criterion of a formal symbol system that defines the essential point of difference between his conception of symbolic systems and representations used in arbitrary computation. The requirement of interpretability in criterion 8 is intended to capture the essence of familiar symbolic systems such as logical theorem proving and rule based systems and to exclude highly distributed, 'holographic' or connectionist representations that do not behave in a symbolic manner (even though they operate on a digital computer). For example, while connectionist approaches to building intelligent systems can be framed so as to meet criteria 1–7, connectionist methods do not typically allow for a systematic assignment of real-world interpretation to hidden layer neurons (*i.e.*, hidden neurons learn with a bias towards performance rather than any particular 'meaning'). They, therefore, do not satisfy criterion 8 and are therefore not (directly) subject to Harnad's criticism.

Harnad's 8th criteria for a symbol system is essential to understanding the symbol grounding problem. I will consider how changes to this criterion (whether in its phrasing or in comprehension) influence an understanding of the symbol grounding problem. Specifically, I further generalize the symbol grounding problem as follows: "What *kinds of reasoning* can be performed by systems constrained by different *representational criteria*?" In this formulation, I regard different research groups as working from both different assumptions of what constitutes intelligence (*i.e.*, the kind of reasoning) and different representational constraints.

3.2. Notational Preliminaries

Problems

I begin by assuming the existence of a set, $\mathcal{P}$, that contains all problems that may be posed to an intelligent system. Each problem is a declarative sentence (in some formal language) about the world and an agent is said to be able to solve a problem if its determination of the truth of that statement matches the 'real world' truth. A problem might be a query posed by a person to a theorem prover or a question-answering system, or it might represent an encoding of the inputs and outputs of a robotic system (*i.e.*, "given certain sensory inputs x, the appropriate behavior at time t is to perform action a"). While a real life agent may encounter complex situations and exhibit nuanced performance that is neither success nor failure, I assume that these situations can be analyzed as a large set of binary sub-problems and the agent's performance is a measure of how many of the sub-problems can be successfully solved.

If an agent, f, believes statement p to be true, I denote this as $f \vdash p$. If an agent, f, can correctly determine the truth or falsity of a problem, $p : \mathcal{P}$, then I denote this as $f \sim p$.

Problem Sets

I define a *problem-set* as a set of problems: an object of type $\mathbb{P}\mathcal{P}$. An agent, f, can solve a problem-set, $ps : \mathbb{P}\mathcal{P}$, if it can solve all problems within that set. This is denoted $f \sim ps$ and is defined as $f \sim ps \Leftrightarrow \forall\, p : ps \bullet f \sim p$.

Intelligence

I use problem-sets to define intelligence. In this work, I do not choose any particular definition of intelligence. I assume a range of definitions so as not only to denote the largely subjective and non-formalizable 'I'll know it when I see it' attitude towards intelligence but also to offer scope for formal definitions of intelligence. As such, a given definition, I, of intelligence is a set of problem-sets; *i.e.*, $I : \mathbb{P}\mathbb{P}\mathcal{P}$. An agent, f, is considered intelligent with respect to a definition of intelligence I, if it can solve *all* problems in *some* problem-set. This is denoted $f \sim I$ and is defined as $f \sim I \Leftrightarrow \exists\, ps : I \bullet \forall\, p : ps \bullet f \sim p$.

This approach to representing definitions of intelligence admits many definitions beyond that of simple IQ tests or fixed checklists of skills. It allows for flexible and context-dependent definitions of intelligence. Consider that one may regard Albert Einstein and Elvis Presley as both possessing exceptional intelligence, even though their genius is expressed in different ways. Their distinct skills correspond to different problem-sets within a common interpretation of 'genius'.

This framework allows many definitions of intelligence, for example:

- A set $I_{\text{Harnad}} : \mathbb{P}\mathbb{P}\mathcal{P}$ for those systems that Harnad would regard as exhibiting intelligence
- A set $I_{\text{IQ=100}} : \mathbb{P}\mathbb{P}\mathcal{P}$ to denote sets of problems that a person of average human intelligence would be able to solve
- A set $I_{\text{Market}} : \mathbb{P}\mathbb{P}\mathcal{P}$ of buying decisions that a trading agent would need to solve successfully in order to exceed break-even on a market over a particular time interval
- Given a formal definition of intelligence with a precise threshold, one may have a set $I_{\text{Formal}} : \mathbb{P}\mathbb{P}\mathcal{P}$ denoting those problem-sets that a formally intelligent system could solve

Formal Systems

I define $\mathcal{F}$ as the set of all finite formal systems that satisfy criteria 1–7 of Harnad and are finitely realizable[*]. I define $\mathcal{T}$ as the universal set of symbols and assume that each formal system comprises a set of fixed symbolic transition rules and a dynamic execution state. I assume (without loss of generality) that an execution trace of a formal system, f, on a problem, p, is comprised of a two-dimensional grid of symbols. I denote this as $\boldsymbol{t}(f, p) : \mathbb{N} \times \mathbb{N} \twoheadrightarrow \mathcal{T}$. The two axes of the grid correspond to the state of the system (analogous to a CPU clock) and the position or address of each symbol. The value of each grid-cell is the single symbol stored in the 'address' in that state (be it a byte value stored in RAM, a mark on a tape, a neural network weight or an 'atom' in a logical programming language).

Representational Units

In non-trivial computational systems, individual symbols do not convey meaning alone; intelligent behavior stems from the manipulation of non-atomic sequences or sets within the system's symbolic state. These sequences are not selected or manipulated randomly. The system operates by manipulating only certain subsets of the system state, according to the legal 'guards' and 'arguments' of the system's transition rules. For example, if the numbers 1, 25 and 334 are denoted as the fixed-length sequence of digits <001025334> at a given step of the system trace, then a system's transition rules might only accept sequences aligned to three-digit boundaries (*i.e.*, 001, 025 and 334, but neither 00102 nor 253). For a given formal symbolic system $f : \mathcal{F}$ and problem $p : \mathcal{P}$, I define the set of all representational units, $\boldsymbol{a}(f, p) : \mathbb{P}(\mathbb{N} \times \mathbb{N} \twoheadrightarrow \mathcal{T})$ as the set of all subsets of the system trace, $\boldsymbol{t}(f, p)$, that can match part or all of a guard or parameter of a transition rule in f.

[*]By finitely realizable, I mean that the systems's representations and computational processes that can be described in finite space on a Turing machine by a finite agent within the universe and that the corresponding computations of the system in solving a problem occur in finite time.

3.3. Semantic Interpretability

'Semantic interpretability,' the cornerstone of Harnard's 8th criteria for a symbol system, presents a challenge to the formalization of the symbol grounding problem. The philosophical difficulties of the symbol grounding problem lie in the elusiveness of 'semantic interpretability'.

In defining the *semantic interpretability* of a program, it is necessary to first explore *semantic interpretations*. In a model-theoretic approach, one might define a semantic interpretation to be a mapping from the set of symbols (or representational units) to the set of meaningful 'real world' counterparts.

Unfortunately, even this view of semantic interpretation raises some philosophical challenges. Since symbols may be used to describe abstract, historical, hypothetical and fictional concepts, the set of 'real world' counterparts must include the complete universe of objects, actions, categories, relations and concepts, both real and imagined. Defining such a set has the potential of raising a number of paradoxes, including Russell's paradox: is the universal set itself a 'real' concept? does it belong to itself? is there a set of concepts that aren't 'real'? is that set a 'real' concept?

Nevertheless, it seems implicit in the question of semantic interpretability that there exists sets of meaningful 'real' objects and concepts to which symbols can be symbolically interpreted. Demonstrating that there is no such universal set may, in effect, show the incoherency of *semantic interpretability* and ultimately of the symbol grounding problem itself. Exploring this particular question remains as future work but, for the time being, I will assume the existence of a universal set of real concepts and denote this $\mathcal{U}$. If inconsistency is a problem, it may be possible to resolve it by handling malformed concepts through indirection or quotation. For example, a malformed or incoherent concept may be viewed superficially as a string of words or as a reified definition, rather than the absurd or incoherent target of the definition.

Now, given the universal set $\mathcal{U}$, one may define a semantic interpretation, *m*, as a mapping from sets of representational units to the universal set, that is $m : \mathbb{P}(\mathbb{N} \times \mathbb{N} \nrightarrow \mathcal{T}) \nrightarrow \mathcal{U}$. I will denote the type of a semantic interpretation (*i.e.*, $\mathbb{P}(\mathbb{N} \times \mathbb{N} \nrightarrow \mathcal{T}) \nrightarrow \mathcal{U}$) with the symbol $\mathcal{SI}$.

Given a *semantic interpretation*, the definition of *semantic interpretability* does not necessarily follow simply by declaring a program to be semantically interpretable if its representational units have a semantic interpretation. If the symbols 'horse-like' and 'stripes' combine in a program to give the symbol 'interrogation' (rather than 'zebra'), there may still be meaningful interpretations of the software, while the operation of the program as a whole appears to be largely meaningless. Semantic interpretability of a formal system demands inspection of not only its internal symbols but also the *use* of the symbols.

Unfortunately, it is difficult to analyze *use* without first defining a specific model of computation. In this dissertation, I intend to retain a level of abstraction from such particulars. In future work, I plan to explore abstract definitions of usage but, in lieu of an accepted definition, I propose here a working definition of semantic interpretability based on an execution trace. In this working definition, I view a system as being semantically interpretable if the combined semantic interpretation of the system's *execution trace* in solving a problem, p, entails the truth value of the semantic interpretation of that problem p.

That is, with $\mathcal{SI}$ denoting the type of semantic interpretations of representational units, $\mathcal{SI} = \mathbb{P}(\mathbb{N} \times \mathbb{N} \twoheadrightarrow \mathcal{T}) \twoheadrightarrow \mathcal{U}$, where $\mathcal{U}$ is the universal set of all objects. Then, given a (model-theoretic) semantic interpretation $m : \mathcal{SI}$, that maps from a set of representational units, $r : \mathbb{P}(\mathbb{N} \times \mathbb{N} \twoheadrightarrow \mathcal{T})$, to elements of $\mathcal{U}$, I say that a formal system, f, in solving a problem, p, is semantically interpretable if *syntactic entailment* (*i.e.*, computation) corresponds to *semantic entailment* from the model implied by the conjunction of the semantic mapping of the system's entire execution trace. That is:

$\boldsymbol{si}(m, f, p)$

$\Leftrightarrow$

$\left(\boldsymbol{t}(f, p) \subseteq \cup\cup\mathrm{dom}(m)\right) \wedge \left(f \vdash p \Leftrightarrow (\{\, u \mapsto e : m \mid \cup u \subseteq \boldsymbol{t}(f, p) \bullet e\} \vDash p)\right)$

◊ "A system f is semantically interpretable on problem p with respect to interpretation m, if the trace of the system in solving p has an interpretation and the system's truth determination for p matches the 'real world' truth of p that follows from the mapping of each unit in the system's trace"

While this working definition ignores the internal use of symbols and may be overly stringent for any particular system, it does not limit the generality of this work. Representational units with different purposes may be expanded to include the neighboring section markers, delimiters, type definitions, annotations or positional information that indicates their purpose, thereby embedding the *use* of a representational unit in the formal system onto its surface structure.

The precise definition of $\boldsymbol{si}(m, f, p)$ is not crucial for the rest of this chapter. I view its definition, in addition to the definition and nature of the universal set $\mathcal{U}$, to be primitives upon which my formalization is built. That is, my objective in the remainder of this dissertation is to build upon these assumed primitives so that the symbol grounding problem is *reduced* to the problem of defining semantic interpretability and the universal set of concepts.

Like commonsense, *semantic interpretability* seems intuitively straightforward. For typical systems the determination is clear cut. Difficulties arise when attempting to define the exact boundary between semantic interpretability and non-interpretability. As long as one has a sensible definition of semantic interpretability, the precise boundaries do not matter to the definitions in this chapter. Extreme or unconventional definitions of semantic interpretability will still work within the remainder of this chapter. However, such definitions will undermine the usefulness of both the formalization and the symbol grounding problem. For example, if one considers every formal system to be, itself, a meaningful ‘concept’, then any system is trivially semantically interpretable by a mapping of its symbols onto itself. That is, one could trivially ‘solve’ the symbol grounding problem on a technicality by excessively generalizing $\mathcal{U}$, so that the ‘real world’ concept of any symbol can be taken as “those situations, entities and environments that stimulate the generation of the symbol”. Such contrived entities would seem absurd to a human-observer and are also highly context dependent so, therefore, do not correspond to intuitions of meaningful ‘real world’ entities that belong in $\mathcal{U}$.

Thus, the definition of $\boldsymbol{si}(m, f, p)$, the nature of $\mathcal{U}$ and their influence on this formalization of the symbol grounding problem is an interesting problem with exciting prospects for future work.

3.4. Representational System Classes

The notions of and notation for semantic interpretability may now be used to formalize the different attitudes toward the symbol grounding problem. My goal is to analyze the space of finite formal systems, $\mathcal{F}$, and categorize these into a hierarchical ontology based on the semantic interpretability of their representations. That is, Harnad’s criteria 1–7 are assumed and a hierarchy is built from specializations and generalizations of Harnad’s 8th criteria.

For example, $\mathcal{SIR}$ is one such class intended to denote the set of systems with fully Semantically Interpretable Representations. This class is intended to match the original statement of Harnad’s 8th criteria, so that Harnad’s thesis that a symbolic system cannot be intelligent is formally expressed as follows:

$\forall f : \mathcal{SIR} \bullet \neg (f \sim I_{\text{Harnad}})$

◊ “Every machine consisting of only *Semantically Interpretable Representations* is insufficient for solving the problems corresponding to Harnad’s sense of intelligence”

An extension of the ‘solves’ operator to representational classes so that $c \sim i \Leftrightarrow \exists f : c \bullet f \sim i$ further simplifies Harnad’s thesis to:

$\neg\, \mathcal{SIR} \sim I_{\text{Harnad}}$

By exploring variations of Harnad’s definition of symbolic systems, I have identified a range of representational system classes beyond $\mathcal{SIR}$. In particular, I have identified six representational system classes that appear to capture the philosophical position of many AI researchers and that form (by their definition) a hierarchy of representational expressiveness. Each class represents a set of formal systems and is therefore of the type $\mathbb{P}\mathcal{F}$.

The following subsections describe the classes ordered from most restrictive to most general. Each class has been defined so that it subsumes (*i.e.*, is a super-set of) those classes that have been presented before it. The subsumption property can be proven syntactically from the formal definitions in this work and holds irrespective of the foundational assumptions of this work.

3.4.1. Context-Free Semantically Interpretable Representation

$\mathcal{CFSIR}$: Every symbol must have a unique semantic interpretation.

$\mathcal{CFSIR} = \{f : \mathcal{F} \mid \exists\, m : \mathcal{SI} \bullet \forall\, p : \mathcal{P} \bullet \boldsymbol{si}(m,f,p) \wedge \bigcup(\mathrm{dom}\ m) \subseteq \boldsymbol{a}(f,p) \wedge \forall\, r : \mathrm{dom}\ m \bullet \#r = 1\}$

Systems in $\mathcal{CFSIR}$ are those in which every single symbol has some meaning (given by valuation function, m): symbols do not acquire meaning from their context in some larger representational unit such as a sentence or structure.

A system that, for example, uses the symbol ***Mouse*** to represent the common house mouse but also uses that same symbol in the context of a Disney movie to state that Mickey Mouse is a mouse *could not* be regarded as making use of symbols with universal and unambiguous meanings. Consider, for example, posing the system the question of whether a mouse can speak English. In a symbolic system with context-free semantic interpretations, such distinctions would first need to be translated into separate symbols: *e.g.*, ***Natural_Mouse*** and ***Cartoon_Mouse***. Whether complete translations are possible for all symbolic systems remains an open question and is, in fact, the question of whether $\mathcal{SIR} = \mathcal{CFSIR}$.

Systems in $\mathcal{CFSIR}$ include:

1. Semantic web systems based on RDF: every resource is denoted by a globally unique URL that is intended to capture some unique context-free interpretation. RDF provides no facility to contextualize the truth of an RDF triple without complex reification [15].
2. Traditional database systems: in typical database designs, each record is intended to have a unique and context-free interpretation.
3. Internal representations of industrial robotic systems: every variable in the control system of an industrial robot can be assigned a unique meaning (*e.g.*, joint position, current distance sensor reading, *x*-coordinate of a recognized widget).

3.4.2. Semantically Interpretable Representation

$\mathcal{SIR}$: Every representational unit must have a semantic interpretation.

$\mathcal{SIR} = \{f : \mathcal{F} \mid \exists\, m : \mathcal{SI} \bullet \forall\, p : \mathcal{P} \bullet \boldsymbol{si}(m, f, p) \wedge \cup(\mathrm{dom}\ m) \subseteq \boldsymbol{a}(f, p)\}$

The set $\mathcal{SIR}$ corresponds to those systems that match Harnad's original definition of formal symbolic systems. Every representational unit in the system must have a semantic interpretation and every symbol used by the system belongs to a representational unit.

Systems in $\mathcal{SIR}$ (but not $\mathcal{CFSIR}$) include:

1. John McCarthy's early proposal to incorporate context into formal symbolic systems [119] and related efforts that have arisen from this, such as PLC and MCS [153].
2. The Cyc project's symbolic engineering, wherein symbols have meaning and that meaning is given within context spaces [102].

3.4.3. Iconic and Symbolic Representation

$\mathcal{ISR}$: Representational units may have semantic interpretation. Non-interpretable representational units must be composable as sets that in aggregate have semantic interpretation and resemble their meaning.

$\mathcal{ISR} = \{f : \mathcal{F} \mid \exists\, m : \mathcal{SI} \bullet \forall\, p : \mathcal{P} \bullet \boldsymbol{si}(m, f, p) \wedge \boldsymbol{iconic}(\cup(\mathrm{dom}\ m) \setminus \boldsymbol{a}(f, p))\}$

In $\mathcal{ISR}$, individual representational units need not have a semantic interpretation but may be part of an aggregate that is semantically interpretable as a whole. Such aggregations in $\mathcal{ISR}$ have a structure that somehow "resembles" the meaning of their referent (*e.g.*, by projection or analogy)—they must be iconic.

For example, the individual pixels of a high-resolution image could not typically be regarded as having a particular meaning when considered individually; they simply record the intensity and color of light at a given angle taken from a given perspective. However, in aggregate may be understood as denoting the object that they depict. A system with hybrid visual/symbolic representations could refer to its symbolic knowledge to answer factual queries but use high resolution images to compute answers to queries about nuanced physical traits or to compare the appearances of different people. Iconic representations in some way resemble their meaning, be they low-level resemblances such as images, 3D models and perspective invariant features or more abstract forms such as graphs representing the social networks in an organization or the functional connections of components.

Precisely what, then, does it mean for a symbol to resemble its meaning? If a system resembles its meaning, then a small representational change should correspond to a small semantic change. That is, for a set of iconic representations, i, there should exist a computable representational distance function, ***rdist***, and a semantic distance function (with some real world meaning and therefore a member of $\mathcal{U}$), ***sdist***, and error limit, ε, such that:

$$\textbf{\textit{iconic}}(i)$$

$$\Leftrightarrow$$

$$\forall\ i_1, i_2 : i \bullet |\textbf{\textit{rdist}}(i_1,i_2) - \textbf{\textit{sdist}}(m(i_1),m(i_2))| \leq \varepsilon$$

System proposals in $\mathcal{ISR}$ include:

1. Harnad's [65] proposed 'solution' to the symbol grounding problem via the use of visual icons.
2. Reasoning performed within Gärdenfors' conceptual spaces framework, especially as a mechanism for embedding greater 'semantics' into symbolic systems such as the Semantic Web [49]. The cases or prototypes of a case-based reasoner may also be regarded as a similar form of iconic representation.
3. Setchi, Lagos and Froud's [154] proposed agenda for computational imagination.

3.4.4. Distributed Representation

$\mathcal{DR}$: Representational units may have semantic interpretation. Non-interpretable representational units must be composable as sets that in aggregate have semantic interpretation.

$$\mathcal{DR} = \{f : \mathcal{F} \mid \exists\ m : \mathcal{SI} \bullet \forall\ p : \mathcal{P} \bullet \textbf{\textit{si}}(m,f,p)\}$$

Every element of the set $\mathcal{DR}$ is a finite system that makes use of two kinds of representations: those that can be systematically assigned meaning and those that only have meaning in aggregate (and may be of arbitrary form). That is, $\mathcal{DR}$ requires semantic interpretability but does not require that the units of semantic interpretation correspond to the same representational units that are manipulated by the rules of the formal system.

Consider, for example, a neural network that has been trained to identify the gender of a human face. Some of the network's output nodes may be specifically trained to activate in the presence of masculine features. These output nodes, in addition to the hidden layer neurons that feed into the output nodes, may in aggregate be seen as meaning 'facial masculinity'. Even though it may be impossible to assign a coherent semantic interpretation to the representations and values of the hidden layer neurons that the formal system manipulates, the aggregated network can be seen as capturing specific real-world meaning.

Examples of systems that make representational assumptions or restrictions consistent with $\mathcal{DR}$ include:

1. Hybrid systems wherein neural networks have been trained under supervised learning to recognize symbols of a higher level symbolic reasoning processes. Indeed, all forms of supervised machine learning where the internal structure of the induced representations are regarded as a black box would be consistent with the restrictions of $\mathcal{DR}$.
2. Neural-symbolic systems that, for example, perform symbolic reasoning within connectionist mechanisms (*e.g.*, [90]).

3.4.5. Unconstrained Representation

$\mathcal{UR}$*: Representational units may or may not have any particular semantic interpretation.*

$\mathcal{UR} = \mathcal{F}$

Every element of the set $\mathcal{UR}$ corresponds to a problem-set that may be solved by a finite formal system (*i.e.*, a Turing-complete machine). The set $\mathcal{UR}$ therefore corresponds to the capabilities of computational systems, irrespective of whether their internal representations can be assigned particular semantic interpretations.

Examples of systems that make representational assumptions or restrictions consistent with $\mathcal{UR}$ include:

1. Neural-network-based systems in which output or activity is triggered entirely by arbitrary connectionist processes (*e.g.*, [113,33]). In such systems, input nodes correspond to raw sensory data, output nodes are motor commands corresponding to actions and internal hidden nodes are trained without regard to the development of meaningful cognitive symbols (*i.e.*, black-box intelligence). None of the nodes in the network can be said to capture meaningful semantic interpretations.
2. Universal computable models of intelligence such as $\text{AI}\xi^{tl}$ [76]. Such approaches emphasize computation or modeling that maximizes a reward function without regard for the semantic interpretability of the computational processes (though there is an implicit assumption that the most successful representations are likely to be those that best capture the environment and therefore are likely to acquire semantic interpretability in the limit).
3. Reactive systems, such as those concrete implementations of Brooks [20]. Such systems do not attempt to explicitly model the world (or may only partially model the world) and so lack semantic interpretability.

3.4.6. Non-Formal

$\mathcal{NF}$: Representation units may or may not have any particular semantic interpretation and may be manipulated by rules that are beyond formal definition (such as interaction with the environment or hyper-computational systems).

$\mathcal{NF} = \mathcal{F} \cup \mathcal{F}^*$

The class $\mathcal{NF}$ extends $\mathcal{UR}$ with a set of 'enhanced' formal symbolic systems, $\mathcal{F}^*$—systems with distinguished symbols that are connected to the environment[*]. While problems associated with action in the physical environment may already be found in the set $\mathcal{P}$, and these may already be solved by systems of other representational system classes (such as $\mathcal{UR}$), the set $\mathcal{NF}$ includes those systems that use embodiment directly as part of its deductive processes: systems where the environment is 'part' of the reasoning, rather than merely the 'object' of a solution. $\mathcal{NF}$ encompasses systems that, for example, need the environment to generate truly random sequences, to perform computations that are not finitely computable on a Turing machine but may be solved by physical systems, to exploit some as-yet-unknown quantum effects, to build physical prototypes or, more simply, to solve problems about objects and complex systems that simply cannot be described or modeled in sufficient detail on a realizable computer system.

Examples of systems and research that make representational assumptions or restrictions consistent with $\mathcal{NF}$ include:

1. Embodied robotics, in the true spirit of Brooks' vision, that treat 'the world [as] its own best model' and that refute the possibility of a disembodied mind [94]. Such work regards direct sensory experience and manipulation of the physical environment throughout problem solving as an essential part of intelligent thought. It considers that intelligence has co-evolved with the environment and sensory abilities and that it is not sufficient merely to have a reactive system but that higher order intelligence arises from the complex interactions between reactivity and the environment. Note, however, *actual* reactive robots/systems to date would, in fact, be better classified in $\mathcal{UR}$ (as I have done) because they do not yet operate at a level of interaction beyond primitive reactivity.

2. Models of intelligence and consciousness that are not Turing-computable or constrained by Gödel's incompleteness theorem. Examples of these may be found in work such as that of

[*]For example, a Turing machine can interact with the environment by reserving a segment of its tape as 'memory mapped' I/O. Symbols written to this segment of the tape will manipulate actuators and sensory feedback is itself achieved by a direct mapping back onto symbols of the I/O segment of the tape.

Penrose [140] postulating significant (but currently unspecified) quantum effects on intelligent thought and consciousness.

3.5. Discussion

The representational system classes do not only formalize many of the loose and ambiguous concepts that appear in debate on symbol grounding but are also useful for rapidly communicating assumptions, analyzing the symbol grounding problem and generating new research questions. For example, Harnad's claims may be succinctly summarized as $\neg\ \mathcal{SIR} \sim I_{\text{Harnad}} \wedge \mathcal{ISR} \sim I_{\text{Harnad}}$ (*i.e.*, semantically interpretable representations are not enough; one needs iconic representations).

Syntactic proof techniques can show that the representational classes form a hierarchy: $\mathcal{CFSIR} \subseteq \mathcal{SIR} \subseteq \mathcal{ISR} \subseteq \mathcal{DR} \subseteq \mathcal{UR} \subseteq \mathcal{NF}$. This can be seen by observing that each class is defined by removing or generalizing one criteria in the definition of the proceeding class. Given that the representational classes form a hierarchy, it follows that the problems that each class may solve also forms a hierarchy (*i.e.*, a hierarchy of intelligence). It remains an interesting question whether these hierarchies are strict. Are there classes of representational systems C_1 and C_2 such that $C_1 \subseteq C_2$ but there exists some definition of intelligence I where $\neg\ C_1 \sim I$, and $C_2 \sim I$? This is, is C_2 strictly more intelligent than C_1 (with respect to a given definition of intelligence)? I denote a strict hierarchy as $C_1 < C_2$.

With an extreme definition of semantic interpretability such that every system is semantically interpretable as a model of itself (see Section 3.3), then the hierarchy collapses into a single equivalence. However, my intuitions are that, for a realistic definition of semantic interpretability and intelligence, the hierarchy is strict. I plan to show this in future work but I briefly outline some reasons for believing so here:

- $\mathcal{CFSIR} < \mathcal{SIR}$, because even though the context-sensitive symbols of $\mathcal{SIR}$ could be systematically mapped into sets of context-free symbols in $\mathcal{CFSIR}$ (*e.g.*, ***Mouse*** → ***Cartoon_Mouse***), the potentially unbounded regress of contexts may make it impossible to ensure that this mapping remains finite when problem-sets are unbounded (*i.e.*, it can be done for any *particular* problem but not *in general*, in advance of knowledge of the problem).
- $\mathcal{SIR} < \mathcal{ISR}$, following the arguments of Harnad.
- $\mathcal{ISR} < \mathcal{DR}$, because I believe that there are pathological concepts that *emerge* from complex chaotic systems so that iconic representations of structure or appearance hinder rather than enhance performance (*i.e.*, systems in which emergence is crucial to understanding the global system behavior but for which properties of emergence cannot be predicted from local or structural analysis).

- $\mathcal{DR} < \mathcal{UR}$, because I believe that there are pathological situations where an attempt to analyze the situation using concepts diminishes the ability to learn appropriate behaviors (compare this to the manner in which human beings 'discover' false patterns in truly random data, hindering their ability to make optimal decisions using that data).
- $\mathcal{UR} < \mathcal{NF}$, because even though the universe may be formally computable, it may not be possible for any agent *situated within* the universe to describe the universe in sufficient detail such that a Turing machine could compute the solution to all 'intelligent' problems.

Finally, I emphasize again that I do not claim to have *solved* the problem. Instead, this framework reduces the symbol grounding to two long-standing philosophical challenges: the selection and definition of intelligence, I, and the problem of the nature of 'meaningful' entities in the universe (*i.e.*, the set $\mathcal{U}$ and consequently how to define $\boldsymbol{si}(m,f,p)$). While the framework does not yet offer precise guidance towards solving these sub-problems, it provides straightforward machinery by which the symbol grounding problem can be understood in such terms. The contribution of this work lies in formalizing the connections between sub-problems and, thereby, narrowing the ambiguity in the problem and closing opportunities for circular reasoning.

3.6. Implementing Frameworks

The symbol grounding problem concerns the creation of systems that have intrinsic meaning and that model the real world. Symbol grounding is therefore an important consideration that should inform the design and creation of intelligent systems and artificial commonsense. For example, if iconic reasoning is a *necessary* condition for intelligent behavior, then one can immediately conclude that the semantic web is insufficient for deeply intelligent behavior because it falls in the class $\mathcal{CFSIR}$ (even though it may have many other useful applications in query, integration and search).

My motivation for exploring the symbol grounding problem in this dissertation is to better understand the representational needs for an intelligent system. The best chance of successful implementation is an effort that meets all *necessary* requirements and ideally a minimal set of *sufficient* requirements. The representational system classes form a hierarchy and so identifying appropriate bounds on representational requirements can assist in selecting suitable representations and mechanisms for commonsense reasoning.

For example, the point of Harnad's Chinese-Chinese dictionary 'merry-go-round' argument is to show that symbolic methods alone are insufficient for solving the symbol grounding problem and therefore for creating intelligent behavior. He then goes on to argue that iconic and symbolic methods in combination are sufficient for such behavior.

The work described in this thesis is philosophically aligned and inspired by Harnad's thesis but I also view the problem of selecting representations from a more pragmatic perspective, prioritizing efforts where they are most likely to succeed.

Consider the situation—as it appears to be the in present case—in which there is insufficient knowledge of symbol grounding to conclusively rule out any given class of representations. With no prior knowledge, unlimited resources and unlimited time, one would develop the most general class, $\mathcal{NF}$, as it subsumes every possible representation scheme. However, as engineers with finite resources and limited time, it is necessary to prioritize development efforts towards cases that have the highest expected reward when considering likely benefits, probability of success and development cost.

It turns out the class of systems that use iconic and symbolic representations, namely $\mathcal{ISR}$, appears to be an ideal candidate for engineering intelligent systems:

1. $\mathcal{CFSIR}$, $\mathcal{SIR}$ and $\mathcal{ISR}$ use representations that have universal structure and interpretation. Such structure lends itself to ready comprehension by human engineers and therefore makes them prime candidates for engineering and debugging (as opposed to more general classes that do not readily have meanings or interpretations). $\mathcal{ISR}$ is therefore the most general class that lends itself to large-scale engineering.

2. The classes $\mathcal{CFSIR}$ and $\mathcal{SIR}$ are very well understood, comprising the vast majority of traditional AI ('Good Old Fashioned AI') research, while $\mathcal{ISR}$ is the next most complex class that remains comparatively untested. Given that traditional AI techniques have not resulted in general purpose intelligence, it is worthwhile considering the next most general class, $\mathcal{ISR}$.

3. As the hierarchy is climbed to more general representational system classes, it becomes more expensive and difficult to integrate with established systems. For example, $\mathcal{NF}$ and $\mathcal{UR}$ act like 'black boxes' (from a semantic perspective) and their representations do not readily lend themselves to representation-level integration with engineered systems. Given the extensive work in $\mathcal{CFSIR}$ and $\mathcal{SIR}$ that today remains largely unsuccessful, $\mathcal{ISR}$ is the next most specific class likely to integrate with existing systems and have a dramatic financial impact.

If intelligent and commonsense-enabled systems development has a pragmatic focus, the symbol grounding problem suggests that effort is best directed towards the class $\mathcal{ISR}$ and, where possible, towards the most generic and reusable iconic representations in $\mathcal{ISR}$.

In this dissertation, I seek to create a hybrid system that combines symbolic and iconic reasoning into a unified framework. I use a general-purpose scheme that integrates and reasons with a wide range of iconic and symbolic methods. While there is an infinite variety of possible forms of iconic representation (as there is also an infinite variety of logical formalisms), the representation that I propose is intended to be sufficiently general such that if intelligent commonsense reasoning can be automated in $\mathcal{ISR}$, then it should be possible to automate intelligent commonsense reasoning in the Comirit Framework.

$\mathcal{ISR}$ is a large and general class. Systems that combine iconic and symbolic reasoning can be built in frameworks other than Comirit. Whether or not Comirit is ultimately successful in commercial commonsense reasoning systems, the analysis of the symbol grounding problem presented in this chapter may still find potential application in directing the efforts of any development of intelligent or commonsense-enabled systems.

Finally, I recognise that the kind of meta-argument presented here may not be as compelling as a specific argument directed towards a particular system design. A more natural argument for iconic representation may be deduced, for example, from empirical findings in cognitive science or biology. However, I prefer neither to be drawn into the debate over the interpretation of experimental data nor to propose yet another model of human cognition. Instead, I hope that the research agenda in the remainder of this dissertation is positioned as best chance of avoiding previous mistakes and exploring new approaches.

3.7. Conclusion

This formal framework of representation grounding helps clarify the contention in important work on symbol grounding as stemming from arguments about different kinds of representational system classes. The framework consists of six classes to this end: $\mathcal{CFSIR} \subseteq \mathcal{SIR} \subseteq \mathcal{ISR} \subseteq \mathcal{DR} \subseteq \mathcal{UR} \subseteq \mathcal{NF}$. These capture many perspectives on the symbol grounding problem: the classes have significant power both for explanation and investigation. The classes may be used to quickly express assumptions in addition to providing abstractions useful for further exploration of the problem, comparison of existing work and the development of novel conjectures.

The analysis of this chapter serves as my inspiration and motivation for hinging the development of commonsense reasoning systems on general purpose iconic representations. In the remainder of this dissertation, I design a framework around this principle: applying graph-based iconic representations to the major representational challenges of commonsense reasoning and combining them with the full capabilities of symbolic logical languages.

Finally, while this framework guides my research program, I view it also as an open challenge to other researchers in Artificial Intelligence. I readily admit that this analysis has a number of weaknesses: it is based on assumptions that may be subject to dispute, it offers only one perspective of the grounding problem and it is not yet sufficiently complete to be able to prove or disprove the equivalence of representational classes. I hope that this work motivates other researchers to attempt their own formalization so that discussion can be lifted from informal speculation to the level of formal mathematical proof. In fact, I would regard disproof of part or even all of this formalization as a success in that disproof would demonstrate the falsifiability of the formalization and contribute to the understanding of the symbol grounding problem by finally eliminating one explanation of the problem *with certainty*.

Chapter 4

Inside Comirit

Comirit is a framework for implementing commonsense reasoning systems. Comirit represents the principal and novel contribution of this dissertation and I propose it to affirm my thesis that *artificial commonsense can be created within constrained time and resources, from present-day technologies* (per Section 1.3). In this dissertation, I describe the theory, design and implementation of the framework. I also explain how the framework can be applied to commonsense reasoning problems.

In Comirit, I use well-understood technologies in new and innovative ways. An analogy may be made to music: a composer can create new experiences without inventing new notes, by combining existing notes in novel forms. Thus, the contribution and value of Comirit lies in the novel manner that existing technologies have been packaged and re-purposed for commonsense reasoning and elegantly integrated into a coherent system.

Having already explained the functional and methodological characteristics of this dissertation and my objectives (Chapters 1–3), the purpose of this chapter is to place the design of Comirit into a high-level context, to introduce the overall design of the framework and explain how it relates to my objectives. I will describe Comirit in detail over the following three chapters (5–7) and then evaluate the framework in Chapter 8.

4.1. What is Comirit?

Comirit is the name of this research project and the broad range of designs and software systems that it concerns. I have developed Comirit to advance my thesis by directly and constructively showing how commonsense reasoning systems may be built. A commercially robust system for direct evaluation is beyond the scope of this project so I evaluate the design and several prototypes to demonstrate my thesis.

This written dissertation concerns the creation of several artifacts: the Comirit Design, Comirit Framework, Comirit System and Comirit Embedding. There are important but subtle distinctions between each.

1. *This written dissertation* is, in part, a detailed and open-ended design: the *Comirit Design*. I use object-oriented designs, pseudo-code and careful explanations in this dissertation to describe Comirit in sufficient detail for implementation by a skilled engineer.

2. The Comirit Design can guide a skilled engineer in the implementation of a *concrete software artifact* known as a *Comirit Framework*. The framework is the skeleton of a working commonsense reasoning system. It need only be implemented once because it can be adapted to many domains and problems. In this dissertation, I highlight the many opportunities for optimizing an implementation of the framework.

2. A working implementation of the Comirit Framework can then be used by software and knowledge engineers to produce a *Comirit System*. That is, the Comirit Framework is extended and adapted to particular problems or applications to create a Comirit System. In this dissertation, I provide examples and suggestions that indicate how this may be accomplished for typical reasoning problems.

4. A Comirit System (*i.e.*, a specific commonsense reasoning system) can then be deployed or embedded into a software agent or a robotic agent to provide the agent with commonsense reasoning capabilities, thereby resulting in a *Comirit Embedding*.

These four artifacts may be seen as analogous to the design of a database management system (DBMS). A written *design* explains the implementation of the DBMS, that, when *implemented*, may in turn be applied to a given situation as a database *instance* and which, finally, may be *embedded* in, or queried by, a Business Information System.

The primary emphasis of this dissertation is the design (and evaluation) of the Comirit Framework: its details, trade-offs, benefits and disadvantages, in addition to guidelines and commentary on how the Framework may be best applied to create a useful System.

In the following subsections, I consider each of these artifacts of Comirit.

4.1.1. What is the Comirit Design?

In this dissertation, I aim to demonstrate my research thesis. My methodology is based on a constructive method of implementation and evaluation and so I include in this dissertation (Chapters 4–8), a detailed design of the artifacts that I use for evaluation.

I have used formal notation, object-oriented design, pseudocode and informal description to explain the implementation of the Comirit Framework in such a way that any trained software engineer can repeat my work. That is, Comirit could be successfully implemented by presenting Chapters 4–7 to a software engineer (or a team of engineers), without requiring prior knowledge of the art in commonsense reasoning.

4.1.2. What is the Comirit Framework?

I use the term *framework* as it is used in the object-oriented design community. The Comirit Framework provides base abstractions for solving problems and a facility for inversion of control:

- The Comirit Framework offers a set of reusable and general-purpose abstractions that are readily adapted to commonsense reasoning problems. Representation schemes and reasoning algorithms are implemented as specializations of these abstractions.
- The Comirit Framework provides a common-use container that, through 'inversion of control', orchestrates the interaction and integration of a diverse and open-ended set of representations, reasoning mechanisms and learning mechanisms.

The Comirit Framework is general-purpose and can accommodate many forms of reasoning. However, its base abstractions have been designed with particular emphasis on the combination of simulation and logical deduction into an integrated and coherent whole.

4.1.3. What is a Comirit System?

The Comirit Framework does not specify the actual commonsense knowledge or the rich models of the real-world that commonsense reasoning requires. For the Comirit Framework to be useful, commonsense knowledge and meta-knowledge must be encoded as simulations, logical formulas, deduction rules, modal operators and learning algorithms. An instance of the framework configured with such commonsense knowledge is referred to as a *Comirit System*.

The actual population of such a knowledge base would represent the bulk of a commercial development and is beyond the scope of this dissertation. Populating a comprehensive commonsense knowledge-base in Comirit is likely to be a project involving a small team of knowledge engineers for one to three years or would require a dedicated environment that is conducive to autonomous (and supervised) machine learning. Nevertheless, the abstractions used by Comirit are highly conducive to cost-effective knowledge engineering so that such undertakings would be less expensive than otherwise possible. In Comirit, acquisition of simulations is primarily a low-cost, low-skill activity that can be performed quickly. In domains such as physical reasoning, Comirit can autonomously populate its own knowledge base 'on the fly' using online-learning rules for 3D visual reconstruction and associative learning. A small amount of traditional knowledge engineering is required for populating abstract knowledge and meta-reasoning strategies but Comirit is designed to minimize such effort.

The representations used in Comirit Systems are, ideally, shared globally. A Comirit System might combine a small amount of local knowledge with the commonsense knowledge in an expansive centralized database. This knowledge-sharing avoids the need to manually re-engineer all commonsense knowledge in every deployment but still allows flexibility in describing application-specific knowledge.

4.1.4. What is a Comirit Embedding?

A Comirit System may be seen as analogous to a database, a theorem prover or a machine learning algorithm. It is only of academic interest when used in isolation but has practical use when applied to a problem and embedded in a complex system.

Comirit may be embedded into a complex system in three different ways:

1. In a passive role that provides commonsense 'theorem' proving or query answering services to the software or robotic system. In this role, Comirit would be embedded via a standard synchronous query-response API.
2. In an active role that uses commonsense knowledge to generate plans or to orchestrate complex behaviors in the software or robotic system (to achieve or maintain predefined goals). In this role, Comirit would direct the behavior of the system, either by hosting domain-specific components directly within the framework or via an asynchronous API based on remote callback functions.
3. In a hybrid combination of active and passive roles, autonomously directing some behaviors in order to achieve goals while enacting behaviors under the direction of yet other components in the software or robotic system.

The preferred approach to the embedding will depend upon the particulars of each application environment. In this dissertation, I do not describe the specific implementation details and low-level interfaces of these embeddings. However, I will describe the concessions that the Comirit Framework makes to allow for any such scheme.

4.2. Design of the Comirit Framework

Being a framework, the design of Comirit concerns a set of base representations and a mechanism for inversion of control:

- **Base Representations.** Comirit, as an integrative framework, supports all representation schemes. Its most basic abstractions are opaque objects, without constraints on the structure and type of subclasses. However, Comirit has been designed with particular emphasis on graph-based numerical simulation and so includes a hierarchy of abstractions for iconic representation and reasoning. Graph-based simulations allow for efficient and expressive reasoning across many domains, with a uniform representation that reduces complexity and simplifies knowledge elicitation.
- **Inversion of Control.** Comirit permits a range of specialized reasoning and learning algorithms to be combined within a single system. The various algorithms in Comirit are orchestrated by a top-level search strategy that is a generalization of the proof method of analytic tableaux. Numerical simulation, symbolic deduction and connectionist learning algorithms are treated equally when combined in a generalized analytic tableau. They are orchestrated as a coherent and unified system.

Traditionally, tableau algorithms have been used for automated reasoning, rather than as a framework for machine learning. For clarity, I therefore develop the control mechanism in two steps: first, I describe a general purpose hybrid *reasoning* algorithm and, second, extend it to support *learning* algorithms.

In Comirit, graph-based representations are combined with the control mechanism. Comirit may, therefore, be seen as comprising a total of three 'layers' of functionality:

1. A set of abstractions designed with an emphasis on *representing* simulations
2. which are used by a *control mechanism* for hybrid reasoning
3. and that also integrates *learning*.

4.3. Representations

In Chapter 3, an analysis of the symbol grounding problem identified iconic representations as important in the construction of practical artificial commonsense.

One such iconic representation is computer simulation. Computer simulations are models that resemble their meaning: a small change to a simulation typically corresponds to a small change in the real-world correlate that the simulation is modeling (and *vice versa*). Recall that this is the criteria for defining ***iconic*** in Section 3.3.

Not only do simulations meet the essential criteria of being iconic but also there are many practical reasons why simulation is a desirable representation for commonsense knowledge. They are efficient, they are highly visual, they are easy to develop and they are easy to populate. Furthermore, modern simulations that occur in computer games, computer animations and virtual worlds have detailed and life-like environments that may be rich resources for commonsense knowledge.

As such, Comirit uses numeric simulations as the principal iconic mechanism for commonsense representation and reasoning. In Comirit, simulations are constructed in a generic multi-purpose representation, based on mathematical graphs that can uniformly model a wide range of problems. Indeed, the original inspiration for Comirit was the flexibility and power of graph-based simulations; the rest of the Comirit architecture evolved in my research from a desire to compensate for the deductive limitations of simulation. The framework has grown into a hybrid commonsense reasoning architecture, where iconic representations in the form of graph-based simulations remain the crucial representational component.

A simulation in Comirit encapsulates practical knowledge about a commonsense situation. For example, when reasoning about how a human might eat lunch, Comirit can instantiate the virtual meal of Figure 4.1. The simulation is complete with models, physical laws and parameters to accurately simulate real-life behavior so that deducing future states is reduced to evaluating the simulation. Comirit can instantiate many copies of these simulations to explore possibilities and what-if scenarios during reasoning.

For example, simulation may be used to conduct virtual experiments with the cup of coffee and determine the safety of different interactions. Simulations can predict the effects of an interaction and analyze the reactions (such as whether or not a 'mess' has occurred when a sideways force is applied to the tip of the cup), as illustrated in Figure 4.2.

The flexibility of simulation is a key strength of the representation scheme. Simulations that incorporate 3D physics are sufficient to model the behavior of most mechanical systems. However, the representation scheme is not just limited to physics: any system that is driven by or can be predicted by laws which have local effects is well suited to simulation-based reasoning.

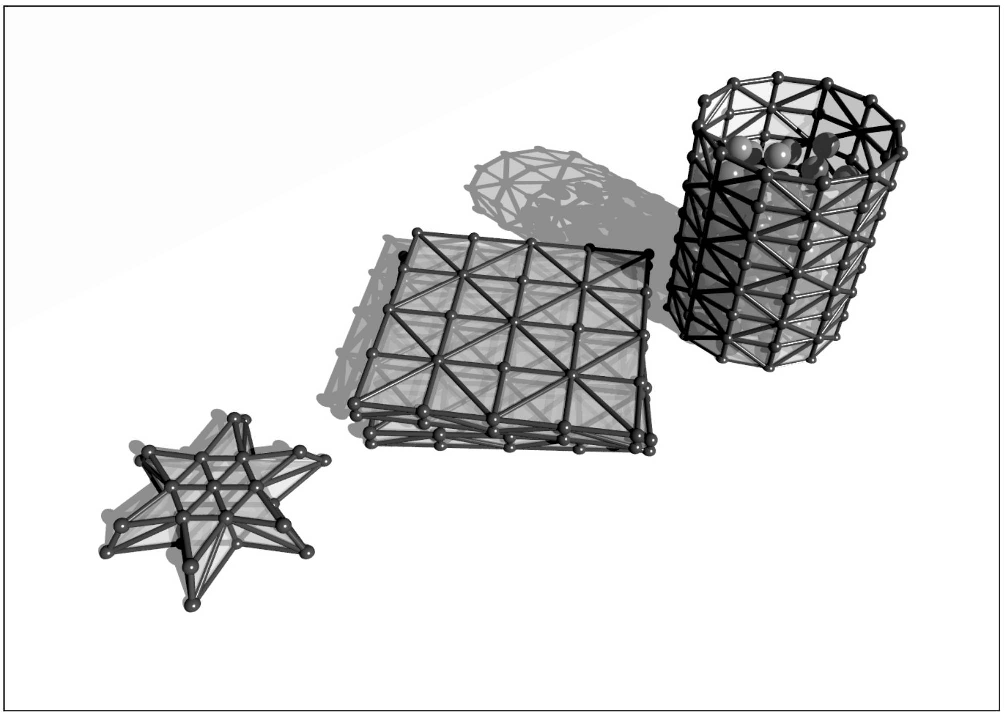


Figure 4.1: A virtual meal (cookie, sandwich and cup of coffee)

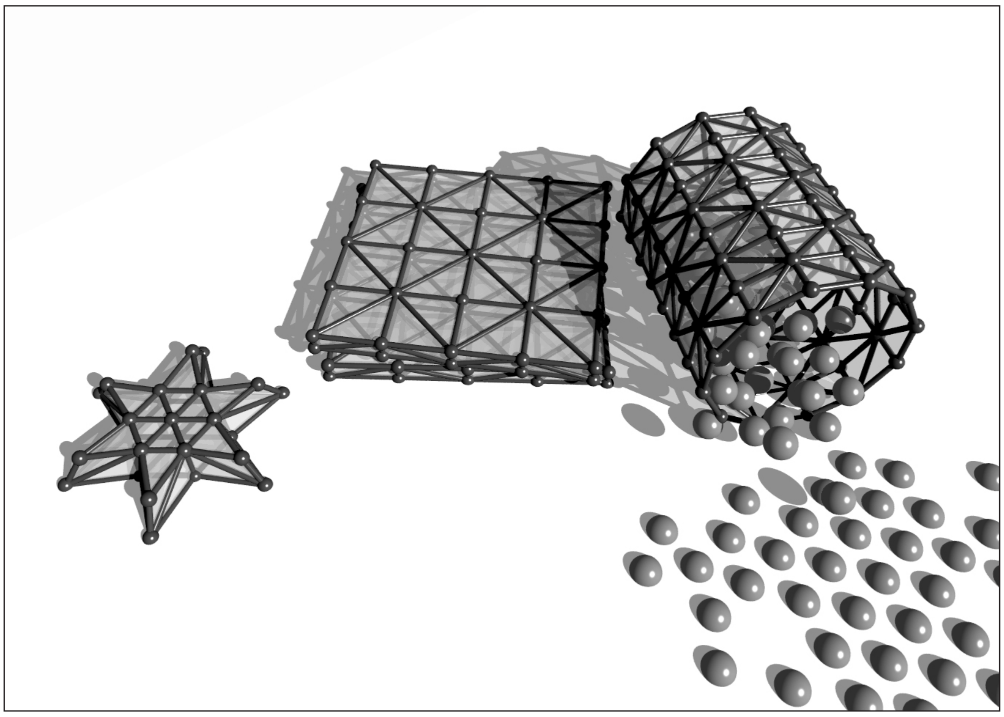


Figure 4.2: A virtual meal with a spilled cup

While simulation is crucial to Comirit and is intended as the primary form of knowledge representation, the framework is not restricted to iconic simulation-based reasoning. Comirit is a general purpose framework that can incorporate a wide variety of representations and mechanisms. In particular, simulations are intended to be combined with the representations and methods of logical deduction (such as first order logic) and the framework also provides generic abstractions that allow for future integration with new representation schemes as they are invented.

The control mechanisms and strategy for instantiating and integration simulations are described in the following section.

4.4. Control

Simulation is a powerful and efficient tool for commonsense reasoning but it only supports a 'forward chaining' inference mode. Simulations must follow the 'arrow-of-time', deducing possible or likely future states from current conditions. One cannot simulate spilled milk in reverse to discover a likely cause. It is more effective to simulate a particular event such as dropping an open milk carton and then test if the outcomes are consistent with observations.

For many problems of commonsense, simple anticipation is sufficient and can be performed by the 'forward chaining' inference. Indeed, the egg-cracking benchmark problem and its variations (see Section 2.2) largely concern anticipating the likely outcome of actions performed on a variety of objects and materials. In general, however, commonsense is more than simple anticipation. Simulation may need to be combined with back-chaining to infer possible causes for an event, to speculate on what-if scenarios and to plan complex actions. Furthermore, there are some problems of commonsense for which simulation is irrelevant to the problem-solving process and may be better solved purely by logical deduction, database recall or other mechanisms ('Who is the Queen of England?', 'Where do fish live?').

The top-level control-flow in Comirit is therefore designed to allow diverse reasoning mechanisms to be integrated into a single and coherent whole.

