UNIVERSITY OF GOTHENBURG

Reasoning with Bounded Cognitive Resources

Abdul Rahim Nizamani

Ph.D. thesis

Department of Applied Information Technology
Chalmers University of Technology & University of Gothenburg

Gothenburg, Sweden 2015

IT Faculty

Reasoning with Bounded Cognitive Resources

Thesis for the Degree of Doctor of Philosophy

Reasoning with Bounded Cognitive Resources

Abdul Rahim Nizamani

UNIVERSITY OF GOTHENBURG

Department of Applied Information Technology
Chalmers University of Technology & University of Gothenburg
SE-412 96 Gothenburg, Sweden

2015

Reasoning with Bounded Cognitive Resources

Abdul Rahim Nizamani

Division of Cognition and Communication,
Department of Applied Information Technology,
University of Gothenburg,
SE-412 96 Göteborg, SWEDEN.

Email: abdulrahim@nizamani.net

ISBN 978-91-982069-8-2

URL http://hdl.handle.net/2077/40579

This thesis was typeset with LaTeX 2ε and compiled with XeLaTeX using the MiKTeX software package. The bibliographical references are generated with BibTeX.

Printed by Chalmers Reproservice
Gothenburg, Sweden, 2015

Dedication

To the forgotten scribes of antiquity, long before the advent of printing press and digital communication, who preserved the knowledge and ideas by their hard and tedious work of accurately copying books with handwriting.

Abstract

Reasoning is an essential element of intelligence. Automated reasoning in formal and symbolic systems is a major topic in computer science and artificial intelligence (AI). Programs for computer-assisted and automated theorem proving are being constructed and used by mathematicians and logicians. Recently, there has been a surge in research in inductive reasoning systems. Inductive programming is an example of applying induction to symbolic systems. Most of the reasoning systems in AI are narrow and specialized to particular tasks and domains. Research work in artificial general intelligence (AGI) aims at producing models of AI that are fully or partially independent of task types. AGI systems are ideally able to learn new knowledge and rules that were not intended during their construction.

Automatic reasoning systems are traditionally developed by using heuristics to limit the computational resources. This thesis aims to produce models of reasoning that use bounded cognitive resources. Since reasoning is a cognitive function, and human cognitive resources (such as working memory) are severely restricted, one possible method of reducing the computational complexity of reasoning systems is to introduce cognitive resources in the system and put limits on them similar to human cognitive limits. Another important aim of this thesis is to unite the deductive and inductive reasoning in symbolic systems, by using Occam's razor as a guiding principle for induction. The thesis uses an exploratory approach to search for a unified model of reasoning in arbitrary domains.

This thesis is a collection of published scientific papers, each contributing iteratively to the construction of a larger and domain-independent model of reasoning in symbolic domains. The first two papers present proof formalisms for first-order logic and description logic that produce comprehensible proofs. The second part comprises of five papers, each extending a model of inductive reasoning that can learn axioms of any arbitrary symbolic domain from random examples. Some of the models in the thesis were able to outstrip human performance in arithmetic, logic and number series problems. This is an interdisciplinary thesis that contributes to a number of scientific areas, mainly formal logic and AGI. Further research in this area can potentially lead to a universal reasoner that is able to learn and reason in more complex symbolic systems such as higher order logics and computer programming.

Acknowledgements

First and foremost, my utmost gratitude goes to my supervisor Claes Strannegård. He guided me throughout the course of my doctoral studies, from the initial phase of getting admitted, to the very end of writing this thesis. We worked together closely during all stages of research and studies.

I am highly grateful to Jens Allwood, my previous supervisor, with whom I had insightful philosophical discussions on widely varying topics. He guided me through some of the courses arranged by Swecog. We worked together in areas other than this thesis work, particularly corpora-based analysis of communication. I also thank Ulf Persson, my secondary supervisor, who worked closely in our research team and whose comments were invaluable in some of the papers.

I cannot forget to thank Kajsa Nalin who helped me in practical issues during my initial time and acted as a mentor. I am also grateful to all other colleagues at the division of cognition and communication, including Alexander Almér, Elisabeth Ahlsén, Gustaf Lindblad and Arvid Karsvall. In addition, I owe my gratefulness to Christer Svennerlind for all the wonderful philosophical and cultural discussions and chats.

It is not possible to enlist all the teachers of the courses, workshops and summer schools, but they deserve to be thanked for their efforts in transferring the knowledge and insights to their students including myself.

I should also mention other PhD students who shared their experiences with me and discussed everyday issues of the doctoral program. These include Muhammad Azam Sheikh and Shafqat Virk at the department of CSE, and Fahd Omair Zaffar, Christian Stöhr, Åsa Fyrberg and Rickard von Haugwitz at the department of AIT. My gratitude also goes to numerous other persons whom I cannot mention all in this short piece of acknowledgements.

After mentioning persons, I feel obliged to mention the institutions and organizations that sponsored my work or helped in my studies. The foremost is Quaid-e-Awam University (Pakistan) that partially sponsored the scholarship for my doctoral studies, together with the University of Gothenburg that provided partial funding towards my research. The graduate school of Swecog was invaluable for its doctoral courses and regular conferences.

Abdul Rahim Nizamani
Gothenburg, Sweden, September 2015

List of Abbreviations

ACT-R	—	Adaptive Control of Thought—Rational
AGI	—	Artificial General Intelligence
AI	—	Artificial Intelligence
AIW	—	Alice In Wonderland
aka.	—	also known as
BNF	—	Backus–Naur Form
BNFC	—	BNF Converter
CHREST	—	Chunk Hierarchy and REtrieval STructures
DAML	—	DARPA Agent Markup Language
DL	—	Description Logic
e.g.	—	exempli gratia (for example)
etc.	—	et cetera (and the rest)
et al.	—	et alii (and others)
FOL	—	First-Order Logic
GGP	—	General Game Playing
GHC	—	Glasgow Haskell Compiler
GUI	—	Graphical User Interface
HTM	—	Hierarchical Temporal Memory
i.e.	—	id est (that is)
IQ	—	Intelligence Quotient
IST	—	Intelligence Structure Test
LNAI	—	Lecture Notes in Artificial Intelligence
LNCS	—	Lecture Notes in Computer Science
LTM	—	Long Term Memory
NARS	—	Non-Axiomatic Reasoning System
OEIS	—	Online Encyclopedia of Integer Sequences
OIL	—	Ontology Inference Layer

OWL	—	Web Ontology Language
PJP	—	Predicting Job Performance
PM	—	Procedural Memory
PSYCOP	—	PSYChology Of Proof
RTS	—	RunTime System
SAIS	—	Swedish Artificial Intelligence Society
URL	—	Uniform Resource Locator (*e.g.,* a web address)
US	—	Universal Search
WM	—	Working Memory

List of publications

This thesis is based on the work contained in the following seven papers, all published in peer-reviewed journals and conferences. The papers are referred by roman numerals throughout the thesis.

Paper-I
C. Strannegård, F. Engström, A. R. Nizamani, and L. Rips, Reasoning About Truth in First-Order Logic, Journal of Logic, Language and Information 22 (1) (2013), pages 115–137.

Paper-II
F. Engström, A. R. Nizamani, C. Strannegård, Generating Comprehensible Explanations in Description Logic, in: M. Bienvenu, M. Ortiz, R. Rosati, M. Simkus (Eds.), Informal Proceedings of the 27th International Workshop on Description Logics, Vol. 1193 of CEUR Workshop Proceedings, Sun SITE Central Europe, 2014, pages 530–542.

Paper-III
C. Strannegård, A. R. Nizamani, A. Sjöberg, F. Engström, Bounded Kolmogorov Complexity Based on Cognitive Models, in: K. Kühnberger, S. Rudolph, P. Wang (Eds.), Artificial General Intelligence, Vol. 7999 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2013, pages 130–139.

Paper-IV
C. Strannegård, A. R. Nizamani, F. Engström, O. Häggström, Symbolic Reasoning with Bounded Cognitive Resources, in: P. Bello, M. Gaurini, M. McShane, B. Scassellati (Eds.), 36th Annual Conference of the Cognitive Science Society, Cognitive Science Society, Austin, Texas, 2014, pages 1539–1544.

Paper-V
C. Strannegård, A. R. Nizamani, U. Persson, A General System for Learning and Reasoning in Symbolic Domains, in: B. Goertzel, L. Orseau, J. Snaider (Eds.), Artificial General Intelligence, Vol. 8598 of Lecture Notes in Computer Science, Springer International Publishing, 2014, pages 174–185.

Paper-VI
A. R. Nizamani, C. Strannegård, Learning Propositional Logic From Scratch, The 28th Annual Workshop of the Swedish Artificial Intelligence Society (SAIS), May 22–23, 2014, Stockholm.

Paper-VII
A. R. Nizamani, J. Juel, U. Persson, C. Strannegård, Bounded Cognitive Resources and Arbitrary Domains, in: J. Bieger, B. Goertzel, A. Potapov (Eds.), Artificial General Intelligence, Vol. 9205 of Lecture Notes in Computer Science, Springer International Publishing, 2015, pages 166–176.

Awards related to this thesis

Kurzweil Prize for Best AGI Paper
awarded to Paper-III in the AGI conference 2013.

Kurzweil Prize for Best AGI Paper
awarded to Paper-V in the AGI conference 2014.

Cognitive Science Society Prize for Best Student Paper
awarded to Paper-VII in the AGI conference 2015.

Table of Contents

Part I

Preamble

1

Introduction

This chapter introduces the notions of intelligence, artificial intelligence, artificial general intelligence and logical reasoning, and outlines the aims and research questions addressed by this thesis. This is an informal introduction to these concepts, further description is provided in the next chapter. The last section lists the scientific papers included in this thesis with their publication details, and specifies my individual contribution in each of them.

1.1 Intelligence

Intelligence is a complex concept and has a large meaning potential [8] (range of meanings in varying contexts). The word "intelligence" literally means the ability to understand. It is derived from the Latin verb *intelligere* which means to comprehend or perceive [9]. Dictionary definitions of intelligence are not very precise and objective. It is usually defined with synonymous words such as understanding, mind, mental ability, or with its sub-concepts like learning, reasoning, planning, etc.