The control-flow in Comirit is a generalization of the method of analytic tableaux. The method is an algorithm for proving logical formulas but I have generalized it so that it may also contain non-logical representations such as simulations. The crucial insight behind this generalization is that the tableau search algorithm is effectively a constructive method of proof that searches over the space of satisfying models (or 'possible worlds'). Any mechanism that can be adapted to reasoning over spaces of satisfying models can be integrated into the tableau search algorithm.

Most simply, the method of analytic tableaux is an algorithm for systematic analysis of a problem by cases. The method works by performing a top-down syntactic traversal of the system's input and constructing models of each case. For example, imagine being stuck in the office on a rainy day with the choice of taking either a bus or taxi home. The commonsense task of deciding whether it is possible to get home without getting wet can be solved using the tableau method. The method constructs models of each case—catching the bus home and catching a taxi home—and then checks for consistency. See Figure 4.3 for a depiction of a tableau-like analysis of this simple problem. The system constructs the tableau from the top down, branching into each of the two cases, and then deduces the effects of the cases to see if they are consistent with the goal of not being wet.

More complex problems can be analyzed by detailed decompositions into many different cases. The potential explosion in cases that might occur with a complete analysis is avoided by using an efficient reasoner that prioritizes the most important and informative cases first. An inefficient reasoner might decompose the problem into every possible bus and taxi route before making a decision about how to get home, whereas an efficient reasoner will make the crucial decision early, using only the abstract properties of the two cases: bus versus taxi.

In Comirit, the tableau method is generalized so that the branches of the tableau search tree can contain not only the logical formulas of standard tableaux but also simulations, belief states, functions

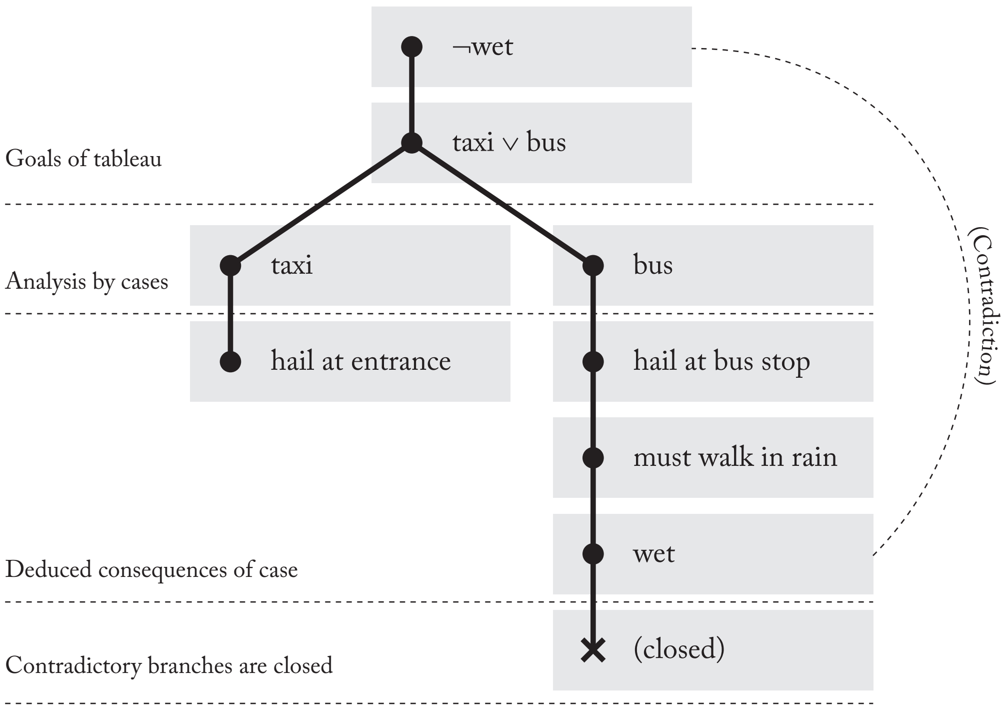


Figure 4.3: Tableau analysis of getting home without getting wet

for computing heuristics and even the update rules of the tableau algorithm itself. The insight behind the successful integration of simulation, tableau reasoning and machine learning is that these mechanisms each analyze cases (or possible worlds): simulations are numerical analyses of particular cases, analytic tableaux are collections of cases and cumulative learning is a process of selecting good models (or cases) of the world. Furthermore, there is no reason that the framework be limited to these forms of reasoning: Comirit is an open-ended architecture that can accommodate virtually any mechanism that can evaluate, create or manipulate cases in the tableau-based analysis.

Finally, with tableau reasoning as the basis for hybrid reasoning in Comirit, there is the opportunity to draw from the large body of literature on efficient tableau reasoning, in addition to the opportunity to incorporate legacy knowledge bases (such as Cyc) that are expressed in purely logical languages (since logical reasoning is a subset of the capabilities of Comirit).

4.5. Control and Learning

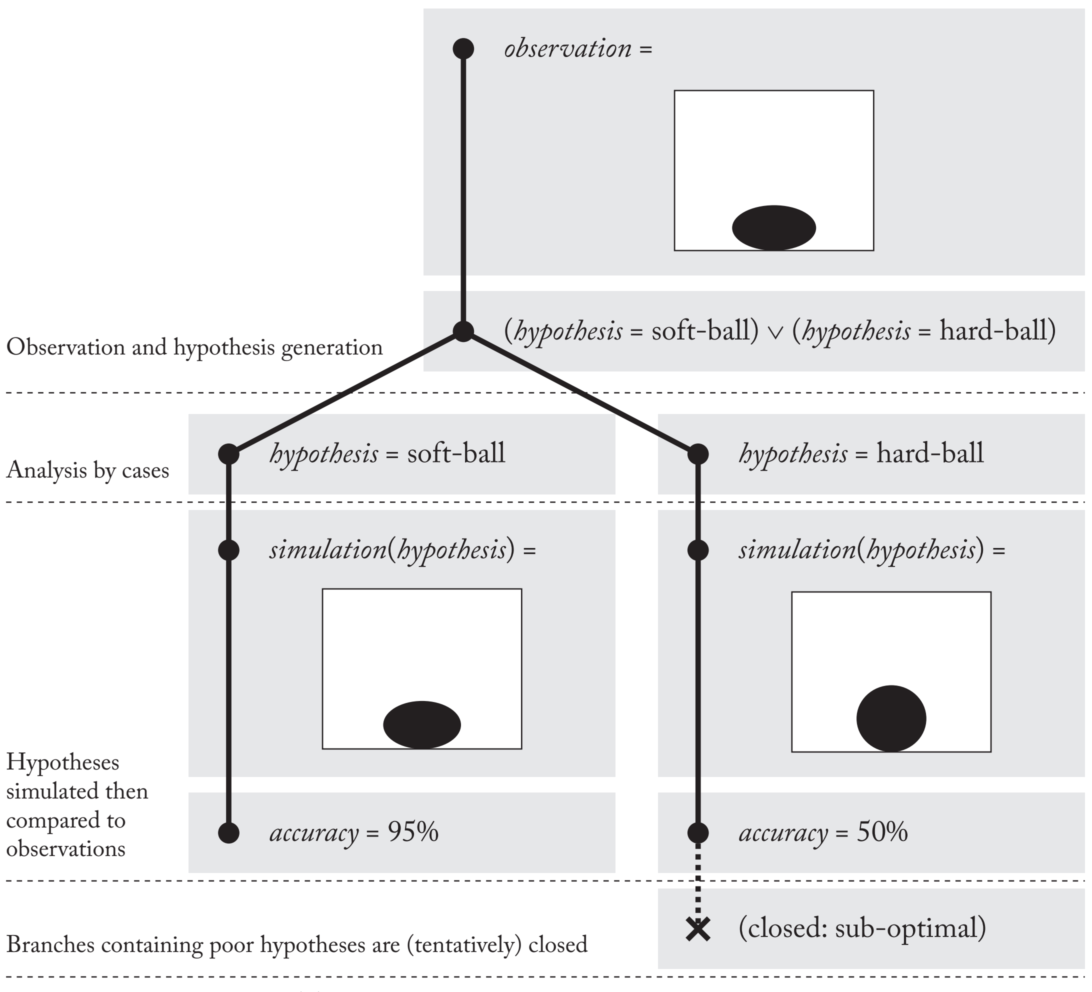
Any attempt to engineer a commonsense reasoning system will inevitably face the question of how the diverse experience of day-to-day life may be input into the system. The rationale behind using simulation as a core representation in Comirit is that it enables the rapid accumulation of deep knowledge and has an ability to generalize quickly from small sets of core laws. However, commonsense is extremely broad and constantly changing so that, even with simulation, the knowledge engineering process is a significant undertaking.

Powerful tool support can drive down the human costs associated with knowledge engineering. 3D modeling tools can assist in the construction of simulations and it would not be difficult to incorporate a visualization tool into a graph editor to allow engineers to instantly test new knowledge. It is reasonable to expect that many parameters of simulations could even be configured automatically. In the case of physical simulations, the 3D structure of an object may be directly digitized by stereoscopic cameras or laser scanners and the parameters that drive a simulation may be learned by motion analysis. In Comirit, these ideas have been extended to allow some forms of knowledge to be acquired fully autonomously.

In Comirit, autonomous learning is performed by standard machine learning algorithms such as Self-Organizing Maps (SOM) and hill-climbing parameter search. In order to integrate learning with the tableau-based control strategy, machine learning is conceptualized as an iterative process of hypothesis generation and testing. That is, learning is akin to postulating a set of possible models of the world and then selecting the models with the highest correlation with the real world. Integration with tableau-reasoning then follows by treating the beliefs (the hypotheses) of the agent as part of the possible models or cases of the world. Learning whether a ball is soft or hard is akin to observing the environment and then selecting between two cases: a world in which one has hypothesized that the ball is soft and a world in which one has hypothesized that the ball is hard, subject to the constraint that beliefs must closely match observations. This process for one round of hypothesizing and evaluation is depicted in the tableau of Figure 4.4 (again, the tableau is read top to bottom).

Learning in Comirit requires further generalization and extension of the basic tableau algorithm. Hypotheses that are generated during learning may be neither true nor false: selection is a matter of choosing the best hypothesis (or set of such best hypotheses). I extend the tableau algorithm to allow branches (or cases) to be ordered and, thereby, prioritize computation and selection of those branches that have the most accurate beliefs.

Learning is a continual and ongoing process. Since new situations unavoidably result in new experiences and since commonsense 'truth' is itself also in a state of constant flux, a commonsense reasoning system must continuously learn. In Comirit, there is a background process that constantly performs learning in the tableau-reasoning framework. It continuously observes the outside world and the



observation =
Observation and hypothesis generation
(hypothesis = soft-ball) ∨ (hypothesis = hard-ball)
Analysis by cases
hypothesis = soft-ball
hypothesis = hard-ball
simulation(hypothesis) =
simulation(hypothesis) =
Hypotheses simulated then compared to observations
accuracy = 95%
accuracy = 50%
Branches containing poor hypotheses are (tentatively) closed
(closed: sub-optimal)

Figure 4.4: Tableau based learning of whether a ball is hard or soft

system's interactions with the world and ensures, through machine learning, that the system's beliefs remain consistent with observations.

4.6. Conclusion

The Comirit Framework combines three major mechanisms to achieve efficiency, flexibility, cost-effectiveness and adaptability:

- **Graph-based generic simulations (iconic representations)**

  Simulations are an extremely efficient way of reasoning about and predicting the consequences of real-world activity and situations.

- **Hybrid integration framework**

  The open-ended integration framework complements the efficiency of graph-based simulations with the flexibility of traditional logical methods of deduction.

- **Automated learning mechanisms**

  Mechanisms for automated learning dramatically reduce the cost of populating and fine-tuning the system's knowledge base, allowing the system to autonomously adapt to novel situations and changing environments.

Over the remainder of this dissertation, I explain and evaluate these mechanisms in detail.

Chapter 5

Representations

Over the past two decades, advances in 3D simulations and computer graphics have led to great improvements in the depth and richness of blockbuster computer games and digital animation. While such virtual worlds are typically intended as works of fiction, they offer convincing and detailed representations of many aspects of everyday experience. Given the difficulty of capturing these same aspects of everyday experience using the traditional techniques of commonsense reasoning, an obvious question then arises: is it possible to directly adapt the methods of virtual worlds to the challenge of commonsense reasoning?

In this chapter, I explain how simulations—the same kind of simulations that underlie virtual worlds, animations and computer games—can be adapted to perform commonsense reasoning. Simulation is an iconic representation: simulations are non-logical and non-symbolic processes that resemble their real-world counterparts. In Comirit, simulation serves as an iconic representation for efficient and realistic reasoning about a wide spectrum of everyday or 'commonsense' scenarios with an ease that is difficult to achieve by other methods.

Even though simulation is used routinely in virtual worlds and many technical domains, it has found surprisingly little application in commonsense reasoning. An explanation for this may be found in the mismatch between the breadth of commonsense reasoning and the typical specificity of a given simulation. This mismatch is addressed in Comirit by an innovative underlying representation that is sufficiently generic to represent problems across many domains but that remains efficient, easy to implement and easy to use.

The objective of this chapter is, therefore, to explain the use of simulation as a primary representation for commonsense knowledge. First, I advocate the use of simulation for commonsense reasoning and explore connections to the literature. I then formally define a Comirit Simulation and show how it may be used to represent everyday objects. Finally, I outline the implementation process and discuss both the benefits and limitations of the representation. In later chapters of this dissertation, I will explain how to compensate for the limitations by integrating simulation within a complete commonsense reasoning system that incorporates symbolic methods.

Note that, in early publications [85,84], I gave this iconic representation the name *Slick* and invented unique terminology to describe its parts. For clarity, I now use more standard mathematical terminology to describe the formal foundations and prefer to refer to the representation as a *Comirit Simulation*.

5.1. Why Simulation?

I argued in the literature survey of Chapter 2 (and specifically in Section 2.7) that existing techniques for commonsense reasoning trade power for cost-effectiveness (or *vice versa*). This leaves an opportunity for new systems that simultaneously achieve power and cost-effectiveness at the expense of other, less important factors. The rich detail and realism of modern computer games, animations and commercial simulators point towards simulation providing that opportunity.

Many modern computer games provide players with sophisticated open-ended environments that they can freely explore on their own terms. These games incorporate elaborate environments that, while set against a fictional back-story, realistically mimic much of the everyday environment of typical human experience. In-game worlds include everyday objects, buildings, natural environments and virtual human characters, all realistically modeled *in real-time*. Consider some of the examples below (in-game screen-shots appear in Figure 5.1):

- *Grand Theft Auto IV* allows the player to freely explore a realistic city-scape modeled upon New York City.
- In *Bully*, the player solves problems and challenges in a rich, realistic and explorable model of a schoolyard and nearby town.
- In *The Sims*, the player is free to create domestic environments for virtual characters and is responsible for their physical, emotional and financial wellbeing. The player can populate the environment with household objects that the virtual characters will use in their daily routines.

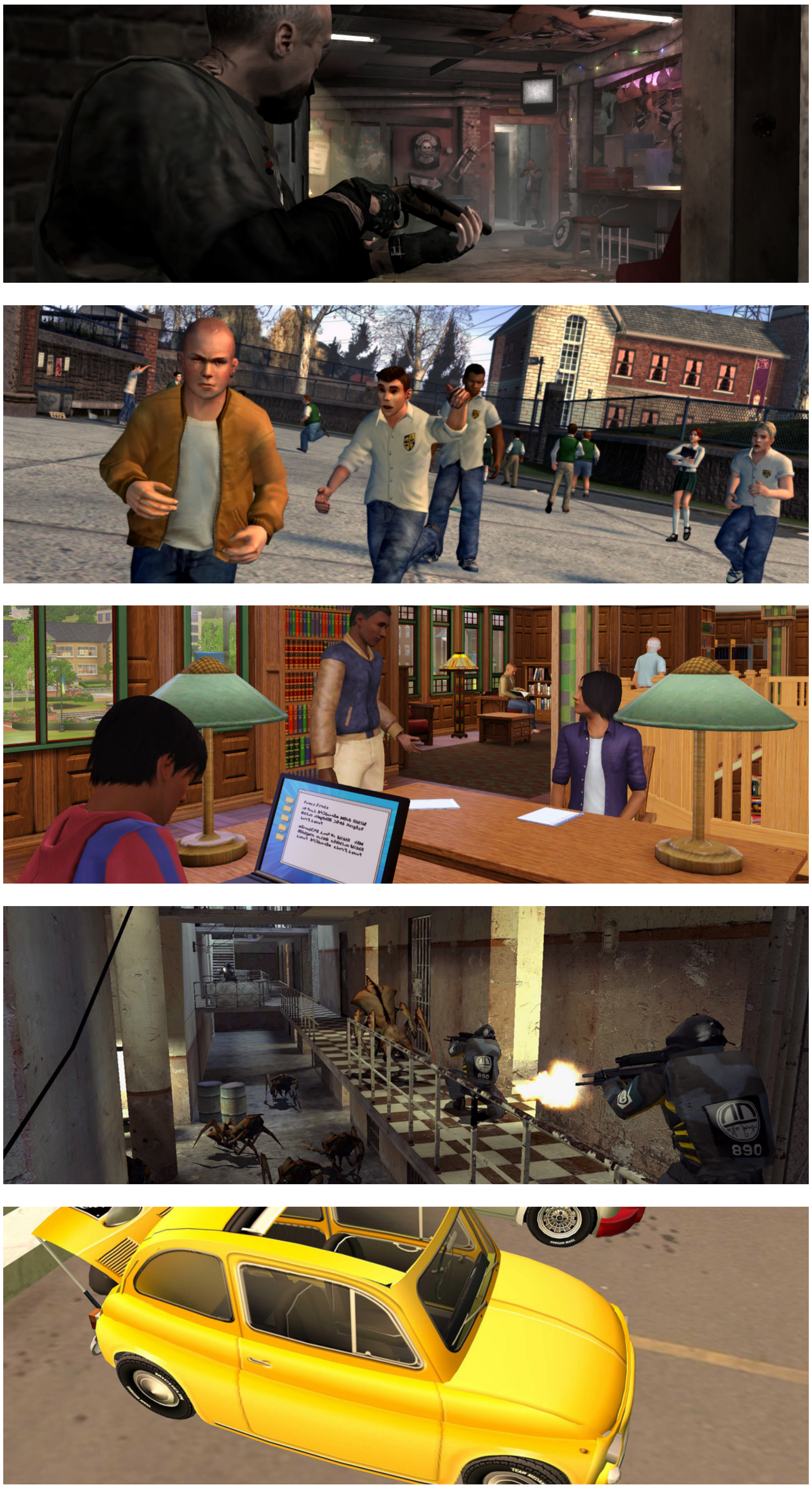


Figure 5.1: Screen-shots of (top-to-bottom) Grand Theft Auto IV, Bully, The Sims, Half-Life and Second Life (Source: publisher's web-sites; car by Flickr user 'Torley')

- While the *Half-Life* series of games has players engaged in combat with alien creatures, they are set in familiar environments such as office buildings, streets and public transport facilities. Most in-game objects can be manipulated and behave as one would expect in real-life.
- The *Second Life* online service allows players to buy land and construct any 3D objects that they can imagine. While some players have built fanciful structures that can only exist in a virtual world, others have built sophisticated models of actual objects and locations that resemble and behave like their real-world counterparts.

Beyond the realm of computer games, industrial simulations and computer animation offer similarly rich models. Examples abound from the pervasive use of simulation throughout industry:

- Commercial training simulators, such as those for flight, military and driver training
- Industrial simulations such as for crash testing, architectural walk-throughs and crowd behavior modeling
- Systems modeling of processes that are difficult or expensive to inspect, such as those for physics, astronomy, economics, hydrodynamics, medicine, chemistry and biology research
- Physical simulation, agent simulations and simulations of the characteristics of light, such as those used for the 3D animation and special effects in movies to create believable visuals

While games and simulations do not *explicitly* model commonsense knowledge using traditional representations of Artificial Intelligence, they *implicitly* represent a great deal of commonsense. For example, the explicit knowledge generated over the 20 years of the Cyc project appears to pale in comparison to the amount of implicit knowledge represented in a modern blockbuster computer game like *Half-Life*. If Cyc is queried about *doors*, it returns facts stating that a door is a kind of 'portal covering' and that a door and a doorway go together, in addition to abstract categorizations and information about their typical part-hood of buildings and vehicles. In contrast, a computer game like *Half-Life* could be said to implicitly represent not just this knowledge but also a door's particular responses to different actions, how they may block or partially obstruct a view according to how open the door is, how they limit physical access, how they are typically hinged to a wall and their typical dimensions in everyday environments. I am not aware of any computer game that explicitly incorporates challenges involving cracking an egg or figuring out whether Tweety can fly (to use the examples of Section 2.2) but it is not difficult to imagine a game that implicitly incorporates knowledge of all the popular challenge problems of commonsense reasoning researchers (ignoring, for a moment, the likely market failure of such a game).

The benefits of simulation provide a compelling argument for the direct application of computer games, industrial simulations and computer animations as a mechanism for commonsense reasoning. There is, however, the pragmatic difficulty of adapting these purpose-specific systems to commonsense reasoning. Existing simulation architectures make no concessions for extracting implicit knowledge and they are optimized for particular classes of problems in given domains. Simulation is a powerful method for reasoning about the everyday world but it must first be redesigned or 're-imagined' with commonsense reasoning in mind. In the remainder of this chapter, I show how this can be achieved.

5.1.1. Advantages and Compromises

Simulation is not a 'silver bullet'. While simulation offers clear benefits, these benefits must inevitably come at a cost. My contention is that the benefits of simulation outweigh the necessary compromises.

First, consider the advantages that simulations may offer for commonsense reasoning:

- **Expressiveness.** Virtually every situation or problem encountered in day-to-day life has already been richly encoded in some simulated environment. Simulations allow the world to be captured with incredible nuance and detail.
- **Cost effectiveness.** Sophisticated simulations of everyday environments are routinely built by the game industry in years rather than decades.
- **Computational predictability and efficiency.** The running times of simulations are highly predictable; they have fixed upper bounds that are a function of the input length and simulation duration and do not suffer from cascading effects of inconsistency. This is in contrast to methods, such as logical deduction, which may not be decidable even for simple formalizations.
- **Simplicity and clarity.** Simulations are routinely developed by professional developers. They do not require advanced degrees or in-depth training in logic or machine-learning techniques. For example, while Morgenstern [129] reflected on the difficulty of using logic to adequately describe the motion of a liquid as it flows from an egg to a bowl, under a simulation such effects can be accurately emulated by the Newtonian equations of motion that may be readily implemented with a high-school knowledge of physics.
- **Visualization and ease of debugging.** Simulations lend themselves to direct visualization of internal state, making it easier to understand, use and debug a simulation than an equivalent logical representation.

However, in choosing simulation, a trade-off is necessary:

- **Generality.** Not all problems are suited to simulation. Rote learning, recall of facts, symbolic mathematics and logical deduction are better suited to explicit representation in databases and theorem provers that efficiently and directly perform such reasoning, rather than the implicit representation of simulations.
- **Accuracy.** Simulations are numerical approximations and use numerical methods. The output of a simulation will therefore differ from a precise analytic solution (although this error should be small in routine cases that do not test the limits of a simulation).
- **Deductive power and flexibility.** Simulations have an 'arrow of time' and typically are not executable in reverse. Furthermore, simulations generally require full problem specifications and perform poorly with uncertain or vague models. This means that while simulations can efficiently predict likely consequences of a situation, they are not efficient at abducing the possible causes.
- **Introspectability.** Since simulations are typically codified in low-level algorithms, it is difficult for a system to introspect into its own simulation capabilities except by a black-box experimental analysis of its own output.

These trade-offs are not as significant as they may first appear, especially in light of the benefits. There is little doubt that simulation is poorly matched to abstract reasoning about logical problems, that it can only generate numerical approximations and that it is best suited to inferring consequences rather than causes. However, it will become clear during evaluation in Chapter 8, that the *typical situations* encountered by an agent performing *real work* in a *realistic environment* tend to involve the kinds of approximate, concrete reasoning that are *well suited to simulation*. Furthermore, by encapsulating simulation in a hybrid framework, as I will demonstrate in the next chapter, it is possible to combine simulation with other techniques so as to compensate for many of the weaknesses of simulation.

5.2. Connections to the Literature

Simulation has a long history of application in engineering and physical sciences but established work in commonsense reasoning finds very little use of simulation. The notable exceptions are the simplistic grid-based reasoning used in ThoughtTreasure (see Section 2.8) and the preliminary work by Gardin and Meltzer [52] and Decuyper, Keymeulen and Steels [35] to which my research is closely related. In fact, the limited applications of simulation to commonsense reasoning is very surprising in light of the close correspondences and analogies between simulation and existing work in commonsense reasoning. In the following sections, I consider first the early work that this research project develops upon and then some of the more distant connections of simulation to the existing literature.

5.2.1. Early Applications of Simulation

The earliest proposals for simulation as an analogical representation for commonsense reasoning are the preliminary works by Gardin and Meltzer [52] and Decuyper, Keymeulen and Steels [35].

Gardin and Meltzer [52] constructed simulations of ropes and liquids in an attempt to adequately model explanations of commonsense intuitions, such as the use of a rope to pull but not to push an object. Their simulations are assembled from round beads or 'particles' that are connected together by local interaction rules. All computation is performed in a 2-dimensional model and update rules appear to be inspired by desired qualitative behavior, rather than any specific appeal to the laws of physics. Figure 5.2 is a facsimile of their original results [52], illustrating how strings under gravity, held at single points, drape over solid objects in the 2D world.

In what appears to be a separate system built upon similar principles, Gardin and Meltzer [52] subsequently attempted to use local update laws to model the behavior of liquids. Figure 5.3 is the output of their system for the effects of a liquid pouring into a glass. The output provides a compelling 'first-order' approximation of liquid motion but lacks detail: the behavior is based on a simple qualitative model of the laws of motion, resulting in liquids that pour in a solid, straight line of one molecule thickness. Indeed, the underlying rules of Gardin and Meltzer's simulations do not appear to allow

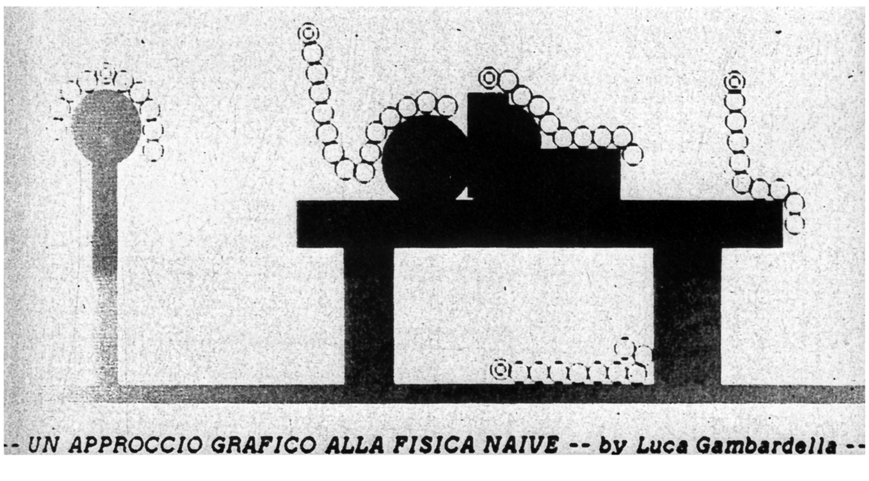

-- UN APPROCCIO GRAFICO ALLA FISICA NAIVE -- by Luca Gambardella --

Figure 5.2: Anchored strings draped over shapes (inverted-color facsimile of Gardin and Meltzer [52])

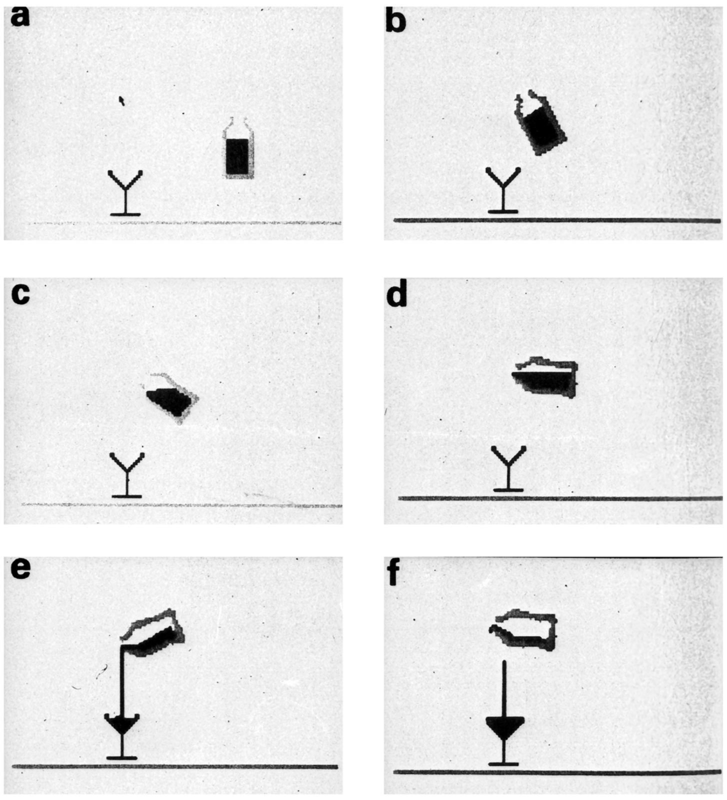


Figure 5.3: A liquid pouring into a wine-glass (inverted-color facsimile of Gardin and Meltzer [52])

for liquids to have momentum, meaning that their simulations would not be able to predict that a full glass will spill if moved too fast or that a fountain can be formed from a projected stream of liquid.

Decuyper, Keymeulen and Steels [35] also explored simulation but, instead, emphasized aspects of the software architecture. Their intention was to create a three-fold combination of particle-based simulation (as with Gardin and Meltzer [52]), space-based models of flow and qualitative formalisms. They outlined an abstract architecture in which a qualitative reasoning process instantiates and analyzes simulations. However, their work appears to be speculative since they do not appear to have subsequently published any form of detailed design or evaluation.

Indeed, as Ernest Davis also notes [32], neither the approaches of Gardin and Meltzer nor Decuyper, Keymeulen and Steels appear to have been developed from their preliminary publications and there is a scarcity of other work that has developed from these core ideas.

Nevertheless, I view such initial work as ideas very much in the motivating spirit of what I am setting out to complete in this dissertation; I am moving beyond speculation to design concrete systems that use simulation to perform practical commonsense reasoning.

5.2.2. Analogies between Simulation and Established Practice

Despite the distinct differences between the underlying algorithms of simulation and automated logical deduction, it is possible to draw compelling analogies between established practice in commonsense reasoning and simulation:

- The use of formal logic in working theorem provers can be conceived as a special form of simulation. Like simulation, logic can be used to anticipate the consequences of a given situation (with assertions or axioms corresponding to the known laws of the environment). A logical system may compute with qualitative models but the underlying process of rule application in deduction is related to the computation that occurs in simulations, such as computer games and animations.
- Conversely, the state of a simulation and the laws of the environment can be considered a formal theory of behavior. The source code of the simulation engine is analogous to a theory and its execution has parallels to an entailment operator for reasoning about the likely consequences of an action.

While I believe that the application of simulation to commonsense reasoning represents a new paradigm for commonsense reasoning, one could take a pessimistic stance and regard simulation as simply a heuristic for numerical reasoning about probable future states. Nevertheless, the benefits of simulation are so significant that whether or not it is 'simply' an heuristic, the advantages are so unmistakable that it deserves careful attention and a central role in the design of a commonsense reasoning system.

5.2.3. Qualitative Simulation

Qualitative simulation [94, 43] may be seen as a concrete development of the analogy between simulation and the established practice of logic-based knowledge representation.

Qualitative simulation is a specialization of qualitative reasoning (see Section 2.6.7 for a critique of qualitative reasoning). A qualitative simulation consists of a qualitative description of the dynamics of a system. Rather than expressing a system in quantitative terms (*e.g.*, the water temperature is 300 Kelvin) it is described in qualitative terms (*e.g.*, the water is 'warm'). Logical inference rules are used to describe the possible behaviors and transitions over these qualitative terms. For example, a model of water may incorporate landmarks 'freezing point' and 'boiling point'. These are related to the state transitions for 'ice state', 'water state' and 'vapor state' in connection with the input or removal of heat into or from the system. Thus, while a qualitative model may not use precise temperatures, it is still possible to conclude that after heating liquid water, it will be in either a liquid or boiling state. Even if a system's precise dynamics are unknown, one can deduce which transitions may occur.

Qualitative simulations may be viewed as an approach to building models for qualitative reasoning that are inspired by simulations. However, while qualitative simulation shares the name 'simulation' and their mechanics are clearly analogous to simulations, I do not regard them as true simulations in the sense that I am proposing in this dissertation. They are not like computer games or industrial simulations, they do not have the same efficiency, simplicity and richness. They do not lend themselves to visualization, visual development and visual debugging in the same way as quantitative simulations. They do not allow complex and intricate systems to be described by a small set of physical laws. That is, they are merely an effective methodology for qualitative reasoning.

That is not to say that qualitative reasoning is irrelevant to the development of Comirit. The smaller state-spaces of qualitative simulations make them preferable for problems involving deep search. I shall explain in Chapter 6 how simulation is combined with logical techniques. It would be entirely appropriate and beneficial for qualitative simulation, implemented as a logical symbolic system per Chapter 6, to serve as a heuristic for quickly analyzing a search space and for postulating previous states (against the 'arrow of time').

5.2.4. Other Applications of Simulation

Even though simulation has found limited application as an internal mechanism for commonsense reasoning, there is growing interest in the use of simulation as a training environment—an external world—for intelligent systems. Simulations offer a safe environment for virtual robots to explore the 'world' free from mechanical wear, short battery lives, hardware costs, sensor limitations and safety concerns. Simulations are finding application in 'shallow' learning of stereotypical activities by identifying events that occur in computer games [109] and for deeper learning in a developmental robotics context, allowing a virtual agent to learn the fundamental properties of the world by experimenting in virtual 'sandboxes' [56,55].

5.2.5. On the Limited Application to Commonsense

In light of these connections and analogies, there remains an obvious question: why are there so few direct applications of simulation to commonsense reasoning?

I believe that its absence is primarily a matter of computational power. The use of simulation necessarily involves compromises (as in Section 5.1.1). As such, simulation is unlikely to be developed as a method of commonsense reasoning, except as part of a pragmatic research program (such as my own), in an era in which simulation is a practical solution.

The modern desktop computer, with its ability to perform detailed computation of large-scale problems, coupled with high-quality accelerated graphics displays, stands in stark contrast to the desktop machines available to earlier researchers, such as Gardin and Meltzer [52] in 1989. Until now, it has been inconceivable to seriously consider use large-scale simulations with millions of components as a viable method of commonsense reasoning.

Given this explanation, it would be reasonable to expect the simultaneous emergence of other research exploring applications of simulation to commonsense reasoning. Indeed, as this dissertation comes to publication, a number of related but independent publications have come from the Soar group, as I discuss in the following sub-section.

5.2.6. Recent Developments

As this writing of this dissertation has neared its conclusion, the University of Michigan's Soar group has been independently exploring the application of simulation and visual reasoning to intelligent reasoning.

In particular, Samuel Wintermute and Scott Lathrop developed SRS (Spatial Reasoning for Soar) [177] and SVI (Soar Visual Imagery) [98] respectively. These efforts were subsequently combined to create SVS (Spatial and Visual System) [176]. SVS is a multi-modal reasoning extension to Soar that combines visual representations and spatial scene simulations with Soar's existing production-rule decision system and memory processes.

Simulations in Soar consist of patterns of motion applied to spatial scenes. A spatial scene is a set of polygons that resembles real-world objects, stored in a tree structure (*i.e.*, a scene graph used in computer graphics). Patterns of motion are implemented as continuous 'motion models' in the spatial level of the system. The decision system can invoke the simulation in a stepwise manner to determine the effects of action selection or to form complex plans. For example, motion modeling would allow Soar to determine the physical stability of balanced objects, collision-free paths for a vehicle or even to perform constraint satisfaction in spatial domains [176].

These new developments in Soar are highly complementary to this research project. However, while our objectives and motivations are closely related, our *contributions* are significantly divergent.

I argued in Section 5.1 that simulation is a powerful resource of implicit commonsense knowledge about the world. My objective is therefore to accurately capture the implicit knowledge of the world for extraction and learning. In contrast, the Soar SRS is designed as an abstract mechanism that uses spatial representations primarily to enhance spatial reasoning. The difference lies in the rationale for constructing simulations and the role in which they are used (not to mention the different methods and algorithms that arise from the divergent rationale).

While I have a pragmatic objective of cost-effectively creating systems with broad understanding of commonsense, I do not deny the emphasis Soar places on the importance of simulation as a tool in high-level and abstract reasoning processes of generally intelligent systems. Though I am pragmatically focussed on creating broad commonsense reasoning systems in the short term, rather than longer-term general intelligence capabilities, I would expect that merging our respective research projects would be highly productive future work. I discuss this further in Section 9.2.3.

5.2.7. A Note of Caution Regarding Mental Imagery

There is a temptation to invoke the apparent simulation-like nature of mental imagery as further argument for simulation in artificial commonsense reasoning. Unfortunately, this argument is dangerous in light of debate in the cognitive sciences about the nature and existence of mental imagery. There is no clear consensus on the nature of mental imagery [165] and there is good reason to believe that it may even be an 'illusion' (for lack of a better word). Pylyshyn [143] provides a comprehensive survey of empirical results to suggest that the intuitive conception of mental imagery is akin to the fallacious 'homunculus theory of mind'.

Indeed, such debate and controversy is, in part, why I prefer to take a practical stance towards artificial intelligence. My objective is not to model the human mind, not to copy its mechanisms and not to create a human common sense. My argument for simulation is based on the computational and practical benefits of simulation, rather than being motivated by biological inspiration. Even if it transpires that human thought is a symbolic and entirely non-visual process, it is still likely that exploiting the rich complexity of simulation will remain the most effective short-term means towards creating artificial commonsense.

5.3. Concept

From an *algorithmic* perspective, it will become clear that a Comirit Simulation is equivalent to the numerical integration of a set of differential equations. The simulation algorithm iteratively advances a graphical model by small time increments.

However, Comirit Simulations are *not* intended as an algorithmic contribution. The novel value of Comirit lies in the *representations*, *framework* and *software architecture* that it offers for managing and exploiting simulations. Comirit provides a general purpose framework for cross-domain reasoning with simulation, including capabilities for integration into a hybrid architecture and for autonomous learning. Comirit provides a framework through which simulations can be *applied* to solve meaningful and useful commonsense reasoning problems.

Note also that, just as theorem provers and action languages alone are not panaceas to the challenges of commonsense reasoning, simulation is not the answer to *every* problem faced by an intelligent system. Simulations implicitly encode real-world knowledge and serve a crucial role in efficient reasoning but they are best used in concert with other mechanisms. In Chapter 6, I will explain how simulation is combined with other methods, resulting in a hybrid architecture where the responsibility for planning, symbolic reasoning and the resolution of high level goals and contextual information are shared among many components.

5.3.1. Parts of a Simulation

A Comirit Simulation consists of the following parts:

1. A *system clock* that increases with each iteration of the simulation
2. A (relatively) static *graph representation* that models the structure of a problem domain
3. A set of highly dynamic *annotations* that record the state of a simulation
4. A static set of computable functions or *constraints* that update the annotations and thereby drive the computation of the simulation

These parts are defined in the following sub-sections.

5.3.2. System Clock

A Comirit Simulation is computed iteratively and each iteration is labeled with a timestamp. The timestamp may be a true measure of the simulated passage of time, it may be a simple counter or it may represent some other abstract quantity.

The system clock is therefore a totally ordered set $T = \{t_0, t_1, t_2, ...\}$ of timestamps.

The initial state of a simulation is assigned the smallest timestamp value, denoted t_0. Subsequent iterations are labeled with the smallest unassigned timestamp (*i.e.*, the successor in the totally ordered set T).

If the system clock, T, is finite, the simulation will halt after $\#T$ iterations.

Given a timestamp, t, the next and previous timestamps in the total ordering of timestamps are denoted by the functions $next(t)$ and $prev(t)$ respectively.

$[T]$

$$\begin{array}{|l} next : T \rightarrowtail T \\ prev : T \rightarrowtail T \end{array}$$

5.3.3. Graph Representation

In a Comirit Simulation, the essential structure of a problem domain is modeled by an edge-labeled directed multigraph. These multigraphs capture those structural aspects of a situation that rarely change during a simulation or, ideally, are completely static.

Informal Definition

The underlying representation of a Comirit Simulation is an *edge-labeled directed multigraph*.

An *edge-labeled directed multigraph* is a generalization of a directed graph consisting of:

1. a set of objects, called vertices, and
2. a set of triples (consisting of a start vertex, an end vertex and a label), called labeled edges.

Note that this definition allows for multiple directed edges between any two given vertices, provided that each edge has a distinct label.

An example of an edge-labeled directed multigraph appears in Figure 5.4.

Formal Definition

Formally, an *edge-labeled directed multigraph,* drawn from vertices V and labels L, is an element of $CGRAPH$ (Comirit graph):

$[V, L]$

$EDGE == V \times V \times L$

$CGRAPH == \{G : \mathbb{P}\ V \times \mathbb{P}\ EDGE \mid \forall\ e : G.2 \bullet e.1 \in G.1 \land e.2 \in G.1\}$

◊ "An edge (of type EDGE) is a triple comprising a start vertex, an end vertex and a label"

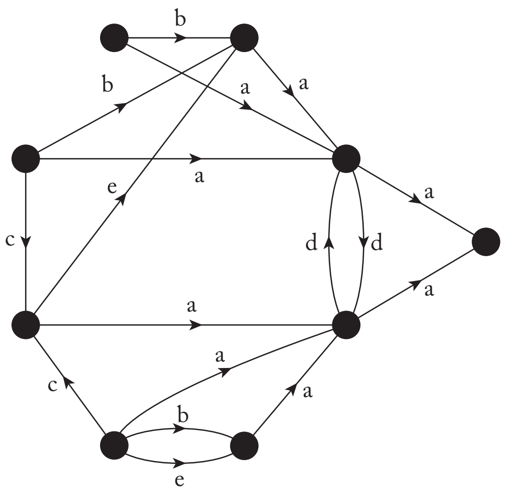


Figure 5.4: An example of an edge-labeled directed multi-graph (labels are a, b, c, d and e)

◊ "A *CGRAPH* is defined as a set of vertices paired with a set of edges, such that each edge starts and finishes at an element of the set of vertices"

Usage

In practice, the graph structure is used to represent the underlying and slowly changing relationships in a problem domain. In contrast, dynamic state and factual knowledge is represented by rapidly changing annotations that are attached to vertices and layered above the graph structure (annotations are defined in the following section, Section 5.3.4).

The use of an underlying graph structure without any fixed semantics is intentional—it permits simulations to be adapted to different domains. Vertices are used to denote the principal entities, relationships or structuring elements in a problem domain. Edges are used to represent the connections between vertices, with labels indicating their purpose or role. For example, consider the following examples of static or slowly changing structures across a variety of problem domains:

- In an office simulation, vertices might represent employees, offices and departments, while edges are used to associate employees with each other and to departments or offices.
- In a physics simulation, vertices might represent objects, agents and forces, while edges are used to represent the target and source of the forces.
- In a social context, vertices might denote people and their relationships to each other (rather than their current mood and attitudes), while edges indicate the role of people in the relationship (*e.g.*, wife of marriage, husband of marriage).

- In an indoor navigation problem, vertices might denote rooms, doors, actors and objects that inhabit the world (rather than the current closed/open status, position or appearance).

By way of example, Figure 5.5 depicts a simple graph that may be used as an underlying basis for simulating an office environment. Note that while I have explicitly represented both directions of the symmetric relationship 'next to' in the diagram, it would be possible to have symmetric, transitive and reflexive relationships automatically expanded by the system.

The graph structure is unchanged throughout the computation of a simulation, so selection of a stable underlying structure is important. In domains where changes to the underlying graph structure are necessary, the changes are made through processes that are external to the simulation: manual edits by an engineer or, automatically, through an agent's observation of the changing environment (a new object in the room, changes in staffing or the discovery of entirely new locations).

While *static* graph structures restrict the potential scope and flexibility of simulations, preliminary experience suggests that this limitation is minor. There is usually some stable structure that applies in any given problem, for example the slowly changing hierarchies or groups in an organization or the conservation of mass in a physical system. This slight restriction comes with benefits for model simplification that provide fixed anchors for connection with symbolic layers and opportunities for optimization.

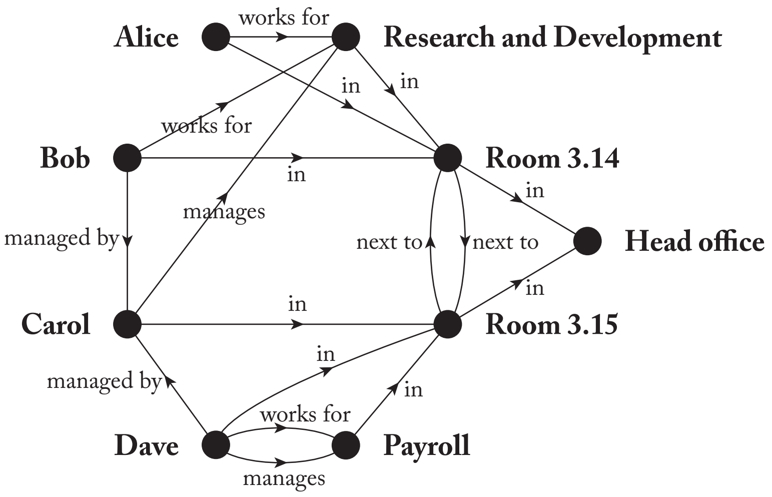


Figure 5.5: A simple graph-based representation of part of a business structure

5.3.4. Annotations

In a Comirit Simulation, the rapidly changing dynamic state of a simulation is recorded against the vertices of the underlying graph. This state is stored in frame-like structures with values that are updated in each iteration of a simulation. While annotations store the dynamic state of a simulation, they may also store static data that cannot be readily represented in graph-based forms.

Informal Definition

The underlying graph is associated with a time varying set of annotations. For each instant in time, and for each vertex, there is a mapping from annotation names to the value of the annotation. With each iteration of the simulation, the values of the annotations may change.

Values may be numeric or non-numeric. The value of an attribute for an uncomputed future iteration (*i.e.*, for a future time in the simulation clock) is undefined.

By convention, some attributes may be used as though read-only. That is, their value is set during initialization, is always constant and is never updated. These do not need to be explicitly declared as such, nor do they require explicit copying of their value, since an automatic copy occurs with each advance of the simulation (see later in Section 5.3.5).

Initial values for attributes are configured during the instantiation of a simulation. Ideally, default values should be stored in a database or catalog of defaults and adapted to a given situation based on perceptions and knowledge of the context.

Formal Definition

Given a set of 'annotation names' of type A, and annotation values of type D, an *annotation* of a graph $G : CGRAPH$ at a given instant in time is an element of the type *ANNOT*. The time varying trace of annotations is correspondingly an element of the type *TANNOT*.

$[A, D]$

$ANNOT == V \nrightarrow (A \nrightarrow D)$

$TANNOT == T \nrightarrow ANNOT$

◊ "An annotation (of type *ANNOT*) is a function that maps from vertices to a function mapping attribute names to values"

◊ "A time varying annotation (of type *TANNOT*) is a function from timestamps to annotations"

If an annotation $AN : ANNOT$ is associated with a given graph $G : CGRAPH$, the vertices of the annotation should be a subset of those of the graph. *i.e.*, dom $AN \subseteq G.1$

Usage

In practice, annotations are used to store both read-only factual knowledge and the rapidly changing and dynamic state of a simulation.

Consider the following examples:

- In an office simulation, annotations on an employee vertex might have attributes for whether they are at their desk and what they are currently working on. The simulation might have read-only attributes for their name, employee identifier and to indicate that the vertex denotes an employee.
- In a simple physics simulation, annotations on an object vertex might have mutable attributes for current coordinates, orientation, velocity and temperature, in addition to read-only attributes for the type of matter, its state transition temperatures and its color.
- In a social context, annotations on a relationship vertex might have attributes corresponding to current relationship strength and nature (romance, friendship, professional) in addition to a read-only attribute indicating that the vertex denotes a reified relationship (rather than a person).
- In an indoor navigation problem, annotations on a door vertex might include its current open/closed status, type of handle, method of opening, distinguishing features and time it was last used.

By way of example, Figure 5.6 illustrates annotations that might be used on vertices in an office simulation.

5.3.5. Constraints

Constraints are the fundamental computational operators of a Comirit Simulation. Constraints are the functions that are responsible for updating the values of annotations. They are invoked for each iteration (or clock tick) of the simulation and thereby iteratively extrapolate from known present values to unknown future values by a process of simulation.

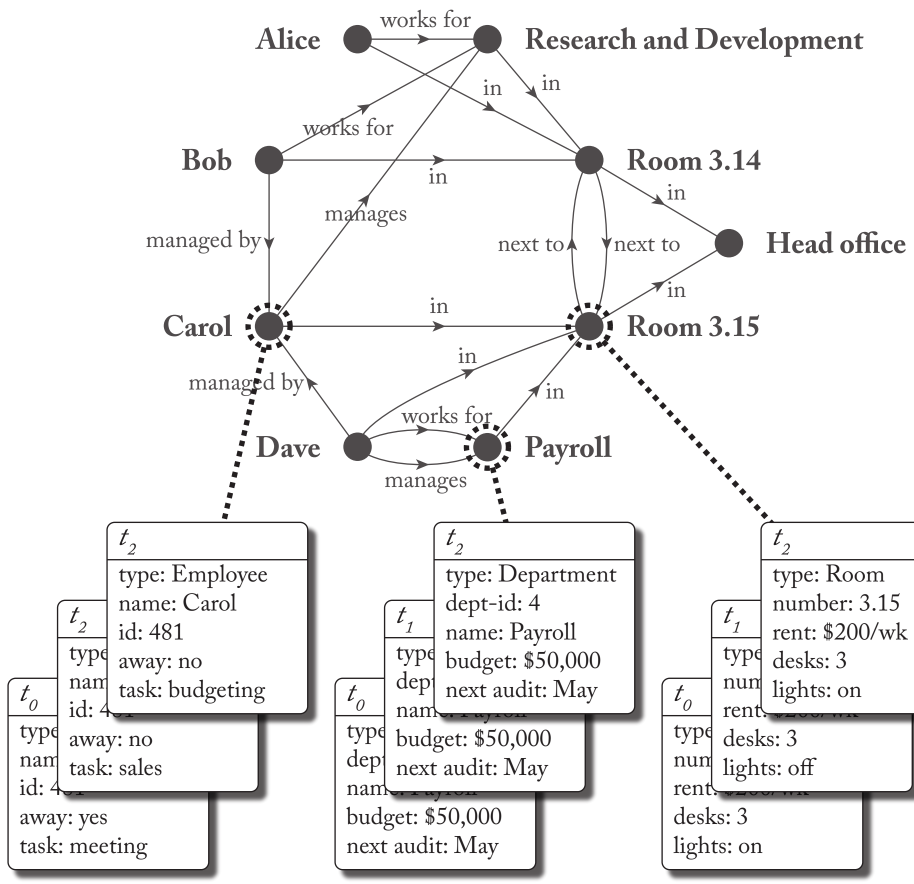


Figure 5.6: Example annotations that may be found on a simulation of a business

Informal Definition

A *constraint* is a function used to update the values of an annotation. Comirit Simulations are advanced by the application of a set of constraints to the current values of annotations.

Constraints are associated with a ranking that determines the sequence in which the constraint is applied. This makes it possible for inviolable (high-priority) constraints to override the results of 'softer' (low-priority) constraints.

Given a trace of annotations, new annotations are computed as follows:

1. The annotations for the most recent time instant are copied into a 'working memory'.
2. In order from lowest priority to highest priority, constraints are applied serially to the working memory to compute new values of annotations.

3. The final result of the working memory is then taken as the final value of annotations for the current time instant.

A Comirit Simulation is externally initialized (at time t_0) with a graph structure, a set of constraints and initial values of annotations. The simulation thus advances in small steps, with each iteration copying to working memory, applying constraints and merging the results back into the trace.

Formal Definition

A *constraint* is an element of the type CONSTRAINT that pairs an update function of type CFUNC with a priority.

$CFUNC == (ANNOT \times TANNOT \times CGRAPH) \rightarrow\!\!\!\!\!\!+ \; ANNOT$

$CONSTRAINT == \mathbb{R} \times CFUNC$

◊ "A function of type *CFUNC* takes a working map of annotations (first parameter), an historical trace of prior annotations (second parameter) and the simulation graph (third parameter) to return an updated (partial) set of annotations"

◊ "A constraint (*CONSTRAINT*) is a function of type *CFUNC* paired with a priority value."

The new annotations generated by a constraint are merged with the working memory by the *override* function:

$$\_\ override\ \_ : ANNOT \times ANNOT \rightarrow\!\!\!\!\!\!+ \; ANNOT$$

$$\forall X, Y : ANNOT \bullet$$

$$X\ override\ Y = \{V : \mathrm{dom}\ X \cap \mathrm{dom}\ Y \bullet V \mapsto X(V) \oplus Y(V)\}$$
$$\cup\ \mathrm{dom}\ Y \ntriangleleft X$$
$$\cup\ \mathrm{dom}\ X \ntriangleleft Y$$

◊ "The result of *X override Y* is given by the 'function override' of annotations on vertices in both *X* and *Y*, combined with the annotations for vertices found in only in *X* or *Y*"

The result of applying a set of constraints to a working map of annotations is given by the following recursive function definition for *apply* (this definition makes use of an auxiliary function, *lowest*):

$lowest\ \_ : \mathbb{P}\ CONSTRAINT \rightarrowtail\ CONSTRAINT$

$apply : \mathbb{P}\ CONSTRAINT \times ANNOT \times TANNOT \times GRAPH \rightarrowtail ANNOT$

$\forall\ C : \mathbf{P}\ CONSTRAINT \bullet$

$\quad \mathbf{let}\ \mathrm{M} == min\ \mathrm{dom}\ C \bullet lowest\ C = M \mapsto C(M)$

$\forall\ A : ANNOT;\ T : TANNOT;\ G : CGRAPH \bullet$

$\quad apply(\varnothing, A, T, G) = A$

$\quad \forall\ C : \mathbb{P}\ CONSTRAINT \bullet$

$\qquad \mathbf{let}\ L == lowest\ C;\ F == \mathrm{L.2};\ C_2 == C \setminus L \bullet$

$\qquad\quad apply(C, A, T, G) = apply(C_2, (F(A, T, G)\ override\ A), T, G)$

◊ "The function *lowest* returns the constraint with priority equal to the smallest priority: it returns the lowest priority constraint"

◊ "The function *apply* takes as input a set of constraints, a working memory, a trace of annotations and a graph to return a new set of annotations for the next iteration. If there are no constraints, *apply* returns a copy of the working memory. If there are constraints, *apply* merges the working memory with the output from the lowest priority constraint and then recursively calls apply with the remainder of the constraints and the new working memory."

Now, finally, given the state of a simulation, the iteration function may be defined by the *Simulation* state and the *AdvanceSimulation* operation:

Simulation

$graph : CGRAPH$

$constraints : \mathbb{P}\ CONSTRAINT$

$time : T$

$trace : TANNOT$

AdvanceSimulation

$\Delta\ Simulation$

$graph' = graph$

$constraints' = constraints$

$time' = next(time)$

let $working == trace(time) \bullet$

$\quad trace' = trace \cup \{time' \mapsto apply(constraints, working, trace, graph)\}$

Usage

In practice, constraints implement simple and local laws of a simulation. While a constraint may be implemented to modify attributes of every vertex in a simulation, the combined effect should have a single purpose and any given attribute change should only depend on the values of annotations on that vertex and its nearest neighbors. That is, complex macroscopic behaviors should not be explicitly programmed in a constraint but should emerge from repeated microscopic and local interactions.

The determination of a vertex's nearest neighbors should also be a simple process. Ideally, it is the vertices that are immediately or closely connected by edges in the graph. In some circumstances, it may be necessary to also consider a neighborhood of *similar attribute values*. For example, two objects may interact due to their physical proximity, even though there is no prior relationship in the underlying structural representation. In this case, they are neighbors, not because of the graph representation but because the distance between their position vectors is below some threshold.

Examples of constraints that might be defined for practical simulations include the following:

- An office simulation might have constraints corresponding to simulated activities, such as employees performing current tasks on a to-do list, delegating tasks and receiving tasks. These constraints would update attributes, such as the employee's current work, their location in the office, fatigue and interest.
- In a physics simulation, constraints would implement physical laws and effects due to gravity, friction, momentum, acceleration, fluid dynamics and thermodynamics.
- In a social context, one constraint might model the slow decay of friendship over times of absence, while another might boost the strength of a friendship during social interactions. While such constraints are naïve simplifications of human relationships, it may be sufficient for useful commonsense reasoning.

- In an indoor navigation problem, one constraint may be implemented as a rule that swaps the open/closed state on a door when a nearby agent applies a force on the door.

5.4. Example

Physical simulation is one of the most compelling demonstrations of simulation and also the original inspiration for the development of the Comirit Framework. The simulation of macroscopic *physical* systems by iterative simulation of low-level microscopic laws is an effective way of providing general purpose, naïve physical reasoning capabilities to a commonsense reasoning system. In this section, I demonstrate how a Comirit Simulation may be used to simulate an arbitrary physical system. For illustrative purposes, I will build a simulation of a mug of coffee. While simple, this example is interesting because it not only illustrates solids, liquids, liquid flow, gravity and a principle of containment but also does so with precision and detail beyond existing frameworks.

5.4.1. Modeling Principle

An analytical attempt to express the macroscopic interaction of many laws of physics over interacting complex objects will result in complex systems of equations for which it is extremely difficult to find closed-form solutions. In these circumstances, it is preferable to use numerical methods. One method of numerical analysis involves the discretization of time into short intervals and the discretization of objects into small 'packets' of mass. The system may then analyze the problem by iteratively integrating the laws of physics over these discrete microscopic models[*].

In fact, this approach is the standard mass-spring model used in many games, fabric simulations and 3D animations [124]. In a mass-spring model, complex physical structures are modeled as ideal point masses connected by ideal springs. A rigid object is modeled as a lattice of masses connected by springs with a high spring constant, whereas an elastic cloth may be modeled by a thin surface of masses connected by freely flexing springs.

[*]If the microscopic effects of a physical law over microscopic time intervals is considered to be an approximation of a differential equation, then iterative computation of a simulation is equivalent to the Euler method [162] of numerically solving ODEs.

A microscopic discretization into masses and springs has additional advantages beyond its efficiency as a method of numerical analysis:

- The same computational mechanism can simulate a wide variety of objects.
- Mass-spring models can be built quickly and easily in 3D modeling tools.
- Different materials can be modeled simply by setting the simulation constants rather than as fundamental types (*e.g.*, rigid materials are given a high spring constant, soft materials have a small spring constant).

5.4.2. Graph Representation and Annotations

Figure 5.7 illustrates how a cup of coffee is represented in a discrete mass-spring model. The masses (balls) and springs (bars) correspond to vertices in the underlying graph structure and are connected to each other by edges (*i.e.*, a vertex representing a spring is connected to vertices representing mass-es).

In this example simulation, the simulation graph is an edge-labeled directed multigraph, where the set of vertices, $V = M \cup S \cup C \cup H$, have been sub-typed in to the following five categories:

- M is the set of point masses that simulate the mass of the ceramic mug and coffee liquid.
- S is the set of 'springs' that connect, by edges, exactly two point masses in M.
- C is a set of corners that connect, by edges, two adjacent springs, to ensure they remain at a fixed, rigid angle in the solid.
- H is a set of convex hulls that form impermeable surfaces around three point masses (*i.e.*, triangles) in M..

The annotations of each vertex is listed in Table 5.1.

5.4.3. Constraints

Given a graph-based model, the laws of physics are implemented as operators over the annotations. The following constraints are sufficient for a plausible simulation of the mug of coffee. They are listed in order, according to the priority of the constraint application from lowest to highest (*i.e.*, the 'Impermeability of Hulls' constraint is the highest so is inviolable).

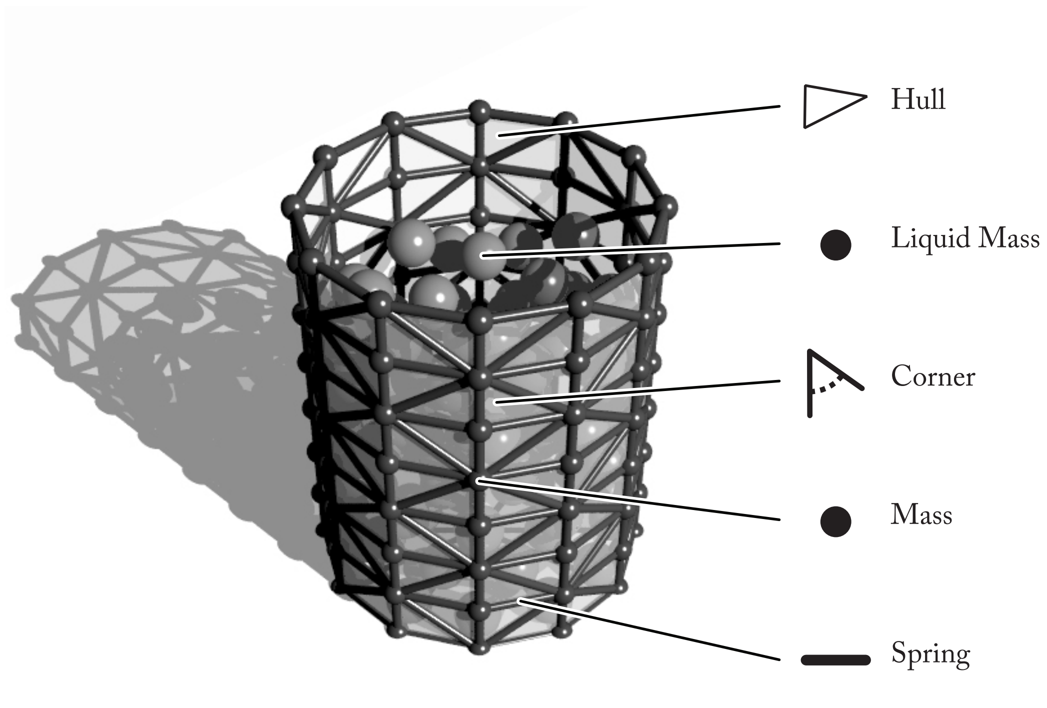


Figure 5.7: Parts of a mass-spring simulation of a mug of coffee

Vertex Type	Annotation Attributes
Masses (M) – Solid	Id, Type, Position (*x*, *y*, *z*), Mass
Masses (M) – Liquid	Id, Type, Position (*x*, *y*, *z*), Mass, Compressibility, Surface Tension
Springs (S)	Id, Spring Constant, Target-length, Snap-length, Broken (a boolean attribute)
Corners (C)	Id, Angle, Spring Constant, Broken (a boolean attribute)
Hulls (H)	Collision Elasticity, Broken (a boolean attribute)

Table 5.1: Annotations used by a mass-spring simulation

Momentum and Universal Damping

Momentum is a constraint that has a multiplying effect on other constraints: small forces in each iteration are accumulated by momentum in an accelerating process. The specification of this function is therefore crucial in order to avoid the spurious amplification of high-frequency oscillations caused by numerical imprecision. It turns out that momentum is best calculated by extrapolating from actual effects in previous moments, rather than by the possibly more intuitive approach of storing a hidden velocity parameter.

That is, momentum is given by:

$x_{\text{working}}' = x_{\text{working}} + (x_{\text{t}} - x_{\text{t-1}})$

$y_{\text{working}}' = y_{\text{working}} + (y_{\text{t}} - y_{\text{t-1}})$

$z_{\text{working}}' = z_{\text{working}} + (y_{\text{t}} - y_{\text{t-1}})$

Incorporating a small universal damping constant also helps to further eliminate spurious oscillations and numerical instabilities, without noticeably affecting the quality of the simulation.

Thus momentum, combined with universal damping, is given by:

$x_{\text{working}}' = x_{\text{working}} + (x_{\text{t}} - x_{\text{t-1}}) \times (1 - \mathit{damping})$

$y_{\text{working}}' = y_{\text{working}} + (y_{\text{t}} - y_{\text{t-1}}) \times (1 - \mathit{damping})$

$z_{\text{working}}' = z_{\text{working}} + (y_{\text{t}} - y_{\text{t-1}}) \times (1 - \mathit{damping})$

Gravity

The effect of gravity is computed as the displacement due to gravitational forces:

$z_{\text{working}}' = z_{\text{working}} - 2 \times (\tfrac{1}{2}\, g\, \Delta t^2)$

The true displacement due to gravity in time Δt is doubled for minor technical reasons. The accumulating effects of momentum are computed from old values and so momentum is consistently underestimated from accelerating forces. Doubling such accelerating forces results in a net effect over time that is precisely the analytic value of the integration.

Solid Bonds

The effect due to solid bonds is computed over the 'springs' in the simulation. The force of the spring may be computed by Hooke's law ($F = -k\, x$) and this force is applied to compute the displacement of each mass at the ends of the bond.

If the bond becomes over-extended by a predefined length, then the bond is regarded as broken. This has the effect of reducing the spring constant, *k*, to zero ($k = 0$).

Rigid Shapes

The rigid structure of an object is maintained by assuming the existence of a torsion spring in the corners, C. Each corner is taken as a torsion spring holding adjacent linear springs at a constant angle. The force applied to each mass in the system is given by applying the angular form of Hooke's law: $\tau = -k\,\theta$. If either of the linear springs in a corner are 'broken', then the corner is also taken to be 'broken'.

Liquid Incompressibility

The incompressibility of liquids is handled by assuming implicit springs between nearby liquid masses.

The distance between any two masses in a liquid is compared to a constant 'radius of effect'. If the masses are further apart than the radius of effect, they are left unchanged. If the masses are closer than the radius of effect, Hooke's law is applied as though there is an implicit repelling spring between the two masses. This has the effect of preventing the particles of a liquid from being compressed, while still allowing them to flow freely.

Liquid Surface Tension

Surface tension is an attractive force within liquids that causes liquids to bead up into droplets and to form menisci when placed in containers. Surface tension is handled analogously to the incompressibility of liquids. When nearby masses of liquid are outside the radius of effect for liquid incompressibility, but within a secondary radius, a small attractive force holds the masses to each other. In prototype simulations, I use Hooke's law again to model this force: the distance from the radius of effect (at which the force becomes zero) is used to compute the attractive force due to surface tension.

Impermeability of Hulls

A hull is a barrier, formed between three point masses, which is impermeable to the passage of other masses. The constraint works much like that of the incompressibility of liquids: when another mass comes within close proximity of the hull, then an application of Hooke's law is implicitly applied at a tangent to the hull to deflect the other mass.

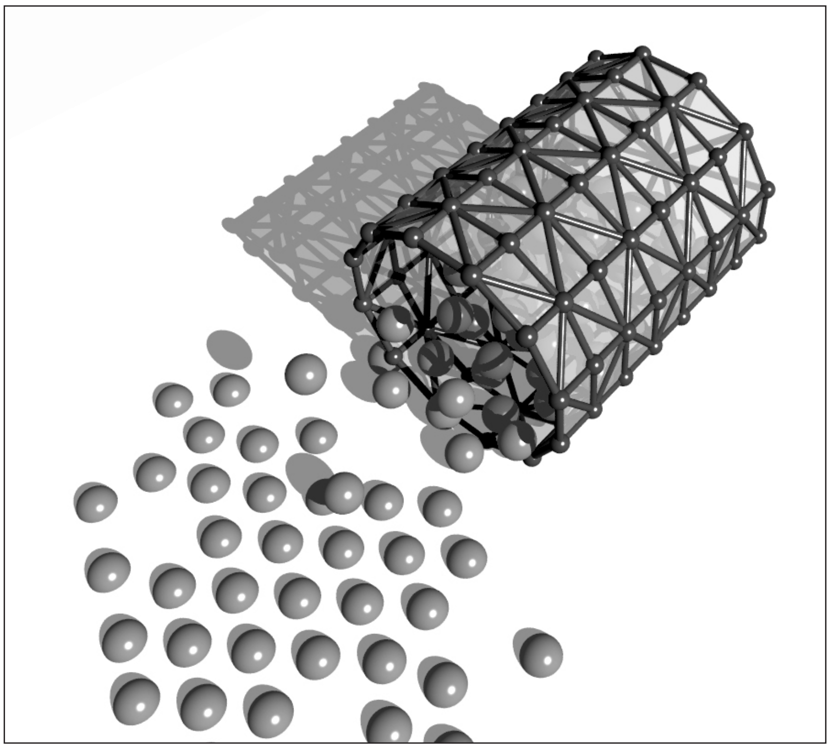


Figure 5.8: A simulated spill of a mug of coffee

Because a hull is impermeable, an additional check is performed to test if a mass is about to pass through the hull (*i.e.*, it has force sufficient to exceed the repulsive force due to Hooke's law). In this case, the potential collision is treated as an elastic collision about the surface normal of the hull.

When the hull causes another mass to be deflected, the corners of the hull are subject to a reactive force. This reactive force is distributed over the point masses at the corners of the hull so that $d_1F_1=d_2F_2=d_3F_3$ and $F_1+F_2+F_3=F$ (where d_i is the distance from the point of impact to the corner mass and F_1 is the force applied to the corner mass in the direction of the surface normal), ensuring that equal torque is applied to each corner.

5.4.4. Discussion

The result of simulating a spill by applying a force to the rim of a cup is depicted in Figure 5.8*. In Chapter 8, I will conduct a thorough evaluation of the use of mass-spring systems to model physical

*Note that the liquid particles have spread out onto the table in a grid-like pattern. This is caused by the 'radius of effect' that prevents liquids from becoming compressed. The figure should be interpreted as though the liquid fills the space between the rendered 'particles of liquid'.

systems (such as the *Egg Cracking Problem*). However, it is important to note that, given sufficient computing resources and modeling resolution, seven physical constraints, such as those of the example, are sufficient for realistically simulating any purely mechanical system of solids and liquids. This is the power of simulation: any novel object can be simulated in rich detail without modifying any constraint (or line of code), simply by configuring the initial attribute values to match real life behavior.

Richer simulations can be created though the addition of further rules. For example, heat diffusion in the coffee may be simulated with the addition of attributes for current temperature and specific heat and by a constraint that equalizes temperature differentials between nearby masses. Convection currents in the coffee may even be simulated by adding a rule that increases the repulsion forces between hot elements of liquid, thereby reducing the effective density of warmer parts of liquid, allowing it to convect to the top of the liquid.

A strength of simulation is that the addition of such rules do not require changes to existing rules. This simplifies the development process, enabling the system to scale up in terms of both computational and system complexity, and means that every rule provides a clear incremental benefit. For example, the addition of a rule for heat diffusion dramatically increases the scope of problems that the simulation can solve, allowing the system to additionally model the thermal effects of such diverse objects as ovens, cups of coffee, bathtubs and refrigerators.

5.5. Implementation

The formal model of a Comirit Simulation lends itself to implementation in modern programming languages. Object-oriented encapsulation is an important technique for managing the complexity and integration of components in the Comirit Framework. Thus, I consider below how simulation may be modeled as an object-oriented design. In Section 5.5.2, I then consider techniques for optimizing the performance of a simulation.

5.5.1. Object Oriented Implementation

The translation of the formal model of a Comirit Simulation to an object-oriented design is relatively straightforward. Figure 5.9 depicts the crucial classes and interfaces of a Comirit Simulation. Note that the top-level objects in the hierarchy (**`Term`**, **`Rule`**) will be explained in the following Chapter; they are necessary for integration into the hybrid reasoning framework.

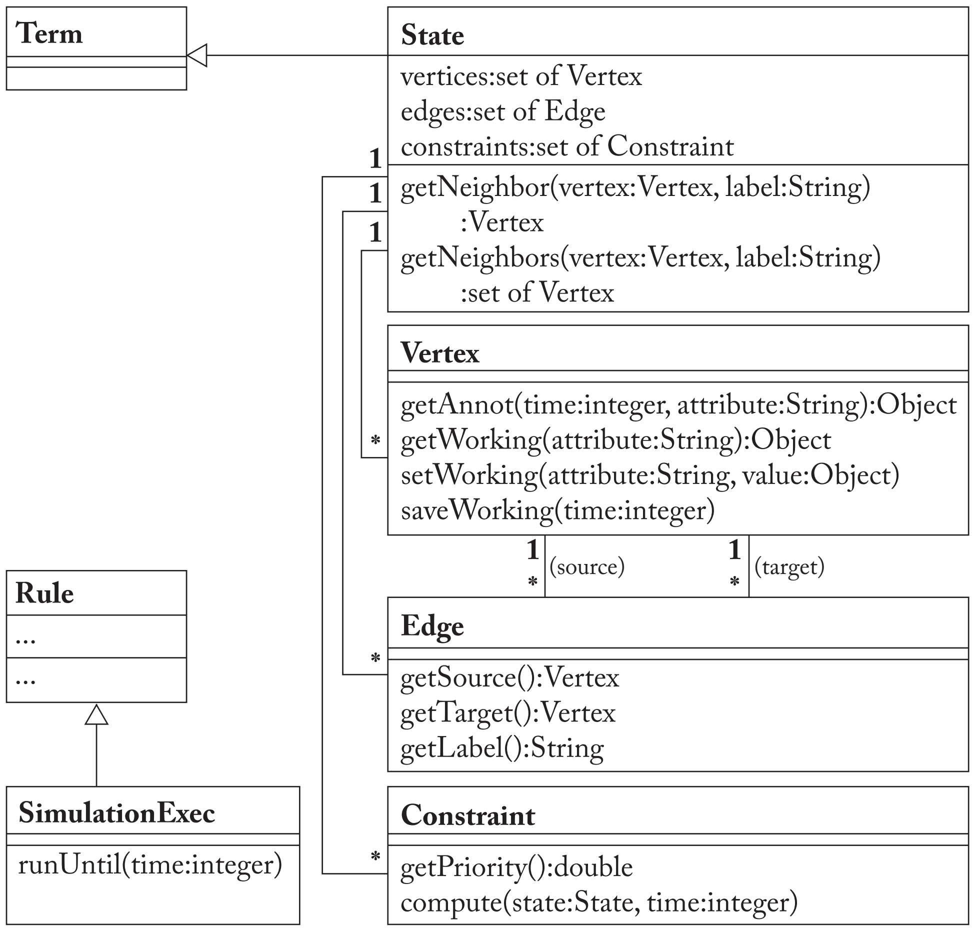

Term
State
vertices:set of Vertex
edges:set of Edge
constraints:set of Constraint
getNeighbor(vertex:Vertex, label:String) :Vertex
getNeighbors(vertex:Vertex, label:String) :set of Vertex
1
1
1
Vertex
getAnnot(time:integer, attribute:String):Object
getWorking(attribute:String):Object
setWorking(attribute:String, value:Object)
saveWorking(time:integer)
*
1
*
(source)
1
*
(target)
Rule
...
...
Edge
getSource():Vertex
getTarget():Vertex
getLabel():String
*
SimulationExec
runUntil(time:integer)
Constraint
getPriority():double
compute(state:State, time:integer)
*

Figure 5.9: Class diagram for a Comirit Simulation

This object oriented design can be adapted to new problem domains by extending (that is, subclassing) the leaf classes. For example, a physics simulation will require subclasses of **`Vertex`** for masses, springs, corners and hulls, in addition to subclasses of **`Constraint`**, corresponding to each of the laws of physics.

5.5.2. Optimization

An un-optimized implementation of the design of Figure 5.9 is sufficient for small-scale problems. My simple prototypes, implemented on standard desktop computers, are capable of simulating thousands of vertices in near real-time.

In principle, *if* constraints are implemented to execute in *constant time*, the algorithm will execute in $O(n\,t\,c)$, time where n is the number of vertices, t is the number of iterations of the simulation and c is the number of constraints. In practice, this may be reasonable as many constraints perform only constant-time work on a given vertex. For example, the constraints for gravity and momentum in section 5.4.3 read and modify a constant set of annotations in a constant operation for each vertex.

Constraints that interact with connected neighbors tend to slightly increase computational expense. In this case, the complexity increases with the degree of connectivity of the graph, a factor that usually remains small. For example, the vertices in a mass-spring simulation are usually connected to just three neighbors but rarely connect to any more than six neighbors. With suitable data-structures, these neighbors can be accessed in constant time.

The most significant computational cost of a simulation comes from complex definitions of a vertex's 'neighborhood'. For example, in a physical simulation, there is a need to detect collisions between unrelated objects that are in physical proximity (*e.g.*, similar attributes for position when computing liquid incompressibility). This may require an $O(n)$ iteration through each of the n vertices in the graph (slowing the entire algorithm to an $O(n^2tc)$ time complexity). The resulting $O(n^2tc)$ algorithm, being polynomial-time, still compares favorably to an undecidable theorem prover. However, if greater performance and scalability is required, indexing schemes can help. For example, an $O(\log n)$ spatial index on position attributes will reduce the complexity of the algorithm to $O(n \log n \cdot tc)$.

Furthermore, because simulations are designed to have primarily local effects (where global behaviors are emergent), spatial partitioning may be used to efficiently distribute the algorithm over a large computing cluster. Each node in the cluster would be responsible for a small subsection of space, with inter-node messaging only required for effects that cross node boundaries.

Other, more sophisticated, opportunities for optimization include the following:

- **Adaptive adjustment of the resolution of objects and constraints**

  Important or complex objects can be modeled with high-resolution mass-spring models, while distant or irrelevant objects are modeled as crudely as possible. Similarly, constraints may operate to different degrees of accuracy depending on the importance of the object in the simulation. For example, when a rough approximation is sufficient, the constraints could be computed by a low-order polynomial Taylor-series expansion.

- **Adaptive adjustment of the resolution of the system timer**

  Crucial inflection points (such as collisions) can be simulated with high precision, while system steady-states are quickly skipped over. The system timer may even operate at different resolutions in distant regions of the same simulation.

- **Dynamic activation of constraints during simulation**

  Constraints only need to be active if they will have a material effect on the simulation. For example, constraints corresponding to the laws of thermodynamics do not need to be invoked when or if the system is in a *thermodynamic* equilibrium, even if it has not yet reached a *mechanical* equilibrium.

- **Selective retention and deletion of historical attribute values**

  While the formal model of a simulation assumes all historical values are retained, memory may be conserved by careful garbage collection of attribute values based on their use. For example, if no constraint makes reference to attribute values older than the previous iteration, then there is no need to store more than two iterations of annotation values. Similarly, if historical values are needed by constraints for only *some* attribute values, then only those attributes require a full historical record.

- **Lazy, partial, incremental and parallel computation**

  If the full configuration of a simulation is not immediately available at instantiation or if only parts of the simulation output have high urgency, then it may be possible to vary the order of computation so that it is no longer performed in stepwise advances. The simulation may begin computing with whatever data it has available or it may begin by prioritizing computations that are more pertinent to timely decision making. For example, it may first compute physical mechanics on nearby objects, it may then compute mechanics on distant objects and then, finally, overlay the results of thermodynamic laws. This could be handled by lazy programming languages, by balancing thread priorities in a parallel computation or by extending the simulation framework so that constraints may return special 'deferred' symbols in a form of cooperative multi-processing.

5.6. General Applicability

The flexibility of Comirit Simulations is its strength: the mug of coffee example is just one example of the framework in action. Not only can the graph representation be used to model a huge range of mechanical objects as mass-spring systems but also it can be adapted to different modeling perspectives, different levels of granularity and entirely novel domains including non-physical reasoning.

In the example of Section 5.4, I have taken a mass/material perspective: the simulation tracks the flow of matter through space. This is by no means the only possible perspective. The physical space in a simulation could be parceled into a grid and represented as a mesh of vertices or a cellular automata. The movement of matter and liquid would then be simulated as changes in density at grid locations*.

*These different perspectives are called the Lagrangian (matter-based) and Eulerian (space-based or field-based) views in work on fluid dynamics.

I have chosen not to use this perspective in the example because the ratio of space to matter is so large. It is far more efficient to track the small proportion of mass particles than it is to divide space into voxels at an equivalent resolution. In other situations, it may be preferable to take a space-based perspective as, indeed, Decuyper, Keymeulen and Steels [35] suggested in their preliminary proposal for physical reasoning systems.

The example of Section 5.4 also used a very rich model of a mug-of-coffee with accurate physics that in many circumstances may not be necessary. For example, in simulating the politics of an office environment, it may suffice to represent an object, such as a mug-of-coffee, as a single vertex with attributes, such as whether it is full or empty, and edges that associate it with vertices to indicate the current owner, room and desk of the mug. That is, the resolution and abstraction of a model is often determined by the intended application of the system. Indeed, it may be the case that multiple models of the same object are needed and used simultaneously by a given system.

Finally, while I have not had the opportunity to experiment with rich non-physical environments, the Comirit simulation framework is *not* limited to physical simulation. For example, even though iterative models of emotional state and game theoretic models of human behavior might be overly simplistic approximations for understanding a specific event, their use in simulation may be sufficient in practical applications where the general properties of emergent behavior is of concern. While a simple model could not accurately predict the precise moment that a hectic schedule would result in burn-out, it may be sufficient for a commonsense reasoning system to understand that hectic schedules do have emotional tolls that can eventually lead to burn-out. Capturing such a principle using formal logics is likely to be significantly more challenging than modeling fatigue as a simulation constraint that gradually affects agents.

5.7. Limitations and Challenges

Comirit Simulations are an extremely powerful and general purpose approach to modeling a wide range of commonsense scenarios but they are by no means perfect or complete. In this section, I consider some of the limitations and open questions of simulation.

5.7.1. Deductive Inflexibility

The most significant limitation of simulation is its deductive inflexibility. Given a complete specification of initial conditions and inputs to a system, a simulation can efficiently predict the future states and consequences of the system. However, simulations perform poorly where there is uncertainty or where it is necessary to infer possible *causes* of a final state. That is, simulations are constrained by the 'arrow of time'. The mug of coffee simulation of Section 5.4 can easily predict the outcome of dropping a mug onto the floor but, given a mug in a puddle of coffee, it is not possible to run the simulation backwards in order to find how the event may have happened*. Instead, to reason about causes or uncertainty, an external mechanism must generate hypotheses of prior states. In such cases, simulation remains useful because it can be used to validate the hypotheses (*e.g.*, by testing that the hypothesis does result in the observed outcome).

Simulation is also a poor match for deduction in those domains that are not governed by a set of fundamental laws or trends from which observable behaviors emerge. Problems of naïve physics are well suited to simulation because Newton's laws of motion are a small set of fundamental laws with local effects that can be applied iteratively. However, systems which have unconstrained non-local effects do not suit simulation. For example, in a legal expert system, small changes to the phrasing of a legal definition can have far reaching and unexpected consequences to many different laws. A subtle change in the legal definition of a trust may have dramatic tax impacts on individuals who attempt to protect their wealth through trusts. Modeling the interactions of such laws would be better suited to a traditional expert system rather than attempting to convert it to a simulation.

5.7.2. Control Languages

In this chapter, I have described the operation of a simulation but have left interfaces for instantiation, manipulation and observation unspecified. This aspect of a simulation is crucial in practical applications. In particular, there is a need for control languages that include operators for the following tasks:

1. Instantiate entirely new instances of a simulation
2. Create new graphical models or retrieve previously stored models
3. Initialize a simulation with default values
4. Apply additional inputs to the running simulation to model externally-controlled variables
5. Control the computation of the simulation (with time-limits, CPU-limits, or CPU-sharing)
6. Read and interpret the results of a simulation
7. Store final values and graph-structures of a simulation for future use

*Non-linearities in the constraints, rounding errors and the effects of friction ensure that an inversion of constraints is computationally difficult and highly uncertain.

Because of the consistent underlying graph-based representation, such a control language will find diverse and novel applications. For example, a general purpose 'copy' operator might allow for the instantiation of a *glass* of coffee (rather than a ceramic mug) by copying the stored annotations for 'glass' over the top of the graph-based structure of a mug.

I initially developed Comirit with a purpose-built control language based on XML and XSLT transforms. Upon integrating simulation within a hybrid reasoning framework, it became apparent that the control language is better embedded into the design of the Comirit Framework, as standard reasoning operators. Thus, the general purpose rules that I will describe in Chapter 6 may also be used to implement any of these control operators.

5.7.3. Tool Support

The implementation of the base classes of a simulation and appropriate constraints is a relatively straightforward task. The time and effort associated with using simulation predominantly stems from the tedium of designing appropriate 3D models and tuning the annotations so that behaviors match reality. In developing prototypes, I built simulations by careful spreadsheet calculations, the adaptation of 3D chemical models[*] and repeated trial-and-error. While manual fine-tuning was suitable for this preliminary research, scalable development of complex simulations in commercial environments will require suitable modeling tools.

The development (or design) of modeling tools is beyond the scope of this dissertation. However, my practical experiences suggest the need for the following core features:

- A visual 3D modeling interface to quickly build accurate models of everyday objects, with (in the case of physical simulations) all masses, springs, corners and hulls appropriately attached to each other
- The ability to import 3D models created by other tools and automatically generate Comirit Simulations from the 3D structure
- The ability to set global values (for example, mass or rigidity) of an object and have those values divided or replicated globally over each vertex of the graph structure
- The ability to test models and their parameters in a live and interactive simulation to ensure that the models behave realistically

[*]For example, I deformed the raw 3D description of a spherical C_{60} buckyball molecule that I found on a chemistry website to create a simulated egg in an early prototype.

The development of a toolset for building and experimenting with simulations will clearly require an investment of development resources. For this reason, simulation as a method of knowledge engineering may be characterized by higher initial investments (tool development) but very small long-term costs (rapid knowledge engineering in such visual tools).

Tools for constructing simulations may also incorporate automation features. One could, for example, imagine a live 3D motion capture system that automatically gathers the physical dimensions of an object, builds an appropriate mass-spring approximation and then uses machine-learning techniques to discover appropriate attribute values so that approximation accurately matches reality. In fact, in Chapter 7, I will discuss how such automation may *eliminate* many manual knowledge engineering efforts altogether.

5.8. Conclusion

There is a significant amount of commonsense knowledge implicitly represented in modern computer games and animations. Inspired by this resource, I have adapted simulations to Comirit by designing a representation scheme for efficient cross-domain commonsense simulation.

While the underlying mechanisms of a Comirit Simulation are equivalent to standard techniques of numerical analysis, the value of Comirit Simulation is the simple and standardized representation that it provides. This representation offers a number of benefits:

- The underlying principles and components may be easily understood and implemented by software engineers without in-depth training in formal logics.
- The structural architecture of simulation is general purpose, allowing for efficient reasoning across many domains.
- The architecture retains a common set of base abstractions that can be uniformly managed and inspected within an integrated system and that allow for the development of powerful tools to assist in the construction of simulations in any domain.
- The constraints that drive simulations operate orthogonally to each other. Individual constraints may be added or removed at will to change the breadth of a simulation (*i.e.*, changing the constraints in a simulation is configuration, not programming).
- The design is effective in a naïve implementation but the mechanisms may incorporate a number of powerful optimizations.
- The framework has been validated through experiments and prototypes (see Chapter 8).

Chapter 6
Control

In Chapter 5, I argued that simulation is a powerful mechanism for reasoning about the commonsense world. It is expressive, cost effective, efficient, simple and easily debugged. In fact, a wide range of commonsense reasoning problems may be solved by simulation alone: consider the many variations on the *Egg Cracking Problem* mentioned in Section 2.2 that simply asked for the outcomes of a range of scenarios and thus could be solved by direct simulation.

However, simulation is not without its limitations. It is approximate, it does not work well with uncertainty, it lacks deductive generality and it does not provide mechanisms for introspection. For example, many problems require an agent to infer possible causes or explanations of a situation. Simulation is simply unable to be used in a 'backward chaining' inference mode, not to mention its complete lack of traditional logical operators.

It would be premature to dismiss simulation solely on the basis of these weaknesses. Simulation and logic have highly complementary strengths and weaknesses. If logic is combined with simulation, then the strengths of simulation can compensate for the weaknesses of logic and *vice versa*.

Thus, I propose an integrated hybrid reasoning system that combines complementary reasoning mechanisms into a unified whole. Through integration, I seek to obtain advantages that are not merely additive. Each mechanism may, indeed, solve certain problems independently. However, the real advantage of hybridization comes from the multiplicative effects when complementary mechanisms combine and cooperate to solve sub-problems of a complex problem.

Of course, no pairing will be perfect. Simulation and logic form an excellent partnership but there is always the possibility of future integration to make up for remaining deficiencies. A design goal of the hybrid reasoning framework of Comirit is, therefore, to create an *open-ended* architecture that harmoniously combines a multitude of reasoning mechanisms with a particular emphasis, in the first instance, on the combination of simulation and logic.

A crucial challenge of integration is to address conceptual and semantic mismatches. It is not merely a matter of compiling the source code of a simulation engine and a logical reasoner into a single module; their behavior needs to be carefully orchestrated. Such integration remains a perennial problem in Artificial Intelligence and, while there are a number of 'routine' techniques used in the field, none are well suited to the solving the peculiar semantic mismatch between simulation and logic. Instead, I propose a novel re-imagining of a theorem-proving mechanism to create a deep integration of both mechanisms and semantics, without detracting from the power of either.

In this chapter, I will describe the core *Comirit Framework*. Hybrid reasoning in Comirit is based on a generalization of the method of analytic tableaux. I begin with a review of the literature and then provide a formal description and an example of the framework. This is followed by an object-oriented design. The chapter concludes with a discussion of the benefits and a summary of the framework.

I evaluate the framework in later chapters. First, I extend the framework in Chapter 7 so that it supports autonomous learning. This demonstrates that the design is open-ended. In Chapter 8, I evaluate the framework from a functional perspective to show that it enables flexible reasoning and has capabilities beyond that of simulation alone.

6.1. Connections to the Literature

The pragmatic objectives of this research project mean that, wherever possible, I prefer to use well-understood and established techniques of Artificial Intelligence (AI). However, while there are many integration techniques used in AI research, none addresses the semantic mismatch of a hybrid reasoning scheme combining simulation and logic for commonsense reasoning.

In this section, I review the standard integration and 'attachment' techniques of AI to explain why it is necessary to look elsewhere. I then review work in tableau reasoning as a background to a new integration technique that I have proposed as part of Comirit.

6.1.1. Integration Techniques

The need to integrate multiple mechanisms is a recurring theme in Artificial Intelligence research. The explicit top-down control structures of routine software engineering practices tend to present difficulties when applied to integration in Artificial Intelligence. This is because the precise sequencing of a system's problem-solving activity is often not known *a priori*: it is dependent on the structure of the problem, the varied capabilities of problem-solving components and the availability of relevant data during the problem-solving process. In typical AI systems, explicit control flow is therefore replaced by an implicit process that emerges from the bottom-up interactions of independently communicating components. A range of common patterns, with somewhat overlapping principles, has been applied with success in many modern intelligent systems, for example:

- Blackboard Systems [71, 28],
- Tuple Spaces [23, 6],
- Publish/Subscribe Message Routing [41, 170],
- Agent-oriented Architectures [78, 157].

In fact, specializations of these patterns have already been proposed as approaches to integration for intelligence and commonsense reasoning (see Section 2.8 for a review of some of these proposals). Furthermore, I have had personal experiences with each of these architectures and have found them useful in a range of academic and professional problem domains. It was therefore my preference to again apply these established patterns to the integration of Comirit.

Over the course of six months in 2007, I developed prototypes of several simulation engines and logical systems (ranging from propositional provers and Prolog SLD resolution to description logics and incomplete first order logic) and attempted to integrate these with blackboards, tuple-spaces, messaging systems and agent architectures. While the components worked well individually, it was not possible to deeply integrate their reasoning, except on an ad-hoc basis. Ultimately, the conceptual mismatch between simulation and logic meant that standard integration techniques resulted in systems that were unworkably complex and difficult to maintain. These difficulties stemmed from two factors:

1. **A mechanism mismatch:** simulation operates by forward-chaining future states in chronological sequence from concrete scenarios, whereas logical proofs are often not even constructive and can use both forward and backward chaining, independent of temporal correlations.
2. **A representational mismatch:** simulation concerns iterative local computation over concrete numerical scenarios, whereas logical reasoning involves 'non-linear' manipulation of abstract symbols that may indirectly reference many scenarios.

These mismatches can only be solved by reinterpreting the underlying mechanisms. Simulation and logic do not integrate well if they are implemented only as cooperating 'black-box' components; they must be unified through a common conceptual and semantic framework.

Comirit therefore stems from an observation that the method of analytic tableaux, a mechanism for automated theorem proving, stands out as also being highly compatible with the semantics of simulation. In the following section, I provide an overview of the method and, in the remainder of the chapter, show how it solves the challenges of integration.

Of course, a caveat to this discussion is that general-purpose design patterns such as the blackboard architecture could, in principle, be used to structure *any* form of computation. Indeed, the Comirit Framework *can* be emulated in a blackboard architecture (and *vice versa*) with appropriate representations. Such emulation may, in fact, be necessary if Comirit is to be integrated into an existing legacy system based on the blackboard architecture. However, 'emulation' adds unnecessary layers of complexity and so new applications of Comirit are obviously better suited to a native representation that does not require the abuse of formalisms or the addition of 'emulation layers'. This is analogous to the theoretical equivalence but practical differences that exist between the abstractions used by Universal Turing machines and the Java Programming language. The generalization of tableau-based reasoning, and its use in structuring control flow and representations, is the important conception behind this work, providing an appropriate language for managing the orchestration of complex multi-paradigm reasoning.

6.1.2. Logical Attachments

While the techniques of Section 6.1.1 are solutions to the general problem of *integrating* mechanisms, an alternate approach would be to assume the primacy of logic in a deductive system where simulation and other methods are merely 'attached' to a logical reasoner.

In particular, simulation could be 'attached' to a logic by either semantic attachment [174], theory resolution [163] or universal attachment [125]. These techniques associate computational processes with expressions (*i.e.*, terms, variables, formulas or conjunctions) in a logical language. These processes are then 'attached' to a logic by adding inference rules that allow an associated computational processes to act as an oracle for proof of matching expressions.

While methods of logical 'attachment' are appealing for their elegant formal foundation and an established body of literature, they are not an appropriate match to hybrid commonsense reasoning. Logical attachments necessarily have only local effects—this is fundamental to their formalization—but such local effects are unsuitable for the kind of contextual and interdependent reasoning that is required for commonsense. For example, a perfect candidate for 'attachment' might be the multiplication operator: 3 × 5 = 15 is difficult to prove in Peano arithmetic but is quickly and reliably computed by machine. The truth of 3 × 5 = 15 is unaffected by the current weather or to the location of a table in a room, thus the multiplication operator has a straightforward 'attachment' to the functor '×' or to expressions of the form A × B = C. In contrast, whether a mug of coffee can be safely carried is dependent on a potentially infinite number and variety of obstructions and influences. In commonsense reasoning, simulation and logic may both simultaneously reference any object in the world, and therefore it is not possible to 'attach' one to the other by standard means (without meaninglessly generalizing the attachment to concern the entire universe of expressions).

6.1.3. Tableau Systems

The method of analytic tableaux is an approach to automatically proving (or disproving) the truth of a logical statement through search for consistent models or worlds. The technique is over 50 years old but has experienced a recent surge in interest due to its applicability to modal logics, description logics and Semantic Web reasoning [31]. A thorough overview of the method, implementation and applications to reasoning within propositional, first-order, modal, description and other logics can be found in D'Agostino *et al.*'s (eds.) handbook [31], Robinson and Voronkov's (eds.) handbook [146] or other standard references for formal computer science. For convenience, I present here a brief review of the technique for the propositional case. Readers familiar with the method may prefer to skim over this section.

A tableau system is a method of proof by contradiction: it proves validity by showing that the negation of a formula is unsatisfiable. It operates by systematically analyzing formulas until it finds inconsistencies between elementary formulas.

The algorithm works by repeated application of rules that transform the syntax of a formula into a tree structure (*i.e.*, a 'tableau'). Starting with the negated input formula as the root, the basic algorithm is simple: when a branch contains a disjunction, the branch is forked into two child branches corresponding to each of the disjuncts; when a branch contains a conjunction, each conjunct is added to the leaf of the branch (see Figure 6.1 on the following page). The effect of applying these rules is that a formula is converted into a tree, where the parent-child relationship can be read conjunctively and the sibling relationship can be read disjunctively.

Conceptually, the tree structure of a tableau corresponds to a search tree for cases or models that satisfy a query. Because the parent-child relationship is read conjunctively, branches (paths from the root to a leaf along the ancestor/descendant axis) are searched for contradiction. If every branch contains a contradiction, it means that no case or model could be found: the negation of the original formula is therefore unsatisfiable and the original formula is consequently valid.

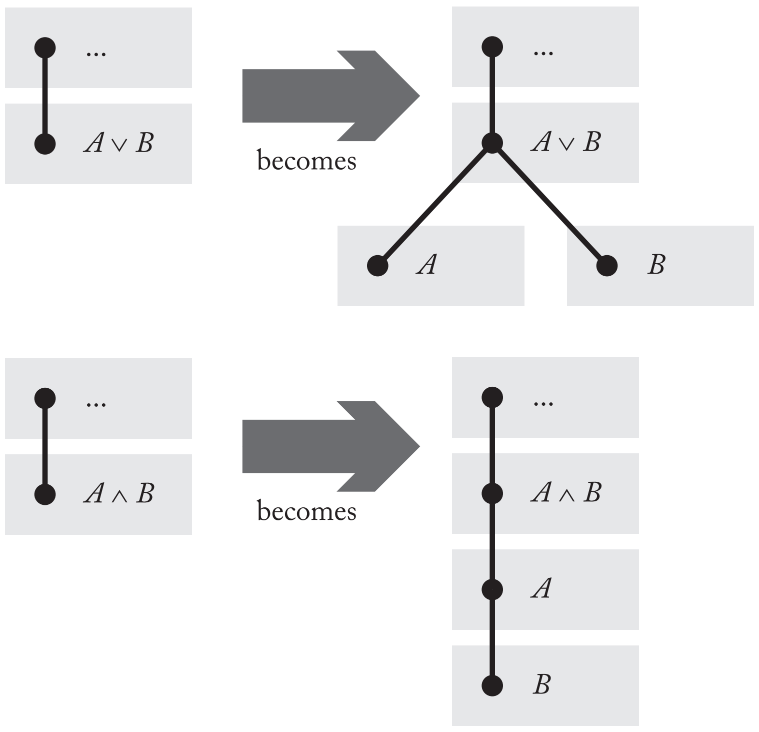


Figure 6.1: The effects of the disjunction rule (above) and conjunction rule (below) on a tableau

That is, the algorithm operates according to the following sequence of steps:

1. The input to the algorithm is negated (*e.g.*, the formula is syntactically enclosed in a ***not*** term).
2. A tree is created with a single root node, annotated with the negated input. Every branch is initially open (*i.e.*, assumed to be consistent).
3. The algorithm repeatedly applies tableau rules to open branches of the tree, until no more rules are applicable:

    3.a. If the branch contains a node that is annotated with a disjunction (or other disjunctive operators), then each of the disjuncts may be appended as *separate child nodes* branching off the leaf of that branch.

    3.b. If the branch contains a node that is annotated with a conjunction (or other conjunctive operators), then each of the conjuncts may be appended in a *chain of new nodes* attached as a child to the leaf of that branch.

3.c. If a branch contains a contradiction among the annotations on nodes in the branch, then the branch is marked as closed.

4. When no more tableau rules are applicable, the final state of the tableau is used to determine the truth of the original input:

   4.a. If all branches have been closed, then the original formula is universally valid (*i.e.*, the formula is valid, because no satisfying model could be found for its negation).

   4.b. If there remains one or more open branches, then the conjunction of the annotations on an open branch represents a counter-example, demonstrating the invalidity of the original formula (*i.e.*, the formula is invalid because a satisfying model could be found for its negation).

Recalling that a formula is *satisfiable* if *its negation is not valid*, one can also use the tableau method to prove satisfiability of a formula. To test for validity, the input formula is negated before applying the tableau algorithm or, equivalently, by disregarding the first step of the algorithm (since two negations cancel each other). Thus, the original formula is satisfiable if open branches remain at the end of the modified algorithm and, indeed, these open branches are satisfying models that serve as witnesses to satisfiability.

For illustrative purposes, I will demonstrate how the tableau method can be used to prove that the formula $((a \lor b) \land \neg a) \Rightarrow b$ is universally valid.

To show that $((a \lor b) \land \neg a) \Rightarrow b$ is valid, it is first negated $\neg(((a \lor b) \land \neg a) \Rightarrow b)$ and then placed at the root of a tableau.

A tableau may contain simplification rules to automatically translate logical implications ($\Rightarrow$), equivalences ($\Leftrightarrow$) and nested negations into simple disjunctions and conjunctions. In this example, I will replace the root formula with its negation normal form: $(a \lor b) \land \neg a \land \neg b$, so as to avoid long chains of uninteresting simplifications from appearing in figures. In practice, it is perfectly acceptable to leave the simplification to the tableau reasoner.

The negation-normal negated formula is then shown to be unsatisfiable (or otherwise) by applying the tableau rules until all branches can be closed. Figure 6.2 illustrates the iterative effects of rule application.

A contradiction can be seen along both of the remaining branches of the completed tableau. On the left-hand branch, the node annotated with a has an ancestor annotated with $\neg a$, resulting in a contradiction. Similarly, the node annotated with b contradicts an ancestor annotated with $\neg b$. Since all branches are contradictory and therefore closed, the original (un-negated) formula is therefore deduced to be universally valid.