Is intelligence a unique ability of human beings or shared by other biological species too? Is it a biologically evolved capability, or an abstract mechanism shared by non-biological systems too? These intriguing questions have been studied in many scientific areas of investigation and are still puzzling us.

> *"These ideas [theories of physics] cannot explain everything. They can explain the light of stars but not the lights that shine from planet earth.*

To understand these lights you must know about life, about minds."
Stephen Hawking

No theory of physics can explain the night-time lights seen from planet Earth. It requires some theory about minds, relatively advanced minds that are able to innovate and transform their environments. That very advanced nature of minds is often understood in terms of intelligence.

The scientific study of intelligence falls into a number of disciplines. The most important of them is cognitive science that investigates the mind as a whole as well as its constituent structures and processes. Cognitive science is a multi-disciplinary area and involves several sub-disciplines to understand and explain human cognition. Interestingly, there is little mention of the term intelligence within cognitive science, probably because cognitive scientists see it in a different perspective (*i.e.,* cognition). Another major discipline is psychology, which studies human intelligence within a number of subdisciplines. The most visible is psychometrics, which seeks to understand the structure of intelligence and its parts, if any. Psychometric theories of intelligence are based on objective data from tests of mental abilities, commonly called IQ tests. Cognitive psychology and cognitive neuroscience are also concerned with intelligence and study the abstract and biological bases of intelligence respectively.

1.2 Artificial Intelligence

Intelligence is usually considered a uniquely human capability, yet there are numerous biological organisms that exhibit some forms or grades of intelligence [10, 11]. With the advent of computing science, humans started wondering if the "machines can think" too? This is the question considered by Turing in his influential essay "Computing machinery and intelligence" [12]. With programmable machines, algorithms were designed that could perform some of the mental tasks that require higher mental abilities such as reasoning and learning. The study and design of such algorithms and methods form the basis of artificial intelligence (AI), a field aiming to produce intelligent machines. Research in AI includes problem solving, knowledge, reasoning, learning, planning, communication (including natural language processing), perception and other traits of intelligence [13].

Most of the approaches to AI are narrow and only focus on solving a particular problem or a set of similar problems. For example, a computer that can

beat world champions in Chess, cannot even compete with novices in Checkers. The ultimate goal of AI is to produce general intelligence in machines. This type of AI has been called strong AI, and more recently has re-emerged under a new title Artificial General Intelligence (AGI). AGI aims to build computer programs that can simulate general intelligence and be able to solve unforeseen problems, and perform at or beyond human level. [14]

One of the goals of AI was to create artifical systems that surpass or at least compete with human abilities. Most of the early research focused on mathematical models of intelligence, disregarding the *human* aspect. Cognitive science on the other hand led to the construction of systems for modeling human cognition and reasoning, but not necessarily intelligence. This area, called cognitive modeling, constructs artificial systems that can simulate mental functions and processes, often based on experimental data.

> *"Cognitive modelers attempting to explain human intelligence share a puzzle with artificial intelligence researchers aiming to create computers that exhibit human-level intelligence: how can a system composed of relatively unintelligent parts (such as neurons or transistors) behave intelligently?"* [15]

Cognitive models are often, though not always, constructed in cognitive architectures. Cognitive architectures aim to emulate what Newell called the Unified Theories of Cognition [16]. Often starting as theories of cognition, they are transformed into a formal as well as computational model of cognition, wherein cognitive models can be developed and tested. Numerous cognitive architectures exist, deriving from a wide range of theories of cognition. Examples of these architectures include ACT-R [17], Soar [18], NARS [19], CHREST [20], MicroPsi [21], and several more.

Cognitive modeling is one of the important methods employed in the models of AGI. A number of cognitive architectures have been proposed for AGI, such as Soar, NARS and MicroPsi. Intelligence, as understood in AI and AGI, is anthropomorphic in nature (higher constructs exist such as universal intelligence). To construct artificial systems that can display intelligent behavior, it is often the most convenient way to build a cognitive model of a person and use that to achieve the relevant behavior. However, for the purposes of AGI, such cognitive models need not be psychologically realistic, in contrast to cognitive science where such models are expected to precisely mimick mental processes.

Occam's razor, also called the principle of parsimony or simplicity, is a fundamental concept in the theory of science. It is used to select a hypothesis from a set of competing theories that all can account for a given observation. Mathematical formalizations of Occam's razor such as Solomonoff's inductive inference and Kolmogorov's complexity theory have been used as a basis for defining algorithmic universal intelligence (section 2.3).

1.3 Aims and scope

The ability of logical reasoning is a distinct capability of intelligent beings and an essential element of intelligence. Logical reasoning has been a major area of research in artificial intelligence.

Reasoning comes in a variety of flavors, sometimes called modes or forms. The most common categorization divides reasoning into deduction and induction. Pierce added abduction as the third form of reasoning [22]. Other modes of reasoning include analogical reasoning, fallacious reasoning, cause and effect reasoning, etc. [23]

An interesting question is, how people apply reasoning in symbolic mathematical areas such as arithmetic and logic. Humans did not evolve for numerical and symbolic processing. The natural environments in which humans evolved their higher form of intelligence was relatively similar to ancient environments. The use of written language, and later mathematical notations, is quite a recent innovation in the human history. It is not a biological evolution, rather is a result of evolution of culture and ideas. This often hinders people from learning and using mathematics effectively and efficiently. Studying how people can reason in abstract domains of symbols can lead to further understanding of underlying mental constructs, and can potentially be helpful to design cognitively enhanced mathematics education.

The first part of this thesis deals with this question in two types of symbolic logic: first-order logic and description logic. The second part concerns artificial systems and presents models of inductive reasoning that can potentially be considered models of AGI. The goal is to produce a unified model of reasoning in symbolic domains, which incorporates the basic reasoning modes of induction, deduction and abduction. The papers included in this thesis show the iterative process towards achieving that goal.

This thesis is interdisciplinary and incorporates theories, methods and results from multiple scientific areas including cognitive science, artificial in-

telligence, psychology, psychometrics, cognitive modeling, mathematical logic and others. The work presented in this thesis is a potential approach towards achieving AGI in symbolic areas, though it is certainly bound to be limited and is only one of the many approaches for realizing AGI. It is restricted to logical rule-based approach to modeling intelligence, with no probabilistic methods involved.

The rest of this thesis is organized as follows. The next section lists the publications included in this thesis, and shows my personal contributions to these papers as a coauthor. Chapter 2 reviews the concepts, theories and methods relevant to the contributions. A brief summary of the contributions is provided in chapters 3 and 4, discussed in an iterative fashion. It is followed by some discussion and concluding remarks in chapter 5. Appendix A describes the computer programs that are products of this research work and have been used to compute the results in the papers. Part II (printed edition only) comprises the published scientific papers that contribute to this dissertation.

1.4 Publications

This dissertation is a collection of scientific papers [1, 2, 3, 4, 5, 6, 7] together with a cover article (this preamble). All these papers were published in peer-reviewed journals and conferences. The papers are written and published by multiple coauthors. This research is a cross-disciplinary work and requires expertise from multiple scientific areas, which led us to publish together instead of independently. The coauthors are specialists in their respective areas, and we have worked together closely to construct the models presented in the papers.

Very often, the work of constructing models and evaluating them was iterative: an initial sketch was created and then implemented as a computer program. Afterwards, it was modified iteratively together with the computer program, until it was sufficiently correct to be published. Most often, the modifications in the mathematical model led to changes in the software, and quite often the computational properties discovered from the program led to modifications in the model. Designing, coding and updating the automatic theorem provers and other software was primarily my responsibility with support from Claes Strannegård, whereas the design changes were the results of long sessions of discussions and collective brainstorming. It will be very difficult, if not impossible, to separate the individual contributions of the authors in the construction of the mathematical models that constitute the core of the research output

of the papers.

Nonetheless, it is important to lay down my personal contribution in the published papers for the assessment of this thesis. Below, I list the papers of this thesis and briefly explain my individual contributions to each of them. Publication and copyright details are also provided. The papers are not numbered in chronological order, rather sorted logically in order to facilitate the writing of this thesis.

Paper I

C. Strannegård, F. Engström, A. R. Nizamani, and L. Rips, Reasoning About Truth in First-Order Logic, Journal of Logic, Language and Information 22 (1) (2013), pages 115–137.

Publication: This paper was published in 'Journal of Logic, Language and Information', volume 22 issue 1, 2013, published by Springer Netherlands, under pages 115–137. The paper is published with open access under the terms of the Creative Commons Attribution License, which means it can be reproduced by anyone in any medium (provided the original work is properly cited). It is reproduced in this thesis in its published form.

Contribution: Claes Strannegård was the main author of this paper. I was a supporting author and also collaborated in writing. Fredrik Engström was a supporting author in logic and Lance Rips provided support in psychological design of the experiment. The experiment was conducted before the start of my studies and is reported in my master's thesis [24]. However, this publication was the result of an extensive revision of the proof system based on a new analysis of the experimental data. I worked on the statistical analysis of the data, collaborated in the redesign of the proof system, and wrote a computer program as an automated theorem prover to generate shortest proofs of the test items.

Paper II

F. Engström, A. R. Nizamani, C. Strannegård, Generating Comprehensible Explanations in Description Logic, in: M. Bienvenu, M. Ortiz, R. Rosati, M. Simkus (Eds.), Informal Proceedings of the 27th International Workshop on Descrip-

tion Logics, Vol. 1193 of CEUR Workshop Proceedings, Sun SITE Central Europe, 2014, pages 530–542.

Publication: I presented this paper in a poster session at the 27th International Workshop on Description Logics, 17–20 July 2014, in Vienna. This workshop was part of the Vienna Summer of Logic, 9–24 July 2014, a large event hosting twelve conferences and several workshops. The paper was peer-reviewed and was published in 'Informal Proceedings of the 27th International Workshop on Description Logics 2014', edited by M. Bienvenu, M. Ortiz, R. Rosati, and M. Simkus, pages 530–542. These proceedings are volume 1193 of the CEUR Workshop Proceedings, an open-access publication service of Sun SITE Central Europe, operated under the umbrella of RWTH Aachen University. The paper is freely available on the web and is reproduced in this thesis in its published form.