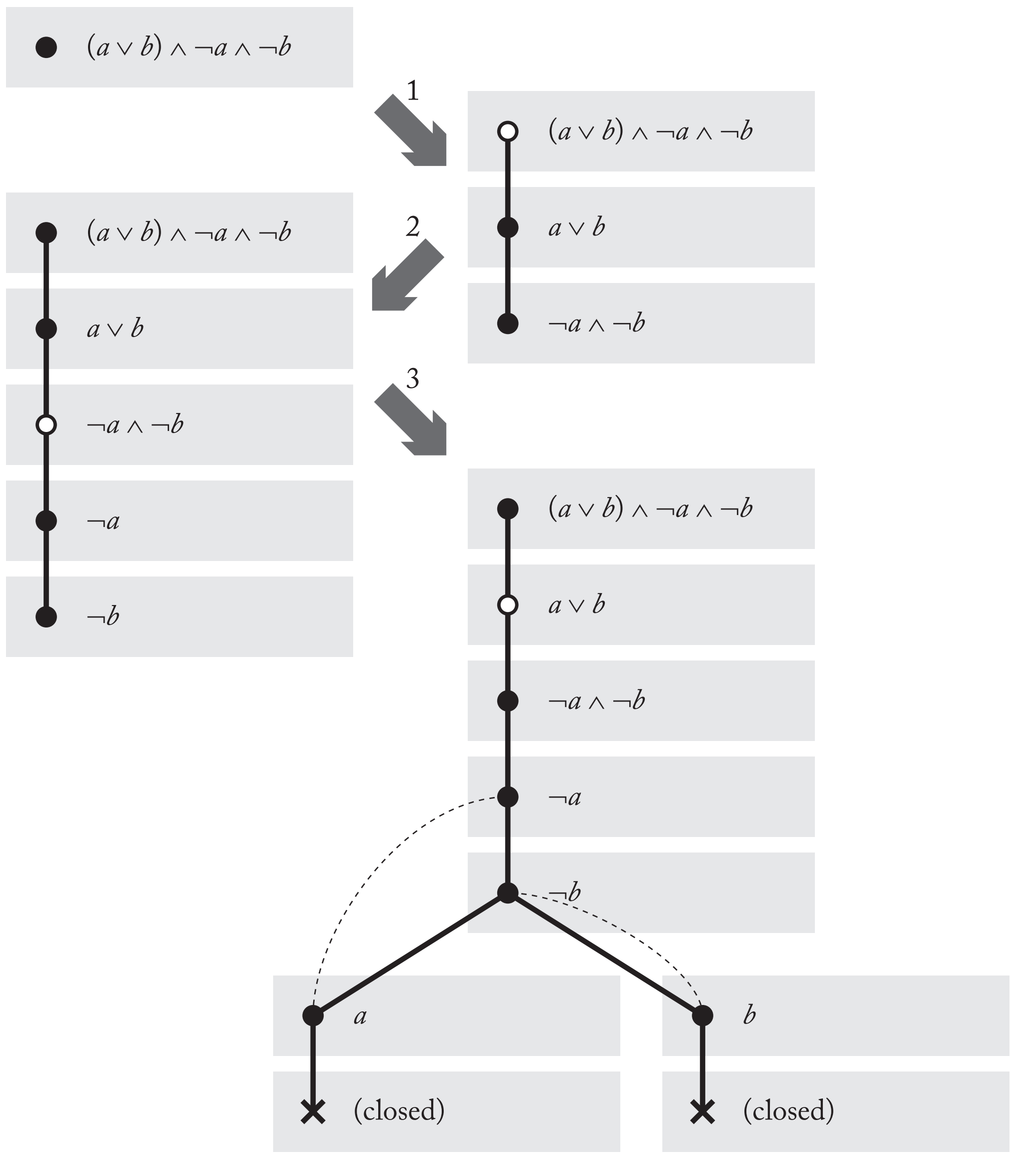

$(a \vee b) \wedge \neg a \wedge \neg b$
1
$(a \vee b) \wedge \neg a \wedge \neg b$
$a \vee b$
$\neg a \wedge \neg b$
2
$(a \vee b) \wedge \neg a \wedge \neg b$
$a \vee b$
$\neg a \wedge \neg b$
$\neg a$
$\neg b$
3
$(a \vee b) \wedge \neg a \wedge \neg b$
$a \vee b$
$\neg a \wedge \neg b$
$\neg a$
$\neg b$
a
b
(closed)
(closed)

Figure 6.2: An example illustrating the steps in tableau reasoning about the formula $((a \vee b) \wedge \neg a) \Rightarrow b$. (1) The conjunction rule is applied to expand the root, giving nodes $a \vee b$ and $\neg a \wedge \neg b$. (2) The conjunction rule is then applied to $\neg a \wedge \neg b$ to give $\neg a$ and $\neg b$. (3) The disjunction rule is applied to $a \vee b$, and then the two branches are found inconsistent (a contradicts $\neg a$, and b contradicts $\neg b$).

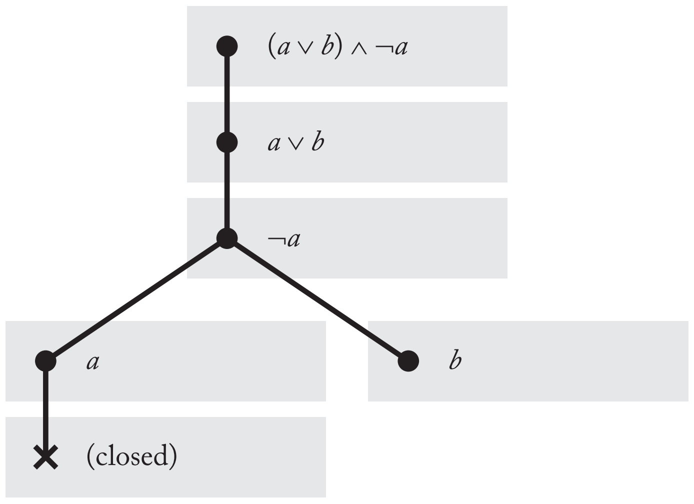


Figure 6.3: A completed tableau for the formula $(a \vee b) \Rightarrow a$.

Similarly, consider the result of applying the method to a formula that is not valid: $(a \vee b) \Rightarrow a$. It is first negated $\neg((a \vee b) \Rightarrow a)$ and converted to negation-normal form $(a \vee b) \wedge \neg a$. Then, the tableau rules are applied to obtain a completed tableau per Figure 6.3.

A contradiction can be seen along the left-hand branch: a is an ancestor of $\neg a$ and so the left-hand branch (when read conjunctively) is in contradiction. The right-hand branch, however, remains open with neither contradictions nor the possibility of applying any rule (without redundant repetition). In fact, the right-hand branch (when read conjunctively) is a counter model for the original formula: $(a \vee b) \Rightarrow a$ does not hold when both $\neg a$ and b hold. The tableau method is an efficient way of searching for such counter models.

While I have used classical propositional logical for my example, the tableau algorithm is readily adapted to a range of logics. For example, modal logics, dynamic logics and logics with quantifiers can be supported by the addition of new tableau rules for introducing labels, Skolem terms, unification and unbound variables. Details of modifications for these and other logics can be found in work such as D'Agostino *et al.* [31].

I have provided only a conceptual overview, ignoring the details of rule selection, search strategy, heuristics and optimization, quantification and modalities, preventing cycles, managing repeated rule application and the complexities of undecidable logics (such as first order logic). Of course, these matters are highly relevant to actual applications and a complete commercial implementation of Comirit but I consider them beyond the scope of this dissertation in which I seek to demonstrate the *feasibility* of the Comirit architecture. In Section 6.2.1 and 6.3.1, I discuss how the generalized tableau reasoning framework permits tableau rules to be themselves included inside the tableau. This allows for configurable (and modular) tableau rules, meta-strategies and data-types. The extensive literature on efficiently implementing tableau systems can be readily adapted into this framework, by implementing new rules that manipulate the tableau and introduce additional optimizations.

6.2. Concept

The key insight within the Comirit Framework is that the tableau method can also be used for non-logical reasoning. The tableau method is a constructive approach to finding contradictions: it attempts to find possible models (or worlds) in which a logical formula is inconsistent. These constructive models are the basis of integration. Rather than merely limiting tableau search to *logical* inconsistency in symbolic models, I extend the tableau search to include inconsistency under mechanisms such as simulation.

The tableau method is very versatile. With appropriate rules, it can reason about a wide range of logics. Hähnle [64] demonstrates that it subsumes many methods of automated logical deduction: model elimination, linear resolution, logic programming and the Davis-Putnam procedure. I am therefore presenting a further demonstration of its versatility, extending tableau reasoning to a framework for uniformly reasoning about both logical *and* non-logical deduction.

Simulation and tableau reasoning can be seen as processes concerned with dividing and subsequently 'closing' the space of possible worlds. In traditional tableau-based reasoning systems, broad regions of the space of possible worlds are divided and closed by syntactic analysis of logical formulas. Simulations, in contrast, work only on narrowly defined spaces of possible worlds but, also, ultimately divide and 'close' spaces of possible worlds. This correspondence is the fundamental principle behind the integration strategy used in Comirit.

6.2.1. Formal Definition

Hybrid reasoning in Comirit is a generalization of the method of analytical tableaux. The tableau is generalized so that it can contain not only logical terms but also any arbitrary object *and even the tableau rules that update the tree*.

Annotations

In Comirit, a tableau is represented as a tree that is annotated with formulas that are either terms, rules or the token '*closed*'. These annotations are defined by the following free-type declaration:

$[TERM]$

$FORMULA ::= closed \mid term⟪TERM⟫ \mid rule⟪\mathbb{F}\ FORMULA \twoheadrightarrow \mathbb{F}\ \mathbb{F}\ FORMULA⟫$

The token *closed* is used to denote that a branch of the tableau is inconsistent (and therefore should be closed). It is generated by rules that detect contradictions.

Terms are arbitrary objects that are opaque to the tableau algorithm but are used by rules. Terms may represent logical symbols, logical connectives, simulations, temporary data or any other representations that the system reasons about. These objects need not be atomic: the tableau rules may be implemented to decompose and analyze the terms.

Tableau rules are functions that take sets of formulas (whether terms or other rules) and create new formulas for injection into the tableau. A tableau rule returns a set of sets, indicating how the new formulas are to be annotated to the tableau. The set of sets is to be read as though it is a decomposition into disjunctive normal form. Each set is used to create a new disjunctive child branch and that set's members are transformed into a conjunctive chain of child nodes.

The rule constructor is defined in terms of finite power-sets ($\mathbb{F}$) and finite functions ($\twoheadrightarrow$) because an infinitary declaration would create a logical inconsistency. This finitary restriction has no practical impact because rules (and tableaux) must ultimately be implemented on a Turing machine.

These generic representations can describe the components of traditional logical tableaux: logical connectives are merely compound terms (*e.g.*, a, b, $a \vee b$, $a \wedge b : TERM$) and logical deduction can be implemented with rules:

- The disjunction rule is implemented as a function that searches for a disjunction matching $A \vee B$ in its input and returns the set $\{\{A\}, \{B\}\}$.
- The conjunction rule searches for a conjunction matching $A \wedge B$ in its input and returns the set $\{\{A, B\}\}$.
- The contradiction rule may search for the presence of both terms matching A and $\neg A$, returning $\{\{closed\}\}$ if it identifies such a contradiction.

Tree Structure

The tree structure of the tableau itself is likewise defined by a free type declaration:

$TABLEAU ::= node⟪FORMULA \times \mathbb{F}\ TABLEAU⟫$

That is, each node of the tableau is annotated by a formula and contains a finite (but possibly empty) set of child nodes.

A branch of a tableau is a sequence of nodes, starting at the root and terminating in a node with no children. The branches of a tableau are given by the *branchof* relation:

$BRANCH == seq_1\ TABLEAU$

$$
\begin{array}{|l}
\_\ branchof\ \_ : BRANCH \leftrightarrow TABLEAU \\
\hline
\forall B : BRANCH;\ T : TABLEAU \bullet \\
\quad B\ branchof\ T \Leftrightarrow \\
\qquad B(1) = T \\
\qquad \wedge \forall i : 2..\#B \bullet \\
\qquad\qquad \exists F : FORMULA;\ C : \mathbb{F}\ TABLEAU \bullet node(F, C) = B(i\text{-}1) \wedge B(i) \in C \\
\qquad \wedge \exists F : FORMULA \bullet node(F, \emptyset) = B(\#B)
\end{array}
$$

◊ "The sequence *S* is a *branch of T* if and only if the first element of *S* is the tableau itself (the root node), each subsequent element is a child of the node that proceeded it and the final element of the sequence has no children"

The annotations of a branch may be extracted and tested for the presence of the *closed* token according to the following definitions:

$$
\begin{array}{|l}
annotations : BRANCH \rightarrow \mathbb{P}\ FORMULA \\
isclosed : \mathbb{P}\ BRANCH \\
\hline
\forall B : BRANCH \bullet \\
\quad annotations(B) = \{F : FORMULA \mid (\exists I : \mathbb{N};\ T : TABLEAU \bullet I \mapsto node(F, T) \in B)\} \\
\forall B : BRANCH \bullet \\
\quad isclosed(B) \Leftrightarrow closed \in annotations(B)
\end{array}
$$

◊ "The *annotations* of a branch are given by the set of formulas that are annotated to nodes in the branch"

◊ "A branch is closed if the token *closed* is an element of the set of its annotations"

The generalized tableau algorithm takes as input a set of formulas (*i.e.*, an input query, along with associated background knowledge, expressed as tableau rules and auxiliary objects). The algorithm proceeds by the selection and execution of rules within the tableau. It operates according to the following steps:

1. The tableau is initialized as a chain of single-child nodes, terminating in a leaf node, with each node annotated by a formula from the input set (*i.e.*, the tableau is a tree with a single branch, annotated with the input set).

2. An open (not closed) branch, *B*, of the tableau is chosen[*] and the following sub-steps are applied in order:

   2.a. If the annotations of *B* contain one or more formulas formed by the rule constructor, then one of those rules is chosen[**] and applied to the annotations of *B* to produce output *R* (a set of sets of formulas).

   2.b. The tableau is modified so that child nodes are added to the leaf of the branch *B*. For each set of formulas in *R*, a chain of child nodes is created from those formulas and added as a child to the leaf of *B*. That is, the tableau is extended as though the structure of output *R* is in an implicit disjunctive normal form. For example, for every $S_i = \{r_{i1}, r_{i2}, r_{i3}, \ldots\}$, that is an element of $R = \{\{r_{11}, r_{12}, r_{13}, \ldots\}, \{r_{21}, r_{22}, \ldots\}, \ldots\}$, the tableau update would result in the addition of a new child, $node(r_{i1}, \{node(r_{i2}, \{node(r_{i3}, \{\ldots\})\})\})$ to the leaf of *B*. This is illustrated in Figure 6.4.

3. If (and while) changes were made to the tableau in Step 2, then repeat Step 2. When no more updates are possible, the algorithm continues with Step 4.

4. When the tableau algorithm makes no more progress, the final state of the tableau is used to determine the output:

   4.a. If all branches have been closed, then the algorithm outputs 'unsatisfiable'.

   4.b. If there remains one or more open (not closed) branches, then the open branches are the algorithm's output and the original input query is known to be satisfiable.

[*]Note that non-determinism is introduced at this step. Selection may be done on a naïve rotational basis, with breadth or depth first search or may make use of sophisticated heuristics that select the most likely branch to remain open.

[**]Non-determinism is also introduced at this step and may also be addressed on a naïve rotational basis or with sophisticated heuristics that select rules that have the most impact, with the least redundant computation and that avoid introducing infinite cycles. In Section 6.3.2, I introduce an indexing scheme that allows for efficient rule selection heuristics.

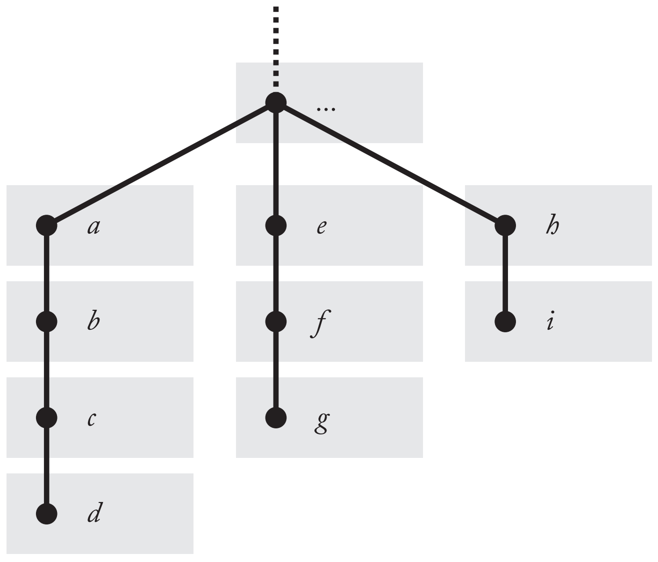


Figure 6.4: The tableau expansion that occurs as a result of a rule which returns the set $\{\{a,b,c,d\}, \{e,f,g\}, \{h,i\}\}$. Note that each set corresponds to a branch and each element of that set is used to create a linear chain of child nodes.

6.2.2. Illustrative Example

The operation of the generalized tableau algorithm may be more clearly understood by way of an example. Consider the following problem:

> A commonsense-enabled, household robot is tasked with cleaning a kitchen. It comes across a mug that is full of coffee and needs to move it safely to the sink. The robot's actuators can move the mug with either a 'low-force' action or a 'high-force' action. Is a coffee spill inevitable? Which actions are suitable for moving the mug without creating a mess?

For illustrative purposes, assume that:

- The 'low-force' exerts a 0.01N force for one second.
- The 'high-force' action exerts a 0.1N force for one second.
- The mass of the mug is 500g.

The problem involves the identification of safe actions. That is, the problem requires models in which an action is performed (*high-force* or *low-force*) and for which no mess is created ($\neg$*mess*). This leads to the following satisfiability query:

$(\mathit{high\text{-}force} \vee \mathit{low\text{-}force}) \wedge \neg \mathit{mess}$

The question of whether a coffee spill is inevitable could be phrased as a related validity query:

$(\textit{high-force} \vee \textit{low-force}) \Rightarrow \textit{mess}$

Recalling that validity may be proven by demonstrating the negation is satisfiable (and *vice versa*), these queries are in fact logically equivalent:

$\neg((\textit{high-force} \vee \textit{low-force}) \Rightarrow \textit{mess}) \equiv (\textit{high-force} \vee \textit{low-force}) \wedge \neg \textit{mess}$

Thus, these two formulations represent different perspectives of the same problem: defining the desired objective versus defining the inverse 'error' conditions.

The generic tableau algorithm, described in Section 6.2.1, lacks specific reasoning rules, so rules must be specified for the robot. I shall assume that the following rules have been implemented:

- Tableau rules for conjunction and disjunction (*e.g.*, $A \vee B \rightarrow \{\{A\}, \{B\}\}$).
- A tableau rule for contradiction (*e.g.*, $A, \neg A \rightarrow \{\{\textit{closed}\}\}$).
- Simplification rules for negated conjunction, negated disjunction, double negation, implication and negated implication.
- Rules that solve for equality (=).
- A rule for expanding (≡) atoms such as *high-force* to their definitions ($\textit{high-force} \equiv \textit{force} = 0.1\text{N}$).
- A simple numerical 'simulation' of a mug of coffee.

For illustrative purposes, I use the following highly simplified and abstracted simulation:

```
rule simulate(force, mass) {
   speed = force / mass * 1s
   if (speed > 0.01ms⁻¹)
      return {{speed, mess}}
   else
      return {{speed, ¬mess}}
}
```

The simulation is initialized by declaring a mug of coffee in the initial conditions: *coffee*. The *coffee* term may be later expanded to its definition: $\textit{coffee} \equiv \textit{mass} = 500\text{g}$.

In a real implementation, the term *coffee* would expand to a 3D model and rich graph-based Comirit Simulations of Chapter 5 would simulate its behavior. Note, however, that the simulation used in this example is functionally equivalent to a more sophisticated simulation. It cannot be used in reverse against the 'arrow of time' to deduce the force from the speed, it can only be applied when its inputs are known and it is only applicable to a single branch of the tableau at a time.

With this configuration, the tableau reasoner can then be used to determine whether a coffee spill is inevitable or if there are suitable actions for avoiding mess. That is, the reasoner is used to test the satisfiability or validity of the input query.

A large set is initialized, containing the input query, the initial conditions and all tableau rules. This becomes the input to the tableau algorithm. The tableau rules are then applied in successive iterations, as depicted in Figure 6.5, until no further progress is possible (*i.e.*, the tableau has 'stalled').

The resulting tableau fuses both logical- and simulation-based reasoning. It contains two branches: the left branch describes a world in which the robot moves its arm with 'high-force' and creates a mess, inconsistent with the $\neg$*mess* goal; the right branch describes a world in which a 'low-force' action did not produce any mess and therefore satisfies the input query. This result can be interpreted in two equivalent ways:

1. The query (*high-force* $\vee$ *low-force*) $\Rightarrow$ *mess* is not valid. It is not the case that a coffee spill is inevitable.
2. The query (*high-force* $\vee$ *low-force*) $\wedge$ $\neg$*mess* is satisfiable and is satisfied by a model in which the *low-force* action is performed. If the mug is moved by the *low-force* action, it should not create any mess.

This example illustrates the principles of the hybrid algorithm and its ability to combine logical and non-logical deduction. In the example, an 'angelic' choice resolved the non-determinism of the tableau algorithm. However, in practice, heuristics may be added as tableau rules for efficient reasoning. My research prototypes use simple estimates of computational cost and a greedy cheapest-first strategy. More sophisticated heuristics, such as measures of expected utility, may provide better performance in demanding applications. In Section 6.3.4, I describe the details of the software implementation that allows for configurable rule selection strategies.

6.3. Implementation

The formal description of the algorithm may be *sufficient* for implementing a system that unifies logical and non-logical reasoning and eliminates the complexity of ad-hoc integration. However, there are many practical considerations in implementation that are not necessarily straightforward. For this reason, I present Comirit with a software design, rather than merely as an abstract algorithm.

In particular, the design of Comirit is intended to address a range of practical considerations and objectives beyond that of the algorithm's core functionality:

1. *Simple* tableau rules should be *simple* to implement.

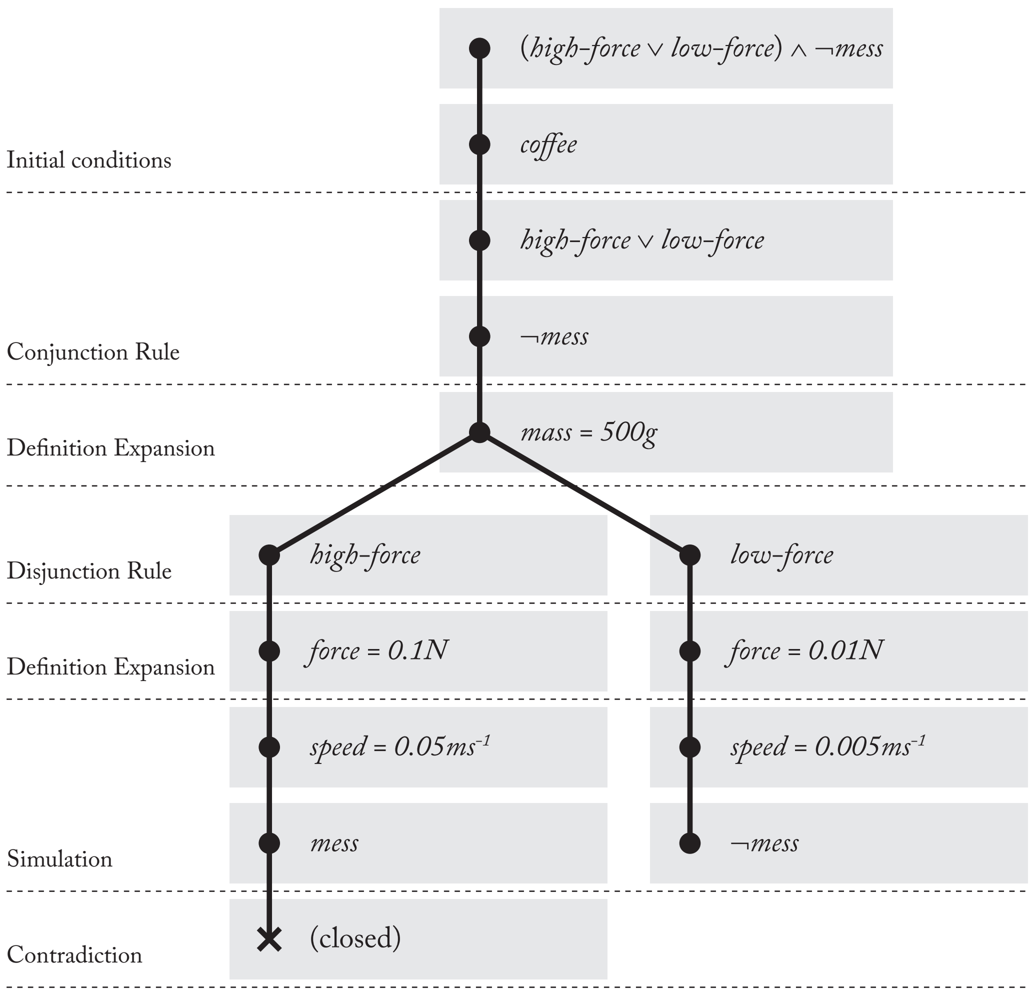


Figure 6.5: A completed tableau for reasoning about how to safely handle a mug of coffee by combining logical deduction and simulation (note that the tableau reasoning rules are also incorporated as nodes in the initial conditions but are not depicted here)

2. The framework should be open-ended to allow sophisticated rules, powerful heuristics and advances in tableau reasoning to be readily incorporated.
3. The framework should support a layering of capabilities (*i.e.*, it should be possible to define 'helper' rules that introduce new abstractions into the tableau mechanism).
4. The framework should not, in itself, be inefficient. It should be possible to write efficient heuristics free of any fundamental limitations of the framework.
5. The framework should not impose any fixed heuristics or methods of evaluation. No heuristic is universal and so the framework should not mandate any particular heuristic or formulation of heuristics.

6. The combination of many representations, reasoning mechanisms and associated meta-data should not complicate the integration. Adding a new data-type or rule should not require the modification of all existing rules and data types.

7. The system should be modular so that the same framework can be used to reason in any problem domain and with any combination of tableau rules.

In Comirit, these challenges are addressed through the application of object-oriented design principles. The entire state, configuration and rules used by deduction in Comirit are encapsulated in objects and stored as formulas within the tableau. This has the effect of narrowing the gap between data and code, thereby simplifying the architecture while simultaneously extending its generality, improving configurability and allowing for self-modification.

In the remainder of this section, I describe the object-oriented design of the Comirit control mechanisms. This work stems from many prototypes of different architectures in an attempt to create an elegantly simple design. In an earlier publication [84], I presented a more complex architecture with a different naming scheme and class hierarchy. The version presented here functionally subsumes the earlier work, yet uses clearer naming conventions and structures that are closer to the formal definition.

Note that this design is not necessarily limited to object-oriented languages. Like many object-oriented designs, there is a straightforward translation into other languages. For example, I have implemented prototypes of Comirit in the Erlang functional programming language. I emulated object-oriented features by representing 'object instances' as ordinary Erlang tuples in which the first element of the tuple denoted the 'class name'. Similar translations are possible for other languages.

6.3.1. Object Oriented Design Principles

Comirit combines representations from multiple reasoning mechanisms in addition to meta-data that guides integration, rule selection and heuristics. Consider some of the data that may appear in a Comirit System:

- **Deductive Data Structures.** Logical terms, simulations, database-backed predicates.
- **Strategic Data Structures.** Heuristic information and scores, search queues, work queues.
- **Supportive Data Structures.** Cached data, indexes.
- **Configuration Data and Rules.** Currently active heuristics, tableau rules, simulation 'laws', meta-strategy rules.

The sheer variety of possible data types presents a data management problem. In fact, many of the particulars of data may not be known in advance at the time of implementing the framework. Even within a complete system, particular problem instances may require the manipulation of some data structures more than others. As a consequence, an object-oriented design is preferred so that data can be encapsulated in classes with simple interfaces.

Unfortunately, this data management and encapsulation problem is constrained by a need for efficiency. The system must be able to efficiently match these structures to the rules that operate over them. Data cannot be treated entirely opaquely because the system must route the data to appropriate rules.

I address the data-management and efficiency challenges with a twofold strategy:

1. Encapsulating all data and code within a single object hierarchy. All tableau state extends from a **`Formula`** class. Encapsulation ensures that the reasoner can store and manipulate the abstract terms and rules, without internal knowledge of the representation's structure.
2. A simple hierarchical key scheme is used to efficiently pair terms with matching rules. The key is a string, resembling a fully qualified file-name or URL, that summarizes the important state of an object. The string representation ensures that efficient matching can be performed with sorted indexes, while retaining the ability for the system to handle the objects opaquely.

In the following subsection, I apply these strategies to a concrete object-oriented design.

6.3.2. Object Oriented Design

Simplicity and uniformity is a driving factor in the design of Comirit. In order to separate reasoning from the core tableau algorithm, *all* state and rules are stored *within* the tableau structure. There is no external meta-data or control rules: all meta-data must be stored within the tableau itself.

Class Structure

The tableau algorithm is centered around a class named **`Tableau`**. An instance of **`Tableau`** references a collection of **`Node`** objects that, as with the formal definition of *TABLEAU*, are structured as a tree.

Each **`Node`** is annotated by a single formula of the abstract class **`Formula`**. The particular formula may be instantiated as one of **`Formula`**'s subclasses (**`Rule`**, **`Term`** or the singleton instance of the class **`Closed`**).

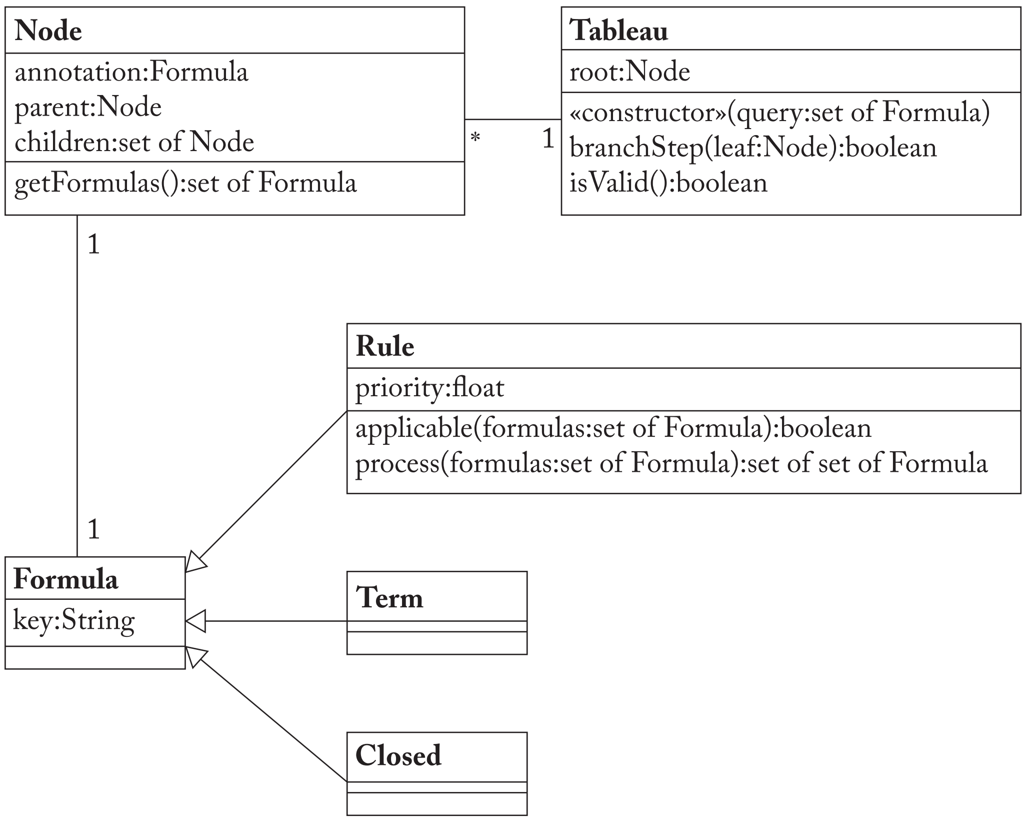

Node
annotation:Formula
parent:Node
children:set of Node
getFormulas():set of Formula
Tableau
root:Node
«constructor»(query:set of Formula)
branchStep(leaf:Node):boolean
isValid():boolean
*
1
1
1
Rule
priority:float
applicable(formulas:set of Formula):boolean
process(formulas:set of Formula):set of set of Formula
Formula
key:String
Term
Closed

Figure 6.6: Class diagram for a tableau in the Comirit Framework

Note that the class names **`Rule`** and **`Term`** should not be taken to imply that these may only be used for syntactic rules and logical terms. Terms may represent *any* kind of data (logical terms, simulations, databases, files, network connections, *etc.*) and rules may be *any* kind of computable operation that manipulates terms or other rules. Indeed, one such important rule is the Comirit Simulation which advances simulations in the tableau.

This class hierarchy is summarized in Figure 6.6.

Core Algorithm

The tableau algorithm is implemented by three methods:

1. The *constructor* of the **`Tableau`** class that takes a set of **`Formula`** and initializes a chain of **`Nodes`**. The initialized tableau therefore contains a single branch that, if read conjunctively, is equivalent to the input query.
2. A `branchStep` method of the **`Tableau`** class that performs a single step of the inner loop of the tableau algorithm to a given branch (*i.e.*, steps 2.a and 2.b of the algorithm in Section

6.2.1). The `branchStep` method extracts the annotations along the branch, selects a rule to apply and updates the branch accordingly.

3. An `isValid` method of the **`Tableau`** class that applies the outer loop of the tableau algorithm (*i.e.*, steps 1–4 of the algorithm in Section 6.2.1) until it can determine the validity of the input query. The `isValid` method operates by calling the `branchStep` method until no more work can be performed and then interprets the resulting tableau.

The constructor has a straightforward implementation as a for-loop that instantiates **`Node`**s. The `isValid` method and `branchStep` methods are detailed below.

The isValid method

The `isValid` method repeatedly selects branches and invokes the `branchStep` method (which performs a single atomic update on the branch) until no more progress can be made towards the tableau.

Since the majority of the work is performed by the `branchStep` method, the implementation of the `isValid` method is therefore relatively straightforward. Consider the following breadth-first implementation:

```
class Tableau {
   ...
   method isValid() : boolean {
      declare changed : boolean

      do {
         changed := false
         for each open branch
            changed := branchStep(branch) or changed
      } repeat until changed == false

      if number of open branches > 0
        return true
      else
        return false
   }
   ...
}
```

While the above example uses a breadth-first search, evenly dividing attention between branches of the tableau, a variety of other search strategies is possible. An obvious alternative choice is a depth first search. Indeed, this is even easier to implement than the breadth first search. More sophisticated approaches include heuristics that allocate attention according to some measure of likely benefit or a cooperative approach that combines with the `branchStep` method to apply those rules with the least cost and greatest expected reward. I have not explored these possibilities in this dissertation. However, in Chapter 7, I introduce an alternative mechanism that allows for selecting and directing attention between branches of a tableau.

The branchStep method

The `branchStep` method is responsible for performing the atomic steps of the tableau algorithm (as used by the `isValid` method). With each invocation of branchStep, the tableau selects and then executes a rule. It then updates the state of the branch to progress the tableau. Thus, there is a three stage process: selection, execution and update.

Execution and update are straightforward operations. Rules are defined in their own class, **`Rule`**. A rule is invoked on a branch and the output is used to construct new instances of **`Node`** to append to the current branch.

The selection of a rule, however, is slightly more complicated. Candidate rules must be identified by searching the branch for annotations of the type **`Rule`**, in order of their priority. Not all such rules will be appropriate for a given iteration, since they may not be able to progress the current state of the tableau. While one could simply execute the rules until a rule returns a non-empty output, it is often easier to compute whether a rule is applicable, than it is to compute the output. Thus, the **`Rule`** class has two methods: `applicable` and `process`.

A simple `branchStep` may be implemented by the following pseudocode:

```
class Tableau {
   ...
   method branchStep(branch:Node) : boolean {
      declare rules : list of Rule
      declare results : set of set of Formula
      declare formulas : set of Formula

      # collect the rules
      for each node in branch
         if node instance of Rule
            rules.append(node)

      sort rules order by Rule.priority

      # search for an applicable rule
      formulas := branch.getFormulas()
      for each rule in rules
         if rule.applicable(formulas)
            # execute it, then process the result
            results := rule.process(formulas)
            for each disjunct in results
```

```
                add a new child to the branch
                for each conjunct in disjunct
                   attach new node to child
             return true

       return false
   }
   ...
}
```

While the above naïve algorithm is effective, it is not necessarily efficient. In many circumstances the addition of a particular kind of term will effectively 'trigger' a particular rule. For example, adding the disjunctive term $a \vee b$, should 'fire' a disjunction rule exactly once and this is the only circumstance in which such a logical disjunction rule should 'fire'.

In order to optimize the matching between **`Formula`** and **`Rule`**s, the tableau builds an index of string-based keys. Every **`Formula`** exposes a key (through an attribute called `key`) and is matched to another **`Formula`** if the key of either is a prefix of the other. For example, a subclass of **`Term`**, denoting a disjunction of logical formulas, may have the key: "disjunction/logical", while a generic disjunction **`Rule`** may have a matching key "disjunction" (see Table 6.1).

The `branchStep` method can then be optimized by storing with each branch a queue of keys corresponding to formulas that have been added to the branch. The rule selection process prioritizes those rules that match keys in the queue:

Term Key	Rule Key	Match?
""	""	Yes
"disjunction/logical"	""	Yes
"disjunction/logical"	"disjunction"	Yes
"disjunction/logical"	"disjunction/logical"	Yes
"disjunction/probability"	"disjunction/logical"	No
"simulation"	"disjunction/logical"	No
"simulation"	"simulation/check_stability"	Yes
"simulation"	"simulation/run"	Yes

Table 6.1: Example matches and mismatches between keys on terms and rules. Keys match if identical, or if one is a prefix string of the other.

- The rule selection process preferentially selects the highest-priority, applicable rule that matches the *oldest* key in the queue. The nature or length of the match is irrelevant. For example, a *high priority* rule with the key “disjunction” matching against “disjunction/logical” is preferred to a *lower priority* rule with a key that precisely matches the target “disjunction/logical” (it is only the rule priority that matters).
- If no applicable rule matches the oldest key, then the oldest key is removed from the queue and the rule selection process continues with the next oldest key.
- If (or when) the queue is empty, then the highest applicable rule is used.
- If no rule is applicable, then no progress can be made towards the tableau.

This scheme has a straightforward and efficient implementation if the rules in a branch are collected into a sorted ‘working set’ (as was indeed the case in the simple `branchStep` pseudocode presented earlier in this section). For each item in the queue, the working set is searched in order of priority for a match. If the matching rule is applicable, then that rule is used, otherwise the matching rule may be dropped from the working set. This ensures that the applicability of each rule is tested only once, while enforcing the prioritization scheme of the rule selection process.

This rule selection scheme is used because it is simple, it helps ensure that formulas are processed in the same order that they are added to the tableau and it consistently applies the prioritization of rules. Other selection schemes could be used. However, it is important that rule selection be consistent and readily understood so that the behavior of tableau reasoning and of rules can be predicted by developers.

This indexing scheme offers a predictable and efficient foundation for developers to implement their own heuristics as appropriate for a given problem. The scheme does not impose any particular heuristic but provides a tool that heuristics may leverage. With careful selection of keys and rule priorities, a developer can wield fine-grained and reliable control of the rule selection process, directing rules to be applied precisely when they have the greatest impact.

Note that the use of the key-based indexing scheme or heuristics is entirely optional. The scheme merely optimizes the rule-matching process. A developer can partially or completely avoid the index by using keys formed from unique strings (which match no other non-empty key) or the empty string (that matches every key):

- If the key of a newly added formula is the empty string (“”), all rules will be tested for applicability in order.
- If the key of any rule is the empty string (“”), then those rules will be selected in order of their priority irrespective of the values in the queue.

- If the key of a rule is a unique string (*e.g.*, "rule only"), then such rules will only be tested after all other rules have been tested for matches.
- If every single key is the empty string (""), then the indexing scheme will have no effect on the rule selection algorithm (*i.e.*, the rule selection is equivalent to the simple pseudocode for `branchStep` presented earlier).

Finally, where more precise control of execution is required than is possible with keys and priorities alone, other techniques are possible. In particular, a heuristic may 'box' terms or rules into special opaque terms that serve as containers. Fine-grained control is achieved by the heuristic revealing the contents of the containers (or 'unboxing') precisely when they are needed. I will provide an example of this in Section 6.3.4.

Rules

Instances of **`Rule`** (or, rather, its subclasses) implement the rules of the tableau system and thereby direct the problem-specific behavior of Comirit. They expose a `process` method that analyzes the formulas in a branch and returns new formulas (in disjunctive normal form) for addition to the branch. If the rule is not applicable (*i.e.*, it cannot make any progress), the process formula should return an empty set.

For efficiency, each rule also exposes an `applicable` method that returns true if the rule will make progress on the tableau. This is because it is often easier to determine whether a rule will make progress, than it is to determine what the actual progress will be. The `applicable` method does not need to be implemented by every subclass. The base **`Rule`** class provides a default (inefficient) implementation that calls `process` and returns true if there is a non-empty result.

Some principles apply when developing concrete subclasses of **`Rule`**:

- Rules should be implemented such that they do not assume any particular order of application. They should be correct irrespective of their order of execution.
- Repeated application of a rule should make real progress towards the completion of a tableau or the rule should eventually stop being applicable. Wherever feasible, it should not be possible to apply a rule indefinitely and grow the tableau infinitely. For example, the disjunction rule should create new terms no more than once for each ground disjunction in the tableau and should not create infinite numbers of redundant branches through repeated application of the disjunction rule.
- Rules should ignore any formulas with types that they do not recognize or cannot process.

While rules should not depend on the order of execution, this principle may be abused in 'quick and dirty' applications. For example, a simple form of default reasoning can be implemented with unsound rules that generate plausible, non-contradictory extensions. Such rules may be set at an extremely low priority and with a unique key, so that they are only applied to an otherwise stalled tableau. Of course, this kind of reasoning may be difficult to describe formally and so important applications should use formally correct tableau rules for a default logic, rather than relying on the execution order of the prioritization scheme.

In the next two subsections, I will provide examples of implementing rules in the framework. In Section 6.3.3, I demonstrate the implementation of a simple disjunction rule and then, in Section 6.3.4, I demonstrate how powerful heuristics may be implemented.

6.3.3. Implementing Simple Rules

Criteria 1 of Section 6.3 required that *simple* rules be *simple* to implement. This is indeed the case with Comirit; a simple **`Rule`** requires only two methods, `process` and `applicable`, in addition to two inherited attributes, `key` and `priority`. In fact, where performance is no concern, these attributes may be left with their default values and implementations so that only `process` need be implemented.

To illustrate the ease of implementing a rule, consider the disjunction rule. In a formal model, the disjunction rule searches for a term of the form $A \vee B$ and returns the disjunctive normal form $\{\{A\}, \{B\}\}$.

In translating the rule to an implementation, there are a number of considerations:

- The disjunction itself is a term and should be implemented as a subclass of **`Term`**.
- The disjuction rule is a rule so it should be implemented as a subclass of **`Rule`**.
- Since a disjunction term is exclusively analyzed by the disjunction rule, the disjunction connective and its rule should be implemented with matching keys (*e.g.*, the string “disjunction”).
- A disjunction rule is of low computational complexity and is important for analyzing the problem domain but it has the effect of increasing the breadth (or fan-out) of the search space. Consequently, the disjunction rule should be given a low priority so that other rules can be invoked first.

Thus, one possible implementation of the disjunction and its rule is given by the following pseudocode:

```
class Disjunction : Term {
   declare key:String := "Disjunction"
   declare lhs:Term
   declare rhs:Term

   constructor new(left:Term, right:Term) {
      lhs = left
      rhs = right
   }
}

class DisjunctionRule : Rule {
   declare priority:float := 100
   declare key:String := "Disjunction"
   method process(formulas:set of Formula):set of set of Formula {
      declare disj:Disjunction
      # look for disjunctions
      for each node in branch
         if node.formula instance of Disjunction
            disj := node.formula
            # check they haven't already been expanded
            if not branch.contains(disj.lhs)
               and not branch.contains(disj.rhs)
                  # and expand...
                  return {{formula.lhs},{formula.rhs}}
      return {}
   }
}
```

This implementation is not complex and so it is clear that simple tableau systems can be readily implemented in the framework. The full set of rules for classical propositional logic can be implemented in a matter of minutes or, at worst, hours.

Basic forms of integration are readily implemented in the architecture as simple rules. Database-driven predicates, 'abbreviated' expressions and also sensor-driven values can be implemented as simple `Rule`s that invoke external routines (*e.g.*, an ODBC database call or a sensor reading) to retrieve expanded forms of terms for addition to the tableau. Their priority can be set proportional to the expected time or resource cost for accessing the external resource (*e.g.*, a file may be priority 50, a database may be given priority 80, an external website may be given priority 150, a 3rd-party commercial web-service may have priority 1000 and interaction with a human user may have an extremely low priority such as 10000).

6.3.4. Meta-Rules and Cost-based Heuristics

The object-oriented design raises the possibility of implementing powerful meta-rules and heuristics. While tableau rules are typically implemented to inspect and transform particular **`Term`**s, this is not an architectural constraint. Rules and terms are both represented uniformly as subclasses of **`Formula`** in the tableau so that either may be transformed or encapsulated within other rules or terms. Thus, meta-rules and heuristics are implemented like any other rule in a tableau; they operate alongside standard tableau rules but the difference lies in the formulas that they transform and the manner in which they operate.

In the following two subsections, I discuss two approaches to implementing heuristics: rule encapsulation and data encapsulation.

Note that in an earlier publication [85], I prescribed a particular scheme for implementing meta-rules and heuristics. The more general version of the tableau algorithm that I present in this dissertation is not only vastly simpler but also allows for more diverse meta-rules and heuristics. The capabilities of meta-rules as described in this dissertation are a super-set of the capabilities of those previously described.

Rule Encapsulation

The functionality and sophistication of simple rules can be extended by encapsulating the rules in other meta-rules. Such meta-rules intercept, filter and transform the inputs and outputs of the encapsulated rules to create a layering of functionality and abstractions.

Such meta-rules are implemented by a **`Rule`** that encapsulates, using a hidden attribute, some other target **`Rule`** (or set of **`Rule`**s). When the meta-rule is invoked, it may transform its input, invoke the encapsulated rule (or set of rules) and then transform its output.

Consider some scenarios:

1. A meta-rule may prevent an encapsulated rule from being applied more than once to a given formula in a branch. This meta-rule would simplify the development of basic rules by automatically preventing cyclic rule applications.
2. A meta-rule could deliver formulas to an encapsulated rule one-by-one, eliminating the need for the encapsulated rule to search the branch.
3. A meta-rule can detect when the output of an encapsulated rule would introduce redundant terms and then filter out those redundancies.

4. A meta-rule may memoize the inputs (or subsets of inputs) and the corresponding outputs of a complex encapsulated rule. This would speed up reasoning where identical subproblems occur several times across a tableau.

5. A meta-rule may encapsulate a set of substitutable rules. It can then apply heuristics to analyze the context, performance needs or expected costs to select the best rule to apply in a given situation. For example, a meta-rule may choose between two sets of rules, one for qualitative simulations and the other for numerical simulations, depending on the accuracy needs.

Meta-rules may be chained to provide a layering of abstractions. For example, a disjunction rule could be naïvely implemented without checks to prevent cyclic rule application. Greater sophistication may be added later by encapsulating the rule in layers that prevent repeated application, that filter out duplicate terms and that deliver each formula one-by-one. This kind of layering or chaining can have a dramatic impact on the efficiency of a tableau and its ease of implementation. A set of extremely unsophisticated tableau rules can be made highly efficient by layering capabilities of memoization, duplicate-removal and heuristic-driven rule selection.

The following pseudocode illustrates one such meta-rule. It encapsulates a single, arbitrary rule and ensures that any redundant effects are filtered out of the output.

```
class RemoveDuplicates : Rule {
   declare encapsulated:Rule
   declare priority:float
   declare key:String

   constructor new(target:Rule) {
      encapsulated = target
      priority = target.priority
      key = target.key
   }

   method process(formulas:set of Formula):set of set of Formula {
      declare results:set of set of Formula
      results := encapsulated.process(formulas)
      for each disjunct in results
         for each conjunct in disjunct
            if conjunct in formulas
               disjunct.remove(conjunct)
      return results
   }
}
```

Data and Rule Hiding

Another way that meta-rules may influence the behavior of a tableau system is by masking rules and terms from other rules in the tableau. Data is masked using 'wrapper' classes that extend **`Term`** but have hidden or private attributes that store an underlying value.

When a **`Rule`** is encapsulated within a **`Term`**, it will not be executed by the `branchStep` method (since **`Term`** does not extend **`Rule`**) and is therefore rendered inoperative. Similarly, because each rule is required to ignore subclasses of **`Formula`** that it does not recognize, if a term or rule has been encapsulated within some other subclass of **`Term`**, then it will not be seen by other rules.

Consider the following usage scenarios:

1. A rule may be encapsulated in an opaque wrapper for delayed release to the tableau algorithm. For example, overly-optimistic and computationally expensive rules may be hidden in opaque wrappers until an 'unhiding' rule has determined that all other reasoning options are exhausted.
2. Values may be encapsulated in an opaque wrapper for delayed release to other rules. For example, default beliefs or speculative knowledge may be encapsulated until it is determined that known factual truth is insufficient for deciding an appropriate action to perform.
3. Potentially contradictory knowledge such as unconfirmed beliefs, speculation, counter-factual reasoning and fiction can be encapsulated in wrappers that hide values from immediate manipulation (and inconsistency checking). Meta-rules may reason with this knowledge, only unwrapping when they have been confirmed to be correct.
4. Like encapsulation, costly rules may be encapsulated in wrappers so that a meta-rule can estimate the cost of each wrapped rule and 'unwrap' the rule with the smallest predicted cost.

In fact, this approach to wrapping and unwrapping knowledge can be used to readily transform any logic to a simple modal logic. Modalities of terms can be represented using modal wrapper classes. Meta-rules can encapsulate standard deduction rules, 'lifting' the existing non-modal rules into each modality to perform specialized reasoning and re-encapsulating the output of the lifted rules. This kind of transformation may offer a stop-gap or light-weight solution where modal reasoning is required, while a proper modal logic is being developed. Even if an effective modal logic is available, this kind of hiding and lifting may be used to transform non-modal rules to be compatible with a modal logic.

The following pseudocode demonstrates one such example of data hiding by meta-rules. The example code is used to delay release of knowledge into a tableau. Instances of **`DelayedTerm`** are used to hide knowledge and are later 'unwrapped' when the low-priority **`UnboxerRule`** is eventually executed.

```
class DelayedTerm : Term {
   declare key:String := “delayed”    # term unique string
   declare encapsulated:Term

   constructor new(target:Term) {
      encapsulated = target
   }
}

class UnboxerRule : Rule {
   declare priority:float := 10000    # very low priority
   declare key:String := “unboxer”    # rule unique string

   method process(formulas:set of Formula):set of set of Formula {
      declare results:set of Formula
      declare delayed:DelayedTerm

      # collect the contents of DelayedTerms (in formulas)
      # into the results set:
      for each formula in formulas
         if formula instance of DelayedTerm
            delayed := formula
            if not delayed.encapsulated in formulas
               results.add(formula)
      return {results}
   }
}
```

6.4. Discussion

In Chapter 8, I will demonstrate the functional claims that the architecture successfully combines simulation and logic. Here, I address the non-functional objectives, discuss how the iconic/symbolic divide may be bridged and explain how framework can be used as a query language.

6.4.1. Design Objectives

The design of the Comirit control strategy satisfies the non-functional objectives that I outlined in Section 6.3:

1. **Simple tableau rules should be simple to implement**

   The generic tableau algorithm is simple, it places few demands on the rule developer and it is closely matched to the behavior of a tableau. I demonstrated the ease of implementing simple rules in Section 6.3.3.

2. **The architecture should be open-ended to allow sophisticated rules, powerful heuristics and advances in tableau reasoning**

   In Section 6.3.4, I demonstrated how sophisticated meta-rules and heuristics can be built into the system. This capability means that the mechanisms of the framework are not fixed and are entirely configurable. Specialized optimizations can be incorporated into the tableau algorithm by implementing them as additional rules that are stored and processed within a tableau like any ordinary rule.