Contribution: Fredrik Engström was the main author in this paper. I collaborated in the development of the model (rules and axioms), and I wrote the software implementation of the automatic explanation generator. The program was run on high-performance computing center at Chalmers University to find the shortest proofs of the test items. I also contributed in writing.

Paper III

C. Strannegård, A. R. Nizamani, A. Sjöberg, F. Engström, Bounded Kolmogorov Complexity Based on Cognitive Models, in: K. Kühnberger, S. Rudolph, P. Wang (Eds.), Artificial General Intelligence, Vol. 7999 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2013, pages 130–139.

Publication: This paper was presented by Claes Strannegård at The Sixth Conference on Artificial General Intelligence, July 31 – August 3, 2013, held at Peking University, Beijing. It was published in a volume named Artificial General Intelligence, 2013, edited by Kai-Uwe Kühnberger, Sebastian Rudolph, and Pei Wang, pages 130–139. This volume contains proceedings of the conference and is included in the subseries Lectures Notes in Artificial Intelligence, part of the series Lecture Notes in Computer Science (LNCS), volume 7999, published by Springer Berlin Heidelberg. This paper received the Kurzweil Prize for Best AGI Paper, sponsored by `KurzweilAI.net`. Springer-Verlag Berlin Heidelberg owns the

copyrights of this paper and permits the paper to be used for defending the thesis. An explicit license was obtained by this author to reproduce this paper in the printed version of this thesis. The paper is reproduced in its official, published form.

Contribution: This paper was mainly authored by Claes Strannegård and myself, with support from the other two coauthors. I collaborated in the construction of the model and the writing. My major contributions were in software development of `SeqSolver`, defining term preference, and computation of results from other software.

Paper IV

C. Strannegård, A. R. Nizamani, F. Engström, O. Häggström, Symbolic Reasoning with Bounded Cognitive Resources, in: P. Bello, M. Gaurini, M. McShane, B. Scassellati (Eds.), 36th Annual Conference of the Cognitive Science Society, Cognitive Science Society, Austin, Texas, 2014, pages 1539–1544.

Publication: This paper was presented by Claes Strannegård at CogSci 2014 conference, the 36th annual meeting of the Cognitive Science Society, held on 23–26 July 2014 in Quebec City, Canada. It was published in the proceedings of the conference edited by P. Bello, M. Gaurini, M. McShane, and B. Scassellati, and published by the Cognitive Science Society, Austin, Texas. These proceedings are open access and are freely available on the web. The copyright for the complete proceedings is held by the Cognitive Science Society, while copyrights for individual articles are held by the authors[1]. The paper is reproduced in this thesis in its published form.

Contribution: In this paper, I mainly contributed in the development of the computational model, collaborated in the brainstorming sessions for constructing the formal model, and wrote the software implementation of Occam. I also collaborated in writing of the paper.

[1]Source: an email message received by this author from the Cognitive Science Society

Paper V

C. Strannegård, A. R. Nizamani, U. Persson, A General System for Learning and Reasoning in Symbolic Domains, in: B. Goertzel, L. Orseau, J. Snaider (Eds.), Artificial General Intelligence, Vol. 8598 of Lecture Notes in Computer Science, Springer International Publishing, 2014, pages 174–185.

Publication: This paper was presented by Claes Strannegård at The Seventh Conference on Artificial General Intelligence, August 1–4, 2014, held in Quebec City, Canada. It was published in a volume named Artificial General Intelligence, 2014, edited by Ben Goertzel, Laurent Orseau, and Javier Snaider, pages 174–185. This volume is part of the subseries Lecture Notes in Artificial Intelligence, in the series Lecture Notes in Computer Science (LNCS), volume 8598, published by Springer International Publishing. This paper received the Kurzweil Prize for Best AGI Paper (for the second time), sponsored by `KurzweilAI.net`. Springer International Publishing Switzerland owns the copyrights of this paper and allows it to be used for the purpose of defending a thesis. An explicit license was obtained by this author to reproduce this paper in the printed version of this thesis. The paper is reproduced in its official, published form.

Contribution: In this paper, I contributed to the formal model and collaborated in writing some of the sections in the paper. I continued the development of the software.

Paper VI

A. R. Nizamani, C. Strannegård, Learning Propositional Logic From Scratch, The 28th Annual Workshop of the Swedish Artificial Intelligence Society (SAIS), May 22–23, 2014, Stockholm.

Publication: I presented this paper at the 28th annual workshop of the Swedish Artificial Intelligence Society (SAIS), held on May 22–23, 2014 in Stockholm. The paper was peer-reviewed and included in the conference proceedings, which will be published by Linköping University Electronic Press. The copyright is held by the authors. The paper is included in this thesis in its submitted form, as official proceedings are not published yet.

Contribution: I was the main author in this paper. I wrote and presented the paper. I wrote the computer program (OCCAM*) and used it to compute the results.

Paper VII

A. R. Nizamani, J. Juel, U. Persson, C. Strannegård, Bounded Cognitive Resources and Arbitrary Domains, in: J. Bieger, B. Goertzel, A. Potapov (Eds.), Artificial General Intelligence, Vol. 9205 of Lecture Notes in Computer Science, Springer International Publishing, 2015, pages 166–176.

Publication: I presented this paper at The Eighth Conference on Artificial General Intelligence, held in Berlin on July 22–25, 2015. The paper is published in a volume named Artificial General Intelligence, 2015, edited by Jordi Bieger, Ben Goertzel and Alexey Potapov, pages 166–176. This volume is part of the subseries Lecture Notes in Artificial Intelligence, part of the series Lecture Notes in Computer Science (LNCS), volume 9205, published by Springer International Publishing. This paper received the Cognitive Science Society Prize for Best Student Paper, sponsored by Cognitive Science Society. Springer International Publishing Switzerland owns the copyrights of this paper and allows it to be used for the purpose of defending a thesis. An explicit license was obtained by this author to reproduce this paper in the printed version of this thesis. The paper is reproduced in its official, published form.

Contribution: I contributed in writing this paper with Claes Strannegård, and collaborated in the construction and development of the computational model. I also contributed with designing and coding the computer program AIW and used it to compute the results in the paper. This model covers most of the functionality in the previous models and required many long sessions of collective brainstorming. The authors met each week and discussed potential changes to improve the model and the results. The computer program evolved together with the formal model, each affecting the design of the other. The paper itself is a short description of the model, and an extended paper is being written to report the complete model in a scientific journal.

2

Theoretical background

Due to the interdisciplinary nature of this thesis, the papers draw upon a number of scientific areas for theoretical and methodological support. Major areas include cognitive science, cognitive modeling, artificial intelligence, and philosophy of science. The next section describes the background of this thesis and relevant previous works. Later sections discuss some of the theories that are important for the discussion of contributions in later chapters.

2.1 Bounded cognitive resources

The model of bounded cognitive resources is central to this dissertation. It has been used in various forms in all the papers contributing to this thesis. It is a simple cognitive model derived from the memory model of Atkinson and Shiffrin [25], which divides the human memory system into three stages: sensory memory, working memory and long term memory. The long term memory was further grouped into declarative memory and procedural memory by Park and Gutches [26]. Sensory memory can be categorized by the various senses, most common types include visual memory and auditory memory.

Human cognitive resources are severely limited in size, creating a bottleneck effect when processing sensory information. The short term memory is known to store 7±2 chunks of information at once in most individuals [27]. Visual short term memory retains visual scene information for only a few seconds. Another interesting cognitive limit is the counting ability, called subitizing. It refers to the ability of most individuals to count instantaneously up to

four items [28]. Although it is not a fixed number, counting more than three or four items requires considerably more time.

Anthropomorphic Artificial Intelligence

The work presented in Paper-I and -II is in continuity of earlier works by Stranneg̊ard and others. The idea of designing computational solutions—particularly logical proof methods—based on psychological experiments was conceived in 2005 [29]. It was called *anthropomorphic artificial intelligence*, in contrast to traditional artificial intelligence that is less connected to human cognitive attributes. The goals of Anthropomorphic AI are two-fold.

1. The first goal is to use cognitive psychology methods (for instance, introspection and experimentation) to guide the design of computational solutions to the problems where humans are usually better than machines. For instance, search algorithms that fail because of a lack of good heuristics, can be guided through cognitive limitations such as working memory capacity.
2. The second goal is to find solutions that are comprehensible for people. For instance, logical proofs computed from traditional automated theorem provers tend to be lengthy, complicated and better suited for machines than for people. Considering the cognitive limitations of humans, proof systems can be designed that produce easily comprehensible proofs for human verification.

The method proposed for anthropomorphic AI is a production system outlined in [29]. The author presents a formal model of a production system restricted by cognitive heuristics, gives some examples, and discusses potential applications of such a model.

Classical AI algorithms that use search can be equipped with an anthropomorphic module that cuts off complex search paths. In this thesis, we tried it specifically on domain-specific AI programs for number sequences, arithmetic and logic. We also used it to construct some domain-independent AGI programs. In all cases the programs without the anthropomorphic module would be computationally expensive. Using such a module, some problems would go from less feasible to more feasible, and others may become computable from undecidable.

Bounded proof systems

Reasoning in propositional logic with bounded cognitive resources was first sketched in [30]. A psychological experiment for studying reasoning processes in propositional logic was reported in [31, 32], similar to the experiment described in Paper-I. University students were recruited as subjects, and their accuracy and latency (mean response time) were recorded for the test questions. Questions were of the format: "*Is this sentence a tautology?*" followed by a propositional sentence. These questions comprised forty tautologies and forty non-tautologies, presented randomly to the participants. The tautologies from this experiment are also used for evaluation in one of the papers contributing to this thesis (Paper-VI).

The authors in [31] also presented a proof formalism for propositional logic constructed with bounded cognitive resources, by modeling and bounding declarative memory, procedural memory, visual memory and working memory, following the Atkinson and Shiffrin memory model [25]. They present two proof systems: **T** for proving tautologies and **NT** for proving non-tautologies. A computer program was developed to automatically generate proofs of the test items in these bounded proof systems. The mathematical complexities of those proofs (formula length, minimum proof length, etc.) were correlated with the psychological difficulty of the problems (accuracy and latency) using Pearson product-moment correlation. Results show high correlations between the psychological and mathematical complexities of the problems. For instance, the correlation between latency and minimum proof length was .89 for tautologies and .87 for non-tautologies.

It should be noted that these high correlations did not occur automatically. Instead, the proof system was designed from the experimental data, and different variations were tested against the experiment results to achieve the highest possible correlations. The experiment was not conducted to evaluate the proof systems, rather to guide their development and make them more psychologically plausible.

The above mentioned approach can also be applied to other kinds of formal logics. Goal-driven bounded proofs in first-order logic have been proposed in [33]. The author suggests a proof formalism for first-order logic that is complete, subformula-preserving and goal-driven, and can be adapted to cognitive modeling for constructing comprehensible proofs, in a fashion similar to those for propositional logic.

Anthropomorphic method for inductive reasoning

The method of generating computations with bounded cognitive resources has been applied earlier to inductive reasoning tasks, *e.g.,* IQ tests. Modern IQ tests involve inductive reasoning and contain specialized sub-tests with inductive questions. Two obvious subsets of problems in IQ tests are number sequence problems and progressive matrices. Some IQ tests consist entirely of such problems (*e.g.,* Raven's progressive matrices).

The method of cognitive modeling using bounded cognitive resources has been applied, in various forms, for solving number sequence problems [34, 35] and Raven's progressive matrices [36, 37]. In [34], the authors describe a model of solving number sequence problems that makes use of a model of human reasoning including a set of cognitive resources. A computer program Asolver is presented that can solve all of the problems of an IQ test called PJP, whereas other mathematical software and online resources could only solve a small subset of the problems. Paper-III in this thesis is a major extension of the work presented in [34] and the results are tested against three different IQ tests.

2.2 Logical reasoning

Reasoning (or simply Reason) is the ability to think logically and consciously derive new information from known facts. It is often associated with logic and rationality, and is also related to argumentation. It is an essential component of intelligence.

Human logical reasoning is studied in a number of scientific disciplines. In cognitive psychology, the mental models tradition and the mental logic tradition are two prominent subfields. Mental logic tradition tries to explain human deductive reasoning in terms of logical rules [38]. PSYCOP is a computer program that simulates human reasoning with mental logic and uses a formalism similar to natural deduction [39]. The theory of mental models argues that formal logic is insufficient to model human deductive reasoning, as the later is affected not only by the rules but also by the content. It postulates that human reasoning occurs by modeling the possibilities that can arise from a discourse [40].

In cognitive science, human reasoning has been modeled in cognitive architectures such as Soar [18], ACT-R [17], CLARION [41], Polyscheme [42] and NARS [19]. Some of the cognitive architectures are specialized for human

cognition and try to accurately model mental reasoning. Others such as Soar and NARS are used for general reasoning by relaxing the anthropomorphic limitations. Many AGI systems are similar to cognitive architectures, though not restricted to mental reasoning.

Logical reasoning is often categorized into different modes or forms. A traditional division distinguishes between deductive and inductive reasoning. Deductive reasoning derives the conclusion based on complete knowledge of the premises. Induction on the other hand is based on deriving rules from incomplete knowledge. Inductive reasoning can be rule-based or stochastic. Stochastic reasoning deals with probability distributions and estimation of random variables from partial observations [43]. This thesis does not deal with stochastic reasoning.

In mathematics, reasoning is often modeled using formal symbolic logics. Although it is highly debated whether mental reasoning can be accurately modeled with formal systems or not, reasoning in general can be expressed and formalized in symbolic logics. There are many systems of formal logic. Below I describe a few of them that are relevant to this thesis.

Symbolic logics

Logic and logical inference have been studied for millennia in the human history. Formal logic started in ancient Greece, India and China, and developed further in the Middle Ages by Islamic and Christian philosophers. Modern formal logic traces backwards to Greek logic, particularly Aristotelian syllogism.

Recently, logical formalisms have adopted symbolic representations with strict semantics. One of the simple symbolic logics is propositional logic. The atomic sentences in propositional logic are indivisible statements represented by a *proposition symbol*. Each such symbol can be either true or false. For example, the statement "it is raining" can be assigned a symbolic label A, where A can either be true or false. Propositional logic does not involve time, while in temporal logics time is explicity modeled. Another classical symbolic logic is the first-order logic (FOL), also called predicate logic or predicate calculus, which uses predicates and quantified variables over objects. It can express quantified statements such as "all apples are green" or "there is a red temple". Many variants of FOL exist, for instance those with different quantifiers.

Description logic is a family of logical formalisms widely used for knowledge representation in real-world applications, for instance in medical ontolo-

gies and the semantic web. It is more expressive than propositional logic but computationally more efficient than first-order logic, and can be regarded as a guarded fragment of first-order logic [44]. A number of computational languages have been constructed to implement DL (or a variant of it), which include Web Ontology Language (OWL) [45], DARPA Agent Markup Language (DAML), Ontology Inference Layer (OIL) and DAML+OIL [46].

Due to its popularity in building ontologies for knowledge-based systems, many automated reasoners are available for DL and its sub-languages. Protégé is one of the well known reasoners for DL, which is a set of tools to construct domain models and ontologies. Other DL reasoners include FaCT++ [47] and Pellet [48]. There have been a number of competitions of automated DL reasoners. One of the workshops in the Vienna Summer of Logic 2014 was OWL Reasoner Evaluation Workshop. In addition to the regular paper presentations, this also hosted a competition of OWL reasoners on different tasks with real-world and challenging ontologies. Results of this competition were announced in the annual DL workshop 2014, attended by this author.

The classical formal logics include a binary version of truth, consisting of true and false values, often denoted with $\top$ and $\bot$ symbols respectively. Non-classical logics can have more than two truth values. Three-valued logics such as Priest logic P_3 include a third value of undefined or indeterminate $\mathtt{I}$. Four-valued logics consider additional truth values. Fuzzy logics deal with approximate truth values in the range $0..1$, with 0 standing for absolute falsity and 1 for absolute truth.

Methods and formalisms developed in this thesis use binary truth values but can be extended to multi-valued logics with discrete truth values. However, they are not appropriate for fuzzy logics which deal with continuous values.

Proof systems

There are several formal systems of deductive reasoning in symbolic logics. A common system of logical proofs is *natural deduction*, which expresses deductive reasoning in terms of inference rules. The modern version of natural deduction was first proposed by Gentzen [49, 50]. Other deductive proof formalisms include Hilbert calculus, sequent calculus [51], natural deduction in sequent calculus style [51], and the method of analytical tableaux [52]. In artificial intelligence, automated reasoning systems include resolution system with unification [53] and Stålmarck's method [54].

Soundness and completeness are two essential properties of a logical proof system. Soundness is the property of a formal system such that the theorems in that system only derive true formulas. If any false formula can be derived in a formal system, then the system is known to be unsound. In comparison, completeness is the property of a logical system that every true formula can be derived in that system. Thus, if there exists a formula that is true in a system but cannot be derived in the system due to the lack of a necessary theorem, then the system is incomplete.

Soundness and completeness are fundamental attributes of any proof formalism. In Paper-I, a proposition is given about soundness and completeness of the proof system $\mathcal{T}_M$. An informal proof for this proposition is also provided. The proof system for DL in Paper-II is also sound and complete (a short sketch of its proof is given in the paper).

The inductive reasoning system described in Paper-V was able to learn a complete theory of integer arithmetic. Thus, it can theoretically solve any problem of elementary arithmetic, provided infinite computational resources. While in Paper-VI, we did not find a complete theory of propositional logic with the same system. This however was not the intended goal of the system.

In addition to soundness and completeness, logical proof formalisms can have other desirable properties. The proof systems of first-order logic and description logic, described in papers I and II, produce proofs that are linear, local and goal-driven. These properties are not necessary for any proof system, but are useful for producing comprehensible and easy-to-follow proofs.

- A proof is *linear* when every sentence in the proof is derived from a single sentence rather than a set of sentences. Such a proof is a linear sequence of sentences.
- A proof is *local* when every step in the proof follows from its immediate predecessor. Thus, to check whether a sequence of steps is a valid proof, it is sufficient to look at two consecutive steps at a time.
- A proof is *goal-driven* if it starts with a goal—the desired conclusion—and by applying rules, ends in the true statement $\top$ (or the false statement $\bot$ for proving falsity).

Principle of Compositionality

The principle of compositionality is the principle that the meaning of a complex whole is determined from the meaning of its constituent parts and the rules

used for combining those parts [55]. In semantics, it is called the principle of semantic compositionality, and is often known as Frege's Principle.

The main rule used in the deductive model of bounded computations, used throughout this thesis, is designed from this principle. Called *Substitution* (in later papers, *Rewrite* or *Deep Rewrite*), it replaces a sub-expression A in a larger expression T with another equivalent expression B. If A and B are equivalent, then the meaning (*e.g.,* truth-value) of T should not change. For instance, in arithmetical expressions, we can always replace an expression with its value. *E.g.,* $(2 + 3) * 4$ will still be computed as 20 when $(2 + 3)$ is replaced with 5.

This principle is not accepted as a universal truth, rather as a philosophical view. In fact, there are many more arguments against it than in its favor [56]. Many of the discussions about its validity revolve around the notion of compositionality and how and where it can be accommodated, as Pelletier puts it:

> "... and that other discussions are best described as 'how compositionality can/cannot be accommodated within theory X' rather than whether The Principle is or is not true." [56]

Apart from Substitution, there are a number of rules used in the papers that do not follow this principle (or follow it in a different style). The rules *Weakening* and *Strengthening* in Paper-I are two instances that do not follow compositionality of logical expressions. These rules can rewrite a term, but not a subterm inside it with another term. Furthermore, the equivalence of two terms A and B is unidirectional for these rules: A can be replaced with B, but not in the other direction.