3. **The architecture should support a layering of capabilities**

   The encapsulation of rules, demonstrated in Section 6.3.4, allows for a rich and sophisticated layering of capabilities.

4. **The architecture should not, in itself, be inefficient**

   The key-based matching of new `Formula`s to `Rule`s and the prioritization scheme ensures that the architecture is not, in itself, inefficient. Given appropriately chosen keys, indexes efficiently 'route' formulas to matching rules.

5. **The architecture should not impose any fixed heuristics or method of evaluation**

   While a previously published version of this work did impose constraints on implementing heuristics, the wide range of schemes described in Section 6.3 demonstrate that the current architecture supports many methods of evaluation and many kinds of heuristics. Furthermore, the key-based matching scheme is a generic mechanism that ensures such heuristics can be efficiently implemented.

6. **Adding a new data-type or rule should not require the modification of all existing rules and data types**

   The object-oriented design ensures that new rules can be added orthogonally to existing rules. The tableau algorithm does not need to inspect the contents of rules or formulas; it simply stores them as opaque, encapsulated objects. Individual rules must ignore formulas that they do not recognize, thus allowing for new types and mechanisms to be added to a tableau, without requiring changes to existing rules.

7. **The system should be modular**

   The generic tableau algorithm makes no assumptions about the rules or data that it contains. Any method of reasoning that can be unified into a tableau framework can be implemented as a set of rules and used by an implementation of Comirit.

6.4.2. Bridging the Iconic/Symbolic Divide

This generalization of the method of analytic tableaux solves the problem of integrating the *mechanisms* of simulation and logical reasoning. However it does not address the need for *coordinating the semantics* of simulation and logic. Logical terms should lead to the instantiation of appropriate simulations and *vice versa*.

For example, a simulation of a mug of coffee constrained by $\neg$*mess* requires a translation between symbols and simulation. In the example given in Section 6.2.2, the simplified simulation generated these symbols directly. However, with graph-based simulations, the hybrid reasoner requires a mechanism for testing whether $\neg$*mess* holds in simulation or, conversely, it requires a mechanism for automatically generating the *mess* symbol when a mess occurs in a simulation.

In prototypes of Comirit, I have developed these translations on an ad-hoc basis, implementing them as rules as they are required. This is not as difficult as one would expect; the interface between the mechanisms is very small on the problems that I have tested.

There is good reason to expect that a very rich commonsense reasoning system would only require a small number of low-level mapping functions. While human languages include rich vocabularies to describe objects in the real world, most terms do not serve as low-level primitives but are based on analogy and metaphor. One may use abstract primitives to describe an object as being *rounded* or *smooth* but details are generally conveyed by analogies, such as *egg-shaped* and *glassy*. The Region Connection Calculus (RCC8) [144] is an excellent illustration of the principle that just a few abstract primitives can have a large descriptive power; RCC8 defines just eight relations concerning the connectivity of objects (all derived from a single primitive) but still makes fine distinctions that are rarely important in everyday speech[*]. Where greater precision is required in language or communication, diagrams and gesture are typically involved.

[*]Samuel Wintermute (University of Michigan) suggested this analogy with RCC8 to me during an informal discussion.

The development of rich and general purpose translations between symbols and simulations remains as future work. However, I expect that only a small number of abstract primitives will be required. Examples of many of these are as follows:

- **Shape:** round, square, thin, long, wide, fat
- **Texture:** smooth, rough, bumpy, dimpled, solid, liquid
- **Appearance:** red, green, blue (and other simple colors), shiny, matte, textured
- **Position and orientation:** above, below, near, beside, front, behind, touching, inside

More complex relationships would be defined as complex combinations of these primitives. For example, *mess* might be defined as being liquids (and small solids) not being inside a container (*i.e.*, some kind of spillage and dirtiness) and larger objects not sharing common orientations (*i.e.*, not being neatly tidied).

On top of a kernel of primitives, a complete system would require a catalog of known objects and an ability to measure similarity. A catalog would allow direct instantiation and description of specific objects. For example, a mug of coffee can be recalled from the catalog when its symbol is encountered. Conversely, an observation may be compared to mugs stored in the catalog to translate the observation into the symbol 'mug'. Sophisticated analogies may be understood by using loose similarity measures, as in 'the cushion was shaped like an egg but had the texture of packing foam'.

In general, it is easier to translate from simulation to symbols and, in circumstances where simulations are cued entirely by external observation, this may be sufficient. However, any translation from symbols to simulations will eventually need to deal with the uncertainty and vagueness of the symbols that are used in human languages. That is, while situations that are prototypical or stereotypical lend themselves to straightforward reasoning from the prototypes of symbolic primitives, there are many situations in which symbols do not have precise mappings to simulations.

As it stands, the Comirit Framework can accommodate search-based approaches to uncertainty and vagueness. This can be implemented by rules that do not simply 'expand' a single definition for a symbol but, instead, sample a parameter space. In this way, the uncertainty of a symbol can be systematically and analyzed and, likewise, 'precisifications' [12] can be generated for vague symbols. For example, a statement such as 'the mug is on a table' can instantiate a number of different simulations arising from uncertainty (different parts of the table) and from vagueness (different kinds of tables). Short simulations can be executed to check the plausibility of generated interpretations. For example, unless there is reason to suspect otherwise, static and stable configurations are preferable so that it is unlikely that the simple statement that 'the mug is on a table' would refer to a mug on its side with liquid pouring out or balancing on its edge in an unstable equilibrium. Of course, uncertainty and vagueness might also be incorporated into Comirit by way of rules that implement logics of uncertainty and vagueness (e.g., [86, 11]).

Finally, in embodied applications of commonsense, the observed world can itself help bridge, or even avoid, the symbol/simulation divide. An embodied robot that acts without symbolic communication may make use of logic to plan complex actions but it can use simulation as a hypothetical or virtual world for experimentation. That is, it uses real-world and virtual-world embodiment but does not require or use explicit mechanisms for interpreting the virtual-world. It simply acts in the world (and its simulated analogs) and senses the response.

In Section 8.1.5, I will furthermore show how some concepts, learned in simulation, may be translated into symbols by representing the learned concept as a vector or address and using the vector as proxy for a symbol.

6.4.3. Tableau-based Queries

In the illustrative example of Section 6.2.2, the remaining open branch of the tableau described a world in which an appropriate action was performed to satisfy the input formula. In many cases, the goal of commonsense reasoning is not to determine whether or not a formula is satisfiable but to identify the particular model that satisfies the query.

Satisfying models can be read by direct inspection of the open branches of a completed tableau. Each open branch represents a satisfying model, whose description can be read as the conjunction of formulas annotated to the branch. If satisfying models incorporate some action (*e.g.*, a *low-force* action appearing in an open branch), then an agent can determine its necessary action by searching the open branches of a tableau for terms of the appropriate type.

In situations where an agent or robot seeks to achieve a goal through action, it is often not necessary to know the particular satisfying model but only an action that is necessary for success. Indeed, if it is known that the selection of a correct action is more important than completely proving the satisfiability (or otherwise) of an input query, then this knowledge may be used to speed up reasoning. The tableau may be able to determine which particular action is necessary for a query to be satisfied, even if it is not possible to determine if the entire query is itself satisfiable (*i.e.*, it may be faster to prove necessity than sufficiency).

Such action-specific querying can be implemented by including variables and unification rules into the tableau. In this approach, terms with unbound variables are added to the tableau and the tableau algorithm is modified to stop when it finds consistent variable bindings.

Syntactically, satisfiability queries can be converted to variable-based queries by expressing the variables in the right-hand side of an implication. For example, given a wff, $F(x)$ that constrains the value of x, the legal values of x are discovered by posing a query $F(x) \Rightarrow x = \mathtt{X}$, where $\mathtt{X}$ is an unground Prolog-style unification variable. When this formula is negated and converted into negation normal form, the tableau will contain the query term $\neg(x = \mathtt{X})$. A contradiction rule will then allow the branch to be closed by the unification of $\mathtt{X}$ with the true assignment for x (*i.e.*, if the tableau contains $x = 3$ and $\neg(x = \mathtt{X})$, a contradiction rule can bind $\mathtt{X}$ to the value 3).

Care must be taken when using a unification rule since it does not add new consequences to a tableau but has the effect of restricting the meaning of a tableau. A poorly chosen application of a unification rule can therefore have the effect of making the algorithm incomplete (though it will not affect correctness).

Fortunately, many problems have a structure in which the query $F(x)$ always stalls with, at most, one open branch. In this case, if the unification rule is set as the lowest-priority rule in the system, it will appropriately unify with the correct result of the query, limiting neither correctness nor completeness.

However, this is not the case in general. In some pathological queries, there is a potentially infinite chain of rule applications so that it is not possible to set a unification rule as the lowest-possible rule (doing so would prevent it from ever being applied). This may be solved by clever heuristics (as is typically the case with theorem proving) and by extending the Comirit Framework to allow backtracking. An efficient general solution to this remains an open problem in tableau reasoning and so exploration of these ideas remains as future work.

6.5. Conclusion

Hybrid reasoning is a powerful method for multiplying the capabilities of a collection of underlying mechanisms. Even in the worst and most pathological cases, the benefits of hybridization will be additive. In problems with richer structures with a variety of sub-problems, hybridization can have a multiplicative effect in which mechanisms cooperate to solve sub-problems. In the case of simulation and logic, simulation can perform rich reasoning about situations and environments (such as the physical world) that are extremely difficult to express in logic, while hybridization with logic adds flexibility and deductive power.

Tableau reasoning has historically been seen as a framework for representing a wide range of logical reasoning mechanisms. An important contribution of the Comirit Framework is the insight that tableau reasoning can be further generalized to support not only logical reasoning but also *non-logical* reasoning such as simulation. Along with this insight, I have offered a careful object oriented-design that simplifies the process of implementing such reasoning systems on real computer systems.

Hybrid reasoning in Comirit is therefore not an ad-hoc integration of logic and simulation, but an elegant, open-ended, general purpose and modular software architecture. It is not only easy to use but also supports the implementation of powerful heuristics and abstractions.

Chapter 7
Learning and Control

In Chapters 5 and 6, I described mechanisms for expressive and efficient commonsense reasoning. These mechanisms were inspired by the relative ease of constructing sophisticated simulations for computer games. If most knowledge in a commonsense reasoning system can be expressed in simulation, then the effort expended in logic-based knowledge engineering is dramatically reduced.

Nevertheless, 'rapid knowledge formation' in either simulation or traditional logics has a non-zero cost. Even if minimal effort is involved in encoding a single fact of commonsense, the vast breadth and depth of commonsense knowledge is such that the accumulated cost of building a complete knowledge-base can be significant. This cost is further compounded by a changing world in which 'commonsense' is itself in a constant state of flux and therefore subject to revision.

An alternative to the expensive process of manual knowledge engineering is the construction of systems that can learn for themselves. Each fact of commonsense knowledge that a system can autonomously discover by itself is one less fact that needs to be manually encoded. Furthermore, a system that can automatically revise its knowledge as it encounters change and new situations will be far more capable of adapting to a changing, complex world.

In this chapter, I describe how cumulative inductive learning is integrated into the Comirit Framework. Learning is performed over an hypothesis space of predictive models; the system continuously observes the environment, matches expectations against observations and refines its predictive models. These models are then used for planning actions, anticipating future events and predicting the effects of potential actions. That is, the system *learns* models and then *reasons* about action: the system does not perform shallow reinforcement learning.

The purpose of this chapter is twofold. In it, I principally seek to integrate learning capabilities into Comirit. However, a secondary purpose of this integration is to demonstrate the ease by which the Comirit Framework can be extended.

I begin the chapter with a conceptual review of machine learning and machine learning algorithms. I then explain how machine learning may be used in the Comirit Framework and how it may be implemented. I conclude with a discussion of future directions and a summary of the chapter. A functional evaluation of the framework on a physical learning problem appears in Chapter 8.

Note, finally, that this work is not intended to rule out the possibility of manual knowledge engineering. Even if the costs of manually constructing a commonsense knowledge-base are substantial, they should be considered relative to the benefit gained from potential applications (see Section 1.2). With suitable tool support for engineering knowledge, the benefits and cost savings generated by artificial commonsense may far exceed the cost of engineering and maintaining a suitable knowledge-base. Nevertheless, even if a system is predominantly hand-engineered, a partial use of autonomous learning will leverage the efforts of engineers to dramatically decrease the cost of construction, while improving accuracy and adaptability.

7.1. Inspiration

To understand this work, it may be useful to understand the context in which it was developed.

After developing the iconic representation and hybrid reasoning schemes (of Chapters 5 and 6), I began exploring ways in which simulations could be more rapidly built. I had been building simulations in a two stage process:

1. I first initialized a graph-based structure from careful geometric calculations in a spreadsheet and by adapting 3D models of chemicals. For example, for an early 3D model of a chicken egg, I deformed a spherical model of a C_{60} Buckminsterfullerene molecule that I found on a chemistry web-site.
2. I then set the attributes of the graph-based structure by trial-and-error. I manually edited configuration files and ran manual experiments until simulations sufficiently resembled reality. For example, if a rubber ball seemed too rigid, I reduced the spring constant in its 'bonds'.

It was clear that the effort behind these tasks could be vastly improved with appropriate tools. Both of these tasks required little abstract thought; they involved a process of visual comparison and iterative fine-tuning. While powerful tool support is beyond the scope of this project, I considered simple tools in the hope of reducing the time that I spent manually adjusting numerical values in configuration files.

In investigating possible tool support, it became clear that the tedious and intellectually undemanding nature of the task meant that suitable tools could so dramatically reduce the effort in building simulations that the work could be fully automated:

1. The underlying graph-based structure can often be directly observed from the environment. In the case of physical simulations, a wire-frame approximation can be automatically assembled from the 3D volumes reconstructed from moving camera images, stereoscopic vision, LASER triangulation, LIDAR, time-of-flight cameras, direct physical contact or other scanning/imaging techniques.
2. The annotations that guide the dynamic behavior of simulations are typically numeric so that their values may be learned with standard machine learning techniques (*i.e.*, by search for numerical attributes that minimize the error between simulation and observed reality).

For example, consider a household robot that uses simulation as an underlying representation and encounters an object it has not encountered before (*e.g.*, a mug of coffee). The robot can build a wire-frame approximation by a robust 3D shape reconstruction algorithm and it can then gently interact with the object and use machine learning techniques to annotate its model with simulation parameters that match observed behavior. This learning process can be performed continuously: every observation provides additional detail to assist 3D reconstruction and provides additional examples for machine learning.

While this does not achieve every possible kind of learning—it neither learns the fundamental laws of simulation nor does it 'learn how to learn'—an ability to automatically acquire practical knowledge of new objects can vastly reduce the cost of large-scale knowledge engineering. A household robot could be provided with simple mechanical knowledge and a logical description of its goals and strategies but be left to its own devices to model and understand the multitude of possible objects in a home.

Learning the laws of a simulation poses a greater challenge. Fortunately, they are relatively easy to implement manually and rarely change. For example, once the laws of Newtonian physics have been implemented, they can be used in simulations of almost all simple mechanical systems. Nevertheless, it may eventually be possible to automatically learn update functions through inductive programming or by treating update functions as a parameter search for a non-linear function (*e.g.*, using a neural network or genetic programming). Indeed, there has been substantial work in the area of autonomously learning laws of physics and other scientific knowledge [37, 97, 172]. For the purposes of this dissertation, I will assume a fixed set of manually implemented laws.

7.2. Models of Machine Learning

The work described in this chapter does not represent a significant advance in the raw algorithms of machine learning but it is a novel application and a new approach to integration. Thus, throughout this chapter, I will only consider simple learning algorithms and standard machine learning techniques. There is, nevertheless, scope for applying the vast literature on state-of-the-art machine learning to this problem domain.

In this dissertation, I use a 'textbook' model of machine learning, so that any modern algorithm may be applied to the problem:

I view learning in the Comirit Framework as a mechanism of iterative inductive learning. The agent observes real world behaviors or transitions and collects these observations as mappings in a set *f*. The learning problem is then to find a function (or set of functions), *h*, that best approximates *f*. This is an iterative process because, with each instant in time, the system collects new observations that enrich the set of observations, *f*, and allow the hypothesis, *h*, to be refined.

In the case of a system like Comirit, the mechanics of the function *h* are fixed by the implemented laws of the simulation. Search for a good hypothesis, *h*, is therefore reduced to the problem of finding a set of numerical simulation parameters. This parameter search can be implemented by an *iterative generate-and-test* algorithm: a process that involves sampling the hypothesis space of parameter values and further refining those hypotheses that perform the best (*e.g.*, a stochastic greedy search or a beam search).

Any iterative learning algorithm that may be described in terms of search over an hypothesis space can be incorporated into this framework. In this dissertation, I use a simple greedy search but there are many alternatives, including (though not limited to) kernel methods, neural and auto-associative networks, Bayesian networks, genetic programming and boosting.

7.3. Concept

Unfortunately, there is a slight conceptual mismatch between the textbook model of learning (in which learning is an iterative generate-and-test process) and tableau based reasoning.

The generation of a set of hypotheses is akin to a disjunction rule; it proposes a set of alternatives that may model the world. If hypothesis generation is thereby implemented in a tableau as a branching operation, then a problem arises when attempting to test the hypotheses and select the best one. A branch of a tableau can only be closed on its own merits because rules are not able to reference the state of other branches. In contrast, in machine learning, there is often no hypothesis that is either completely consistent or completely inconsistent with observations. Hypotheses need to be compared with each other on a relative basis, rather then entirely on their own terms.

In Comirit, the mismatch is addressed by externalizing the selection of the best hypotheses, by extending the tableau algorithm so that it searches not for consistent branches but for *minimal and consistent* branches.

In order to provide a single uniform mechanism and to expose branch selection to the same heuristics, rules and transformations as any other data in a tableau, the tableau branch ordering is defined using symbols that are stored *within* the tableau. New terms are added to the tableau that are assumed to be tautologically true in any logical context but are evaluated externally to determine the minimality of a branch.

These terms may be created using the rank constructor:

$$rank : \mathbb{N} \times \mathbb{R} \rightarrowtail TERM$$

Branches *may* contain $rank(index, value)$ terms which pair a Natural number *index* with a Real number *value*. A given branch must not have any more than one *rank* for a given *index*. The *rank* terms must use *index* values sequentially and without gaps (that is, in every branch, *index*es must increment 1, 2, 3, 4, ...).

The $rank(index, value)$ terms in a branch induce a partial order $\leqslant_{\text{branch}}$ (defined using an auxiliary function, *ranks*, that extracts the *index*es and *value*s of rank terms from a branch):

$$\_ \leqslant_{\text{branch}} \_ : BRANCH \leftrightarrow BRANCH$$

$$ranks : BRANCH \to (\mathbb{N} \rightarrow\!\!\!\!\!+ \mathbb{R})$$

$$\forall\, B : BRANCH \bullet$$

$$\quad ranks(B) = \{R : \mathbb{N} \times \mathbb{R} \mid rank(R.1, R.2) \in annotations(B)\}$$

$$\forall\, A, B : BRANCH \bullet$$

$$\quad A \leqslant_{\text{branch}} B \Leftrightarrow$$

$$\quad\quad \mathbf{let}\ ra == ranks(A);\ rb == ranks(B);\ shared == (\mathrm{dom}\ ra \cap \mathrm{dom}\ rb) \bullet$$

$$\quad\quad\quad ra = rb$$

$$\quad\quad\quad \vee\ \exists\, n : 1..(\#shared-1) \bullet (1..n \lhd ra) = (1..n \lhd rb) \wedge ra(n+1) < rb(n+1)$$

◊ "The function *ranks* returns a functional set of (*index*, *value*) pairs corresponding to those *ranks* found in the annotations of the branch."

◊ "Branch A is less than or equal to ($\leqslant_{\text{branch}}$) branch B if and only if the rank terms in the branches are identical or, among the *rank* terms that overlap, there is a sequence of *index*es with identical *value*s, followed by an *index* whose *value* is strictly smaller for A."

The partial order over branches behaves much like the ordering of sections in a book. In the same way that Section 12.1.1 of a book occurs after Section 9.6.3 but before Section 12.1.2, branches in a tableau are compared by examining the *value* assigned to the smallest *index* number. If the *value*s for the smallest *index* are equal, then the *value*s corresponding to the next-smallest *index* are compared, and so on.

The branches form a *partial* order because it is possible that the ranks of a branch are still being computed. In this case, the definition for $\leqslant_{\text{branch}}$ leaves such otherwise minimal branches incomparable with those branches that are more complete. For example, returning to the analogy of book sections, this is similar to the way that Section 6 occurs after Section 5.2.4 but Section 5 and its subsection Section 5.2.4 are incomparable.

Finally, having now an ordering over branches, the notion of a branch of a tableau being 'open' can be refined. A branch is no longer considered 'open' by default, simply if it is consistent. It is open if it satisfies both of the following conditions:

1. It is consistent (*i.e.*, the '*closed*' token has not been generated in the branch).
2. There is no other consistent branch that compares to be strictly less than it.

A branch that is neither 'closed' (inconsistent) nor 'open' (minimally consistent) is said to be 'weakly closed'.

Formally, a branch B is an open branch of tableau T is open if B *openbranchof* T holds:

$$\_\ \mathit{openbranchof}\ \_ : \mathit{BRANCH} \leftrightarrow \mathit{TABLEAU}$$

$$\begin{aligned}
&\forall\, B : \mathit{BRANCH};\ T : \mathit{TABLEAU} \bullet\\
&\quad B\ \mathit{openbranchof}\ T \Leftrightarrow\\
&\qquad B\ \mathit{branchof}\ T\ \land\ \neg \mathit{isclosed}(B)\\
&\qquad \land \neg\exists\, C : \mathit{BRANCH} \bullet C\ \mathit{branchof}\ T \land\ \neg \mathit{isclosed}(C) \land \left(B \leqslant C \land \neg(C \leqslant B)\right)
\end{aligned}$$

◊ "Branch B is an open branch of the tableau T if and only if it is a branch of T and there is no other branch of T that is not closed and that compares strictly less than B"

Note that this formulation means that it is possible to have several unequal branches open in a tableau (due to the ordering relation being partial) and that if there are any branches that are not closed, there must be at least one open branch (*i.e.*, it is not possible for all branches to be 'weakly closed').

Furthermore, note that the formulation of *openbranchof* allows a weakly closed branch to be reopened as the tableau evolves. If a minimal open branch is later found to be inconsistent (*i.e.*, it generates a closed token), this will cause the next most minimal branch to become open again after having being 'weakly closed'.

The tableau reasoning algorithm (Section 6.2.1) does not need to be modified: the original definition of an open branch (*i.e.*, ¬*isclosed*) in the original algorithm is simply replaced by the *openbranchof* relation defined above. The mechanics of the algorithm remains identical.

This change to the definition of 'open', in effect, transforms the tableau algorithm into a greedy search—the algorithm now selects minimal branches where possible. Other definitions of 'open' are also possible. For example, a beam search can be implemented by modifying the definition of *openbranchof* so that it selects the minimal *N* branches, where *N* is the beam size. Such variations remain as future work but are straightforward to apply.

Thus, this simple extension to the Comirit Framework opens the possibility of selecting between branches and provides a foundation for implementing a range of learning and attention-directing mechanisms. Only the '*closed*' token permanently closes a branch of the tableau, so this extension does not alter the correctness of the underlying tableau algorithm—it merely prioritizes the search algorithm towards minimal branches.

7.4. Example

To illustrate how the ordering of branches in the tableau can be used for implementing learning in Comirit, consider again a household robot tasked with cleaning a house. The robot will need to explore its environment and tidy any objects that appear to be out of place. Given the diverse range of objects found in houses, from artistic objects to home-made items and new technologies, it is inevitable that the robot will encounter objects that it has never encountered before and for which its programmers never prepared it.

To reason about an unknown object, the robot builds a simulation which it will use to determine safe and effective methods of interaction. For example, the robot may use simulation to reason that it must move a mug full of coffee slowly and carefully because the mug is brittle and its contents could spill and pose a danger.

The robot must therefore observe the object so that it can learn a model of its behavior. It may do this by applying its most careful and conservative actions to conduct initial experiments or it may enlist the assistance of another agent or designated 'teacher' to demonstrate how to safely handle the object. If it learns that the object is robust and stable, it can then handle the object quickly and use it to support other objects. Conversely, if it learns that the object is heavy or delicate, it will reason through simulation that it is better to use slow and deliberate action to handle the object.

In this setting, learning may be implemented as a generate-and-test process in the Comirit learning framework. On encountering a new object, the robot assumes a conservative hypothesis about the object's behavior. With each observation, it generates permutations of the current hypothesis, and commits to the new hypothesis that most closely matches observations to date.

If hypothesis generation is treated as a disjunctive rule that creates new branches, then each branch of the tableau may be interpreted as a world in which the agent believes a certain hypothesis. The tableau's preference for minimally ordered branches is a method of selecting those worlds in which the agent's beliefs prove most effective in predicting observations.

That is, the robot follows the following algorithm:

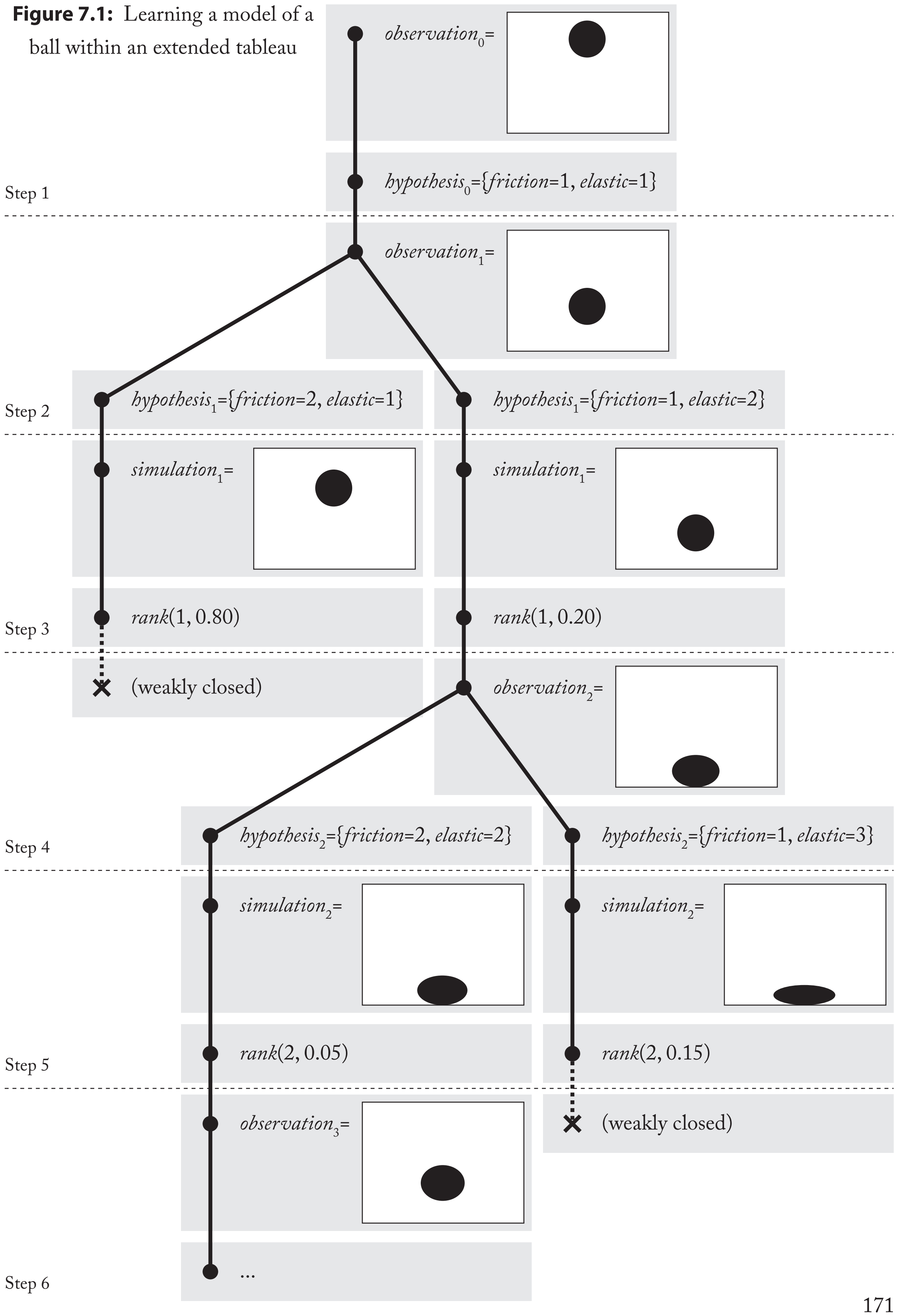
1. Initially, a default hypothesis about the values of annotations is assigned to the novel object.
2. Whenever the robot makes a new observation, it generates a set of alternate hypotheses (random perturbations, in the case of a stochastic hill-climbing strategy) as a disjunction in the tableau. This disjunction produces new branches in the tableau.
3. The alternate hypotheses are each simulated from the prior observation in order to generate predictions for the current observation. The error between this expectation and observed reality is computed and these are stored in the tableau as *rank*(*index*, *value*) terms.
4. The best hypothesis is implicitly chosen by the tableau, due to its preference for minimally ordered branches. The algorithm continues again with Step 2.

Note also that because the tableau can contain logic, simulations and functions, the system may use logical constraints or ad-hoc 'helper functions', even when learning. For example, a constraint such as $mass > 0$ can be used to eliminate a hypothesis without the need to test it against observations.

So, if the household robot were to observe a falling ball, it might automatically build a simulation according to the hypothesis search in the tableau of Figure 7.1:

- **Step 1.** The tableau initially contains the first observation of a ball and the initial hypothesis generated (many other control objects, meshes, functions and other data will be in the tableau but these are not shown for simplicity).
- **Step 2.** The system observes movement in the ball. It generates new hypotheses, seeking to find an hypothesis with minimal error.

Figure 7.1: Learning a model of a ball within an extended tableau



- **Step 3.** The system simulates each hypothesis. The result of the simulation is compared with observations to determine the error in the hypothesis. The right branch has smaller error so the left branch is 'weakly closed'.
- **Step 4.** As with Step 2, the system observes more movement and generates new hypotheses, further refining the current hypothesis.
- **Step 5.** The system then simulates, as with Step 3, but this time the left branch is minimal.
- **Step 6 and later.** The algorithm continues yet again with more new observations and further hypothesizing.

7.5. Implementation

As with extending the formal definition of tableau reasoning, it is an uncomplicated process to extend the concrete implementation of Comirit to support learning. The extensions require minimal change and machine learning algorithms are readily adapted into the framework.

7.5.1. Framework Implementation

The extensions formally defined in Section 7.3 have a direct translation into concrete implementation.

A new class, **`Rank`**, extending **`Term`**, must be defined to represent the *rank*(*index*, *value*) terms. This is depicted in Figure 7.2.

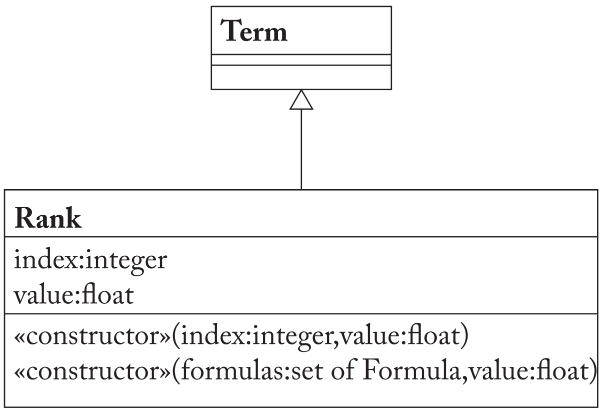

Term
Rank
index:integer
value:float
«constructor»(index:integer,value:float)
«constructor»(formulas:set of Formula,value:float)

Figure 7.2: Class diagram for the tableau-based learning mechanisms of the Comirit Framework.

The definition of **`Rank`** has a number of properties to ease the development process:

- **`Rank`** has a secondary constructor that accepts a set of formulas. This constructor automatically computes an appropriate *index* based on the largest rank *index* already present in the set of formulas.
- **`Rank`** is a subclass of Term and, as such, may be used like any other **`Term`** in the tableau. A **`Rank`** can be instantiated by any rule (provided the *index* is generated in the correct sequence) and instances of the class are stored in the tableau like any other formula.
- Since **`Rank`** is an ordinary class like any other **`Term`**, it may also be sub-classed. This may be done, for example, to incorporate additional annotations that trace the rank back to the 'reason' that it was created (*e.g.*, to the hypotheses and observations that it is associated with).

In addition to the **`Rank`** class, the `branchStep` method described in Section 6.3.2 must be updated to use the new definition of 'open'. This involves only the translation of *openbranchof* from its formal definition to a concrete implementation.

Note that these changes are minor and do not have any impact on existing rules and terms. The extensions introduce little extra complexity and place no greater burden on the rule developer. Indeed, if a tableau neither contains nor creates instances of the **`Rank`** class, then the behavior of the algorithm is *identical* to the unmodified tableau algorithm of Chapter 6.

7.5.2. Implementing Learners

The concrete implementation of a given learning problem requires three classes:

1. A representation of the current hypothesis. This is a class extending from **`Term`**.
2. A **`Rule`** for randomly (or systematically) perturbing the current hypothesis to generate new hypotheses. This rule is cued by observations: when a new observation is encountered, it generates a corresponding set of hypotheses.
3. A rule for comparing new hypotheses against observations and, accordingly, generating instances of **`Rank`**.

I will illustrate the implementation of learners by way of example. I consider a highly simplified (and unrealistic) scenario in which a system must 'learn' the square-root of a number by a stochastic greedy hill-climbing search. Of course, there are efficient algorithms for finding the square-root of a number built into most programming languages. However, the problem makes for a simple illustration of autonomously learning a single unknown parameter.

In this example, the value 2 (=x^2) is 'observed' each second and the system assumes a default hypothesis of 0 (=x). With each observation, the system generates new hypotheses by randomly displacing the current hypotheses (x_h), simulating each hypothesis (${x_h}^2$) and selecting the hypothesis with the least error.

I assume a rule called **`Observer`** that observes the environment and generates time-stamped observations of the type **`Observation`**:

```
declare global counter:integer := 0

class Observer : Rule {
   method process(formulas:set of Formula):set of set of Formula {
      counter := counter + 1
      return {{new Observation(counter, 2)}};
   }
}

class Observation : Term {
   declare time:integer
   declare value:float
   constructor new(timestamp:integer, observation:float) {
      time = timestamp
      value = observation
   }
}
```

Hypotheses are represented by the class **`Hypothesis`**, extending **`Term`**:

```
class Hypothesis : Term {
   declare time:integer
   declare root:float
   constructor new(timestamp:integer, hypothesis:float) {
      time = timestamp
      root = hypothesis
   }
}
```

New hypotheses are generated by random perturbations of the current hypothesis:

```
class Generator : Rule {
   method process(formulas:set of Formula):set of set of Formula {
      declare max_observation:Observation
      declare max_hyp:Hypothesis

      # the perturbations
      declare new_hyp1:Hypothesis
      declare new_hyp2:Hypothesis
      declare new_hyp3:Hypothesis
      declare offset:float

      # find the newest hypotheses and observations
      sort formulas order by Observation.time descending
      max_ob := formulas[0]
      sort formulas order by Hypothesis.time descending
      max_hyp := formulas[0]

      # generate hypotheses if there are newer observations
      if max_observation.time > max_hyp.time
         offset :=  system.getRandomNumber()
         new_hyp1 := new Hypothesis(max_hyp.time+1,
                                    max_hyp.value-offset)
         new_hyp2 := new Hypothesis(max_hyp.time+1,
                                    max_hyp.value)
         new_hyp3 := new Hypothesis(max_hyp.time+1,
                                    max_hyp.value+offset)
         return {{new_hyp1},{new_hyp2},{new_hyp3}}
      else
         return {{}}
   }
}
```

Finally, the **`RankGenerator`** rule measures the error between the hypothesis and observation in a branch to generate an instance of **`Rank`**. The new instance of **`Rank`** implicitly causes the tableau algorithm to select the branch with the least error.

```
class RankGenerator : Rule {
   method process(formulas:set of Formula):set of set of Formula {
      declare ranks:integer := 0
      declare hypothesis:Hypothesis
      declare observation:Observation
      declare simulated:float
      declare error:float

      # count hypotheses that have been ranked
      for each formula in formulas
         if formula instance of Rank
            ranks := ranks + 1

      # is there an unranked hypothesis?
      sort formulas order by Hypothesis.time
      if formulas[ranks + 1] instance of Hypothesis
         # get the unranked hypothesis and its observation
         hypothesis := formulas[ranks + 1]
         sort formulas order by Hypothesis.time
         observation := formulas[ranks + 1]

         # compute the rank/error to optimize
         simulated := hypothesis.value * hypothesis.value
         error := abs(observation.value - simulated)
         return {{new Rank(formulas, error)}}

      else
         return {{}}
   }
}
```

While a simple example, it is easy to see how these core classes may be adapted, with little effort, to a wide range of machine learning algorithms. The **`Hypothesis`** class could represent individuals in a genetic algorithm framework, branches of decision trees, parameters of neural networks and even logical formulas. In Chapter 8, I evaluate a system where the hypothesis space is a Kohonen Self-Organizing Map (SOM) [91] and where observations are based on a virtual 2D camera.

7.6. Discussion and Future Work

The model of machine learning described in this chapter is characterized by rounds of branch fan-out (during hypothesis generation) followed by aggressive branch pruning (during the hypothesis selection). This characteristic can be exploited to improve performance. In the following subsections, I describe how *cuts* and *factoring* may be used to improve performance, save memory and better structure a tableau.

7.6.1. Cuts

While placing an ordering on branches enables the tableau algorithm to emphasize minimal consistent branches and discontinue search on suboptimal branches, 'weakly closed' branches cannot be discarded from memory. This is because it is possible for the open branches to become inconsistent (*i.e.*, generate the *closed* symbol), causing a 'weakly closed' branch to be reopened.

In many cases, however, the rule engineer will know that, once a particular branch has become suboptimal, it will never be reopened. For example, it may be the case that another 'smaller' branch will *always* have at least one descendant that is consistent and therefore open.

The square-root learner of Section 7.5.2 is an example of this situation: once the algorithm has selected a hypothesis, it commits to the hypothesis and has no need to go back to revise an old hypothesis; it will only refine the current hypothesis.

In these cases, the 'weakly closed' branches may be safely discarded from memory.

The management of 'weakly closed' branches could be handled analogously to 'cuts' in Prolog [161]. Rather than attempting to automatically determine the safety of permanently discarding a 'weakly closed' branch, the developer might denote a safe cut using special terms. For example, a *cut*(*index*) constructor may be introduced to the model of the tableau. Whenever a branch contains a *cut*(*index*) term along with a *rank*(*index*, *value*), the branch may be assumed to be permanently closed if the *value* of its ranking is less than that of any other *rank*ing of the same *index* in other branches.

Of course, as in Prolog, the developer must take care to ensure that cuts are genuinely free of unintended side-effects, that they are 'green cuts', to use the terminology of Prolog. In many cases (such as with the square-root learning) the safety of using cuts is obvious but other problems may require careful formal proof before it is used.

There is a large body of work on the semantics, analysis and automatic determination of green cuts in Prolog programs (*e.g.*, [161, 34, 151]). In future, it may be possible to apply some of this work to Comirit, to verify the safety of a cut or to fully automate the generation of cuts. An obvious piece of ‘low-hanging fruit’ is the search for analogs of Prolog clause tail recursion that may exist in tableau reasoning.

7.6.2. Factoring

The tableau algorithm performs well on ‘average case’ problems but there are pathological problems in which the algorithm is unnecessarily slow. In particular, if a tableau is simultaneously solving the conjunction of unrelated disjunctive problems, the branching caused by disjunctions in each of the subproblems may interact to produce a combinatorial branching in the tableau. Careful coordination of the disjunctions, such as by sequencing, by heuristics and by setting disjunctive rules to be low priority, can dramatically improve the situation. However, if the sub-problems are truly separable, then they are more efficiently solved in separate tableaux.

Learning is a good example of this situation. If an agent is continuously learning from its observations, its current problem-solving processes will have no bearing on that learning. That is, while decision-making and planning may depend on learned knowledge, the learning does not depend on the agent’s current decision-making and planning.

Rather than performing learning and reasoning in a single monolithic tableau, it is useful to factor out learning into a separate tableau. The factoring can be made transparent to rule implementation by holding the two tableaux in an implicit ‘conjunction’. That is, the output of the learning tableau can be implicitly imported into rules operating on the main decision-making tableau.

Thus, the system may be factored into tableaux as follows:

- **Upper Tableau.** This tableau is used for learning. The tableau algorithm is applied in isolation, allowing the system to efficiently learn without the concern for managing the interactions with disjunctions produced by the goal or decision-making processes.
- **Lower Tableau.** This tableau is used for the primary decision-making and planning process and can use the results of the Upper Tableau. The tableau algorithm is *only* executed in the Lower Tableau when the Upper Tableau has exactly one open branch and no weakly closed branches (weakly closed rules having been discarded by cuts). Each rule application to the Lower Tableau is then performed against the conjunction of a branch in the Lower Tableau and the contents of the single open branch in the Upper Tableau. The output of such rules are used to update only the Lower Tableau, thus isolating the Upper Tableau from any of the goal-seeking reasoning processes of the agent.

These Upper and Lower Tableaux isolate the background, continuous and time-sensitive process of learning from the longer-running mechanisms of action selection and planning, while still allowing a flow of learned experience to influence action selection and planning.

For greater performance, the Lower Tableau need not wait for the Upper Tableau to converge on a single open branch. Both tableaux can be concurrently evaluated on separate execution threads, with the Upper Tableau saving a checkpoint whenever it converges on a single open branch and the Lower Tableau always reasoning with the last saved checkpoint.

There are other possible variants on factoring. For example, a Comirit System could be extended beyond a two-tableau hierarchy to a complex lattice of tableaux. Furthermore, the need to wait for (or checkpoint against) a single open branch in the Upper Tableau is not strictly necessary. With careful accounting, it would be possible to allow branches in the Lower Tableau to be automatically split into separate branches corresponding to those in the Upper Tableau. Doing so would reintroduce the risk of combinatorial explosion but this may be safe where the combinations are known to be bounded or where there are mechanisms for aggregation. For example, a fixed-width beam search in learning will have a bounded combinatorial effect on goal-seeking behavior. Alternately, the branches of a tableau conducting a beam search may be aggregated into a 'virtual tableau' with a single branch representing the coherent fusion of the best hypotheses of the beam.

7.6.3. Complex Learning Signals and Control Theory

I have presented learning as a process of hypothesis generation and selection. In the example of Section 7.5.2, learning occurred only via the binary decision of the tableau regarding whether to close or retain each branch. It is important to note that this does not mean that the learning algorithms may *only* be implemented as a binary reward process. Hypotheses generators are permitted to examine the deep structure of the environment before proposing candidates, they may make use of the precise value of prediction errors and they may even sample the hypotheses space to compute the local error gradients.

In addition to more sophisticated hypothesis generation, one could apply control theory to the process of parameter search. A simulation can be viewed as a multi-input-multi-output system control problem, where the controller is responsible for fine-tuning and adjusting simulation parameters. In this perspective, the underlying parameters of the simulation are interpreted as the *control signal*, the output of the simulation is interpreted as the *sensed output* and the observed world may be interpreted as the *desired output*.

While these techniques could provide faster convergence and more targeted revision of hypotheses, I have not yet explored them in great depth because, as I shall demonstrate in Chapter 8, naïve learning techniques are already effective. I suspect that, while control theory may have an awkward formulation in Comirit, the literature on control theory will offer useful techniques for improving learning rate and adaptation to a changing environment, while avoiding unwanted hysteresis that might arise when the agent is manipulating the environment.

7.7. Conclusion

In this Chapter, I have explained how the Comirit Framework can be extended to support autonomous learning. In the next Chapter, I will evaluate the framework in the context of a practical learning problem. However, at this point, it is worthwhile to make the following observations:

- Learning can be elegantly incorporated into the Comirit Framework (and, indeed, potentially other tableau-based systems).
- Learning can be integrated into Comirit as a general-purpose, generate-and-test framework that supports not only numerical parameter search but also any machine learning technique that revises and selects hypotheses.
- The extensions of this chapter are completely optional: rules can take advantage of the ordering of branches but need not. A set of tableau rules that makes no use of the learning capabilities will behave in *exactly* the same way as with the *unmodified* tableau reasoner of Chapter 6.
- The extended framework can be used for continuous observation-based learning, isolated from any abstract reasoning processes.

Thus, the Comirit Framework combines simulation, logical reasoning and autonomous learning into a coherent, integrated and readily adaptable framework. This is a powerful combination with wide potential.

In addition, the integration of machine learning into the Comirit Framework serves to further validate my earlier claims of the advantages of the Comirit Framework:

- The ease by which learning is integrated into Comirit reinforces my claims that the control strategy is open-ended and readily accommodates different mechanisms.
- The possibility of combining machine learning with simulation and logical reasoning in a hybrid tableau further highlights the power of Comirit Simulation as a representation of commonsense knowledge. Comirit Simulations, being parameter-driven, are highly amenable to the many highly-efficient machine learning algorithms for parameter optimization.

Finally, I note that the extensions proposed in this chapter may be adapted to purposes other than learning:

- The implicit selection of minimal branches can also be used to guide planning or action selection in reward-driven domains. That is, a tableau can be used to generate plans and simulation may be used to evaluate rewards so that sub-optimal *actions* will be automatically discarded from the search.
- In complex and computationally expensive domains, the $rank(index, value)$ terms may be used to manage 'attention' and implement heuristics. For example, if it is possible to statistically estimate the likelihood of a given branch being satisfiable, then by expressing the negated log-likelihood using *rank* terms, the system will prioritize search in the most probable branch, in effect, achieving a maximum likelihood depth-first search.

Of course, the scope of this dissertation is necessarily constrained and so these exciting avenues for further research remain as future work.

Chapter 8

Evaluation

Throughout the preceding chapters of this dissertation, I have described the design and implementation of a commonsense reasoning and learning system that I have called Comirit. In this chapter, I use this system to advance my thesis that *artificial commonsense can be created within constrained time and resources, from present-day technologies*. A full commercial-grade implementation of Comirit is beyond the scope of this research project so I evaluate prototypes of the framework against established benchmark problems. In these evaluations, I show that Comirit can perform commonsense reasoning in realistic time and resources and from present-day technologies, while outperforming existing state-of-the-art systems.

That is, this chapter serves to demonstrate three claims:

1. **Capabilities.** Comirit is an effective framework for artificial commonsense reasoning.
2. **Cost-effectiveness.** Comirit can be built with realistic time and resources.
3. **Non-speculative technologies.** Comirit is a re-purposing of present-day technologies and is not predicated on speculative assumptions about future innovation.

Given a team of software engineers and funding to support a two-to-three-year development program, a direct evaluation of Comirit would proceed by complete implementation and application to real commonsense problems. Unfortunately, I do not have access to such resources: the *design* of Comirit has consumed my time as a research student. The *implementation* of the system would require renewed time and resources.

How, then, can Comirit be evaluated if it exists only as a design, rather than implementation? Lacking a full implementation, I must make inferences from properties of the design and extrapolate from experiences with prototypes. I do this in seven ways:

Capabilities

1. Showing that the design is derived from a principled analysis of the problem (*i.e.*, "it should work")
2. Constructing prototypes and conducting experiments (*i.e.*, "it seems to work")
3. Showing that the design corresponds to a model of cognition (*i.e.*, "it is similar to something that does work")

Cost-effectiveness

4. Extrapolating from time expended in implementing prototypes
5. Examining the effort involved in building other large-scale, open-ended games and simulations

Non-speculative Technologies

6. Noting that the design consistently draws from routine techniques of Computer Science
7. Implementing functional prototypes

The combined weight of each argument amounts to conclusive evidence of the value of Comirit and therefore the validity of my research thesis.

The structure of this Chapter follows that of the seven approaches listed above, examining each of these arguments in turn.

8.1. Capabilities

Commonsense reasoning is insufficiently understood to *formally* prove the correctness and completeness of any system and Comirit is no exception. Furthermore, because a full implementation of my design lies beyond the scope of this dissertation, it is not possible to directly test the capabilities of the system in its intended applications. Instead, I demonstrate the capabilities of the system by showing it has a sound design, that it performs well on standard benchmark problems and that it has an architecture that corresponds to a model of cognition.

To this end, I present the following arguments (these being expansions of the first three of the seven arguments just listed):

Principled Design

1. Comirit follows from a principled analysis of intelligence and commonsense; the architecture is chosen to maximize the practical likelihood of achieving commonsense intelligence.

Benchmark Evaluation

2.a. The simulations used by Comirit (as described in Chapter 5) can solve the *Egg Cracking Problem* and its elaborations.

2.b. The hybrid reasoning mechanism (of Chapter 6) can solve the *Eating on an Airplane Problem*.

2.c. The learning capabilities of Comirit (described in Chapter 7) can autonomously learn a realistic (but simplified) physical reasoning problem.

Correspondences

3. The architecture of Comirit corresponds to a model of cognition proposed by Peter Gärdenfors. This correspondence is somewhat indicative that the architecture will scale to deep intelligence problems.

Note, however, that I do not claim that Comirit is *necessary* for commonsense reasoning, only that is it *sufficient*. Other powerful techniques may be developed and used independently for commonsense reasoning. The open-ended design of Comirit is such that, should other powerful commonsense reasoning techniques be developed, they could be integrated into Comirit to further enrich the functionality of the framework.

These arguments are presented over the following subsections (Sections 8.1.1–8.1.5).

8.1.1. Principled Design

The principled design process used to develop Comirit is documented throughout this dissertation:

- In Chapter 3, I developed a formal framework for the symbol grounding problem and concluded in Section 3.6 by deducing the importance of iconic representations.
- Starting from this insight into the importance of iconic representations, the Comirit Framework was developed as a platform that integrates iconic with non-iconic methods of reasoning. The combination of iconic and non-iconic reasoning is designed to combine the strengths and

compensate for weaknesses in each method. This design process is documented in Sections 5.1, 6.1 and 7.1, where I explained the development from needs and objectives to a design.

- Comirit, as it is presented in this dissertation, represents the outcome of over 22 separate versions implemented from scratch (documented later in Section 8.2.1), in addition to countless rounds of refactoring and 're-architecting' in each version. This work had the effect of eliminating unnecessary complexity, ensuring a clear focus on the core principles of the design and resulting in a seamless integration of simulation, learning and reasoning.

8.1.2. Benchmark Evaluation: Representations

To demonstrate the effectiveness of simulation as a representation for commonsense reasoning, I consider the *Egg Cracking Problem*. The problem is a standardized, non-trivial benchmark that poses representational challenges and has published solutions that may be used for comparing the effectiveness of the representation.

The Egg Cracking Problem

The *Egg Cracking Problem* was posed by Ernest Davis and appears in Miller and Morgenstern's [127] collection of benchmark problems (see also Section 2.2). It is the task of characterizing the following situation:

> 'A cook is cracking a raw egg against a glass bowl. Properly performed, the impact of the egg against the edge of the bowl will crack the eggshell in half. Holding the egg over the bowl, the cook will then separate the two halves of the shell with his fingers, enlarging the crack, and the contents of the egg will fall gently into the bowl. The end result is that the entire contents of the egg will be in the bowl, with the yolk unbroken, and that the two halves of the shell are held in the cook's fingers.'