The Strengthening rule in Paper-II uses a restricted version of compositionality. A term A can be replaced with B inside a larger term T if A is *negation free* (does not appear inside a negated expression). The equivalence is still unidirectional.

2.3 Occam's razor

The principle of parsimony, also known as the Occam's razor, is a scientific principle for preferring simplicity when deciding between competing theories. It is generally considered a fundamental tenet of modern science and is used in many different scientific disciplines. Most leading scientists have accepted it in one or the other form [57]. Einstein is quoted to have stated:

> "The grand aim of all science…is to cover the greatest possible number of empirical facts by logical deductions from the smallest possible number of hypotheses or axioms." (Einstein, quoted in [58])

The principle of simplicity was first devised by William of Ockham (1285-1349), translated approximately to "Entities should not be multiplied beyond necessity" [59, 57]. Modern versions of this principle are more elaborate than this original statement, and different definitions and formalizations exist. The basic notion of simplicity is a universal scientific proposition, though different interpretations exist that can lead to different formalizations and varying levels of scientific acceptance. For instance, consider the following two common interpretations of the Occam's razor: [59]

1. Simplicity is a goal in itself.
2. Simplicity leads to greater accuracy.

The first is often equated to a preference for selecting more comprehensible hypotheses and is generally accepted to be valid. The second one is problematic and is not considered a universal principle. Accuracy does not necessarily lead to comprehensibility and accurate models are often more complex and less likely to be easily understandable. One possible method to achieve comprehensibility is to extract a simpler version of a more complex but accurate model by abstraction.

The first interpretation is generally accepted; however, simplicity itself can have many definitions, and not all of these can lead to comprehensibility. Simplicity is not always equivalent to comprehensibility. In inductive models, for instance, a rule is more comprehensible if it is consistent with previous knowledge.

This thesis uses the principle of simplicity mostly as a heuristic technique to select an axiom from a set of axioms that all are valid for a given problem (observation). Multiple definitions of simplicity are used for this purpose, for instance, the size of an axiom (*i.e.,* number of symbols) and the size of the computation resulting from the use of an axiom to solve a particular problem. A number of other preferences are also utilized for further comprehensibility and elegance.

A number of mathematical formalizations of Occam's razor exist, for instance, Solomonoff's theory of inductive inference and Kolmogorov complexity.

Solomonoff probability

Ray Solomonoff (1926–2009) proposed a method to assign *a priori* probabilities to observations, which is generally called algorithmic probability. It combines Occam's razor with Epicurus' principle of multiple explanations, and uses the Bayes' rule and Universal Turing Machines. Solomonoff's algorithmic probability is incomputable and can only be approximated. [60, 61]

Kolmogorov complexity

Andrey Kolmogorov (1903–1987) proposed an approach to analyze patterns in sequential data or other data structures. Kolmogorov complexity [62] is a formal description of the complexity and simplicity of data. For instance, a string can be described either by writing it down in full, or by writing a shorter description that can be used to produce the full length string.

Kolmogorov complexity $K(x)$ of an object x is formally defined as the length of the shortest program p that when run on a universal computer U, outputs x and then halts. The definition is formally written as:

$$K(x) := \min_{p}\{\ell(p) : U(p) = x\}$$

where $\ell(p)$ is the length of a program p [63].

The major problem with Kolmogorov complexity is its incomputability. Some bounded versions of Kolmogorov complexity are computable, for example, the time-bounded "Levin" complexity.

Levin complexity

Leonid Levin proposed a computable version of Kolmogorov complexity, called Kt or Levin complexity. The main idea is to add logarithm of the time a program p takes to output a string x to its length $\ell(p)$. It is formally written as:

$$Kt(x) := \min_{p}\{\ell(p) + \log(\mathsf{time}(p)) : U(p) = x\}$$

Essentially, Levin complexity penalizes the slow programs with the time they take to produce their output. The theoretical algorithm that runs the programs in parallel to find Kt is called Levin's Universal Search (US). Though it is bounded by run time and computable in theory, the algorithm is not practically

viable in its original form due to unrealistically large multiplicative constants in the running time [63].

Paper-III presents a version of Kolmogorov complexity in the case of number series extrapolation problems, which is bounded by cognitive resources and psychological difficulty. Although it is a special case, the notion of bounding computational complexity by cognitive limitations is a general idea, which is further explored in later papers.

2.4 Artificial General Intelligence

Research in artificial general intelligence (AGI) receives a variety of approaches and methods. Considering the very nature and complexity of intelligence, no single method may be able to produce realistic models of AGI for real-world applications. Although the goal itself is to have a single model of intelligence that can learn and solve any problem that a human is able to, this has not been achieved yet and might be possible in near (or distant) future. However, today many methods exist for designing artificial systems with general intelligence of a lower grade or specific to a subset of the problems. A historical review of AGI methods and approaches is given in [64] together with a theoretical account of *intelligence* as used in AGI. Modern methods of AGI are summarized in [65].

One of the oldest traditions in AI and recently in AGI is symbolic systems that can learn, produce, store and modify symbolic entities. Symbolic cognitive architectures have some form of explicit representation of working memory, that draws upon long term memory for storing and retrieving knowledge.

Some methods in AGI assume that symbolic processing can emerge automatically from lower level subsymbolic processing. These designs can be called emergentist approaches to AGI. One of the oldest methods in this tradition is artificial neural networks, which come in various forms and designs. A more recent method is Hierarchical Temporal Memory (HTM) [66].

Computational neuroscience approach to AGI is a set of methods to simulate whole brains at varying levels of abstraction and accuracy. The argument is that human-level artificial intelligence can be realized by emulating the human brain down to molecular or atomic level. Major projects relevant to brain emulation include IBM's "Blue Brain Project", currently hosted by EU under the name "Human Brain Project," China-Brain Project [67], and Japanese Fifth Generation Computer Systems project [68]. Emergentist cognitive architectures are also employed in artificial life systems and developmental robotics.

Some architectures and AGI systems take an integrative approach by merging the symbolic and emergentist methods. The symbolic and subsymbolic distinction is already a matter of dispute, the hybrid approach makes it further vague by making use of both methods towards a single goal. Many symbolic systems already use probabilistic and fuzzy logic methods, blurring their status as purely symbolic. For example, ACT-R has subsymbolic structures and NARS uses fuzzy logic, though both are categorized into symbolic cognitive architectures. However, some architectures explicitly use symbolic and subsymbolic structures in their design. Examples of hybrid architectures include CLARION, CogPrime, and MicroPsi.

Another important tradition in AGI is universal algorithmic intelligence. Its roots go back to Solomonoff's probabilistic inference and Kolmogorov complexity. The aim is to study theoretical and computational essence of general intelligence, without considering the actual methods to implement the models. Some particular cases of universalist approach are Marcus Hutter's AIXI system [69] and Juergen Schmidhuber's Goedel Machine [70]. These systems are not practical for realizing AGI as they require infinite computational resources. Scaled-down versions of these systems have also been designed, which are theoretically computable, yet still computationally intractable.

2.4.1 Symbolic AGI

A major part of this thesis is dedicated to constructing generic models of symbolic reasoning, and can be classified as part of the symbolic AGI tradition. I will briefly mention some of the methods in symbolic AGI.

A common approach to symbolic AGI is symbolic cognitive architectures, which are large frameworks for developing, running and testing cognitive models. The models presented in this thesis can potentially be constructing in some of the AGI architectures such as Soar or NARS. However, this requires a considerable amount of effort, not very different than the effort of implementing the models in a standard programming language. Furthermore, every architecture is different and there is a large learning curve before being able to implement a large-scale cognitive model in it. Therefore, the reasoning models of this thesis were implemented in a general-purpose programming language as it was quicker and easier to work with.

An interesting case in symbolic AGI is the General Game Playing (GGP) project and competition at Stanford University [71, 72]. GGP involves creat-

ing computer programs that can play strategy games without knowing the rules of the game, which are provided at runtime. A GGP system can play any game that can be described using some game description language. GGP players are generally able to play a wide array of possible games, including complex games like Chess, games in static or dynamic environment, games with multiple players and with or without communication between players, games involving chance and partial information. Such programs cannot rely on algorithms designed for specific games, and must learn or encode general strategies for playing.

Inductive programming is a subfield of automatic programming aimed at automatically learning computer programs from incomplete specifications, such as input/output examples or program traces [73]. There are many approaches to inductive programming, including inductive functional programming and inductive logic programming. Other similar approaches include machine learning, grammar induction, structural mining or structural learning, constraint programming, probabilistic programming, and genetic programming.

Inductive functional programming uses functional languages for learning programs, including Lisp and Haskell. Systems that can automatically find computer programs include ADATE [74], IGOR2 [75] and MagicHaskeller [76, 77]. Such systems are able to find small programs (or functions) based on a small set of examples. Even though such programs can be helpful in some cases, they are bound by the choice of language and the set of preliminary functions they can use. I tried MagicHaskeller to find patterns for number series problems used in Paper-III without success. Though other programs exist that can efficiently solve complete IQ tests, they are specific to this task and cannot generalize to arbitrary sequences of non-numeric symbols.

2.4.2 Some comments on cognitive modeling

Cognitive models are constructed at different levels of abstraction. The most detailed cognitive model would be a simulation of the whole brain, for instance, the EU's Human Brain Project. However, most often, the cognitive models are built on a much higher level of abstraction, removing or hiding the real complexity of a human brain.

Cognitive architectures are often grouped into symbolic and subsymbolic, although the division is not so tangible. Some are specifically denoted as hybrid, combining symbolic and subsymbolic processes. They can also be graded on a scale of anthropomorphism: some are more relevant to the *human* mind than

others. For instance, ACT-R tries to model human cognition as accurately as possible on a symbolic level. In comparison, Soar is better suited for modeling artificial agents than biological ones. While both of them have many similarities in structure and function, in practice their intended goals are rather unlike.

In the field of AGI, there is a common understanding that the goals of AGI are to reach human-level general intelligence. However, there is no specific requirement for designing models of *human-like* general intelligence. The relation between human and artificial intelligences within AGI is quite loose. However, one of the most popular approaches towards AGI is to utilize human cognitive models, as humans are the best example of general intelligence (biological or otherwise) known to this date.

Cognitive models in AGI are not required to be psychologically plausible and anthropomorphic: they can very well be inaccurate models of human cognition. In fact, the principle of abstraction ensures that every cognitive model is as inaccurate as any other, compared to the biological brain system. The inaccuracy is directly related to the level of abstraction of a cognitive model: the higher the abstraction level, the less accurate and plausible a model will be.

Cognitive models can be grouped by psychological plausibility into bidirectional and unidirectional. The ones that more or less accurately model cognitive processes of human mind can be considered bidirectional; the ones that are inaccurate with regards to human cognition yet useful in a model of artificial intelligence can be considered unidirectional. Direction can be thought of as the relation between computational model and psychological model of a cognitive process.

The cognitive model that is used in this thesis, called the bounded cognitive resources, is a unidirectional cognitive model. It may not be a very elaborate model of deductive reasoning, yet it captures the basic cognitive functions and structures that are employed during deductive reasoning. It models the working memory explicitly as a term-rewriting system in which terms are bounded by a maximum size (similar to short term memory capacity) and the length of a computation is also limited to a max size (inspired by span of attention). The long term memory is modeled as a linear theory containing facts and rules. The last paper elaborates the long term memory contents with frequency and other attributes that can be useful for better memory retrieval.

This simple cognitive model has worked for our objectives as reported in the papers. Nonetheless, it is always possible to improve it for better results.

3

Contributions in Logic

The papers contributing to this thesis can be grouped in two parts. The first part contains two papers that pertain to mathematical logic. This chapter will summarize the contributions of these two papers.

Papers I and II present models of deductive reasoning in symbolic logics and form the grounds for the later papers, which employ the same deductive model in inductive systems. The proof systems described below use the notion of cognitively bounded computations to produce short and comprehensive proofs. The papers described in the next chapter also use bounded computations as the main reasoning system for deduction, and use it as an advantage for induction.

The first section summarizes the proof systems designed with the bounded cognitive resources model, for proving truth or falsity of first-order logic sentences in finite models. The second section describes the proof model of comprehensible explanations of entailments in description logic.

I Reasoning About Truth in First-Order Logic

Paper-I describes a proof formalism for verifying truth or falsity in first-order logic that is designed for human comprehension and incorporates elements of psychology, and uses explicit models of cognitive resources. It is a formal model of human deductive reasoning in the field of model-based first-order logic.

The paper first describes a psychological experiment that was conducted to investigate problem-solving in the domain of logic. Ten computer science

students participated in the experiment, where they were asked fifty questions of the form: "Is sentence A true in model M?", where A was a sentence in first-order logic and M was a finite directed graph with colored nodes. A sample problem is depicted in Figure 3.1. Test items (formulas and graphs) were manually selected from a set of randomly generated samples, half of them were true and the others false. The test was conducted on computers and questions were randomly presented to each participant. Their answers and response times were recorded for each item.

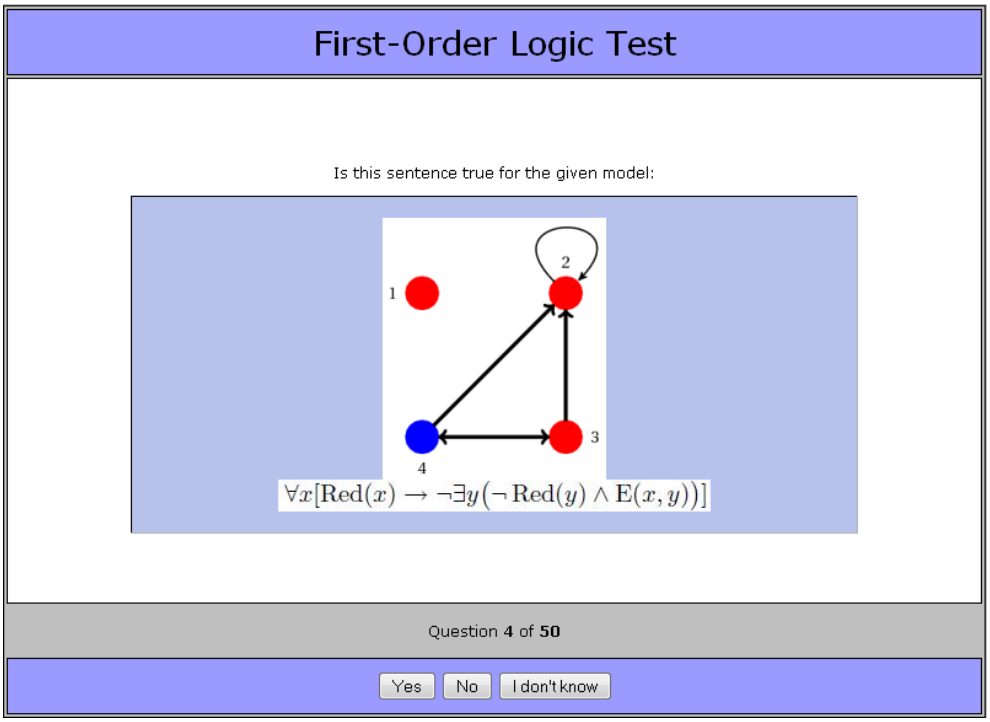

Figure 3.1: Screenshot of the web-based user interface for the experiment. Maximum time to answer a question was 90 seconds. Red(x) means that node x is colored red, and E(x,y) means there is a directed edge from x to y. Solution to the above problem is No for $x = 3$ and $y = 4$.

Next, the paper presents a proof system $\mathcal{T}_M$ for proving truth of logical sentences in finite models, and a corresponding system $\mathcal{F}_M$ for proving falsity. These proof systems are calibrated with the experimental data as explained later.

Proof Systems $\mathcal{T}_M$ and $\mathcal{F}_M$

The proof system $\mathcal{T}_M$ is designed for proving logical sentences to be true in a finite model M. Unlike traditional proof systems in logic, this is a goal-driven proof system that produces local and linear proofs. It is an extension of the bounded cognitive resources model, in which a proof can be seen as a bounded computation from a logical sentence A to the truth value $\top$.

The main components of $\mathcal{T}_M$ are axioms and rules. Axioms model the declarative memory contents of a fictitious logician, and form the knowledge about sentences and models. The results in the paper were computed by using axioms from logic text books that we expected the participants to know by heart. Rules are similar to procedural memory and are used for rewriting sentences. $\mathcal{T}_M$ contains two main rules: Substitution, which is the main rule in the system and is based on the principle of compositionality, and Strengthening which is a non-deep rule for rewriting expressions based on implications. A variant or special case of substitution is the Truth Recall rule which replaces a formula A with $\top$ if A is known to be true (in the long term memory).

The proof system $\mathcal{F}_M$ is designed for proving falsity of a logical sentence in a finite model. The axioms in $\mathcal{F}_M$ are the same as in $\mathcal{T}_M$, rules are designed for falsity. It uses the Substitution rule for rewriting that is identical to the one in $\mathcal{T}_M$, Weakening rule for implications, and a special case of substitution Falsity Recall.

Bounded Proof Systems

$\mathcal{T}_M$ and $\mathcal{F}_M$ are general proof systems without any limitation on the size of formulas or proofs. The proof systems $\mathcal{BT}_M$ and $\mathcal{BF}_M$ are described in the paper as resource-bounded versions of $\mathcal{T}_M$ and $\mathcal{F}_M$ respectively. To that objective, first the length of a formula is defined as (roughly) the number of nodes in its parse tree, counting the uninspected (un-expanded) subtree as size 1. Then, the bounds are defined as follows:

- Working memory capacity: The maximum length of a formula that can fit in WM is set to 8.
- Sensory memory limit: The rule Substitution can only use axioms of length 7.

These proof systems were implemented in a computer program written in Haskell. The program works as an automated theorem prover, and finds the

shortest computations/proofs automatically given the problem and the long term memory (set of axioms). It is described in Appendix A.

Complexity measures

To determine the psychological complexity (difficulty) of a problem, the paper first defines a number of mathematical complexity measures of problems. The difficulty of a problem can be related to the logical sentence (such as formula length, number of quantifiers, number of negations, or a combination of these), or it can be related to the size of the model (number of nodes, edges, etc.). Or, it could be a combination of these two complexities, *e.g.,* a linear or polynomial combination.

However, it is not obvious if any of the above numerical complexities can estimate the psychological difficulty of a problem, which may depend on more intricate relations between the sentence and the model. One such type of relations is the length or size of the computation to solve that problem (in this case, a proof). Thus, we defined two more complexity measures to take into account the complexity of the solution: minimum proof length and minimum proof size. Here, length refers to the number of steps in a proof and size refers to the sum of formula lengths of all proof steps. These were computed using the computer program that searched for and found the shortest proofs for all test items.

Results

The bounded proof systems, their rules and axioms, were the result of many brainstorming sessions for finding the best model given the experiment results. We tested many variations to find the one whose numerical complexities best correlated with the psychological difficulty.

It turned out that model size had no effect on problem difficulty, probably because the graphs used in the experiment were small and homogeneous. Formula length was moderately correlated with difficulty, as was the size of participants' working memory. Highest correlations were obtained between minimum proof size and item latency, validating our initial argument that psychological difficulty is more than a combination of formula and model sizes.

However, the results did not meet our expectations formed from results in an earlier experiment in propositional logic [31]. This may be because first-order logic is considerably more difficult than propositional logic. The mean

latency in propositional logic experiment was 16.0 seconds for tautologies and 14.5 seconds for non-tautologies [31], while in FOL experiment, it was 33s for true items and 37s for false items. In general, more complex tasks require better cognitive models to simulate. Thus, one possible way to improve the results would be to use additional cognitive resources and an improved cognitive model for the proof systems $\mathcal{BT}_M$ and $\mathcal{BF}_M$.