Questions of breadth, fidelity, density, uniformity and elaboration tolerance of a representation can be judged with respect to the 'standard' variations of the problem [127]:

> 'What happens if: The cook brings the egg to impact very quickly? Very slowly? The cook lays the egg in the bowl and exerts steady pressure with his hand? The cook, having cracked the egg, attempts to peel [the shell] off its contents like a hard-boiled egg? The bowl is made of loose-leaf paper? of soft clay? The bowl is smaller than the egg? The bowl is upside down? The cook tries this procedure with a hard-boiled egg? With a coconut? With an M&M?'

Modeling the Problem

In Comirit, this problem would be characterized by constructing a simulation of an egg and bowl. This simulation requires the following parts:

1. A graph-based approximation of the problem domain: a 3D mass-spring model of an egg, its contents and a bowl (Figure 8.1)
2. An implementation of physical laws that characterize the possible behaviors that occur in simulation (see also Section 5.4.3):
   - Momentum and Universal Damping
   - Gravity
   - Solid Bonds (the application of Hooke's law along 'springs' in the graph)
   - Rigid Shapes (the application of Hooke's law for torsion springs to ensure that 'springs' coincident at a 'mass' remain at fixed angles)
   - Liquid Incompressibility (a repelling force between liquid 'masses' that prevents liquids from being compressed)
   - Liquid Surface Tension (a weak attractive force between liquid 'masses' that causes the liquid nodes to group together in droplets)
   - Impermeability of Hulls (a force that prevents 'masses' from passing through hulls)
3. An assignment of values to attributes of the graph-based approximations, so that the simulation will closely correspond to reality. These are set according to known values (*e.g.*, an egg

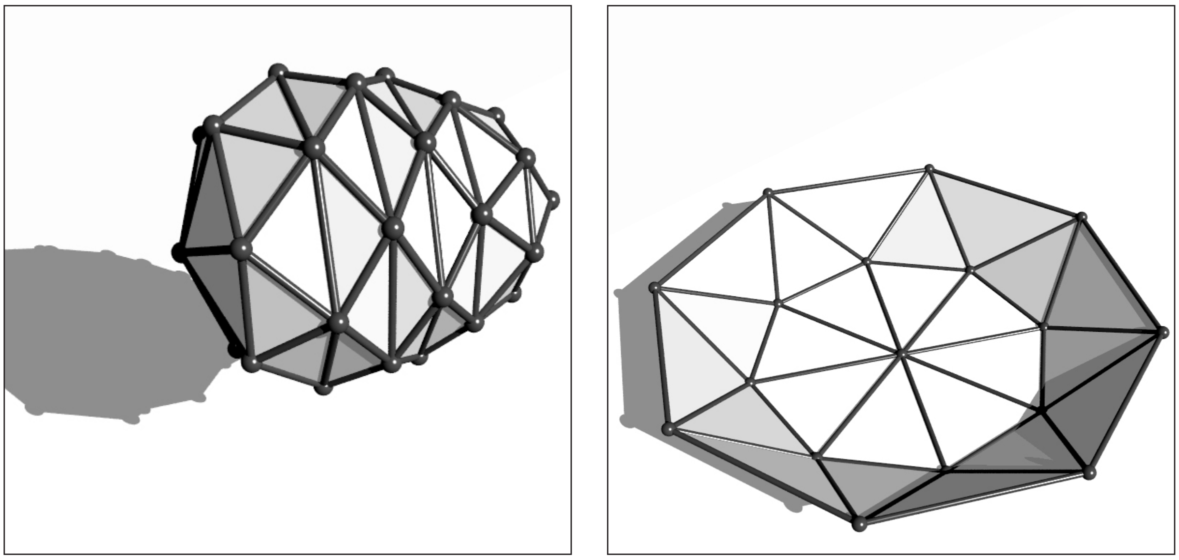


Figure 8.1: Mass-spring models of an egg and a bowl

weighs approximately 60g) and as determined by trial-and-error to match observations (*e.g.*, rigidity is set by comparing simulations against observations of a real egg).

4. A set of operators to bridge the gap between the symbolic description and the simulation, to construct simulations and read their results:

    - Cracked (reports, for each object, whether there exist any rigid edges that have broken)
    - Wet (reports, for each object, if there exists any surface of the object that is in contact with a graph-node that is annotated with a 'liquid' classification)
    - Contains (reports, for each pair of objects, if the nodes of one closed object are fully contained within the other object)

Some or all of these may be autonomously learned by the use of machine learning techniques and the extensions described in Chapter 7 (Indeed, I evaluate learning of attributes in Section 8.1.4). However, the focus of this dissertation is the design of the Comirit Framework, rather than, for example, actually implementing interfaces to 3D scanning devices or creating shape segmentation algorithms to permit 3D models to be learned from observation. For this reason, I conduct this evaluation with hand-crafted models, laws, attributes and bridging operators.

In the *Egg Cracking Problem*, the role of the cook also presents unusual difficulties beyond the scope of this evaluation. The cook is an agent with independent beliefs and objectives and a complex and highly actuated body. While a detailed model of the human body needs only be constructed once, the efforts involved in creating such a rich 3D model of the human body and in modeling the chef's motivations are beyond the scope of this dissertation. Instead, I only consider the inanimate objects subject to disembodied forces. This perspective is typical among the other 'solutions' to the *Egg Cracking Problem* [129, 107, 155].

Observations

On implementing the model of the *Egg Cracking Problem*, the problem and its elaborations can be tested by running simulations.

Figure 8.2 depicts frames from the impact of an egg 'dropping' into the bowl. In this visualization, a crack can be seen to have formed along the edge of the shell, allowing the contents of the egg to spill out into the bowl. The bridging operators accordingly report: *wet*(*bowl*), *wet*(*shell*), ¬*cracked*(*bowl*), *cracked*(*shell*) and ¬*contains*(*yolk*, *shell*). Figure 8.3 depicts two elaborations: a solid, hard-boiled egg (the egg does not crack) and an upside-down bowl (the liquid spills onto the table).

Simulations such as this may be said to accurately characterize a situation because they exhibit the full behaviors expected of the systems that they model. Indeed, a key advantage of simulation is that

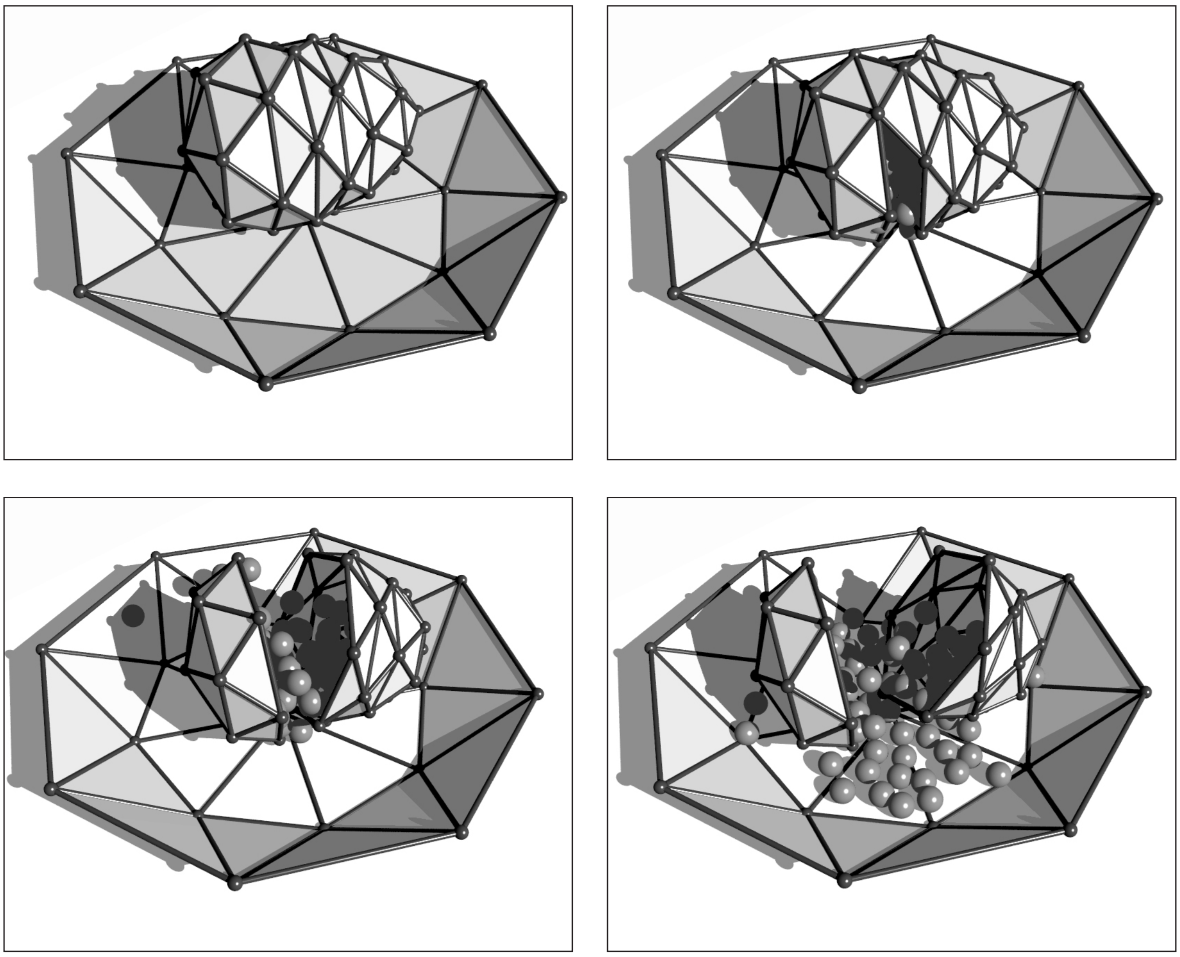


Figure 8.2: Frames of an egg-cracking simulation

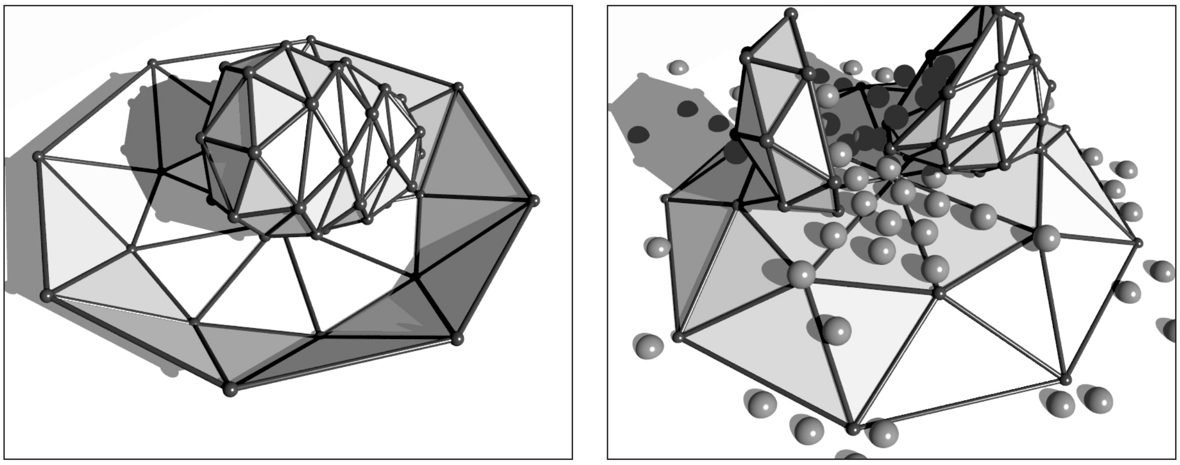


Figure 8.3: Simulated elaborations of the egg-cracking problem: a hard-boiled egg (left) and an upside-down bowl (right)

it does not require the explicit *a priori* definition of abstract behaviors, such as ‘cracking’, because the cracking behavior emerges from the physical laws of the simulation.

Simulation has clear advantages in representing commonsense knowledge, especially when compared to the logical methods used in other studies of the problem domain [129, 107, 155]. Consider the evaluation criteria of Section 2.2:

- **Breadth.** The laws of physics are almost universally broad. Virtually any mechanical interaction with the egg and bowl may be tested without modifying the model. While the formalization presented here is currently limited to the mechanical interactions between eggs and bowls, additional 3D models could be used to model other kinds of mechanical objects (*e.g.*, a glass of water) and additional laws could be introduced to consider non-mechanical effects (such as thermodynamics and state-changes in cooking the egg). Nevertheless, the breadth of this model is vastly richer than logical formalisms: simulation can model scenarios such as ‘what happens if the bowl is cracked but held together’ or ‘what would happened if performed without gravity’ or ‘what would happen if the bowl was a flat plate with no rim’—scenarios that are not supported by the proposed “solutions” that are based on logical formalisms.
- **Fidelity.** Simulations have extremely high fidelity because they have been explicitly designed to emulate reality with precision and detail. Simulations can be rendered as images and directly compared to real life situations—indeed, this is how the representation is fine-tuned. The universality of the laws of physics ensures reasonable and accurate behavior even in out-of-scope and pathological scenarios. In contrast, symbolic terms often suffer ambiguities in reasoning about edge cases. For example, whether a leaky water bottle is said to *contain* a liquid will depend on whether it is to be used for immediate refreshment or for a long hike. Furthermore, the ability to run and visually diagnose simulations assists with the quality of the development process, in contrast to formal logics in which it may not even be evident that an expression is formally consistent.
- **Density.** A strength of simulation is the small number of laws that can produce rich emergent behavior. The *Egg Cracking Problem* requires only seven physical laws in simulation, in contrast to the 92 axioms and definitions of Morgenstern’s logical axiomatization [129]. The density of a simulation-based axiomatization stems from the underlying regularity of the universe: while falling, pouring, overflowing, leaking, splashing, spreading and trickling may be conceptualized differently, they are all caused by and subject to the common effects of liquid flow and gravity.
- **Uniformity.** Comirit Simulations use regular structures of update functions, graphs and annotations. The framework provides a uniformity across and within simulations, allowing for rules and objects to be created and reused independently.

- **Elaboration Tolerance.** Simulations are tolerant of elaborations. Most of the standard elaborations can be simulated by changing the initial configuration or by adjusting the parameters on objects. I will discuss this in more detail, later, in the subsection entitled 'Elaboration Tolerance'.
- **Costs.** Simulations are easy to build and do not require specialist training in formal logics: physical laws come straight from elementary physics textbooks, 3D models can be built with modern modeling tools and the debugging/development process is dramatically enhanced by the ability to run and observe simulations. With a complete implementation of the Comirit Framework and full tool support, the physical laws and the simulation models for the *Egg Cracking Problem* and its elaborations should be constructed in a matter of hours or possibly even minutes. These costs are further evaluated in Section 8.2.1.

I also note that it is unclear whether any of the reported 'solutions' to the *Egg Cracking Problem* ([129, 107, 155]) have been tested on a real theorem proving system. None of the authors mention any experiences with concrete implementation so it seems that their results are purely theoretical.

Elaboration Tolerance

Many elaborations and what-if-scenarios can be entertained simply by adjusting the parameters of the simulation. In fact, it is through the comparative ease by which elaborations are accommodated that the benefit of simulation becomes most evident:

- **Speed and Force.** By virtue of simulation, such elaborations present no difficulty. If there is insufficient force, the egg shell's rigidity will not be overcome and, conversely, if there is too much force, it may even exceed the rigidity and snap-length of the bowl, breaking the bowl. If the chef's force is slow and continuous, directed at odd angles or is an attempt to peel off the shell, the simulation will continue to predict the outcome correctly.
- **Shape and Orientation.** Again, once the shape and position have been configured in the simulation's initial conditions, such elaborations present no difficulty. For example, if the bowl is too small, then the simulated liquids will simply overflow the bowl without any prior need to model concepts such as 'fullness'.
- **Texture.** A wide range of solids, liquids and gasses can be simulated by adjusting parameters for rigidity, breaking points, viscosity and permeability. The different behaviors of chocolate eggs, coconuts, hard-boiled eggs, duck eggs, soccer balls and of bowls made from paper, clay and ceramics may all be modeled via appropriate configuration of these material parameters.
- **Detail.** Finer-grained approximations require further specification but this may be readily done by elaborating the initial conditions (rather than the laws of simulation). For example, in

a real chicken egg there are two membranes between the egg white and the shell which hold a slightly-cracked egg together. The simulation could be enriched by inserting such membranes (a pliable and fragile surface) into the initial conditions of the egg.

- **Scope.** Additional constraints would be required to address elaborations that are far beyond the problem's original scope. For example, additional constraints would be required for 'rotting' and 'cooking' if the action is to be performed over months or at high temperatures. However, it is important to note that the addition of these constraints does not require revision of existing constraints: gravity and momentum continue to work on all matter. 'Cooking' would be simulated by adding an update rule to allow a subset of edges to 'harden' after a certain temperature has been reached.

Simulation is therefore able to evaluate an extremely rich range of questions and elaborations. This advantage is most pronounced in comparison to logic-based formalizations (for the *Egg Cracking Problem*, the most developed of such formalizations is that of Morgenstern [129]). Virtually all of the 'standard' variations are made possible by simple parameter changes, while Morgenstern's axiomatization—as she, herself, explains—is unable to capture all of the changes without significant further work. These variations are listed in Table 8.1 (overleaf).

Davis' Critique

Ernest Davis, in his logical formalization of the 'commonsense' flow and behavior of liquids [32], offered a critique of the few other approaches in the literature. In particular, he made reference to the work of Gardin and Meltzer [52], whose 2D particle-based simulations bear similarity to Comirit 3D simulations (the relationship to Gardin and Meltzer's work was discussed in Section 5.2).

His critique centered around the inability of a particle-based formalism to model reality plausibly. He noted that Gardin and Meltzer's simulated liquids pour in streams that are just one molecule thick (*i.e.*, 'once a particle of water has gone past the lip of [a] pitcher, it is unsupported, and therefore must fall straight down' [32]). While this is a deficiency in Gardin and Meltzer's original publication, it does not represent an underlying weakness in the principle of simulation. The observed real-world behavior of pouring is easily replicated in a particle-based simulation with the following two laws that I used in the *Egg Cracking Problem*:

- **Liquid Incompressibility.** This is a repelling force between particles of liquid that prevents liquids from being compressed. If there is sufficient volume of liquid being poured, then it will not be possible for the liquid to be compressed into a single one-molecule-thick flow.

- **Liquid Surface Tension.** This is a weak attractive force that causes particles to group together to form droplets, to hold the bulging shape at the edge of a rim and to cause a stream of water to take on a cylindrical shape even if poured from a non-cylindrical hole or edge. In fact, such phenomena are excluded from Davis' own detailed formalization due to the logical complexities that it would have added.

Evidence of this claim appears in Figure 8.4, in which a Comirit Simulation is used to model water pouring and splashing over a barrier after it has been suddenly released.

That is, while Davis' was correct in pointing out the weakness of Gardin and Meltzer's representation, such weaknesses are not foundational but represent the preliminary nature of Gardin and Meltzer's publication. The problem is easily fixed and the ease by which the problem is identified and corrected highlights the benefits of simulation as a realistic and accurate model of commonsense.

Other Approaches

The *Egg Cracking Problem* demands deep and sophisticated understanding of commonsense physical phenomena. The problem is therefore challenging for systems and formalisms that are based on shallow knowledge. For example, the following list is a sample of assertions in ResearchCyc[*] that pertain to the *Egg Cracking Problem*:

```
(isa Egg-Chickens DefaultDisjointEdibleStuffType)
(genls Egg-Chickens Egg-Birds)
(genls Egg-Chickens BreakfastFood)
(genls EggWhites-Food (EdiblePartFn Egg-Chickens))
(gensl EggWhites-Food LiquidTangibleThing)
(genls EggYolk-Food YellowColor)
(genls EggShell Container)
(disjointWith Egg-Foodstuff Eggshell-Stuff)
(genls Cracking SeparationEvent)
(genls Cracking IntrinsicStateChangeEvent)
(disjointWith LiquidTangibileThing SemiSolidTangibleThing)
(genls Bowl FoodVessel)
(genls Bowl Basin)
(genls Basin ConcaveTangibleObject)
```

While ResearchCyc and OpenCyc contain millions of assertions, this knowledge primarily concerns the superficial level of typing and classification. This level of detail may be useful for disambiguation of terms in natural language processing problems. For example, this knowledge may disambiguate the verb *cracking*, referring to the physical separation of parts of an object, from the metaphorical meaning, as in *he is cracking under the pressure of completing his dissertation*. However, the level of detail in Cyc is insufficient to explain the specific consequences of a particular event, such as what specifically would happen if a cook applies steady forces to an egg.

[*]ResearchCyc may be obtained from http://research.cyc.com/

Elaboration	Comirit Simulation	Morgenstern’s Theory
Slow impact	Run the simulation with a low force parameter.	**Not supported**: the axiomatization has no model of force or of the nuanced effects of different forces.
Fast impact	Run the simulation with a high force parameter.	**Not supported**: the axiomatization has no model of force or of the nuanced effects of different forces.
Steady pressure exerted	Run the simulation with a low-but-steady force parameter.	**Not supported**: the axiomatization has no model of mixtures, crushing or the chef’s hands.
Shell peeled like a hard-boiled egg	Apply peeling forces to the egg shell.	**Not supported**: the axiomatization does not include a notion of fragmentation.
Loose-leaf paper bowl	Set rigidity parameters of the bowl to match that of paper. However, additional rules are required to model the change in strength of paper when it becomes wet.	Supported by the addition of axioms for hardness and the need for a hard bowl to crack against.
Soft clay bowl	Set rigidity and brittleness parameters of the bowl to match that of soft clay.	Supported by the addition of axioms for hardness and the need for a hard bowl to crack against.
Bowl is smaller than egg	Run the simulation with a small bowl.	Change the initial parameter describing the size of the bowl.
Upside-down bowl	Run the simulation with an upside-down bowl.	Change the initial parameter of the orientation of the bowl. However, the axiomatization can not account for difficulty of cracking an egg against the edge of an upside-down bowl.
Hard-boiled egg	Set rigidity and brittleness parameters of the contents of the egg. Additional rules are required to automatically form and ‘set’ such bonds, if it is necessary to model the cooking process.	**Not supported**: the axiomatization does not support a notion of adhesion, the shapes of openings and the ability of solids to pass through such openings.

Table 8.1 (a): Comparison between Comirit Simulation and Morgenstern’s Axiomatization

Elaboration	Comirit Simulation	Morgenstern's Theory
Coconut instead of egg	Replace the egg with a coconut (*i.e.*, a round object configured with appropriate parameters) and re-run the simulation.	**Not supported**: the axiomatization does not support a notion of adhesion, the shapes of openings and the large forces involved in opening a coconut.
M&M chocolate instead of egg	Replace the egg with an M&M (*i.e.*, a flattened obloid configured with appropriate parameters) and re-run the simulation.	**Not supported**: the axiomatization does not support a notion of adhesion, the shapes of openings and the ability of solids to pass through such openings.

Table 8.1 (b): Comparison between Comirit Simulation and Morgenstern's Axiomatization

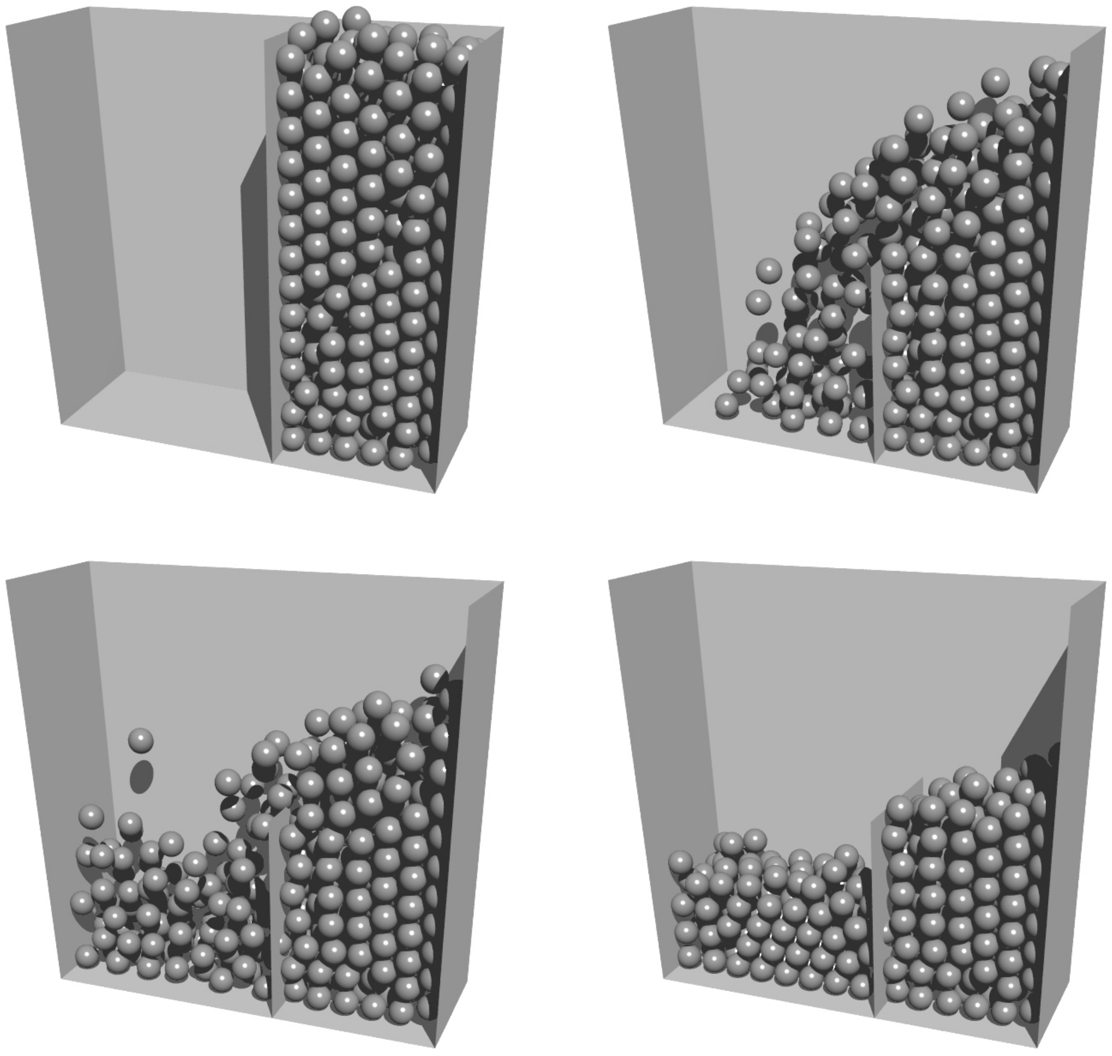


Figure 8.4: Simulation of liquid flow

ResearchCyc includes a general-purpose logical language for expressing knowledge so, while ResearchCyc *does not* solve the *Egg Cracking Problem*, there remains the question of whether it *could* solve the problem. For example, a logical formalization of the *Egg Cracking Problem* could be translated into a ResearchCyc knowledge-base. Doing so would not offer any improvement on the logical formalization; it would simply translate the formalism to an executable form. Furthermore, to do so would represent a divergence from the kind of knowledge currently in ResearchCyc so that the reasoning engine is unlikely to be able to reason with the knowledge efficiently.

Linguistic and non-formal commonsense knowledge-bases fare even worse on the *Egg Cracking Problem*. While they may serve as rich resources for word disambiguation and for identifying commonsense connections and relationships between ideas and words, they lack the rich structure required to perform deep and accurate reasoning. For example, some of the facts in the Open Mind Commonsense / ConceptNet database [67] that concern eggs, bowls and cooks appears in Table 8.2.

Relation	*x*	*y*
you are likely to find *x* in *y*	eggs	the fridge
you are likely to find *x* in *y*	eggs	a chicken
x can be *y*	eggs	eaten
x is *y*	egg	egg-shaped
x is *y*	egg	fragile
x is created by a *y*	egg	chicken
x is created by a *y*	egg	bird
you can use an *x* to *y*	bowl	hold ice-cream
you can use an *x* to *y*	bowl	cut hair
x is *y*	bowl	more than a cup
x is *y*	bowl	round
x can *y*	cooks	prepare a meal
x can *y*	cooks	salt meat
x requires *y*	cook	utensils
x is a kind of *y*	cook	activity

Table 8.2: Knowledge about the Egg Cracking Problem found in ConceptNet

While offering interesting (and somewhat creative) connections between terms, this kind of knowledge would not be suitable for determining if a given bowl could contain a cracked egg or how outcomes might differ between a hard-boiled and a raw egg.

Summary

Simulations offer a simple but powerful way of characterizing commonsense situations. A small number of simple laws can be used to represent commonsense knowledge with breadth, fidelity, density, uniformity, elaboration tolerance and cost-effectiveness. In particular, the strengths of this approach compare extremely favorably against methods based on logics of action.

I have only modeled physical commonsense reasoning but this approach would be highly effective in any domain that is driven by, predicted by or can be approximated by small sets of fundamental laws. Evaluation of Comirit in such non-physical domains remains as future work.

8.1.3. Benchmark Evaluation: Control

I claimed that the tableau-based control mechanism of Comirit combines independent reasoning mechanisms into a powerful whole. In this Section, I demonstrate these claims by testing the framework on a problem that involves both logical reasoning and simulation to solve a problem that is difficult to solve by logical deduction alone.

The Comirit Framework, being a hybrid reasoning system, incorporates logical reasoning capabilities. Thus, Comirit *theoretically* subsumes any purely logical approach to commonsense reasoning. However, a *practical* implementation of Comirit is not merely a logical system with extensions for simulation. Simulations are used to represent knowledge wherever and whenever possible, *instead* of logical formulas. In real applications, the capabilities of Comirit therefore intersect with, but do not necessarily subsume, the capabilities of logical formalizations.

This is not necessarily a limitation. My objective is not to create general-purpose artificial intelligence but to create practical artificial commonsense. Logical formalizations may be necessary to express the precise dynamics of gases and forces in a combustion engine or to represent the true-but-unprovable formula in the proof of Gödel's theorem. Neither of these, however, may be said to be part of commonsense knowledge.

I demonstrated, by the *Egg Cracking Problem*, that simulation offers clear advantages for representing important commonsense knowledge. The objective of this Section is to show that those advantages do not come at the price of inflexibility. Thus, while Comirit may not in practice subsume the full capabilities of a logical formalization of commonsense knowledge, it combines the most useful aspects of simulation and logical deduction into a powerful and integrated whole.

Eating on an Airplane

I consider a simplification of John Bell's *Eating on an Airplane Problem* that appears in Miller and Morgenstern's [127] collection of commonsense reasoning benchmark problems. To my knowledge this difficult problem has not been attempted in the literature.

The problem—presumably thought up during a long and unpleasant flight—concerns the commonsense problems that arise from the difficulties of eating on an airplane and how a hypothetical humanoid robot might address them. While the concept of a robot eating food in economy class is clearly absurd, the problem is relevant because it offers a playful scenario to explore the real commonsense reasoning problems and skills that apply to humanoid robotics: social norms, the manipulation and understanding of everyday objects and the avoidance of danger or mess.

> 'A humanoid robot is flying economy class on a major airline and is required to "eat" the packaged meal that has been served to it. Like its fellow human travelers, the robot can be assumed to be in a standard seat and to have two arms which function similarly to theirs, with similar restrictions on mobility; *e.g.,* because of the cramped conditions, the robot's elbows have to remain close to its chest. In front of the robot is the familiar small table, occupied almost entirely by the tray containing the meal, neatly packaged in little plastic containers with transparent lids, along with a small plastic cup containing a foil-sealed tub of water, and a cellophane envelope containing a set of plastic cutlery, napkin, condiments, etc. For simplicity, assume that eating can be taken to consist of manipulating the food and drink to the robot's mouth, where the utensils are emptied at typical human diner rate. The robot is conventional in eating habits; thus, it tries at all times to not spill, to use the appropriate utensils, and to obey conventions as to when it is permissible to eat with its fingers (chicken, no; asparagus, yes). Moreover, it begins its meal with the starter, follows this with the main course (along with the mini roll which it has spread with butter), then it has dessert, and finally the cheese and biscuits. To complicate matters, the robot drinks water at various stages of the meal. Everything must be kept on the tray or table, including the packaging for the plastic cutlery, the tops of containers, and the containers and their contents. So, like its human companions, the robot quickly becomes involved in an elaborate Chess game, continually maneuvering the containers so that the chosen one is in position.'

This is a deeply complex problem that demands sophisticated planning in addition to both physical and social reasoning. A full implementation is beyond the scope of this dissertation, so I will consider a simplification:

1. A robot faces a mug of coffee, a cookie and a sandwich (two stacked slices of bread).
2. The robot follows a social norm of first eating the main course (the sandwich) before the dessert (the cookie) and of eating the dessert before drinking coffee.
3. The robot can choose from two lifting actions to eat or drink an object: lifting from the top or lifting from the base.
4. The robot can choose from two speeds in performing those lifting actions: slow or fast.
5. The robot considers an action to be performed properly and without mess if the object stays *together* in "one piece" (*i.e.*, the coffee stays in the mug or the sandwich stays in one piece).
6. The objective of the problem is to have the robot decide what it should perform next: what should it eat? what action should it use to eat it?

The tableau-based reasoning system could use a rich action logic for reasoning about sequencing and ordering of steps. However, for the purposes of this example, I consider a simple logical form that considers only the question of 'what do I eat next?'.

Thus, the problem is specified as the conjunction of the following logical formulas:

Goals

- *together* (*i.e.*, the robot does not want to make a mess)
- $eat(sandwich) \vee eat(cookie) \vee eat(coffee)$[*]

Initial Conditions

- $available(coffee) \wedge available(cookie) \wedge available(sandwich)$

Social Norms

- $before(cookie, coffee) \wedge before(sandwich, cookie)$
- $\forall\, x, y, z : FOOD \bullet before(x, y) \wedge before(y, z) \Rightarrow before(x, z)$
- $\forall\, x, y : FOOD \bullet available(x) \wedge before(x, y) \wedge available(y) \Rightarrow \neg\, eat(y)$

Actions

- $\forall\, x : FOOD \bullet eat(x) \Rightarrow lift(\mathrm{x}, top) \vee lift(x, base)$
- $\forall\, x : FOOD \bullet eat(x) \Rightarrow speed(x, fast) \vee speed(x, slow)$

[*]Given appropriate rules for existential quantification, this goal might alternately be expressed as $\exists\, x : FOOD \bullet eat(x)$.

Note that universally quantified formulas, such as those listed above, are usually processed by tableau rules that perform substitutions for each grounded form in a branch or, equivalently, instantiate copies of the formula where the quantified variables have been substituted by a unification variable. Comirit allows for rules to be defined alongside formulas so, in this example, rather than implementing generic quantification rules, I convert each quantified formula into a simple rule that performs pattern matching over the branch and adds terms as appropriate. For example, $\forall\ x, y, z : FOOD \bullet \mathit{before}(x, y) \wedge \mathit{before}(y, z) \Rightarrow \mathit{before}(x, z)$ is converted into a rule of the form $\mathit{before}(A, B) \wedge \mathit{before}(B, C) \rightarrow \{\{\mathit{before}(A, C)\}\}$.

This logical formulation is then combined with a simulation engine. The simulation engine is initialized according to *available*(*object*) terms. For example, given the availability of a cookie, a sandwich and a cup of coffee, the simulation may be initialized as depicted in Figure 8.5. The system then simulates actions according to *lift*(*object*, *where*) and *speed*(*object*, *rate*) terms in the current branch of the tableau. When the simulation executes, it generates *together* and $\neg$*together* symbols according to the effects that occur in simulation. That is, if the parts of any food are not together after the action (and thus forming a 'mess'), then the simulation generates a symbol $\neg$*together*.

The tableau resulting from the joint logical and simulation reasoning is depicted in Figure 8.6.

Note that in this completed tableau, logical deduction ruled out the option of eating the cookie or the coffee, while the simulation ruled out the possibility of lifting the sandwich by its top.

Note also that the tableau has found two suitable actions for eating the sandwich. The selection between actions could be performed outside the tableau by a mechanism beyond the design of Comirit. Alternately, the selection could be performed implicitly by incorporating the prioritization mecha-

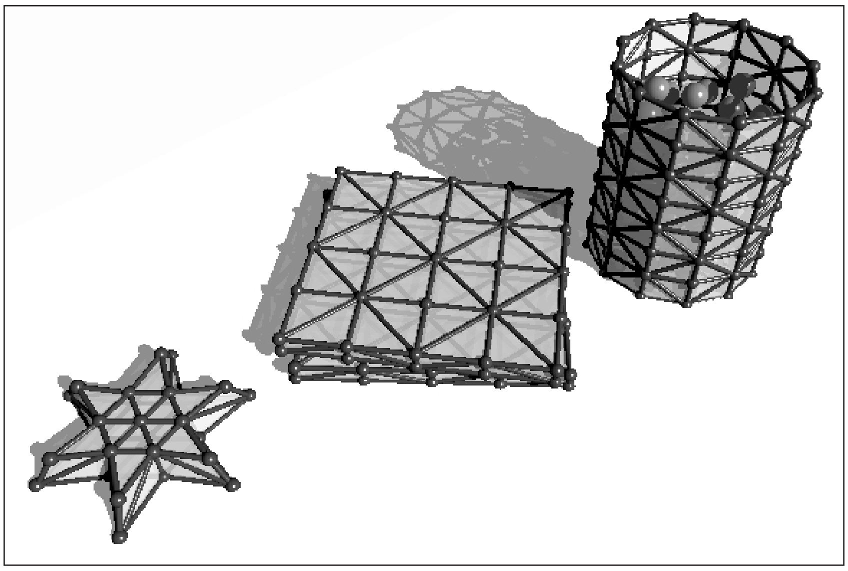
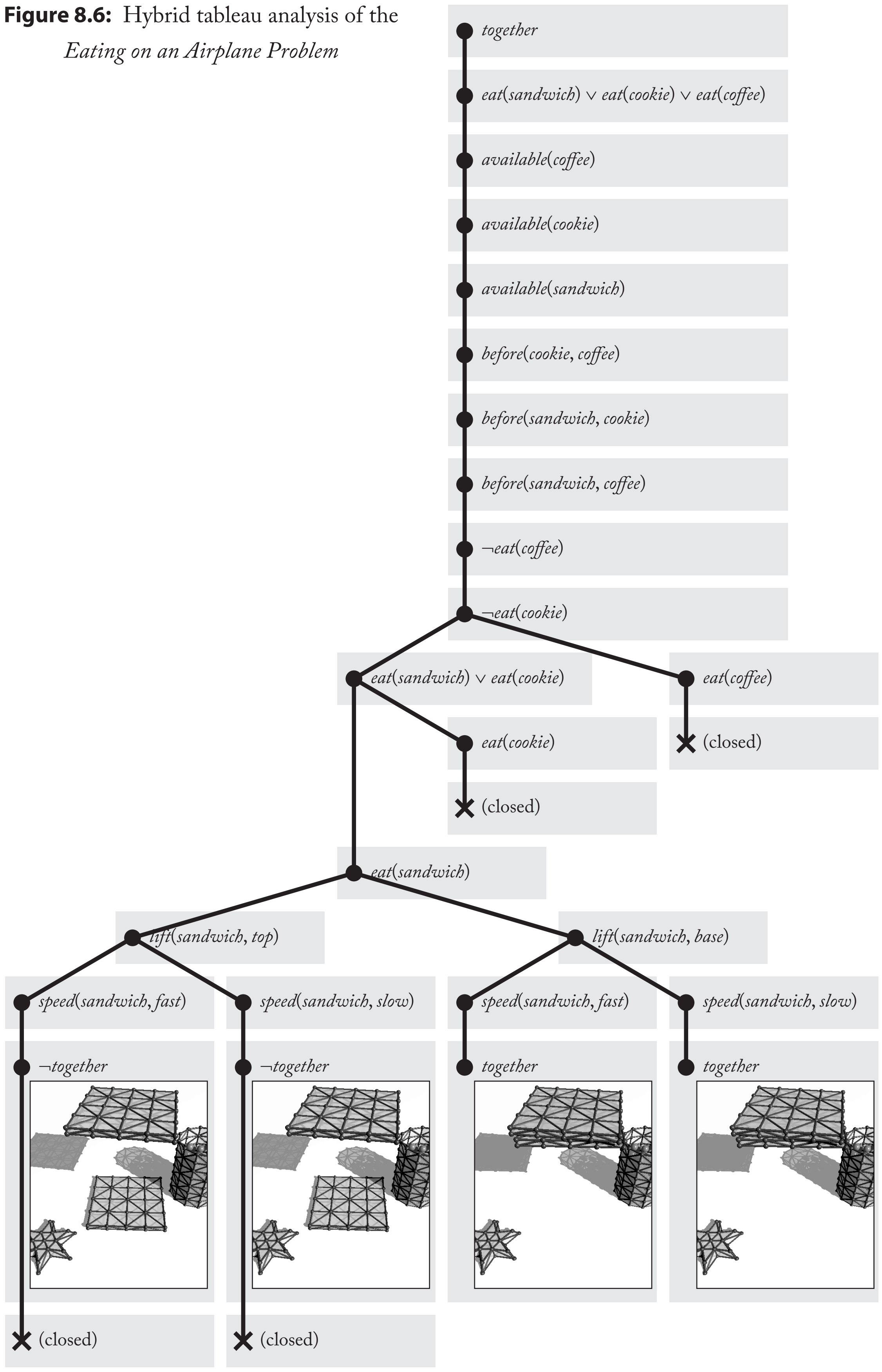


Figure 8.5: Initial conditions of a simulation for the *Eating on an Airplane Problem*

Figure 8.6: Hybrid tableau analysis of the *Eating on an Airplane Problem*



nisms of Chapter 8 into the tableau (*i.e.*, the learning mechanisms for hypothesis selection can also be used for managing non-logical preferences that form a total order).

Observations

This example demonstrated the elegance of combining both logical expression and simulation. The logical formulas enabled terse description of social norms and the relationship between the decision to eat and the lifting action for that decision, while the simulation enabled the complex physical behaviors of the objects to be efficiently expressed and rapidly developed.

Thus, this approach to commonsense reasoning has advantages over logical approaches to commonsense reasoning. Whereas a simple formalization of the *Egg Cracking Problem* required some 92 axioms and definitions [129], this entire problem is specified in just eight logical formulas that operate in conjunction with a simulation.

Similarly, this example illustrates the expressive advantages of hybrid reasoning over simulation used alone and over other ad-hoc approaches to reasoning with simulation. In particular, with social norms directly expressed in logic (including the transitivity rule), the behavior the system can be readily inspected and manipulated by examining the logical definitions.

Of course, the next stage in developing this example is to scale up to the complete problem. The many details and steps, described in the statement of the *Eating on an Airplane Problem*, mean that such development would involve a non-trivial effort, especially without tool support for rapidly building simulations. However, there does not appear to be any fundamental technical reason why the effort would not be a straightforward process.

Summary

Reasoning in this hybrid architecture combines both the flexibility and power of formal logics with the ease of specification and the implicit commonsense 'know-how' of simulations. Even in this simple example, the architecture elegantly and efficiently solves problems that are difficult to express using other methods.

8.1.4. Benchmark Evaluation: Learning and Control

In this section I seek to evaluate the integration of learning into the control strategy of Comirit. Autonomous acquisition of knowledge from the environment represents a 'holy grail' of artificial commonsense. Any knowledge that can be autonomously acquired or maintained through interaction with the environment results in corresponding saving in the costs of manual data entry.

The ultimate objective of autonomous learning is to acquire broad knowledge with the same (or better) speed, competence and ease as children. There is little understanding of how such remarkable skill can be translated to computer systems so there will remain a need for manual knowledge engineering for the foreseeable future. Nevertheless, it remains reasonable to expect that a sophisticated commonsense reasoning system should be able to autonomously populate and maintain at least some aspects of its own knowledge.

So while the creation of a fully-fledged learning system is beyond the scope of this dissertation, my objectives in this section are to demonstrate that autonomous learning in Comirit is, in principle, possible. In particular, I demonstrate that it is feasible for a system to autonomously acquire the parameters of simulations.

The Toy Box Problem

The problems in Miller and Morgenstern's [127] collection of commonsense reasoning benchmark problems are designed to be of moderate complexity: sufficiently challenging so as to be non-trivial but not so demanding as beyond the realm of present possibility. The problems are posed as sets of fixed scenarios in which an agent must deduce some consequence or decide an appropriate action. While the problems are designed to compare and contrast representational schemes, they are less well suited to testing an agent's ability to automatically discover such representations in a changing environment.

In order to evaluate the ability of Comirit to autonomously acquire models of unknown objects, I propose a new open-ended object learning problem, *The Toy Box Problem*:

> A robot is given a box of previously unseen toys. The design of the toys may vary in shape, appearance and construction materials. Some toys may be entirely novel, some toys may be identical and yet other toys may share certain characteristics (such as shape or construction materials). The robot has an opportunity to first play and experiment with the toys but is subsequently tested on its knowledge of the toys. It must predict the responses of new interactions with toys and the likely behavior of previously unseen toys made from similar materials or of similar shape or appearance.

This is an open-ended problem that concerns two kinds of learning. Not only must the robot learn how individual toys behave but also it must abstract general principles about the toys so that it can predict the behavior of previously unseen toys.

2D Toy Box Problem

The challenge involved in the *Toy Box Problem* is highly dependent on the complexity of the toys inside the box; toys could be simple solids or they could involve complex parts, electronics and motors.

My objective in this evaluation is only to demonstrate the principle that simulations can be learned from observation. I do not seek to precisely define the mechanisms by which every object in the real world may be autonomously learned. I therefore consider a simplification of the *Toy Box Problem* in which the world is only 2D and toys are soft solids of even density.

By approaching the *Toy Box Problem* in a 2D world, I avoid the technical challenges of 3D reconstruction and 3D model building which have been out-of-scope in this dissertation. In particular, by working in a 2D world, a model can be constructed directly by color segmentation of the 'camera' image, rather than requiring stereographic reconstruction.

Of course, the real world has three dimensions and so a 2D world must be entirely virtual.

Thus, I have constructed a simple 2D world in which objects are constructed from simple masses and springs (just like a Comirit Simulation). The world consists of four kinds of toys: boxes, balls, donuts and bananas. Instances of several such toys are depicted in Figure 8.7, including examples of how they can be deformed. The figure may be interpreted as a side-view in which the objects have been dropped on a table. Some objects are more rigid than others, this is why they have variations in deformation (*e.g.*, the box is a highly rigid solid, while the donut is quite flexible).

Note that even though the 'mass-spring' model that underlies the simulation of toys is the same mechanism used for representing internal beliefs of the system, the precise construction of the model is kept hidden from the learning algorithm. The learning algorithm is only able to observe the toys through a 2D camera that reduces the toys to a bit-mapped graphic and therefore masks the individual masses and springs that drive the virtual world.

Learning and Evaluation

For a particular toy, the learning problem involves a number of parameters:

- Three directly observed parameters of the toy: shape, size and color of the toy
- The known duration and magnitude of the force of the robot's actuators involved in creating the observation
- Two unknown parameters to be *learned*: the mass and elasticity of the object.

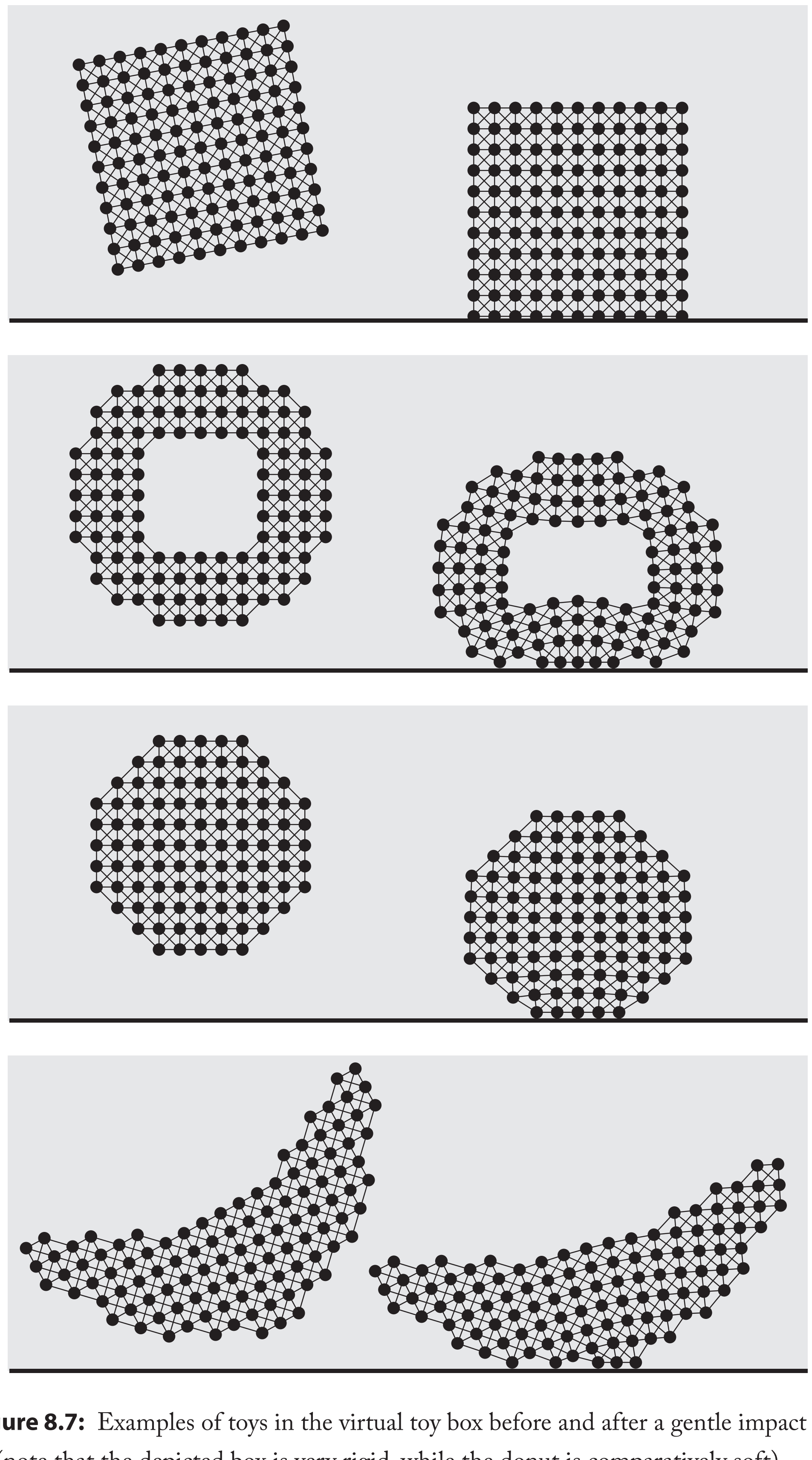


Figure 8.7: Examples of toys in the virtual toy box before and after a gentle impact (note that the depicted box is very rigid, while the donut is comparatively soft)

The system has the chance to interact with the toys and observe their physical responses. It must then predict the outcome of new interactions and novel toys.

In Comirit, the system learns by hypothesizing predictive models of the observation and refining its hypothesis over time.

Recall that autonomous learning in Comirit is a two stage process (Section 7.1):

1. Constructing graph-based approximations from observations
2. Learning values for annotations by observation

In this simplified problem, graph-based approximations can be generated directly from a camera view. The virtual world is projected into a 2D bit-mapped image which is then segmented and converted to mesh of points. This mesh is used by Comirit to approximate the structure of the image. Comirit then performs a numerical search for the values of the annotations on that mesh.