Comments

Proofs in the proposed proof systems are *psychologically* linear. The substitution rule uses an axiom A to rewrite a proof step T to T'. Mathematically, it would not be correct to call this a linear proof rule, as T' is derived from both A and T. However, A comes from the declarative memory contents that form the common knowledge of first-order logic, thus the main inference is from T to T'.

Example 1. Following is an example of a proof that is psychologically linear, although the derivation comes from more than one piece of knowledge. $\top$ is derived from $\top \vee \mathsf{C}$ and $\top \vee \mathsf{x} \rightarrow \top$ (where C is a formula and x a variable).

$$\frac{\top \vee \mathsf{C}}{\top} \text{ Substitution } (\top \vee \mathsf{x} \rightarrow \top)$$

In logic, proving that A is false is equivalent to proving that $\neg$A is true. From a logician's point of view, the system for truth $\mathcal{T}_M$ can be used to compute proofs for false test items, with only a small increase in the length of a FOL sentence ($+1$ for negation symbol). However, negations are cognitively demanding and there is overwhelming evidence that negated statements are more difficult to prove [78, 79, 80]. The results of the experiment reported in the paper also support this: negations were correlated to reduced accuracy. Considering this, using $\mathcal{T}_M$ for false items may not be a good choice for systems designed for cognitive accessibility.

The design of long term memory (set of axioms) in the automated proof finder is an important task. To achieve better correlations between proof complexity and experimental results, the axioms in the long term memory (LTM) must be reasonably simpler and presumptuously known by the participants. The skill level of the participants will also affect the design of LTM axioms—expert participants will presumably know expert rules; novices will have simpler rules

in memory. Thus, it is quite possible to find shorter and simpler proofs by including advanced axioms in the LTM, at the cost of making those proofs accessible only to the experts. The participants in the reported experiment were university students of logic with only moderate knowledge of first-order logic. Therefore, advanced axioms such as the following were excluded from the LTM, even though such axioms may be found in standard textbooks of logic:

$$\forall x \mathrm{P}(x) \rightarrow \neg \exists x [\neg \mathrm{P}(x)]$$

The paper gives mean lateny and accuracy of the experimental data, however the variance is missing. Table 3.1 provides the means and standard deviations of the data.

	True		False	
	mean	sd dev	mean	sd dev
Latency (seconds)	32.6	9.4	37.2	11.6
Accuracy (%)	74.2	15.0	66.9	20.0

Table 3.1: Means and standard deviations of the experimental data in Paper-I

II Generating Comprehensible Explanations in Description Logic

In Paper-II, the method of bounded computations was applied to description logic, to construct a deductive proof system and an automated theorem prover to produce comprehensible explanations in DL. This proof system is designed to compute cognitive complexity of justifications in DL, while simultaneously producing bounded explanations similar to logical proofs.

Description logic is the dominant logical formalism for constructing ontologies, for example, medical knowledge bases and the semantic web. Many automated reasoners can validate logical entailments of a query by finding a justification, *i.e.,* a small number of facts and rules that are sufficient to show whether the query is true.

A justification is the least number of theorems sufficient to prove a query. While such justifications may be optimal in mathematical sense, many such jus-

tifications are difficult to understand for a human user. The authors in [81] explored the cognitive complexity (or psychological difficulty) of abstract justifications, and carried out an experiment with a small number of subjects to explore it. Using the experimental data from that study, we tried to construct a proof model for deriving proof of such justifications using basic axioms of DL, similar to the one built for first-order logic. Such a system was not just useful for explaining to the user why the query was true, it could also be used to measure cognitive complexities of different justifications in terms of the smallest proof length or size.

Explanations in DL

A user can query an ontological reasoning system and ask the question: is the logical statement φ valid in the set of hypotheses Γ? If the answer is yes, this is an *entailment* of φ in Γ, written:

$$\Gamma \vdash \varphi$$

If the reasoner deduces φ to be valid in Γ, it can output the smallest subset of axioms in Γ from which φ can be deduced. This subset of axioms $\Gamma' \subseteq \Gamma$ is the *justification* for the entailment $\Gamma \vdash \varphi$, such that $\Gamma' \vdash \varphi$.

Modern reasoners such as Protégé can find justifications in large ontologies in reasonable time. However, they go no further than this, and produce no explanations of why the entailment is valid given the justification. Such an explanation may be necessary for the user, who otherwise may not be able to deduce $\Gamma' \vdash \varphi$.

Proof system

A proof system is proposed that produces deep and linear proofs (*aka.* explanations) to the justifications in DL. It operates with sets of formulas and a number of manually constructed rewrite rules, and produces bounded proofs for better human understandability.

A computer program called DLvalidity was implemented to search for shortest proofs to justifications in the proposed proof system. The program was implemented in Haskell and used the standard breadth-first algorithm to search for shortest explanations. A number of restrictions were introduced in the program to reduce computational complexity of the search, such as acyclic proofs. See Appendix A for further details.

Evaluation

We used our explanation generator program to find the shortest proofs to the problems used in the experiment [81]. The complexity measures of the generated proofs were matched with those provided by the authors of [81]. Due to the scarcity of data, it is hard to find a numerical relationship between the two proposed complexities. However, based on our earlier experiments, we believe that our proof system for DL is better suited for human understandability due to its use of bounded cognitive resources.

Comments

An evaluation of the proposed proof system with regards to comprehensibility of explanations is not given in the paper. The reason is the scarcity of data on which the system was built: only six justifications were used in the experiment. Furthermore, those justifications consisted of symbolic concepts which adds to the difficulty as concrete concepts may be easier to understand. The proof system can indeed be evaluated properly by an additional experiment with similar justifications. However, it was beyond my resources to conduct such an experiment.

4

Contributions in Artificial General Intelligence

The first two papers, summarized in the previous chapter, used bounded cognitive resources to construct proof systems in mathematical logic. The models were deductive and the rules were manually crafted to match the results with the empirical data. No inductive reasoning or learning was possible. In the following papers, the method of bounded proofs is redesigned as *bounded computations* and is merged with the Occam's razor to construct models of reasoning that include both deduction and induction.

Bounded computations

The papers summarized in the present chapter use bounded computations as their core deductive mechanism. Bounded computations are similar to bounded proofs mentioned in the previous chapter, and are computations produced by a term-rewriting system that follow the restrictions defined by bounded cognitive resources.

Following is a generic version of the bounded computations model described semi-formally. Each paper in this chapter redefines this generic model and adds extensions that will be briefly described in the respective sections. Refer to the papers for a complete description.

Definition 1 (Term)**.** A term is a finite binary tree whose nodes contain values or variables. A term is open if any of its nodes contains a variable; otherwise it is

closed.

The nodes in a term contain values that can be of various types. For instance, the model in Paper-III uses positive integers as values, and the model AIW in Paper-VII uses strings. For a system that is aimed to be independent of types, string values can be used, preferably in Unicode encoding.

From here onwards, terms will be written as strings using standard practice of applying brackets to disambiguate subtrees. Binary operators will be written in infix form.

Example 2. Following are examples of closed terms:

Arithmetic: (1), $(2+3)$, $(5 > 1)$, $(\mathsf{even}(3))$, (-1), $((1\#0)+5)$, $(2\times(9\div 3))$

Logic: $(\top)$, $(\neg\top)$, $(\mathsf{P}\vee\mathsf{Q})$, $(\mathsf{A}\vdash\mathsf{B})$, $(\mathsf{C}_1\sqsubseteq\mathsf{C}_2)$, $(\mathsf{raven}(\mathsf{A})\vee\mathsf{swan}(\mathsf{A}))$

English: (Alice # runs), (Bob # plays), (She # (walks # fast))

Definition 2 (Term size)**.** Term size is defined as the number of nodes in the tree. Special subtrees can be assigned a size of 1.

In the first two papers summarized in previous chapter, the uninspected parts of a term were assigned a size of 1. This reflected the psychological difficulty of an unexamined formula stored in external memory (*e.g.,* computer screen).

Definition 3 (Rule)**.** A rule is an expression of the form $t_1 \oplus t_2$, where t_1 and t_2 are terms and $\oplus$ indicates the type of rule.

The terms in a rule can contain variables, which carry values from t_1 to t_2. An added condition on rules is purity, *i.e.,* t_2 can only have variables that appear in t_1. Rules can be of different types. In general, there are deep and shallow rules. Deep rules can rewrite a subexpression inside a larger expression, while shallow rules can only rewrite a complete expression but not its constituent parts.

Definition 4 (Theory)**.** A theory is a finite set of rules.

Theories are collections of rules. An agent's long term memory is modeled as a theory. In Paper-VII, theories can contain rules annotated with additional numerical values, such as frequency and time of last usage. These additional values are experimental and not all are used in the system.

Definition 5 (Agent)**.** An agent is a tuple (P,T,C), where

- P is a collection of parameters,
- T is a theory, and
- C is a set of concepts.

Definition 6 (Computation)**.** Given a theory T, a computation is a sequence of closed terms $(t_1, t_2, \ldots, t_n)$, such that every t_{k+1} is obtained from t_k by applying one of the rules of T.

Computations are a general term-rewriting system, where each term is rewritten using a rule. Computation length is the number of terms in a computation.

Definition 7 (Bounded computation)**.** A bounded computation is a computation bounded by a maximum *computation length*, with each term bounded by *term size* $\leq$ working memory capacity. If theory T is bounded in size, then a computation in T is also bounded by rules.

Bounded computations are a simple cognitive model of working memory, inspired by Baddeley's model of human memory system.

Example 3. Following is an abstract example of a bounded computation. A, B and C are terms with a maximum term size ts, r_1 and r_2 are rewrite rules, and computation length is 3.

$$\begin{array}{l} A \\ \hline B \\ \hline C \end{array} \begin{array}{l} \\ r_1 \\ r_2 \end{array}$$

Example 4. Here is a bounded computation in basic arithmetic (without the rules).