Since the predictive models of the system will not precisely match the true models, it is not possible to evaluate the learning performance by matching the values of the learned parameters. For example, a 500g toy may be accurately modeled as a mesh of 100 × 5g masses or 500 × 1g masses. Instead, the predictive model is evaluated as a 'black box' by using it to predict the behaviors of previously unseen objects or forces. The accuracy of the prediction is then quantified by projecting the predicted model into a 2D camera image and comparing the individual pixels of the predicted and actual camera images (measured as a percentage of total pixels used to render the object). This process is depicted in Figure 8.8. Note that pixel-by-pixel accuracy is a highly demanding criterion: it has no tolerance even for errors involving pixels rendered just one cell away from the correct location.

The Hypothesis Space

Learning in Comirit is modeled as a hypothesis search. Since the *Toy Box Problem* concerns both the behavior of individual instances and the underlying structure of the problem domain, the hypothesis space requires a two-layered representation combining beliefs about specific instances with beliefs about problem structure.

- **Beliefs about individual instances**

  The system constructs models of objects that it perceives and it must learn the annotations of these models. These beliefs are represented as a vector of values that are refined through stochastic hill-climbing.

- **Beliefs about the problem structure**

  The underlying structure of the problem space requires a sophisticated representation that can generalize knowledge across instances. A novel toy with a similar shape or appearance

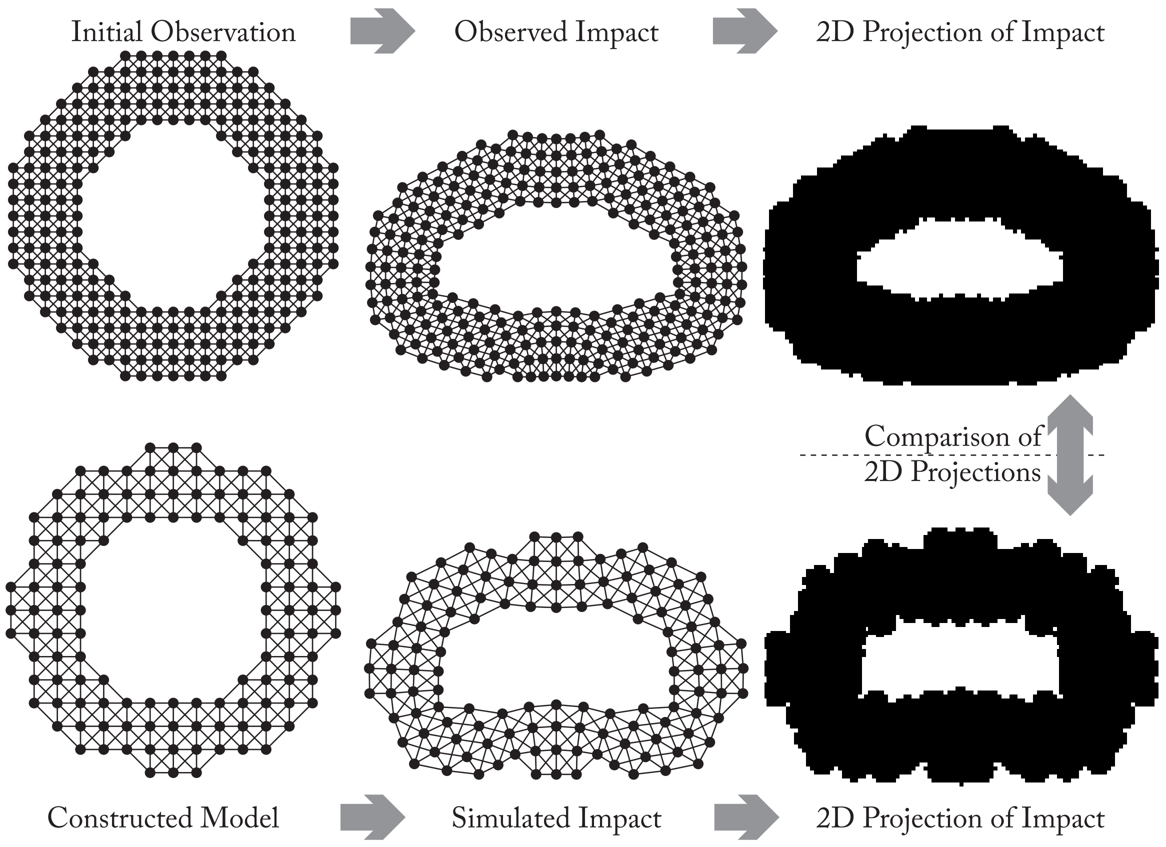


Figure 8.8: Measuring the accuracy of a predictive model using a 2D bit-mapped projection

as a previously observed toy should initially be assumed to have similar behaviors. The Self-Organizing Map (SOM) [91] is a data structure that clusters vectors by similarity into a grid and, as such, is one possible representation for storing abstract knowledge of the underlying problem space.

These two belief representations are combined into a single hypothesis space: at any given time, the system hypothesis is a set that contains vectors for the annotations of observable toys and a SOM of annotation vectors for the entire problem space.

I evaluate each of these layers of the hypothesis space in turn.

Beliefs about Individual Instances

Since the graph-based models of an individual toy may be constructed directly by segmenting the camera and approximating the shape by a 2D mesh, the learning problem for an individual instance concerns only the discovery of a vector of parameters. Thus, the hypothesis space for learning indi-

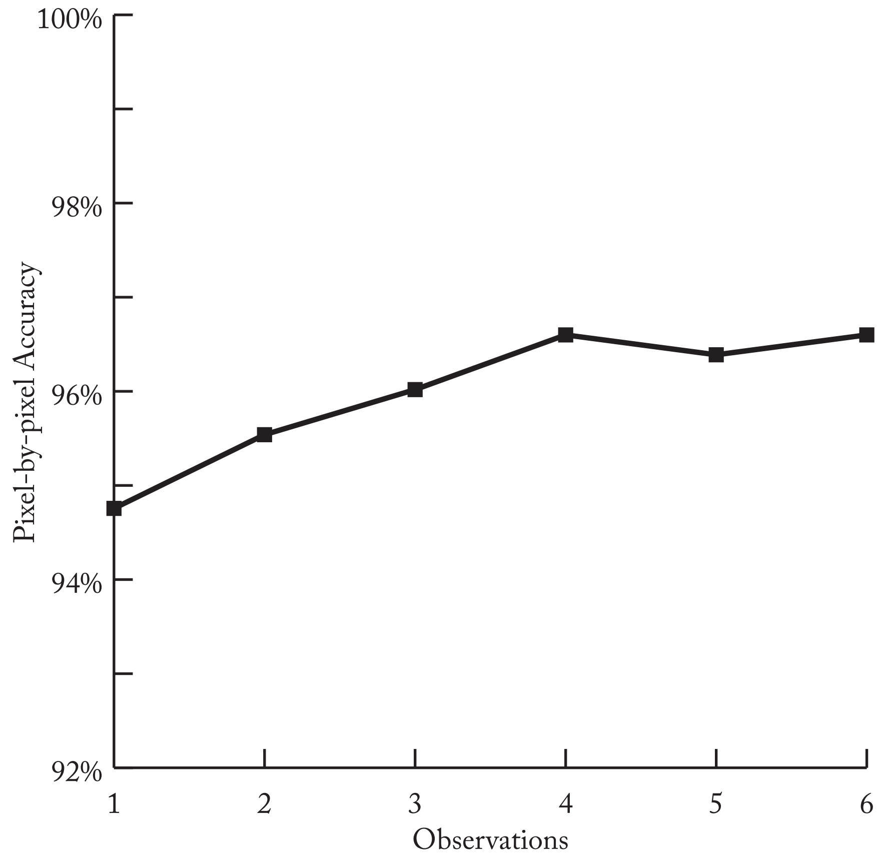


Figure 8.9: The predictive accuracy of models learned from observations of a single toy (using a simple hill-climbing stochastic search)

vidual instances is a space of numerical vectors comprising a weight (of masses in the mesh) and a spring constant (the elasticity of the bonds).

I use a simple learning algorithm in this evaluation: hill-climbing stochastic search. For each observation, the learning algorithm performs several rounds of hypothesis generation and selection. Hypotheses are blindly generated from random permutations of the current hypothesis and selection is performed according to the hypothesis that most closely predicts the known observations.

Astonishingly, with just a *single* pair of observations (*i.e.*, the system observes an image of the toy before and after an interaction) and no prior knowledge, the system is able to learn annotations that predict behavior (in unseen situations) to 95% accuracy (see Figure 8.9). These low error rates are achieved despite the different resolutions used to create the underlying mesh for the simulation. That is, the learning algorithm is robust to variation in the underlying graph-based structure used to model the observation (*e.g.*, if the underlying mesh is very sparse, the search automatically compensates by finding larger mass annotations).

In Figure 8.9, it is clear that additional observations slightly improve the accuracy of simulation; the observations eliminate the slight over-fitting that occur with a small number of training examples. However, the magnitude of these improvements is minor. In fact, the system is converging on the optimal values of the parameters; the process of projecting models into a 2D camera image means that it is simply not possible for predicted models to match observations with 100% accuracy if the underlying mass-spring models differ.

This is an extremely rapid learning rate. It stems from the fact that a single image comprises many pixel observations and is, as such, highly informative. Compare this to day-to-day experiences in real life. It takes a person little more than a momentary interaction with an object to gain a useful understanding of its behavior, to determine if an object feels light, solid, fragile, rubbery, rigid, flexible, hard or soft.

Beliefs about the Problem Space

Given that a single observation is sufficient to learn an extremely precise model of an object, it is difficult to evaluate the incremental benefit that comes from modeling the problem-space itself.

Thus, in order to show the benefits of learning a problem space, I artificially reduce the learning rate (by using a less aggressive hill-climbing strategy) and use only balls and boxes (these regular shapes are less informative to a learning algorithm). In this weakened mode, a single observation achieves only 35% accuracy and 8 pairs of observations are required to reach 90% accuracy.

Now, in order to test the ability of Comirit to generalize experience, I structure the problem domain so that there are correlations between the annotations of particular toys:

- Toy color is inversely correlated with elasticity. This is similar to the real life tendency where shiny metallic objects are usually rigid.
- Heavy (dense) toys are generally inelastic, as is often observed in real life.
- Round objects are generally inelastic. This is analogous to correlations between shape and behavior in real life: a mug is generally rigid so that it may safely hold hot liquids.

The system then attempts to learn this structure of the problem domain, by observing and learning about particular instances of toys and generalizing this knowledge into a Self-Organizing Map.

A Self-Organizing Map [91] is an auto-associative neural network formed from a 2D grid (or map) of vectors. A SOM is first initialized with random vectors. Training vectors are then 'learned' by searching for the vector in the SOM that most closely matches the training vector. The SOM is updated so that the similar vector, and its neighborhood, is adjusted to be closer to the training vector (this learning is performed at a decaying rate and over a diminishing neighborhood). Eventually the SOM converges to produce a similarity-preserving, low-dimensional map of the space of training vectors. Initial hypotheses for novel objects may be generated by searching the map for vectors that closely match observable parameters and by using the corresponding unobservable parameters as the initial annotations.

When a new object is first encountered, the system will be able to directly observe superficial and observable properties (such as size, shape, color and texture) and search the SOM for closely matching vectors. That 'winning' candidate is then used as an initial hypothesis about the unobservable parameters of the novel object. The system can use the initial values to roughly reason about the object and evaluate actions. When it later observes real responses of the object to interactions, it uses stochastic hill-climbing to refine its beliefs about the unobservable parameters and propagate these values back into the SOM by a SOM update.

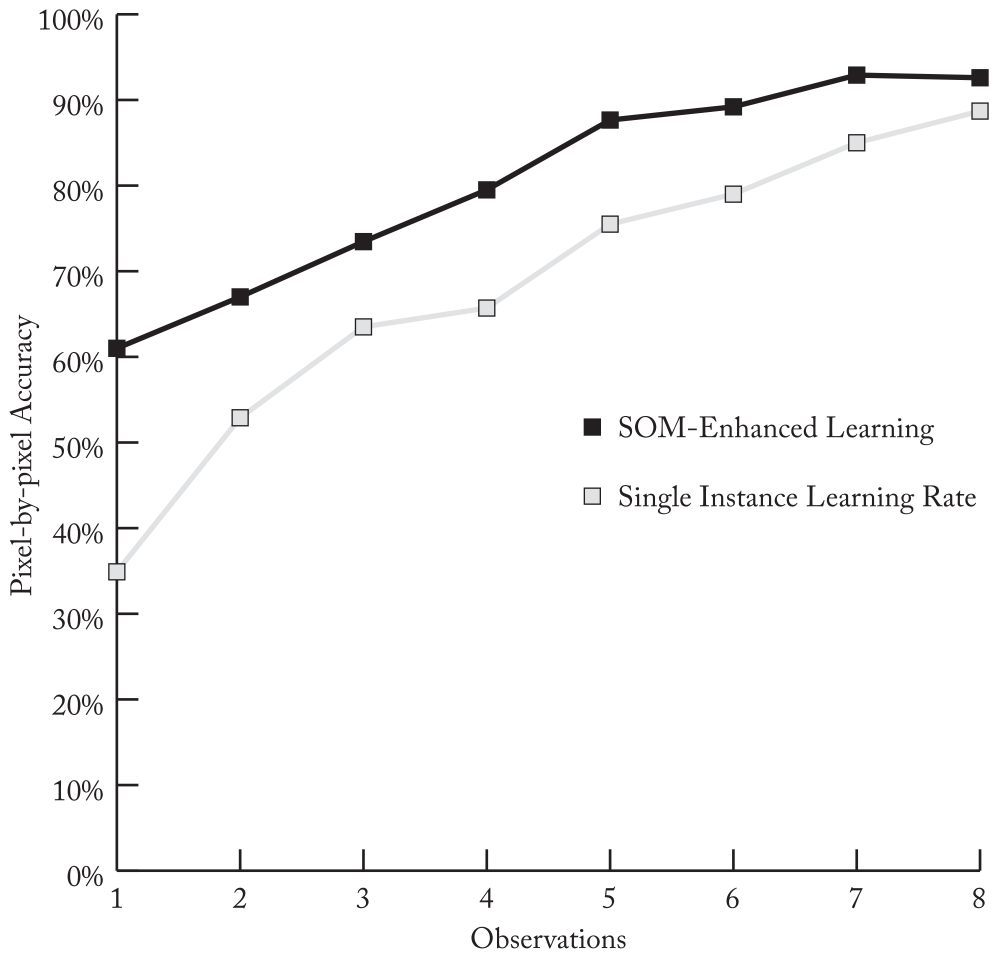


Figure 8.10: The predictive accuracy of models learned using a Self-Organizing Map to generalize knowledge across instances, compared to learning each instance separately without a map

When the SOM is randomly initialized, it achieves less than 5% accuracy in the toy room tests. However, after observing 14 objects once each, it achieved 61% pixel-by-pixel accuracy across all toys. That is, the SOM is generalizing knowledge across objects so that it learns the behavior of individual objects faster. With just six sets of observations (rather than eight), the system reaches 90% accuracy, even though the SOM is itself also learning. This is illustrated in Figure 8.10.

The increased rate of learning demonstrates that the system is generalizing its experiences across related objects and is producing better initial hypotheses for novel objects, even during the earliest learning phases.

Meanwhile, the ability of the framework to rapidly learn the particulars of a given instance from very few observations prevents it from being committed to an hypothesis generated by an SOM when it encounters 'outlier' or unusual objects.

Future Extensions

I consider these trials as early demonstrations of the soundness of the basic concept and framework for autonomous learning. There is, nevertheless, much potential in applying more sophisticated learning algorithms and more challenging domains. In future, I will move beyond the virtual world of these preliminary experiments. I will identify and adapt a robust 3D reconstruction technology and run these experiments on real world objects within real world settings. I have used a naïve implementation of a Self-Organizing Map because I expect to make major changes when I fine-tune the technique to real world problems. I am, however, extremely encouraged by the success of the naïve implementation. Likely extensions to the SOM include larger vectors for more complex annotations, general descriptions of arbitrary shapes, affordance-based indexing and using the SOM to assist in reconstructing parts of known objects that have been visually obscured.

Conclusions

Not only are parameter-driven simulations an effective mechanism for rich commonsense reasoning but also they lend themselves to rapid and autonomous acquisition. I have shown how models of learning can be elegantly incorporated into the Comirit Framework (and potentially other tableau-based systems). The extended framework combines the effectiveness and efficiency of simulation with autonomous knowledge acquisition as well as with the full power and generality of logical formalisms.

To date, I have demonstrated the system on simple (but plausible) learning problems. Early experimental results are extremely encouraging: the system learns rapidly and from very few observations.

As a long term goal, I plan to have Comirit autonomously learn even the fundamental laws of a simulation and the mechanisms for model building. In particular, I hope that interaction in a complex environment (such as the 3D world) may be modeled as a problem of determining the hierarchical non-linear flow of 'particles' (or its equivalent in other non-physical simulations) within an environment. To whatever degree such automation is possible, the framework can accommodate any learning that can be expressed as an optimization problem, while allowing for difficult knowledge to be manually expressed in logical form or as hand-built simulations.

8.1.5. Correspondences

Comirit is neither intended as a biologically plausible model of common sense nor has it been designed as a cognitive architecture: my objectives are purely functional. Nevertheless, it may be useful to analyze the Comirit Framework as though it were a cognitive architecture or a model of cognition.

There are, of course, many models of cognition including analogy-making (such as in Hofstadter's Copycat [75]), abstract models of the mind's functions (such as in Soar [96]) and biological models of low-level simulations of neural activity [113]. With so many models, there is a danger of critical comparison being meaningless: it would be surprising to create a system that did *not* correspond at some level of abstraction to at least one model of cognition. Nevertheless, it is useful to explore the connection between Comirit and a cognitive architecture, because doing so can reveal structures or representations that are either missing or potentially unnecessary.

In this Section, I consider the evolution-inspired cognitive model proposed by Peter Gärdenfors [51, 48, 47]. The model is attractive for comparison with Comirit because it focuses on representational capabilities, rather than specific 'mind modules', physical forms or mechanisms. Comirit is a *framework* for implementing commonsense reasoning capabilities so it does not make sense to compare specific lists of modules. Such lists would be more useful in analyzing a given concrete instance of a Comirit System, rather than the framework.

Gärdenfors' model assumes that an agent is a self-contained autonomous system which interacts with its environment and that it is the processes involved in formation and management of the agent's inner mental worlds that are essential to understanding higher level cognition. That is, Gärdenfors' model is concerned with the kinds of knowledge represented within the inner world of an agent and the process by which this knowledge is mapped to, or grounded with, the outer world.

The model is inspired by the animal kingdom and an apparent hierarchy of cognitive abilities. All animals exhibit similar kinds of low-level abilities but more cognitively-aware animals demonstrate evidence of increasingly greater cognitive faculties. These steps correspond to cognitive building blocks in the model.

Briefly speaking, all animals—even the simplest organisms—have sensations that report on both the environment and the body, allowing the creature to reactively respond to its surroundings. Evidence can be seen in more advanced animals that those sensations are interpreted and transformed into perceptions of what is going on in the agent's immediate surroundings. Mammals and birds demonstrate evidence of predictions and simulations in their inner world that they can use to direct their thinking about goals or objects that are not immediately perceived in the surrounding environment. Conceptions and plans represent the highest level of cognition most clearly found in humans and other primates. These concise representations of ideas and plans for action are built from perceptions and simulations and are important for communication, collaboration and abstract reasoning.

That is, Gärdenfors cognitive model proposes the following interrelated representations that inhabit an agent's inner world (see also Figure 8.11):

- **Non-representational**
    - **Sensations**: the immediate sense of what is happening right now in the agent's world and to the agent's body, forming the foundation of representational knowledge

- **Representational**
    - **Perceptions**: the interpreted or processed sensory impressions
    - **Simulations**: the perceptual experiences of possible or probable scenarios formed within an agent's inner world
    - **Conceptions**: the concise representations of classes of entities and/or individuals which may exist in either the inner (introspective entities) or outer world (actual entities)
    - **Plans**: the recipes for actions (that are represented before they are executed, in contrast to trial-and-error behavior)

With this model framing an analysis of the Comirit Framework, it becomes clear that there is a close correspondence and coverage of the dimensions that Gärdenfors has identified. Indeed, perceptions, simulations, symbolic conceptions and plans are all represented in Comirit. I analyze each dimension over the remainder of this section.

Sensations

In a robotic or information-agent based implementation of Comirit, sensations are low-level or minimal transformations of direct sensor readings.

Ideally, the engineer's role in assigning interpretations or meanings to sensory data should be minor. Sensations that have been significantly transformed and abstracted can limit the adaptability

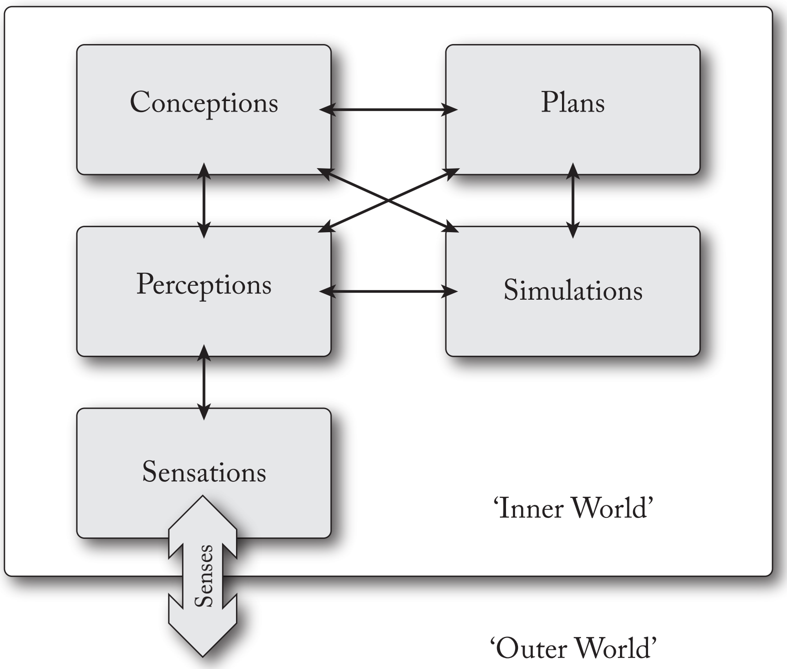


Figure 8.11: Core representations of Gärdenfors' cognitive framework

of the agent. Indeed, the engineer's role in assigning meanings to sensory data is the fundamental issue of the Symbol Grounding Problem [65]. For example, in a system where a camera (or sensory subsystem) is programmed to report sensations of the location of brightly colored balls, no amount of higher level abstract reasoning could be used by the agent to adapt to new problems involving the identification of cubes or of people.

The graph-based representations used by simulation were chosen, in part, because of their generic and iconic nature. Sophisticated transformations of sensations are not required to construct useful simulations: many attributes of a simulation can be acquired and configured directly from observation (as was seen in the *Toy Box Problem* of Section 8.1.4).

In this dissertation, I do not prescribe a concrete design for sensations because sensations are dictated by the design objectives and environment of the agent. Ideally, sensations should be as richly detailed and free of design bias as possible but the practical realities of time, space and computational limitations will require a degree of hard-coded sensory processing that the agent cannot itself meaningfully introspect.

Perceptions

Perceptions are the representation of the inner world created by the fusion of sensory data. In Comirit, all perceptions are represented by graphs and the annotations on such graphs. While I certainly do not claim that the particular graphical representation that I use in Comirit resembles the mental architecture of perceptions or 'mental imagery' (see also Section 5.2.7), it is the case that the functional role of the graph-based representation matches the role of perceptions.

In fact, Graph-based structures were chosen because they provide a bridge between raw sensory input, simulations and conceptions. The translation of images into graph-based models is an easier problem to solve than conception and categorization—in physical settings this is especially true if graphical models are built by direct 3D scanning. The benefit of simulation is that it operates directly on such graph structures.

Furthermore, the cognitive science literature suggests that perceptions may in fact be just special cases of simulations [7], as is the case in the Comirit Framework.

Perhaps the translation of sensations into perceptions represents a present weakness of the Comirit Framework. To date, I have assumed such translations are performed outside Comirit so that the sensory input into Comirit closely resembles perceptions. Where Comirit is applied to information agents that have semi-structured and symbolic inputs, the distinction between sensation and perception becomes minimal and therefore it is less crucial that this mapping be explicitly represented. Nevertheless, the formalization of this aspect of Gärdenfors' model remains an interesting direction for future work.

Simulations

Simulations are detached perceptions of predicted or imagined scenarios. They map directly to the graph-based simulations of Comirit. A simulation in Comirit is created by applying simulation rules to a graph-based representation of a perception.

In the context of Gärdenfors' model, simulation is one of the strengths of Comirit. It is a core representation of the framework and its operation over graph-based structures ensures that the mechanism is robust to noise, error and unforeseen situations. If an object can be perceived in a graph-based form, it can be simulated. In contrast, in logical formalizations it may not be possible to simulate an unknown object, if it cannot be assigned a symbolic categorization.

Conceptions

Conceptions are concise representations of entities in the inner or outer world. For example, when we observe a cat, we perceive it as small entity with four legs, fur and a gentle purr. However, the identification of those perceptions as a *cat* is a conception.

Symbols and symbolic terms are a conceptual representation supported by Comirit. Simple conceptualization is implemented in Comirit by tableau rules that translate from the graph-structures of simulation to symbolic forms (see Section 6.4.2). These are manually engineered rules that analyze the structure of a simulation and generate appropriate symbols to describe the structure. For example, the conceptualization of an egg as being cracked is achieved by a rule that determines if some of the connecting bonds in a simulation of an egg are no longer intact.

More sophisticated conceptualization is possible by the self-organizing maps (SOM) outlined in Section 8.1.4. Where a rich and complex entity is mapped to a coordinate in the low-dimensional space of an SOM, the coordinates may themselves be treated as an opaque symbol for higher-level processing and symbolic reasoning, similar to the 'Knoxels' of Chella *et al.* [25]. For example, an unknown object that maps to the vector with coordinates (182,32) on an SOM, may be denoted by the symbol *SOM(182,32)*. The simulation parameters of the object *SOM(182,32)* can then be learned and the agent may draw other symbolic assertions. If the agent knows that 'fragile' means easily 'broken', it may construct simulations to test whether objects of type *SOM(182,32)* can be easily broken and may then assert $\forall\, x : SOM(182{,}32) \bullet fragile(x)$. In fact, it may eventually be possible to automatically correlate unknown symbols (such as unrecognized words in speech) with active coordinates in the self-organizing map to autonomously learn groundings for such new symbols.

Self-organizing maps are also similar to the *conceptual spaces* that are also posited by Gärdenfors [50, 48] as part of his cognitive framework. A conceptual space is a region with axes describing properties or qualities. A real-world object (*e.g.*, a particular cat) corresponds to a point in the complex multi-dimensional conceptual space. A concept is formed from convex regions in this space (*e.g.*, a space encompassing all kinds of cats). In the conceptual spaces formalism, the multidimensional space of properties is complex and does not necessarily represent a single metric space. Self organizing maps have a similar structure with entities mapping to points in a space and with nearby points representing similar entities.

Where an agent is required to learn the conceptualization of dynamic changes and ongoing processes, this approach may be extended to use recurrent self-organizing maps [168] or by incorporating temporal information into the training vectors of the SOM. In practice, I anticipate that it is less crucial for dynamic concepts to be autonomously learned. For example, while a domestic cleaning robot must be able to learn and conceptualize any new object that is brought into the home, it is less important that the robot be able to understand new processes that the object might enable. Indeed, even if it were able to recognize and conceptualize a new process (such as the unique 'dance' of a new toy robot), it is unlikely to see how the agent may be able to appropriately respond to it.

Comirit therefore supports effective methods of conceptualization and usable representations of conceptions but it is by no means close to the capabilities of a human being. Conceptualization remains a long standing challenge of Artificial Intelligence and it lies at the heart of the symbol-grounding problem. While future work is, and will always be, required in this area, the Comirit Framework serves as a solid foundation for effective commonsense today and for incorporating future innovation.

Plans

Planning is a process by which an agent searches the space of plausible simulations for actions in which its objectives or goals are maximized with high likelihood. If the agent's objectives or goals are non-trivial, then the preferred and most likely subspaces will tend to be scenarios in which the agent has enacted a change; planning, therefore, is a process of generating scripts for action.

In Comirit, planning may be recast as an attempt to constructively prove that there exist a series of actions by the agent that result in or maximize the agent's objectives or goals. The hybrid reasoning architecture of Chapter 6, being fundamentally based on the *constructive* proof-search of the tableau method, subsumes the need for explicit planning routines.

Of course, while proof-search will eventually discover appropriate models, and therefore implicitly construct appropriate plans, efficiency is an important concern for real-world agents. Modern planning and game-playing techniques could eventually be integrated into Comirit as meta-rules and heuristics (using the methods described in Section 6.3.4).

8.2. Cost-Effectiveness

The pragmatic objectives of this research project mean that cost-effectiveness has played a central consideration in its design and in the selection of design trade-offs. The question of whether the Comirit Framework is a cost-effective platform for constructing commonsense reasoning systems is difficult to answer without first implementing a complete system. However, the advantages of Comirit may be understood by extrapolating from prototypes of Comirit and from the efforts of related projects.

8.2.1. Extrapolation from Prototypes

In the process of developing and refining Comirit, I started 22 separate prototypes, each of which was subject to many rounds of re-factoring and re-architecting. In large part, these attempts were an effort to fully explore the space of possible designs. However, my willingness to start afresh so regularly was motivated by a philosophy that it should be possible to implement the core of a *good design* in weeks or even days, rather than months or years.

In Table 8.3, I have summarized the software development efforts that I have expended directly on Comirit over this dissertation. I only have records of days, rather than the individual hours spent

Version	Code Name	Weeks	Language	Reason for Implementation
1	Mind	4	Java	Simulation
2	Simulator	5.5	Java	Simulation reimplementation
3	Simulator2/Slick	4	Java	Simulation reimplementation
4	Simulator3	5	Java	Simulation reimplementation
5	SimulatorProlog	0.5	Java/Prolog	Prolog integration
6	SimulatorProlog2	1	Java/Prolog	Prolog integration
7	SlickProlog	2	Prolog	Pure-Prolog version
8	SlickLibrary	3	Java/Prolog	Prolog integration
9	Slick2Test	2	Java/Prolog	Prolog/tableau integration
10	CycClient	3	Java/Cyc	Cyc integration
11	Blacktable	6	Java	Tableau-based hybrid reasoning
12	Blacktable2	10	Java	Tableau reimplementation
13	Blacktable3	1	Java	Tableau reimplementation
14	PhysicsLab	1.5	Java	Improved low-level representations of physical simulations
15	Vision	5	Java	Reconstructing shapes from visual input
16	Maps	3	Java	Learning with Kohonen maps
17	Bouncer	9	Java	Robust learning in simulated environ-ments
18	Vision2	3	Java	Learning simulations from visual input
19	Vision3	2	Java	Learning simulations from visual input
20	Vision4	4	Java	Learning simulations from visual input
21	Learner	4	Erlang	Learning simulations
22	Experiment	2	Erlang	Learning simulations reimplementation
Total		**80.5**		

Table 8.3: The labor involved in developing prototypes of Comirit

during development. However, given a typical loading of other commitments and non-programming research, these durations represent only a 50% of full-time workload.

Clearly, a basic implementation of the Comirit Framework in any mainstream programming language should be possible within 2–10 weeks. In an extreme worst-case scenario, this should be no more than 1.5 person-years of solid development at 50% full-time load—the total time that I spent developing all prototypes (both failed and successful). Such a pessimistic worst-case estimate ignores the value of this dissertation in providing a workable and readily implemented design.

Throughout this dissertation, I discussed many opportunities for optimization and improving efficiency. In small scale problems, optimization may not be necessary. Inefficiencies in moderate-sized problems may be addressed by brute-force: distributing the framework over a cluster of computers. Indeed, the side-effect free nature of tableau reasoning combined with spatial indexing of simulations would make this comparatively straightforward. Large scale problems will inevitably require powerful indexes and heuristics that will demand additional development effort. Nevertheless, the design incorporates facilities for indexing and optimization so that such work represents an incremental addition, rather than a broad redesign. Tree-based indexes are not difficult to implement—they are part of the standard library in some languages—so it is extremely unlikely that converting a solid unoptimized Comirit Framework into an optimized Framework would demand more than a person-year of development time, even in a worst-case scenario.

Of course, these estimates do not address the engineering effort involved in populating instances of Comirit with commonsense simulations. In part, this effort will be dramatically reduced with suitable tools and by autonomous learning. However, an upper bound on these costs may be estimated by examining the costs that large game-development studios incur in developing the open-ended simulations used in games. I examine these costs in the next section.

8.2.2. Costs of Similar Efforts

In Section 5.1, I viewed the success and cost-effectiveness of modern computer games, as (partial) motivation for the use of simulation as a mechanism for commonsense reasoning. Given the similarity between open-ended computer games and simulation, the development costs of computer games may serve to indicate the likely costs of populating an open-ended commonsense reasoning system.

Unfortunately, while game studios are open about their sales figures and profits, they do not tend to publicize the production costs of their games because these costs are a commercial secret. However, production costs can be inferred from public financial reports and published statements about team sizes.

EA Games is a multi-billion dollar computer game studio, listed on the NASDAQ, and responsible for game franchises including *The Sims*, *Need for Speed*, *Command and Conquer* and *Spore*. According to its 2009 annual report [38, 39], it released 31 titles over the year, with a cost of goods sold of US$2,127 million. Taking this as typical of annual production, it follows that the average development cost of a blockbuster computer game is in the order of US$70 million. For Ubisoft, listed on Euronext Paris, comparable development costs of approximately US$30 million can be inferred from their 2008 financial report [167].

Individual console technology operates in cycles of four to six years and games are often connected with the production cycles of film studios [38]; thus games must be developed within one to three years. Longer development time-frames typically get mocked as 'vaporware'. Consider, for example, the twelve year production schedule for *Duke Nukem Forever* which earned it widespread derision for its endless delays [8].

Of course, a multi-million dollar budget, spent over a one-to-three-year period, represents a significant cost and risk but is not out of the realm of possibility, when viewed in contrast to the billion-dollar production pipelines for drug production in the pharmaceutical industry. Furthermore, these statistics represent the costs of highly polished games involving royalties to voice-actors, polished graphics, careful story-telling, licensing fees, extensive testing on multiple platforms, internationalization and delivery of the product to an international audience.

Small teams with start-up mentalities can produce rich virtual worlds with resources that are, by comparison, minuscule. Consider the rich open-ended 3D world of space combat in the online game *Vendetta Online*. The game is widely recognized as an example of the success that is possible by a small independent publisher. A team of four developed the rich and immersive virtual world from the basement of a home within some four years [62]. With minimal overheads or operating expenses, it is unlikely that this privately held company would have spent more than US$1 million on development, this representing living costs of approximately US$60,000 per person per annum.

Simulations for commonsense reasoning do not need to be highly polished. They do not need realistic texture mapping or appealing graphical effects. If one is willing to sacrifice polish and to use third-party 3D models, it may be possible to develop simulations in less time and with fewer resources. There are widespread 3D modeling resources online: many amateur designers offer 3D models in commercial and open-source 3D file formats on their web-pages, while other sites aggregate such models into convenient free or by-subscription repositories, modeled on the stock-photography business model. Indeed, the demand from architects and interior designers for off-the-shelf 3D models has resulted in widespread availability of exactly those day-to-day models that would be required for simulating situations encountered within a home or a city-scape. With appropriate translation tools, it would be trivial to convert such 3D polygonal models into the graph-based representations of Comirit.

It seems reasonable to conclude that a relatively complete and robust implementation of Comirit, like *Vendetta Online*, may be implemented in a budget of approximately US$1 million. Early release of the commercial applications, listed in Section 1.2, may fund further development and thereby enrich the sophistication of those simulations until they reach or exceed the level of polish in modern computer games.

While these comparisons are useful for estimating the cost of developing simulations, they do not address the costs of logical knowledge engineering in a hybrid reasoning scheme. Comirit has been designed with the intention that knowledge should be expressed in simulation *wherever* possible. The extent to which logical knowledge is required remains unclear. However, I am confident that the vast majority of routine commonsense knowledge can be expressed in simulation using standard software engineering techniques, with logical formalisms taking only a small role in enabling the simulations to be exploited in more general query modes (such as back-chaining). Furthermore, it may be possible to draw the logical knowledge that is required from preexisting knowledge-bases such as OpenCyc [30].

8.3. Non-speculative Technologies

The pragmatic objectives of this research project have also influenced the technologies that underpin the design of Comirit. The claims that I have made within this dissertation do not assume any spectacular developments in Artificial Intelligence research, software design or hardware capabilities. It is easy to demonstrate the non-speculative nature of Comirit, simply by examining the design itself: it makes no unreasonable assumptions and it has been implemented in prototype form. I consider these factors in the following two subsections.

8.3.1. Technologies in the Design

While I have designed the Comirit Framework as an open-ended platform that may incorporate new innovations as they are invented, the benchmark evaluations of Section 8.1.2–8.1.4 show that simulation, logic and learning are sufficient for practical commonsense reasoning.

The design and working prototypes discussed in this dissertation and evaluated in this Chapter are grounded only in well understood and non-speculative technologies:

- Simulations are constructed from graph-based structures with functions that perform local updates. Physical simulations are implemented by translating mass-spring models to the graph-based structures of Comirit. Other simulations, driven or approximated by well-understood laws, may similarly be translated to the graph-based structures.

- Reasoning is performed by a generalization of the method of analytic tableaux. This method is over 50 years old and today plays an important role in reasoning on the Semantic Web [31]. Systems of logical rules traditionally used in tableau reasoning can be implemented in Comirit. The tableau method has been generalized in Comirit to allow hybrid reasoning but neither reduces expressiveness nor fundamentally changes the character of tableau reasoning.
- The proposal for autonomous learning remains the most speculative dimension of the framework. I have demonstrated the successful learning of novel objects using established learning algorithms (*i.e.*, greedy search and Kohonen networks). While it may prove to be difficult to scale autonomous learning to handle complex or distracting environments, the mechanism may still be useful to reduce the costs of knowledge engineering in controlled environments. In any case, learning is not a crucial part of the framework: it can be used where it reduces costs and ignored where it is of no use. In fact, the extensions designed for learning may still be used for other purposes, such as action selection in reward-driven environments (as discussed in Section 7.7).

8.3.2. Prototypes

The evaluations of Sections 8.1.2–8.1.4 would not have been possible without the development of prototypes. The development of these prototypes serves as clear evidence that the Comirit Framework is not pure speculation. Indeed, the 22 iterations of software development (described in Section 8.2.1) were necessary to reduce the framework to a design that can be readily and practically implemented.

8.4. Conclusion

While the evaluation of *any* concrete commonsense reasoning system is, in part, foiled by the difficulty of performing objective quantitative evaluations, an honest multi-pronged approach can offer valid insight into the capabilities and effectiveness of the system.

In this Chapter, I have therefore approached evaluation by presenting seven arguments about the capabilities, cost-effectiveness and non-speculative nature of Comirit. The combined weight of evidence demonstrates the merits of the Comirit Framework. That is, Comirit can be used to perform useful and accurate reasoning on several commonsense reasoning benchmark problems. It can be built cost-effectively. It can be built from technologies and principles that are well within the current art in Artificial Intelligence and Computer Science. Comirit, therefore, serves as a constructive affirmation of my thesis that *artificial commonsense can be created within constrained time and resources, from present-day technologies*.

Chapter 9
Conclusion and Future Work

In Chapter 8, I demonstrated that Comirit is real progress towards the creation of artificial commonsense and serves to validate my thesis that *artificial commonsense can be created within constrained time and resources, from present-day technologies*. The purpose of this final chapter is to step back from the technology and reconsider my original objectives.

Starting within the firmly theoretical foundations of the symbol-grounding problem, I approached the challenge of commonsense reasoning with a pragmatic attitude, sacrificing completeness and accuracy wherever they did not serve the needs of commonsense reasoning. This perspective has been fruitful, the hybrid system combining simulation, logic and learning that I have proposed is vastly different to what might be expected from a purely theoretical endeavor.

While Pat Hayes probably did not have non-formal systems, such as Comirit, in mind when he authored his Naïve Physics Manifestos [69, 70], this work may be seen as a continuation of his philosophy, namely that we should draw on everything we know to go ahead and actually build *real* commonsense reasoning systems. From this perspective, we can look back to see how to improve the theory and foundations of the field, rather than purely speculating about 'castles in the air'. Indeed, this pragmatic paradigm appears to be a growing trend in scientific research; consider, for example, how applied technologies have changed the nature of biology, spawning the entirely new field of biotechnology.

In the remainder of this dissertation, I analyze each of my original objectives to show that I have consistently followed and delivered on my research agenda. I then conclude with a view towards possibilities for future work, including new research problems and a long term vision.

9.1. Dissertation Objectives

In Sections 1.1 and 2.7, I outlined the contributions and research gap that would characterize the work described in this dissertation. I now revisit each characteristic to argue that I have indeed followed through on my original ambitions.

9.1.1. Primary Contribution

In Section 1.1, I stated that the primary contribution of this dissertation is to motivate, propose and validate an approach for implementing artificial commonsense reasoning.

Of course, I am not the first to propose an approach to artificial commonsense reasoning (I summarized many such efforts Chapter 2). However, my work is distinguished by an emphasis on deep reasoning that remains grounded in the pragmatic realities of concrete implementation. In this regard, consider again my claims of Section 1.1 about the unique character of this primary contribution:

1. **Common sense reasoning, rather than common sense knowledge**

   Knowledge representation in Comirit is principally centred around the use of simulations that embody real-life commonsense scenarios. Simulations provide deep commonsense 'know-how' and allow a system to reason about probable cases and consequences of actions. While Comirit may eventually incorporate mechanisms for fast recall and manipulation of vast databases of commonsense facts and knowledge, the framework currently emphasizes mechanisms for implementing 'know-how' rather than 'know-what'.

2. **Computational efficiency, rather than computational accuracy or formal completeness**

   The numerical simulations that feature prominently in Comirit are constructed as approximations of reality. They are 'good enough' for practical commonsense reasoning, though they might perform poorly as a source of *expert* knowledge, and have the advantage of being extremely efficient and reliable reasoning methods.

3. **Ease of use, rather than excessive intellectual and educational demands on knowledge engineers**

   The representations used in Comirit, and simulations in particular, are relatively well understood by most software engineers. This is in stark contrast to the advanced training in formal logic and philosophy that is required of a knowledge engineer working on a logic-based system.

4. **Open-ended architectures, rather than closed and complete systems**

   The Comirit Framework is designed as an *open-ended* combination of simulation with logic and machine learning. Other methods can also be integrated into Comirit in the same way that these existing methods have been integrated. In particular, methods that are readily mapped into a search over spaces of possible worlds are likely to present the least troublesome integration.

5. **Short- or medium-term, rather than long-term development time frames**

   The Comirit architecture represents a novel combination of well understood technologies. I demonstrated in Chapter 8 that a Comirit system can be built with a limited budget and without ongoing fundamental research activity. Furthermore, the simplicity of simulation as a representation, and the learning capabilities of the system, help ensure that knowledge engineering is not an expensive or drawn-out process.

6. **Reusable components, rather than fully autonomous systems**

   Comirit is not a complete software or robotic system. An implementation of the Comirit Framework provides reasoning, learning and planning services. A software engineer must embed Comirit into a functioning system. For example, the engineer may use the system as a query engine, providing commonsense reasoning services in a manner analogous to the data retrieval services of a database system.

7. **Commonsense, not general intelligence or 'Strong AI'**

   Throughout this dissertation, I have emphasized the comparatively modest goal of achieving useful and practical commonsense reasoning capabilities. Comirit exploits simulation and hybrid reasoning to provide practical answers in an efficient and timely manner. Comirit is not capable of 'learning to learn', of deep problem-solving or, by its own accord, exceeding the commonsense domains for which it has been designed (though these are interesting possibilities for future work).

8. **Pragmatic, rather than idealistic attitudes towards intelligence**

   Every aspect of the design of Comirit reflects the objective of creating a useful and usable system for commonsense reasoning. Comirit is not based upon an idealized definition of intelligence or commonsense but has been designed to solve the kind of problems that are faced by real-world agents and that are tested by commonsense reasoning benchmark problems.

9.1.2. Secondary Contributions

In Section 1.1, I further noted that, while my primary objective is to create a commonsense reasoning system, the theory and artefacts described in this dissertation are of independent value beyond this application:

1. **A new formalization of grounding and the symbol grounding problem**

   In Chapter 3, I presented a formalization of the symbol-grounding problem. The formalism serves as a precise vocabulary for discussing and reasoning about the problem and it provokes challenging new research questions. Furthermore, the work demonstrates that formalization of the symbol-grounding problem is possible and that formal methods may serve to lift the precision of its discussion.

2. **Identification of representational constraints for pragmatic and effective commonsense systems**

   The formalization of the symbol grounding problem highlights the crucial importance of iconic and symbolic representations in practical commonsense reasoning. Comirit has been guided by this insight and is designed as a practical implementation of these principles. However, Comirit is not the only way of translating these insights into a software design. These insights into the grounding problem are applicable to the development of entirely new commonsense reasoning or intelligent systems.

3. **Development of new representations, formalisms and algorithms that have general application beyond the implementation of commonsense reasoning systems**

   I have generalized and adapted several technologies: simulation, tableau-based deduction and machine-learning. These generalizations may find application in domains beyond commonsense reasoning and in systems other than Comirit. Some of these potential applications are discussed later, in Section 9.2.5.

4. **Benchmarks and approaches that serve as a counterpoint to existing methods used within the commonsense reasoning community**

   In Section 8.1.4 of the evaluation, I described a new benchmark problem, the *Toy Box Problem*, that may be of use in testing other commonsense reasoning and intelligent systems, especially those that use a 'developmental approach' to learning. Furthermore, the existence of Comirit itself may prove to be of value simply by virtue of its existence and as a counterpoint to the entrenched approaches used in the literature. That is, the competition provided by Comirit may stimulate better commonsense reasoning systems, even if its theoretical foundation fails to have significant impact.

9.1.3. Research Gap

In the literature survey of Chapter 2, I conducted a systematic analysis of the literature, summarizing the core capabilities and characteristics of existing methods and systems. The analysis highlighted the trade-off between power and cost-effectiveness that is pervasive among existing approaches. That is, systems with richly sophisticated representations require careful and expensive knowledge engineering, while systems that are cheaper to build typically work with more shallow knowledge.

Instead, I set out to challenge this trade-off by creating an architecture that simultaneously achieves both power and cost-effectiveness. Rather than sacrificing one for the other, I set out to jointly achieve both goals at the expense (where necessary) of general purpose applicability. That is, Comirit is designed for efficiently reasoning about typical problems of commonsense 'know-how', rather than general purpose problem-solving or general purpose linguistic knowledge.

In Comirit, simulation is central to achieving the new trade-off. Consider, again, the features and capabilities of Comirit viewed from the same perspective of the identified research gap of Chapter 2:

Power

- **Breadth.** While the implementation of a broad knowledge base lies beyond the scope of this dissertation, the graph-based representation lends itself to rapid knowledge engineering, autonomous learning and scalable storage in databases. Comirit may never achieve the breadth of a shallow system built by mining text on the web. However, the low cost of construction means that it is easier to achieve breadth than it is with other manually engineered systems (such as Cyc).
- **Fidelity.** Simulations are high-fidelity approximations of reality. Simulations can reason with nuance that is impossible to capture within linguistic models or symbolic abstractions. For example, the suitability of using a spatula like a knife to cut an omelette, or of using a gym ball as an office chair, may not follow from their symbolic description but may be best understood by visualizing or simulating the object in its novel use.
- **Density.** A key strength of simulation is that there are few concepts that must be defined as primitives in the formalism. Simulation does not require the formalization of every concept: a given chair, for example, may be modeled by a physical object like any other, rather than a

vague set of action potentialities. Simulation is driven by a core set of simple laws (the laws of physics, in the case of physical simulation), from which all behavior and knowledge stems.

- **Uniformity.** Comirit is designed as a uniform framework for commonsense reasoning. All simulation-based knowledge is represented by standard graph-based models. Simulations are, in-turn, integrated with other mechanisms by a uniform control mechanism based on a generalization of the tableau method.
- **Elaboration tolerance.** Because simulations are driven by a small set of laws that correspond to the underlying reality, elaborations—even highly unusual elaborations—are readily admitted without modification to the formalisms.

Cost-effectiveness

- **Construction cost.** The development of a commercial-grade implementation of Comirit Framework is a non-trivial task. However, this dissertation serves as a detailed design that would enable a small team of developers to complete such a project in a reasonable time (one to three years).
- **Adaptation cost.** The graphical representations used by simulation are conducive to rapid knowledge engineering and autonomous knowledge acquisition. With suitable tools, Comirit may be adapted to novel problem domains in short time-frames.
- **Operation cost.** A key advantage of simulation, especially when compared to purely logical deduction, is its runtime characteristics. Simulations can be implemented efficiently and run in time that is polynomial in the problem size. Consider, as evidence, the rich real-time environments of modern computer games. Furthermore, autonomous learning can minimize the cost of maintaining the accuracy of a knowledge base, as it evolves over time.
- **Confidence.** I have developed concrete prototypes of Comirit that can solve established benchmark problems. This experience, combined with the success of large-scale sandbox computer games, indicates that Comirit will scale to real implementations of rich and general purpose commonsense reasoning.

Thus, I argue that Comirit is, indeed, both powerful and cost-effective, where existing systems have made a trade-off between these two dimensions.

Of course, some trade-offs have been made towards achieving these joint benefits. Comirit is neither suitable as an efficient general purpose theorem prover nor is it effective for large scale linguistic processing. Simulation works best on domains driven by a small set of fundamental laws and where abductive inference against the 'arrow of time' is rare. While the hybrid reasoning framework goes a long way towards minimizing these weaknesses, Comirit will never perform as well on abstract reasoning problems as optimized theorem provers. For example, while tableau-based reasoners can solve SAT with relative efficiency, the most efficient SAT-solvers today are based on DPLL [134].

Thus, Comirit addresses a gap in current research and practice. While Comirit is not applicable to every single problem of Artificial Intelligence, it does offer a powerful solution to practical problems of commonsense reasoning, for which no other system previously existed.

9.2. Future Work

This dissertation reports on a doctoral research project, necessarily constrained by resources and time. While this dissertation is complete in the sense that the thesis is evaluated with respect to the framework as it is presented in the dissertation, many interesting questions, which I intend to explore in future, have arisen from the work.

These research questions and opportunities stem not only from the wide scope of the work described within but also from its potential applications to commonsense reasoning and other fields.

I have discussed many important future research questions throughout this dissertation, as they have been relevant to the discussion. Over the following subsections, I will summarize some of the more interesting avenues for future work.

9.2.1. Formal Foundations

In Chapter 3, I presented a formalization of the symbol-grounding argument and presented a challenge to other researchers to contribute their own formalizations. There are a number of future directions in which this preliminary work may be continued:

- The non-equivalence (or equivalence) of separate classes may be formally proven (*i.e.*, proving the postulates of Section 3.5). This may lead to a conclusive identification of necessary and/or sufficient criteria for building intelligent systems
- The formalization program may be continued to eliminate the assumed primitives: the definition of intelligence itself, the nature of a universal set and the meaning of semantic interpretability.
- Other perspectives on the symbol-grounding problem may be formalized in a similar methodology to create a comparable framework.

- Once formalized, other perspectives on the symbol grounding problem may be proven equivalent or, if not equivalent, analyzed for their differences.
- In the long-term, it may even be possible to construct a suitably rich formal model of symbol grounding, intelligence or commonsense, so that commonsense reasoning systems can be engineered entirely by synthesis from formal requirements.

9.2.2. Framework Formalization

While this work stems from a practical vision, it is an adaptation of methods with sound theoretical foundations and it is presented with a partial formalization. Deeper formalization of the framework and its mechanics may not only improve Comirit but may also feed back into other advances in commonsense reasoning.

In particular:

- Formal models of Comirit may be verified against a notion of consistency and completeness.
- Formal models of Comirit may be validated for correctness against their real world interpretation. In particular, models of error and accuracy could be incorporated into the definition of Comirit.
- Formal models of approximate simulation-based reasoning, hybrid reasoning and learning may be translated back into the established literature on commonsense reasoning. For example, Comirit Simulation could be combined with traditional logics of nonmonotonic reasoning, rather than treated as a rule for tableau-based reasoning.

Indeed, there has already been some preliminary progress towards these goals. Guoqiang Jin, in his Chinese-language undergraduate thesis [82], explored a preliminary embedding of Comirit Simulation into a formal framework based upon Moszkowski's Interval Temporal Logic (ITL) [130]. In this embedding, an iteration of Comirit Simulation is modeled by discrete changes in the temporal logic. While this work has not been developed beyond initial speculation, such an embedding may eventually allow for statistical models of simulation errors. These statistical models may be used to quantify the degree to which simulation approximates reality and to create simulations that precisely adapt their precision and resolution to the goals of a system.

9.2.3. Framework Design

Because Comirit is designed as an extensible and open-ended framework, there are ample opportunities for extending the design of Comirit:

- Integrating new representations and reasoning mechanisms into the control strategy; some possibilities include cellular automata, belief revision, spreading activation, probabilistic logics and non-monotonic logics
- Extending the tableau reasoning algorithm (or developing new meta-rules) to incorporate the latest developments and heuristics of tableau reasoning
- Applying new learning algorithms and exploring the possibility of autonomously learning the laws of a simulation (*i.e.*, including simulation constraint functions or tableau reasoning rules in the hypothesis space)
- Developing techniques for automatically diagnosing and repairing inconsistency or incompleteness in sets of tableau rules
- Extending the tableau algorithm of Comirit to support back-tracking and variable unification

In Section 5.2.6, I described recent independent work by the Soar group directed towards integrating visual representations and simulation into the Soar cognitive architecture. While the work differs significantly from this dissertation in both research direction and contributions, it may be possible to adapt some of the new Soar extensions into the Comirit Framework (and *vice versa*).

The fundamental architectures of our respective proposals are largely incompatible. The Soar extensions—the Spatial and Visual System (SVS) [176]—*add* spatial and visual buffers and operators to the Soar production rule system. In contrast, Comirit is designed 'from the ground up' as an integrative learning and reasoning system. Nevertheless, the SVS has a number of novel ideas and designs that are well worth exploring for Comirit. In particular, the SVS includes more detailed design and consideration of the translation between symbolic and iconic representations (predicate extraction and projection) and also includes a 'visual buffer' that is a 2D projection of a spatial model, enabling an agent to reason about perspectives and images.

Conversely, this dissertation provides a generic representation for spatial knowledge (graph-based simulation) and provides a coherent framework for learning spatial simulations, both of which may be adapted to Soar.

9.2.4. Framework Implementation

Aside from the research questions arising from this dissertation, there also remains the practical task of following through on the design to implement a full scale commonsense reasoning system:

- Developing powerful tool support for rapid knowledge engineering, including automatic 3D reconstruction of graph-based models from images or scans
- Developing a comprehensive knowledge-base of commonsense facts, objects and scenarios
- Integration of preexisting resources such as games, databases, ontologies and knowledge bases (including OpenCyc)
- Implementing a robust, marketable system for commonsense reasoning
- Performing an exhaustive *quantitative* analysis of the complete implemented system on a real-world problem

While the full implementation of Comirit may create commercial opportunities, such effort is unlikely to be void of research implications. Full scale comirit development is likely to raise interesting new problems concerning integration, migration and scalability, in addition to the system's usability and social impact.

9.2.5. Applications

In Section 1.2, I presented a list of potential applications of commonsense reasoning. Each of these problems are commercial opportunities for Comirit.

In the short- to medium-term future, I intend to focus on two high-impact problem domains:

- **Household robotics.** The home is an ideal application environment for Commonsense reasoning. While the home is a complex environment with a relatively restricted range of objects, these objects come in virtually unlimited embodiments (*e.g.*, consider the many kinds of sofas) that are difficult to richly describe manually. As such, the domain is a non-trivial test-bed for commonsense reasoning: the contents of a house can be modeled by simple naïve physics and the operation of electronic devices and complex mechanical systems (such as clocks), which do not readily lend themselves to physical simulation, may be described as logical systems.
- **Financial Risk Management.** Risk management is an interesting and high-impact domain that concerns markets, market participants, expectation, planning and simulation. It is partly subject to rule-driven behavior but demands extensive commonsense knowledge in order to deal with unexpected situations and to avoid exploitation at the hands of other market partici-

pants who can anticipate rule-driven behavior. Risk management, therefore, offers a deep and sophisticated test-bed that will challenge the technology of Comirit and serve as a proving ground for its ability to generalize to domains beyond physical and mechanical systems.

As a software framework for multi-representation reasoning, Comirit may also find application in fields beyond commonsense reasoning. In particular, Comirit may be used as a framework for managing computations involving large numbers of simulations. For example:

- **Simulation-dependent computational sciences.** Hybrid reasoning may assist with the management of virtual experiments in chemistry, protein folding, nuclear physics, meteorology and biotechnology.
- **Computer animation.** Hybrid reasoning can manage the creation of rough story-boards and monitor animations to ensure that global constraints of consistency and continuity are enforced.
- **Quantitative finance.** Comirit can provide an integrated environment and formal framework for monte-carlo simulation and other analyses of financial systems.

9.3. Final Thoughts

I began my PhD studies with an innocent ambition of creating 'intelligent systems'. I believe that intelligent software systems have the potential to dramatically change society for the better: those evils in the world that stem from abuse of power, lack of accountability, unfair resource allocation and inefficiency will no longer be able to persist in a world in which the machinery of society is illuminated by pervasive intelligent networks.

Of course, this long term-ambition is far beyond the scope of a PhD dissertation but I feel that commonsense reasoning is the most important first step that we can make today towards this goal.

The objective of this dissertation was therefore to contribute to the discussion and development of systems with artificial commonsense. I have emphasized pragmatism, in the belief that new perspectives are needed to help move past stagnation in theoretical work.

Many many questions, problems and opportunities remain, not only concerning the work described in this thesis but also the long term ambitions of commonsense and artificial intelligence.

I hope that this dissertation contributes some measure of progress towards a better world.

Bibliography

[1] Abraham, R. and Erwig, M. (2007) 'UCheck: A Spreadsheet Type Checker for End Users', *Journal of Visual Languages and Computing*, vol. 18, no. 1, pp. 71–95.

[2] Ahuja, S., Carriero, N. and Gelernter, D. (1986) 'Linda and Friends', *IEEE Computer*, August 1986, pp. 26–34.

[3] Anderson, J.R., Bothell, D., Byrne, M.D., Douglass, S., Lebiere, C. and Qin, Y. (2004) 'An Integrated Theory of the Mind', *Psychological Review*, vol. 111, no. 4, pp. 1036–1060.

[4] Artstein, R., Gandhe, S., Gerten, J., Leuski A. and Traum, D. (2009) 'Semi-formal Evaluation of Conversational Characters', In Grumberg, O., Kaminski, M., Katz, S. and Wintner, S. (eds.), *Languages: From Formal to Natural - Essays Dedicated to Nissim Francez on the Occasion of His 65th Birthday (LNCS 5533)*, Springer.

[5] Associated Press (2006) 'Deep Fritz Checkmates Chess Champ', *The Sydney Morning Herald*, 6 December 2006, <http://www.smh.com.au/news/technology/deep-fritz-checkmates-chess-champ/2006/12/06/1165080982547.html>, Accessed 20 September 2009.

[6] Balzarotti, D., Costa, P. and Picco, G.P. (2007) 'The LighTS Tuple Space Framework and its Customization for Context-aware Applications', *International Journal on Web Intelligence and Agent Systems*, vol. 5, no. 2, pp. 215–231.

[7] Barsalou, L. (1999) 'Perceptual Symbol Systems', *Behavioral and Brain Sciences*, vol. 22, pp. 577–660.

[8] BBC News (2009) 'Duke Nukem Developer Goes Bust', *BBC News*, 7 May 2009, <http://news.bbc.co.uk/2/hi/technology/8037688.stm>, Accessed 26 March 2009.

[9] Belfiore, M. (2007) 'Safety Last for Robo-Cars', *Wired Magazine*, 28 October 2007, <http://www.wired.com/dangerroom/2007/10/safety-last-for/>, Accessed 23 September 2009.

[10] Belfiore, M. (2007) 'Carnegie Takes First in DARPA's Urban Challenge', *Wired Magazine*, 4 November 2007, <http://www.wired.com/dangerroom/2007/11/darpa-names-win/>, Accessed 23 September 2009.

[11] Bennett, B. (1998) 'Modal Semantics for Knowledge Bases Dealing with Vague Concepts', *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR88)*.

[12] Bennett, B. (2001) 'What is a Forest? On the Vagueness of Certain Geographic Concepts', *Topoi*, vol. 20, iss. 2, pp. 189–201.

[13] Bishop, M.A. and Trout, J.D. (2002) '50 Years of Successful Predictive Modeling Should be Enough: Lessons for Philosophy of Science', *Philosophy of Science*, vol. 69, pp. S197–S208.

[14] Bobrow, D.G. (1964) *Natural Language Input for a Computer Problem Solving System*, PhD Thesis, Massachusetts Institute of Technology.

[15] Bouquet, P., Serafini, L. and Stoermer, H. (2005) 'Introducing Context into RDF Knowledge Bases', *Proceedings of SWAP 2005, the 2nd Italian Semantic Web Workshop*.

[16] Brachman, R.J. (2006) '(AA)AI More than the Sum of Its Parts', *AI Magazine*, vol. 27, no. 4.

[17] Bredeweg, B. and Struss, P. (2003) 'Current Topics in Qualitative Reasoning (Editorial Introduction)', *AI Magazine*, vol. 24, no. 4, pp. 13–16.

[18] Broadie, A. (2009) 'Scottish Philosophy in the 18th Century', *Stanford Encyclopedia of Philosophy*, <http://plato.stanford.edu/entries/scottish-18th/>, Accessed 25 September 2009.

[19] Brooks, F.P. (1987) 'No Silver Bullet - Essence and Accidents of Software Engineering', *IEEE Computer*, vol. 20, no. 4, pp. 10–19.

[20] Brooks, R. (1991) 'Intelligence Without Representation', *Artificial Intelligence*, vol. 47, pp. 139–159.

[21] Brooks, R. (1991) 'Intelligence without Reason', *Proceedings of the 1991 International Joint Conference on Artificial Intelligence (IJCAI91)*.

[22] Campbell, M., Hoane Jr., A.J. and Hsu, F. (2002) 'Deep Blue', *Artificial Intelligence*, no. 134, pp. 57–83.

[23] Carriero, N. and Gelernter, D. (1989) 'Linda in Context', *Communications of the ACM*, vol. 32, no. 4, pp. 444–458.

[24] Cassimatis, N. (2005) 'Integrating Cognitive Models Based on Different Computational Methods', *Proceedings of the Tenth Annual Conference of the Cognitive Science Society*.

[25] Chella, A., Frixione, M. and Gaglio, S. (1997) 'A Cognitive Architecture for Artificial Vision', *Artificial Intelligence*, vol. 89, pp. 73–111.

[26] Chklovski, T. (2003) 'LEARNER: A System for Acquiring Commonsense Knowledge by Analogy', *Proceedings of the 2nd International Conference on Knowledge Capture (K-KAP 2003)*.

[27] Coradeschi, S. and Saffiotti A. (eds.) (2003) *Robotics and Autonomous Systems (Special Issue on Perceptual Anchoring)*, vol. 43, no. 2–3.

[28] Corkill, D. (1991) 'Blackboard Systems', *AI Expert*, vol. 6, no. 9, pp. 40–47.

[29] Curtis, J., Matthews, G. and Baxter, D. (2005) 'On the Effective Use of Cyc in a Question Answering System ', *Papers from the IJCAI Workshop on Knowledge and Reasoning for Answering Questions*.

[30] Cycorp Incorporated (2008) 'OpenCyc Brings Meaning to the Web', Press Release, 18 August 2008, <http://cyc.com/company/news/OpenCyc%20Brings%20Meaning%20to%20the%20Web>, Accessed 23 September 2009.

[31] D'Agostino, M., Gabbay, D.M., Hähnle, R. and Posegga, J. (1999) *Handbook of Tableau Methods*, Springer.

[32] Davis, E. (2008) 'Pouring Liquids: A Study in Commonsense Physical Reasoning', *Artificial Intelligence*, vol. 172, pp. 1540–1578.

[33] de Garis, H., Zhou, C., Shi, X., Goertzel, B., Pan, W., Miao, K., Zhou, J., Jiang, M., Zhen, L., Wu, Q., Shi, M., Lian, R. and Chen, Y. (2009) 'The China-Brain Project: Report on the First Six Months', *Proceedings of the Second International Conference on Artificial General Intelligence (AGI-2009)*.

[34] Debray, S.K. and Warren, D.S. (1990) 'Towards Banishing the Cut from Prolog', *IEEE Transactions on Software Engineering*, vol. 16, no. 3, pp. 335–349.

[35] DeCuyper, J., Keymeulen, D. and Steels, L. (1995) 'A Hybrid Architecture for Modelling Liquid Behavior', In Glasgow, J., Chandrasekaran, B. and Narayanan, N.H. (eds.), *Diagrammatic Reasoning*, MIT Press.

[36] Dreyfus, H. (1992) *What Computers Still Can't Do*, MIT Press.

[37] Džeroski, S. and Todorovski, L. (eds.) (2007) *Computational Discovery of Scientific Knowledge: Introduction, Techniques, and Applications in Environmental and Life Sciences*, Springer.

[38] Electronic Arts, Incorporated (2009) *Form 10-K Annual Report*, <http://investors.ea.com/secfiling.cfm?filingID=1193125–09-116895>, Accessed 26 March 2009.

[39] Electronic Arts, Incorporated (2009) 'EA Reports Fourth Quarter and Fiscal Year 2009 Results', Press Release, 5 May 2009, <http://investors.ea.com/releasedetail.cfm?ReleaseID=381903>, Accessed 26 March 2009.

[40] Epstein, R. (2007) 'From Russia, with Love: How I Got Fooled (and Somewhat Humiliated) by a Computer', *Scientific American Mind*, October/November 2007.

[41] Eugster, P., Felber, P.A., Guerraoui, R. and Kermarrec, A-M. (2003) 'The Many Faces of Publish/Subscribe', *ACM Computing Surveys*, vol. 35, no. 2, pp. 114–131.

[42] Falkenhainer, B., Forbus, K. and Gentner, D. (1989) 'The Structure-Mapping Engine: Algorithm and Examples', *Artificial Intelligence*, vol. 20, pp. 1–63.

[43] Forbus, K.D. (1989) 'Introducing Actions into Qualitative Simulation', *Proceedings of the 1989 International Joint Conference on Artificial Intelligence (IJCAI89)*.

[44] Friedland, N.S., Allen, P.G., Matthews, G., Witbrock, M., Baxter, D., Curtis, J., Shepard, B., Miraglia, P., Angele, J., Staab, S., Moench, E., Oppermann, H., Wenke, D., Israel, D., Chaudri, V., Porter, B., Barker, K., Fan, J., Chaw, S.Y., Yeh, P., Tecuci, D. and Clark, P. (2004) 'Project Halo: Towards a Digital Aristotle', *AI Magazine*, vol. 25, no. 4, pp. 29–47.

[45] Fritz, C. and McIlraith, S. (2006) 'Decision-Theoretic GOLOG with Qualitative Preferences', *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR06)*.

[46] Fu, W.T. and Pirolli, P. (2007) 'SNIF-ACT: A Model of Information-Seeking Behavior in the World Wide Web', *Human-Computer Interaction*, no. 22, pp. 355–412.

[47] Gärdenfors, P. (1996) 'Cued and Detached Representations in Animal Cognition', *Behavioural Processes*, vol. 35, pp. 263–273.

[48] Gärdenfors, P. (2000) *Conceptual Spaces: The Geometry of Thought*, MIT Press.

[49] Gärdenfors, P. (2004) 'How to Make the Semantic Web More Semantic', In Varzi, A. and Lieu, L. (eds.), *Formal Ontology in Information Systems*, IOS Press.

[50] Gärdenfors, P. (2007) 'Mindreading and Control Theory', *European Review*, vol. 15, pp. 223–240.

[51] Gärdenfors, P. and Williams, M-A. (2007) 'Multi-agent Communication, Planning and Collaboration Based on Perceptions, Conceptions and Simulations', In Schalley, A.C. and Khlentzos, D. (eds.), *Mental States, Volume 1*.

[52] Gardin, F. and Meltzer, B. (1989) 'Analogical Representations of Naive Physics', *Artificial Intelligence*, vol. 38, pp. 139–59.

[53] Genesereth, M. and Love, N. (2005) 'General Game Playing: Overview of the AAAI Competition', *AI Magazine*, vol. 26, no. 2.

[54] Goertzel, B. (2008) 'A Pragmatic Path Toward Endowing Virtually-Embodied AIs with Human-Level Linguistic Capability', *Proceedings of the 2008 IEEE World Congress on Computational Intelligence (WCCI)*.

[55] Goertzel, B. (2009) 'What Must a World Be That a Humanlike Intelligence May Develop In It?', *Dynamical Psychology*, <http://goertzel.org/dynapsyc/2009/BlocksNBeadsWorld.pdf>, Accessed 12 January 2009.

[56] Goertzel, B., Heljakka, A., Bugaj, S.V., Pennachin, C. and Looks, M. (2006) 'Exploring Android Developmental Psychology in a Simulation World', *Proceedings of the ICCS/CogSci-2006 Workshop on Android Science*.

[57] Goertzel, B., Iklé, M., Goertzel, I.F. and Heljakka, A. (2008) *Probabilistic Logic Networks*, Springer.

[58] Goertzel, B., Pennachin, C., Geissweiller, N., Looks, M., Senna, A., Welter, S., Heljakka, A. and Lopes, C. (2008) 'An Integrative Methodology for Teaching Embodied Non-Linguistic Agents, Applied to Virtual Animals in Second Life', *Proceedings of the First International Conference on Artificial General Intelligence (AGI-08)*.

[59] Graham, G. (2009) 'Scottish Philosophy in the 19th Century', *Stanford Encyclopedia of Philosophy*, <http://plato.stanford.edu/entries/scottish-19th/>, Accessed 25 September 2009.

[60] Graves, A. and Schmidhuber, J. (2008) 'Offline Handwriting Recognition with Multidimensional Recurrent', *Proceedings of 2008 Conference on Advances in Neural Information Processing Systems (NIPS'22)*.

[61] Gregor, S. (2006) 'The Nature of Theory in Information Systems', *MIS Quarterly*, vol. 30, no. 3, pp. 611–642.

[62] Guild Software (2009) 'Guild Software News', Press Release, <http://www.guildsoftware.com/news.html>, Accessed 25 September 2009.

[63] Haase, K. (1986) 'Discovery Systems', *Proceedings of the 1986 European Conference on Artificial Intelligence (ECAI-86)*.

[64] Hähnle, R. (2001) 'Tableaux and Related Methods', In Robinson, J.A. and Voronkov, A. (eds.), *Handbook of Automated Reasoning Volume 1*, MIT Press.

[65] Harnad, S. (1990) 'The Symbol Grounding Problem', *Physica D*, no. 42, pp. 335–346.

[66] Hart, D. and Goertzel, B. (2008) 'OpenCog: A Software Framework for Integrative Artificial General Intelligence', *Proceedings of the First International Conference on Artificial General Intelligence (AGI-08)*.

[67] Havasi, C., Speer, R. and Alonso, J. (2007) 'ConceptNet 3: a Flexible, Multilingual Semantic Network for Common Sense Knowledge', *Proceedings of Recent Advances in Natural Languges Processing 2007*.

[68] Hawkins, J. and George, D. (2006) *Hierarchical Temporal Memory: Concepts, Theory and Terminology*, Numenta Incorporated White Paper, <http://www.numenta.com/Numenta_HTM_Concepts.pdf>, Accessed 24 September 2009.

[69] Hayes, P.J. (1978) 'The Naive Physics Manifesto', In Michie, D. (ed.), *Expert Systems in the Micro-Electronic Age*, Edinburgh University Press.

[70] Hayes, P.J. (1983) 'The Second Naive Physics Manifesto', Cognitive Science Technical Report URCS-10, University of Rochester.

[71] Hayes-Roth, B. (1985) 'A Blackboard Architecture for Control', *Artificial Intelligence*, vol. 26, no. 3, pp. 251 - 321 .

[72] Hediger, H. (1981) 'The Clever Hans Phenomenon from an Animal Psychologist's Point of View', *Annals of the New York Academy of Sciences*, vol. 364, no. 0, pp. 1–17.

[73] Herman, I. (2001) *W3C Semantic Web Frequently Asked Questions*, <http://www.w3.org/2001/sw/SW-FAQ>, Accessed 25 September 2009.

[74] Herre, H., Heller, B., Burek, P., Hoehndorf, R., Loebe, F. and Michalek, H. (2007) 'General Formal Ontology (GFO): A Foundational Ontology Integrating Objects and Processes. Part I: Basic Principles', Research Group Ontologies in Medicine (Onto-Med) Technical Report Version 1.0.1, <http://www.onto-med.de/Archiv/ontomed2002/en/theories/gfo/part1-drafts/gfo-part1-v1–0-1.pdf>, Accessed 24 September 2009.

[75] Hofstadter, D.R. and Mitchell, M. (1995) 'The Copycat project: A Model of Mental Fluidity and Analogy-Making', In Holyoak, K.J. and Barnden, J.A. (eds.), *Advances in Connectionist and Neural Computation Theory*, Ablex.

[76] Hutter, M. (2000) 'Towards a Universal Theory of Artificial Intelligence based on Algorithmic Probability and Sequential Decisions', *Proceedings of the 12th European Conference on Machine Learning (ECML-2001)*.

[77] Hutter, M. (2002) 'The Fastest and Shortest Algorithm for All Well-Defined Problems', *International Journal of Foundations of Computer Science*, vol. 13, no. 3, pp. 431–443.

[78] Iglesias, C.A., Garijo, M. and González, J.C. (1998) 'A Survey of Agent-Oriented Methodologies', In Müller, J.P., Singh, M.P. and Rao, A. S. (eds.), *Intelligent Agents V. Agents Theories, Architectures, and Languages (LNCS 1555)*, Springer.

[79] International Organization for Standardization (2002) *Information Technology—Z Formal Specification Notation—Syntax, Type System and Semantics*, ISO/IEC 13568:2002, <http://standards.iso.org/ittf/PubliclyAvailableStandards/c021573_ISO_IEC_13568_2002(E).zip>, Accessed 10 September 2009.

[80] iRobot Corporation (2006) 'Sales of iRobot Roomba Vacuuming Robot Surpass 2 Million Units', Press Release, 22 May 2006, <http://www.irobot.com/sp.cfm?pageid=86&id=234>, Accessed 21 September 2009.

[81] Jenkins, M-C., Churchill, R., Cox, S. and Smith, D. (2007) 'Analysis of User Interaction with Service Oriented Chatbot Systems', In Jacko, J. (ed.), *Human-Computer Interaction, Part III, HCII 2007, LNCS 4552*, Springer.

[82] Jin, G. (2007) 'Commonsense Reasoning System: Integrated Simulation and Planning (常识仿真推理系统中的规划机制研究)', Unpublished Honors Dissertation, University of Science and Technology, China.

[83] Johnson, A.E. and Hebert, M. (1999) 'Using SPIN-images for Efficient Multiple Model Recognition in Cluttered 3D Scenes', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, no. 5, pp. 433–449.

[84] Johnston, B. and Williams, M-A. (2007) 'A Generic Framework for Approximate Simulation in Commonsense Reasoning Systems', *Proceedings of the AAAI 2007 Spring Symposium on Logical Formalizations of Commonsense Reasoning (Commonsense 2007)*.

[85] Johnston, B. and Williams, M-A. (2008) 'Comirit: Commonsense Reasoning by Integrating Simulation and Logic', *Proceedings of the First International Conference on Artificial General Intelligence (AGI-08)*.

[86] Jøsang, A. (2001) 'A Logic for Uncertain Probabilities', *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 9, iss. 3, pp. 279–311.

[87] Just, M.A. and Varma, S. (2007) 'Cognitive, Affective and Behavioral Neuroscience', *The Organization of Thinking: What Functional Brain Imaging Reveals About the Neuroarchitecture of Complex Cognition*, vol. 7, no. 3, pp. 153–91.

[88] Kaku, M. (1999) *Visions: How Science Will Revolutionize the Twenty-First Century*, Oxford University Press.

[89] Kieras, D. and Meyer, D.E. (1997) 'An Overview of the EPIC Architecture for Cognition and Performance with Application to Human-Computer Interaction', *Human-Computer Interaction*, no. 12, pp. 391–438.

[90] Kilkerry Neto, A., Zaverucha, G. and Carvalho, L. (1999) 'An Implementation of a Theorem Prover in Symmetric Neural Networks', *Proceedings of the 1999 International Joint Conference on Neural Networks (IJCNN 1999)*.

[91] Kohonen, T. (1998) 'The Self-Organizing Map', *Neurocomputing*, no. 21, pp. 1–6.

[92] Kokinov, B., Nikolov, V. and Petrov, A. (1996) 'Dynamics of emergent computation in DUAL', In Ramsay, A. (ed.), *Artificial intelligence: Methodology, Systems, Applications*, IOS Press.

[93] Kowalski, R. and Sergot, M. (1986) 'A Logic-Based Calculus of Events', *New Generation Computing*, vol. 4, pp. 67–95.

[94] Kuipers, B. (2001) 'Qualitative Simulation', In Meyers, R.A. (ed.), *Encyclopedia of Physical Science and Technology, Third Edition*, Academic Press, pp. 287–300.

[95] Kvarnström, J. and Magnusson, M. (2003) 'TALplanner in the Third International Planning Competition: Extensions and Control Rules', *Journal of Artificial Intelligence Research*, vol. 20, pp. 343–377.

[96] Laird, J.E. (2008) 'Extending the Soar Cognitive Architecture', *Proceedings of the First International Conference on Artificial General Intelligence (AGI-08)*.

[97] Langley, P., Simon, H.A., Bradshaw, G.L. and Zytkow, J.M. (1987) *Scientific Discovery: Computational Explorations of the Creative Process*, MIT Press.

[98] Lathrop, S. and Laird, J. (2009) 'Extending Cognitive Architectures with Mental Imagery', *Proceedings of the Second International Conference on Artificial General Intelligence (AGI-2009)*.

[99] Legg, S. (2006) 'Is there an Elegant Universal Theory of Prediction?', IDSIA / USI-SUPSI Dalle Molle Institute for Artificial Intelligence Technical Report IDSIA-12–06, <http://www.idsia.ch/idsiareport/IDSIA-12–06.pdf>, Accessed 3 August 2008.

[100] Legg, S. and Hutter, M. (2007) 'A Collection of Definitions of Intelligence', In Goertzel, B. and Wang, P. (eds.), *Advances in Artificial General Intelligence: Concepts, Architectures and Algorithms*, IOS Press.

[101] Legg, S. and Hutter, M. (2007) 'Universal Intelligence: A Definition of Machine Intelligence', *Minds and Machines*, vol. 17, no. 4, pp. 391–444.

[102] Lenat, D. (1998) 'The Dimensions of Context Space', Cycorp Technical Report, <http://www.cyc.com/doc/context-space.pdf>, Accessed 25 September 2009.

[103] Lenat, D.B. and Brown, J.S. (1984) ' Why AM and Eurisko Appear to Work', *Artificial Intelligence*, vol. 23, pp. 269–294.

[104] Lenat, D.B. and Guha, R.V. (1990) *Building Large Knowledge Based Systems*, Addison Wesley.

[105] Lieberman, H., Liu, H., Singh, P. and Barry, B. (2004) 'Beating Common Sense into Interactive Applications', *AI Magazine*, vol. 25, no. 4, pp. 63–76.

[106] Lieberman, H., Rosenzweig, E. and Singh, P. (2001) 'Aria: An Agent for Annotating and Retrieving Images', *IEEE Computer*, July 2001, pp. 57–61.

[107] Lifschitz, V. (1998) 'Cracking an Egg: An Exercise in Commonsense Reasoning', *Proceedings of the 1998 Internation Symposium on Logical Formalizations of Commonsense Reasoning (Commonsense 1998)*.

[108] Liu, H., Lieberman, H. and Selker, T. (2002) 'Automatic Affective Feedback in an Email Browser', MIT Media Laboratory Software Agents Group Technical Report, November 2002.

[109] Lockerd, A.L. (2002) 'Acquiring Commonsense Through Simulation', Unpublished White Paper, <http://web.media.mit.edu/~alockerd/papers/CommonsenseSimulation.pdf>, Accessed 23 September 2009.

[110] Lowe, D.G. (1999) 'Object Recognition from Local Scale-invariant Features', *Proceedings of the 1999 International Conference on Computer Vision*.

[111] Lukasiewicz, T. (2006) 'Probabilistic Description Logic Programs', *International Journal of Approximate Reasoning*, vol. 45, no. 2, pp. 288–307.

[112] Magnusson, M. and Doherty, P. (2008) 'Logical Agents for Language and Action', *Proceedings of the 4th Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-08)*.

[113] Markram, H. (2006) 'The Blue Brain Project', *Nature Reviews: Neuroscience*, vol. 7, pp. 153–160.

[114] Masolo, C., Borgo, S., Gangemi, A., Guarino, N. and Oltramari, A. (2003) 'WonderWeb Deliverable D18: Ontology Library', Laboratory for Applied Ontology ISTC-CNR Technical Report, <http://wonderweb.semanticweb.org/deliverables/documents/D18.pdf>, Accessed 25 September 2009.

[115] McCable, T.J. and Schulmeyer, G.G. (2008) 'The Pareto Principle Applied to Software Quality Assurance', In Schulmeyer, G.G. (ed.), *Handbook of Software Quality Assurance, Fourth Edition*, Artech House.

[116] McCarthy, J. (1958) 'Programs with Common Sense', *Proceedings of the December 1958 Teddington Conference on the Mechanization of Thought Processes*.

[117] McCarthy, J. (1963) 'Situations, Actions and Causal Laws', Technical Report Memo 2, Stanford University Artificial Intelligence Laboratory.

[118] McCarthy, J. (1992) 'Elephant 2000: a Programming Language Based on Speech Acts', Unpublished Manuscript, <http://www-formal.stanford.edu/jmc/elephant.pdf>, Accessed 11 October 2006.

[119] McCarthy, J. (1993) 'Notes on Formalizing Context', *Proceedings of the 1993 International Joint Conference on Artificial Intelligence*.

[120] McCarthy, J. (1998) 'Elaboration Tolerance', *Proceedings of the Fourth Symposium on Logical Formalizations of Commonsense Reasoning (Commonsense 98)*, <http://www-formal.stanford.edu/jmc/elaboration.html>, Accessed 24 January 2008.

[121] McCarthy, J. and Hayes, P. (1969) 'Some philosophical problems from the standpoint of artificial intelligence', *Machine Intelligence*, vol. 4, pp. 463–502.

[122] McDermott, D. and Doyle, J. (1978) 'Non-monotonic Logic I', Masssachusetts Institute of Technology AI Memo 486.

[123] McKinstry, C., Dale, R. and Spivey, M.J. (2008) 'Action Dynamics Reveal Parallel Competition in Decision Making', *Psychological Science*, vol. 19, no. 1, pp. 22–24.

[124] Mesit, J., Guha, R.K. and Brust, M.R. (2009) 'Simulation of 3D-deformable Objects Using Stable and Accurate Euler Method', *Proceedings of the 11th International Conference on Computer Modelling and Simulation (UKSim 2009)*.

[125] Meyers, K.L. (1991) 'Hybrid Reasoning using Universal Attachment', *Artificial Intelligence*, no. 67, pp. 329–375.

[126] Milch, B. (2008) 'Artificial General Intelligence through Large-Scale, Multimodal Bayesian Learning', *Proceedings of the First International Conference on Artificial General Intelligence (AGI-08)*.

[127] Miller, R. and Morgenstern, L. (2006) *The Commonsense Problem Page*, <http://www-formal.stanford.edu/leora/commonsense/>, Accessed 10 May 2006.

[128] Minsky, M. (1986) *The Society of Mind*, Simon and Schuster.

[129] Morgenstern, L. (2001) 'Mid-Sized Axiomatizations of Commonsense Problems: A Case Study in Egg Cracking', *Studia Logica*, vol. 67, no. 3, pp. 333–384.

[130] Moszkowski, B.C. (1986) *Executing Temporal Logic Programs*, Cambridge University Press.

[131] Mueller, E.T. (1998) *Natural Language Processing with ThoughtTreasure*, Signiform, <http://www.signiform.com/tt/book/>, Accessed 15 August 2006.

[132] Mueller, E.T. (2000) 'A Calendar with Common Sense', *Proceedings of the 2000 International Conference on Intelligent User Interfaces*.

[133] Mueller, E.T. (2006) *Commonsense Reasoning*, Morgan Kaufmann.

[134] Nieuwenhuis, R., Oliveras, A. and Tinelli, C. (2006) 'Solving SAT and SAT Modulo Theories: From an Abstract Davis–Putnam–Logemann–Loveland Procedure to DPLL(T)', *Journal of the ACM*, vol. 53, no. 6, pp. 937–977.

[135] Niles, I. and Pease, A. (2001) 'Towards a Standard Upper Ontology', *Proceedings of the 2nd International Conference on Formal Ontology in Information Systems (FOIS-2001)*.

[136] Nute, D. (1994) 'Defeasible Logic', In Gabbay, D.M., Hogger, C.J. and Robinson, J.A. (eds.), *Handbook of Logic in Artificial Intelligence and Logic Programming Volume 3*, Oxford University Press.

[137] Open Mind / Global Mind Project (2006) *Global Mind Raw Sentences Database*, Available at the Internet Archive, <http://web.archive.org/web/*/globalmind.media.mit.edu/gm_rawdata.zip>, Accessed 4 August 2006.

[138] Oxford English Dictionary (1989) 'Common sense', *The Oxford English Dictionary Second Edition*, OED Online, Oxford University Press, <http://dictionary.oed.com/cgi/entry/50045151>, Accessed 3 August 2008.

[139] Panton, K., Matuszek, C., Lenat, D., Schneider, D., Witbrock, M., Siegel, N. and Shepard. (2006) 'Common Sense Reasoning—From Cyc to Intelligent Assistant', In Cai, Y. and Abascal, J. (eds.), *Ambient Intelligence in Everday Life (LNAI 3864)*, Springer.

[140] Penrose, R. (1999) *The Emperor's New Mind*, Oxford University Press.

[141] Poole, D. (1988) 'A Logical Framework for Default Reasoning', *Artificial Intelligence*, vol. 36, pp. 27–47.

[142] Prentzas, J. and Ioannis Hatzilygeroudis, I. (2007) 'Categorizing Approaches Combining Rule-Based and Case-Based Reasoning', *Expert Systems*, vol. 24, no. 2, pp. 97–122.

[143] Pylyshyn, Z.W. (2003) *Seeing and Visualizing: It's Not What You Think*, MIT Press.

[144] Randell, D.A., Cui, Z. and Cohn, A.G. (1992) 'A Spatial Logic Based on Regions and Connection', *Procedings of the 3rd International Conference on Knowledge Representation and Reasoning*.

[145] Reiter, R. (1980) 'A Logic for Default Reasoning', *Artificial Intelligence*, vol. 13, pp. 81–132.

[146] Robinson, J.A. and Voronkov, A. (2001) *Handbook of Automated Reasoning*, MIT Press.

[147] Rossi, S. (2007) 'Cyber Lovers Warned Beware of Flirtatious Robots', *Computer World*, 11 December 2007, <http://www.computerworld.com.au/article/199134/cyber_lovers_warned_beware_flirtatious_robots>, Accessed 23 September 2009.

[148] Sahami, M., Dumais, S., Heckerman, D. and Horvitz, E. (1998) ' A Bayesian Approach to Filtering Junk Email', *Proceedings of the 1998 AAAI Workshop on Learning for Text Categorization*.

[149] Schank, R. and Abelson, R. (1977) *Scripts, Plans, Goals and Understanding*, Erlbaum.

[150] Schmidhuber, J. (2009) ' Ultimate Cognition à la Gödel', *Cognitive Computation*, vol. 1, no. 2, pp. 177–193.

[151] Schrijvers, T. and Alexander Serebrenik, A. (2008) 'Improving Prolog Programs: Refactoring for Prolog', *Theory and Practice of Logic Programming*, vol. 8, no. 2, pp. 201–215.

[152] Searle, J.R. (1980) 'Minds, Brains, and Programs', *Behavioral and Brain Sciences*, vol. 3, no. 3, pp. 417–457.

[153] Serafini, L. and Bouquet, P. (2004) 'Comparing Formal Theories of Context in AI', *Artificial Intelligence*, vol. 155, no.1, pp. 41–67.

[154] Setchi, R., Lagos, N. and Froud, D (2007) 'Computational Imagination: Research Agenda', *Proceedings of the 2007 Australian AI Conference*.

[155] Shanahan, M. (2004) 'An Attempt to Formalise a Non-Trivial Benchmark Problem in Common Sense Reasoning', *Artificial Intelligence*, vol. 153, pp. 141–165.

[156] Shepard, L.A. (2005) 'Inflated Test Score Gains: Is the Problem Old Norms or Teaching the Test?', *Educational Measurement: Issues and Practice*, vol. 9, no. 3, pp. 15–22.

[157] Shoham, Y. (1993) 'Agent-oriented Programming', *Artificial Intelligence*, vol. 60, pp. 51–92.

[158] Sowa, J.F. (2002) 'Architectures for Intelligent Systems', *IBM Systems Journal*, vol. 41, no. 3, pp. 331–349.

[159] Spivey, J.M. (1992) *The Z Notation: a Reference Manual*, Prentice Hall.

[160] Sridharan, M. and Stone, P. (2006) 'Towards Eliminating Manual Color Calibration at RoboCup', *RoboCup 2005: Robot Soccer World Cup IX (LNCS 4020)*, Springer.

[161] Sterling, L. and Shapiro, E. (1994) *The Art of PROLOG: Advanced Programming Techniques*, MIT Press.

[162] Stewart, J. (2007) *Calculus, 6th Edition*, Brooks Cole.

[163] Stickel, M.E. (1985) 'Automated Deduction by Theory Resolution', *Proceedings of the 1985 International Joint Conference on Artificial Intelligence (IJCAI85)*.

[164] Thaler, S. (2002) 'Device System for the Autonomous Generation of Useful Information', US Patent 6,356,884.

[165] Thomas, N.J.T. (2010) 'Mental Imagery', *Stanford Encyclopedia of Philosophy*, <http://plato.stanford.edu/entries/mental-imagery/>, Accessed 20 August 2010.

[166] Turing, A.M. (1950) 'Computing Machinery and Intelligence', *Mind*, vol. LIX, no. 236, pp. 433–60.

[167] Ubisoft Entertainment (2008) *Annual Report 2008*, <http://www.ecobook.eu/ubisoft/2008/rapport_annuel/ra2008uk/ra2008uk.pdf>, Accessed 26 March 2009.

[168] Varsta, M., Heikkonen, J., Lampinen, J. and Millan, J.D.R. (2001) 'Temporal Kohonen Map and the Recurrent Self-Organizing Map: Analytical and Experimental Comparison', *Neural Processing Letters*, no. 13, pp. 237–251.

[169] Verney, S.P., Granholm, E., Marshall, S.P. Malcarne, V.L. and Saccuzzo D.P. (2005) 'Culture-Fair Cognitive Ability Assessment: Information Processing and Psychophysiological Approaches', *Assessment*, vol. 12, no. 3, pp. 303–319.

[170] Wang, J., Jin, B. and Jing Li, J. (2004) 'An Ontology-Based Publish/Subscribe System', *Proceedings of the 2004 International Conference on Middleware*.

[171] Wang, P. (2004) 'Toward a Unified Artificial Intelligence', *Proceedings of the 2004 AAAI Fall Symposium on Achieving Human-Level Intelligence through Integrated Research and Systems*.

[172] Washio, T., Motoda, H. and Niwa, Y. (2000) 'Enhancing the Plausibility of Law Equation Discovery', *Proceedings of the 2000 International Conference on Machine Learning (ICML 2000)*.

[173] Weizenbaum, J. (1966) 'ELIZA - A Computer Program for the Study of Natural Language Communication between Man and Machine', *Communications of the ACM*, vol. 9, no. 1, pp. 36–45.

[174] Weyhrauch, R.W. (1980) 'Prolegomena to a Theory of Mechanized Formal Reasoning', *Artificial Intelligence*, no. 13, pp. 133–170.

[175] Williams, M-A., Gärdenfors, P., Karol, A., McCarthy, J. and Stanton, C. (2005) 'A Framework for Evaluating the Groundedness of Representations in Systems: from Brains in Vats to Mobile Robots', *Proceedings of the IJCAI 2005 Workshop on Agents in Real Time and Dynamic Environments*.

[176] Wintermute, S. (2009) 'An Overview of Spatial Processing in Soar/SVS', Technical Report CCA-TR-2009–01, Center for Cognitive Architecture, University of Michigan.

[177] Wintermute, S. and Laird, J.E. (2008) 'Bimodal Spatial Reasoning with Continuous Motion', *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI-08)*.

[178] Wood, L. (2005) 'Cycorp: the Cost of Common Sense', *Technology Review*, vol. 108, no. 3, pp. 33–33.

[179] Wzorek, M., Conte, G., Rudol, P., Merz, T., Duranti, S. and Doherty, P. (2006) 'From Motion Planning to Control - A Navigation Framework for an Autonomous Unmanned Aerial Vehicle', *Proceedings of the 21st Bristol UAV Systems Conference 2009*.

[180] Ziemke, T. (1997) 'Rethinking Grounding', *Proceedings of 1997 Austrian Society for Computer Science Conference on New Trends in Cognitive Science*.

Appendix A

Summary of the Z Notation

Throughout this dissertation, I seek to improve the clarity and precision of my explanations by complementing them with formal definitions wherever possible.

Unfortunately, even though there is a consistent use of notation in set theory and other foundational mathematics, there is little consistency in the denotation of more sophisticated mathematical concepts and complex set operators.

Since this work closely related to concrete implementation in a real programming language, I have chosen to use the Z notation to eliminate ambiguity and confusion. The Z notation is explicitly designed for the specification and design of software, it is rigorously defined in an ISO standard [79] and it comes with a large 'tool-kit' of mathematical operators.

To a reader unfamiliar with the Z notation, a number of symbols may appear strange or confusing. The objective of this appendix is to offer an informal guide to reading the Z notation.

For an in-depth analysis, explanation and formalization of the Z notation, I recommend Spivey's reference manual [159]. His reference manual is a pleasure to read and freely available online[*].

Note that because I am describing an open-ended framework, rather than a complete system, I do not make use of the complete Z notation. Instead, I have used Z to sketch the important principles

[*]http://spivey.oriel.ox.ac.uk/mike/zrm/

of the design. An engineer may complete the design and formalize sequential systems, as appropriate for their particular application domain.

A.1. A 60-second Summary

In my experience, the two aspects of the Z notation that readers often find jarring is the use of the colon (:) to introduce variables (or declare types), and the large dot (•) as a separator.

Take the following formula as an example:

$$\forall\, x : X \bullet P(x) \lor Q(x)$$

In this formula:

- The colon (:) introduces the variable x and restricts it range over the set X.
- The dot (•) is punctuation, used to separate the variable introduction from the quantified predicate.

Thus, the above formula might be classically denoted as:

$$\forall\, x \,.\, x \in X \land \big(P(x) \lor Q(x)\big)$$

In this dissertation, I also use the colon to write $x : X$ as shorthand for 'x is of type X'. In the Z notation, the differences between set membership (*i.e.*, $x \in X$) and typing are extremely subtle. As such, for the purposes of this dissertation, $x : X$ may also be understood as being loosely equivalent to $x \in X$.

Similarly, the formula $x : \mathbb{P}X$, can be read in two ways. It may be read as $x \in \mathbb{P}X$, implying that x is a member of the powerset of X; or the formula may be read as a type declaration that x is of the type 'sets of X'.

A.2. Mathematical Toolkit

The Z notation is based on ZF set theory. While ZF set theory is constructed entirely from sets (members of which may only be other sets), the Z notation assumes set-based encodings for tuples, integers, real numbers, names and user-defined sets of atomic 'elements'. On such assumed primitives, the Z notation provides a rich mathematical toolkit (and extended syntax) for describing formal systems. These are summarized over the following subsections.

A.2.1. Logical Relations

$\neg R_1$	**Negation (not R_1)**
$R_1 \wedge R_2$	**Conjunction (R_1 and R_2)**
$R_1 \vee R_2$	**Disjunction (R_1 or R_2)**
$R_1 \Rightarrow R_2$	**Implication (if R_1 then R_2)**
$R_1 \Leftrightarrow R_2$	**Equivalence (R_1 if and only if R_2)**

A.2.2. Quantifiers

$x : X$	**Variable constructor/declaration**
	Introduces or declares the variable x and restricts the values of the variable to members of the set X
$x == F$	**Abbreviation**
	Introduces x as an abbreviation or short-hand for the formula F
$\forall\, x : X; y : Y \bullet R(x,y)$	**Universal quantification**
	For all x in the set X, and all y in the set Y, it holds that $R(x,y)$
$\exists\, x : X; y : Y \bullet R(x,y)$	**Existential quantification**
	There exists an x in the set X, and a y in the set Y, such that $R(x,y)$
$\{x : X \mid R(x)\}$	**Set comprehension**
	Constructs a set from the elements of X that satisfy the relation R
let $x == y \bullet R(x)$	**Substitution expression**
	Declares a local scope for introducing new abbreviations/substitutions (*i.e.*, this example formula is equivalent to $R(y)$)

A.2.3. Set and Tuple Constructors

$\mathbb{P}\,X$	**Powerset** The set of all subsets of X
$\mathbb{F}\,X$	**Finite powerset** The set of all finite subsets of X
$X \times Y$	**Cartesian product** The set of all possible pairs (x_i, y_j) where x_i is an element of X and y_j is an element of Y
$X \times Y \times Z$	**Ternary Cartesian product** The set of all possible triples (x_i, y_j, z_k) where x_i is an element of X, y_j is an element of Y and z_k is an element of Z

A.2.4. Function and Sequence Constructors

In Z, a function or relation is simply a set of pairs. For example, if a function f is defined as $f = \{(1, 2), (2, 4), (3, 6)\}$, then it follows that $f(1) = 2$ and $f(3) = 6$.

$x \mapsto y$	**Maplet** Using a maplet is equivalent to creating a pair, however the maplet is often used to assist the reader in distinguishing between pairs intended simply as tuples, and pairs that are intended to represent mappings in a function (*i.e.*, $x \mapsto y = (x, y)$)
$f : X \nrightarrow Y$	**Function declaration** Declares f to be a function from the set X to the set Y (note that this is equivalent to introducing f as a variable whose value is an element of the set of all functions from X to Y)
$X \nrightarrow Y$	**Function** The set (or type) of all partial functions from the set X to the set Y

$X \twoheadrightarrow\!\!\!\!\!\!\!\shortmid\;\; Y$	**Finite function** The set of all finite partial functions from the set X to the set Y (*i.e.*, those functions with a finite domain)
$X \rightarrowtail Y$	**Total injection** The set of all total injections from X to Y (*i.e.*, a one-to-one function with a total domain but a partial range)
$X \leftrightarrow Y$	**Relation** The set of all relations from X to Y (*i.e.*, many-to-many functions); this is equivalent to $\mathbb{P}(X \times Y)$
$seq_1\ X$	**Non-empty sequence** A sequence is a partial function that maps from the natural numbers onto a value for that sequence position (*e.g.*, $<a,b,c> = \{(1, a), (2, b), (3, c)\}$; $seq_1\ X$ is the set of all *non-empty* sequences whose values are drawn from the set X

A.2.5. Set Operators and Relations

$\varnothing$	**Empty set ({})**
$x \in X$	**Set membership (*x* is an element of *X*)**
$X \subseteq Y$	**Subset**
$X \cap Y$	**Intersection**
$X \cup Y$	**Union**
$\cup X$	**Generalized union** If X is a set of sets, then $\cup X$ is the set given by union of every set in X
$X \setminus Y$	**Set difference** The elements of X that are not in Y

A.2.6. Function and Tuple Operators

$x.1$	**Tuple selection (first element)**
	The first element of a tuple; $(a, b).1 = a$
$x.2$	**Tuple selection (second element)**
	The second element of a tuple; $(a, b).2 = b$
$\mathrm{dom}\ F$	**Function domain**
	The domain of the function X, or equivalently, the first element of every tuple in F
$F_1 \oplus F_2$	**Function override**
	Combines all the mappings in F_2 with those mappings in F_1 outside the domain of F_2 (*i.e.*, F_2 overrides the behavior of F_1)
$X \lhd F$	**Domain restriction**
	Limits the domain of F only to those elements in X (*i.e.*, $(X \lhd F)(x)$ is only defined for $x : X$)
$X \trianglelefteq F$	**Domain subtraction**
	Eliminates the elements of X from the domain of F (*i.e.*, $(X \trianglelefteq F)(x)$ is undefined for $x : X$)

A.2.7. Numbers

$\mathbb{R}$	**Real numbers**
$\mathbb{N}$	**Natural numbers**
$\#X$	**Set size**
	The size of the set X (as a natural number)
$i..j$	**Number range**
	The set of numbers between and including i and j (*i.e.*, $1..4 = \{1, 2, 3, 4\}$)

min X	**Minimum**
	If X is a set of numbers, gives the smallest number within the set

A.3. Type and Schema Declarations

Given Types

A *given type* is assumed to be defined outside of a Z specification and is assumed to be primitive. While there are subtle differences between a type and its carrier set, a given type definition may also be understood as declaring a set of 'atomic' elements that may be used in a specification:

[T]

Free Type Declaration

A free type declaration is a shorthand notation for declaring recursive structures (such as trees).

The following definition declares T to be a type constructed from either an element (referenced by the name *atom*) or by elements recursively constructed from pairs of elements of type T (using the pair constructor).

$T ::= atom \mid pair\langle\!\langle T \times T \rangle\!\rangle$

For example, given the above free type definition, it holds that: $pair(pair(atom, atom), atom) : T$

Declaration Blocks

The following two blocks are used to declare a global variable x and a global variable y subject to the restriction, $P(y)$:

$x : TYPE$

$y : TYPE$

$P(y)$

Sequential Systems

Sequential systems are also declared using blocks, but are labeled and may be used to describe state changes.

A state is an abstract data type defined in a named block. For example:

StateName

$var_1 : TYPE_1$

$var_2 : TYPE_2$

...

An operation may update an abstract data type, and is defined in a named block such as the following:

OperationName

Δ *StateName*

$var_1' = f(var_1)$

$var_2' = f(var_2)$

...

Note that the current and updated values for the abstract data type are given by the undecorated and decorated (′) names respectively.