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{19 * 2}{(20-1)*2}}{(20*2)-(1*2)}}{(20*2)-2}}{40-2}}{(30+10)-2}}{30+(10-2)}}{30+8}}{38}$$

Bounded computations in arithmetic

In the start of my PhD studies, my research interest was to construct a model of bounded computations for arithmetic that can be used to produce intuitive and easy-to-follow computations for integer arithmetic. This was intended for education and would help children in learning arithmetic. A model of bounded computations similar to the one mentioned earlier and a web-based user interface was developed, that could produce arithmetic computations based on individualized bounded resources. By setting the working memory capacity, maximum length of computations and long term memory (factual knowledge assumed to be possessed by a child of school grade n), the program produced computations bounded by these limitations.

This model of individualized education in arithmetic was to be used by an automatic tutoring agent that would act as a virtual teacher for children. However, this part (developing a virtual humanoid emotional tutor) failed to realize and the system was never tested and used in real environment. Later, it was realized that this model can be used for other purposes as well, such as solving number sequence problems of standard IQ tests. The next section summarizes the paper about inductive pattern discovery in number series problems.

III Bounded Kolmogorov Complexity Based on Cognitive Models

Paper-III employs bounded computations for realizing inductive reasoning and tests its model with intelligence tests that are designed for humans. It is the first among the five papers that aim to produce a unified model of reasoning. The model is built iteratively in this series of papers, so I will discuss each paper and describe the improvements and extensions in relation to the previous paper.

Pattern discovery is a major component of modern IQ tests, and is the ability tested in number series prediction problems and geometric matrix completion, among others. Some IQ tests consist solely of such problems, for example, Raven's progressive matrices [82] and PA [83]. Number sequence problems are often considered stereotypical examples of inductive reasoning.

Number series extrapolation is a typical example of inductive reasoning. This is a common task in standard IQ tests, appearing in various forms such as extrapolation and interpolation in linear number series or matrices of numbers.

A common pattern of such questions is to find the next number from a finite series of integers, illustrated below.

1, 1, 2, 3, 5, 8, ?

The model of bounded computations used in this paper is similar to the one described in the beginning of this chapter. Important differences are listed below.

- Natural numbers are used as basic values in terms. In addition, a special variable n is a term, representing the index of a number.
- $f(n-c)$ is a term representing the number at index $n-c$ for any positive integer c. The term size of $f(n-c)$ is 1.
- For any terms t_1 and t_2, $t_1 + t_2$ is also a term (other binary operators include: $-$, $\times$, $\div$).

Using the model of bounded computations, we developed a model of inductive reasoning to solve number sequence problems of standard IQ tests. The model was implemented as a computer program called `SeqSolver` in Haskell, and this program was used to solve number series problems of some of the standard IQ tests, including PJP (11 items) [84], PA (29 items) [83] and IST (38 items) [85]. The problems of the IST test were new and unseen during the development of the program. Yet, it scored above average human performance. For benchmarking purposes, we used a number of other mathematical software that could extrapolate number sequences, including Maple [86], Mathematica [87] and Wolfram Alpha [88], and an online database of interesting integer sequences called OEIS [89]. These software performed poorly on the problems. The results are summarized in table 4.1.

Program	**PJP**	**PA**	**IST**
SeqSolver	11	29	28
Mathematica	6	9	9
Maple	6	6	9
OEIS	4	11	11
Wolfram Alpha	6	9	12

Table 4.1: Score of different software on the number sequence problems of IQ tests.

Comments

There are a number of explanations why the standard mathematical software packages failed so poorly and scored very low. They are intended for mathematical sequences and are devoid of any cognitive component. For instance, they utterly fail on modulo m sequences (*e.g.,* alternate terms of a sequence forming a subsequence), while many of the items in the IQ tests were of this type. In addition, they consider index number as more basic than previous values, while many items in the tests used more than one previous values in their pattern.

Kolmogorov complexity is a mathematical formalization of Occam's razor, which does not consider psychological difficulty of the problems. IQ tests are designed for humans and by humans, and have a strong psychological basis. Pattern discovery in number sequence problems or any other items in the IQ tests is dependent on how humans perceive patterns, rather than strict mathematical definitions of such patterns.

This paper proposes a computable version of Kolmogorov complexity, bounded by psychological and cognitive limitations. The method is not limited to number series only, and can be applied analogously to build models of pattern discovery in other term-rewriting systems and production systems. On these grounds, a more general method of pattern discovery was developed for arbitrary symbolic domains, which is discussed in the next section. A formal version of the bounded Kolmogorov complexity can be stated as:

$$K_T(x) := \min_p \{\ell(p) : p \ \vdash_T^* \ x\}$$

where x is a sequence, T is a theory, and $*$ indicates bounds (*e.g.,* working memory capacity). Intuitively, the Kolmogorov complexity of a sequence x with respect to a theory T is the shortest program p that will produce x with a computation that uses T and is bounded by $*$. This is a simpler version of the model used in the program, which uses additional constraints on p.

IV Symbolic Reasoning with Bounded Cognitive Resources

In Paper-IV, the model for pattern discovery in number series problems was extended and generalized from numerical values to arbitrary symbolic input.

This new model was designed to meet human performance in more than one domains, while keeping a single set of principles and methods. Together with the deduction method of bounded cognitive resources, the present model used a formulation of the Occam's razor to choose the simplest axiom from a randomly generated list of axioms that could account for the observations (examples). This model was implemented as a computer program in Haskell and was named OCCAM, in memory of William of Ockham who first formulated the principle of parsimony, what today is known as Occam's razor.

To transcend the limitation of a single domain and learn multiple domains, OCCAM needs to know the grammars of the domains it is going to learn. In the previous paper, the grammar of patterns for decoding number sequences was built into the program. In the current model, the grammar is represented as languages: grammar rules that define the correct form of expressions for a particular domain. For any language L, its elements are called $L\text{-}terms$. Some examples of $L\text{-}terms$ are listed below:

English "Alice plays", "Bob walks", "OK".
Arithmetic 2, $3 + 5$, $0 * 9$, $\mathrm{even}(2)$, $\mathrm{f}(\mathrm{x})$.
Logic $\top$, P, $\mathrm{P} \vee \mathrm{Q}$, $\mathrm{x} \wedge \bot \to \bot$.
Haskell `reverse([2,3])`, `[] ++ [1,2]`, `f(x:xs)`.

A rule in this model is called $L\text{-}axiom$ and has the form $t \twoheadrightarrow t'$, where t and t' are L-terms. A finite set of L-axioms is a $L\text{-}theory$.

OCCAM contains a single rewrite rule. Given an axiom $t_1 \twoheadrightarrow t_2$ and an expression t that contains t_1 as a subterm, the rule will rewrite t to t' by replacing (or substituting, see [4] for details) exactly one occurrence of t_1 in t to t_2. This is a deep rule.

$$\frac{t(t_1)}{t'(t_2)}\,(t_1 \twoheadrightarrow t_2)$$

Here is an example of a bounded computation in this model:

$$\frac{\dfrac{3 + (0 * 2)}{3 + 0}\,(0 * \mathrm{x} \twoheadrightarrow 0)}{3}\,(\mathrm{x} + 0 \twoheadrightarrow \mathrm{x})$$

Note that the numbers above are mere strings, unlike previous section where numbers were represented as integer values in the program. Thus, the following computation is equivalent to above:

$$\frac{\text{Three} + (\text{Zero} * \text{Two})}{\dfrac{\text{Three} + \text{Zero}}{\text{Three}}\,(\text{x} + \text{Zero} \twoheadrightarrow \text{x})}\,(\text{Zero} * \text{x} \twoheadrightarrow \text{Zero})$$

An *agent* is defined as a tuple (L,T,W,S,D), where

L a language (set of grammar rules)
T L-theory (declarative memory)
W working memory capacity (a natural number)
S assimilation capacity (a natural number)
D accommodation capacity (a natural number)

The *size* of a L-term t is defined as the number of nodes in the (smallest) parse tree of t. The size of t may be considered as a simple measure of the cognitive load of t on working memory. The size of an axiom is the sum of the sizes of its left and right terms. The size of a theory is the sum of the sizes of its axioms.

The paper includes a short description of *Occam function* that is responsible to find the shortest and simplest solution to a new problem. It takes as arguments an agent A and an induction problem I, and outputs a small theory Δ such that adding Δ to the A's existing theory will produce an optimal solution to I. To do this, OCCAM generates a small set of possible Δ's and tries them over I. If more than one Δ solve I optimally, following selectors are applied in the given order to choose one from the set of optimal Δ:

1. Δ has the minimum size.
2. Δ contains the maximum number of variable occurrences.
3. Δ contains the minimum number of variables.
4. Δ is lexicographically minimal.

A more precise and algorithmic description of the Occam function is stated in Paper-V.

OCCAM was implemented in Haskell with approximately 1,500 lines of code. It was tested in a number of domains, including basic English grammar, integer arithmetic, propositional logic and Haskell-like syntax.

The learning process in OCCAM begins with an agent containing empty theory of a given domain. This theory is gradually extended with new rules learned from positive and negative examples. The main mechanism is the simple cognitive model with bounded cognitive resources, which is responsible for

deductive functions and bounds the combinatorial explosion in the inductive tasks.

One limitation of the current version of Occam is that it is unable to learn the grammars, and the grammar of a new domain is manually provided. Once Occam knows the grammar, it is able to learn the axioms for rewriting expressions from the examples provided to it. Additionally, the examples provided to Occam must be designed properly in order for Occam to make sense of them. Each example is a small set of positive and negative facts tailored for a particular axiom. These deficiencies are addressed in the later papers.

Example 5. Suppose that Occam is given the following example:

$5 + 0 = 5$
$6 + 0 = 6$
$5 + 1 \neq 5$

Then, Occam learns the following axiom:

$x + 0 \twoheadrightarrow x$

Example 6. Following computation shows the reversing of a short list by a function inductively learned by Occam (in Haskell-like syntax).

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\texttt{rev([1,2])}}{\texttt{rev([2]) ++ [1]}}\quad \texttt{rev(x:xs) ↠ rev(xs) ++ [x]}}{\texttt{(rev([]) ++ [2]) ++ [1]}}\ \texttt{rev(x:xs) ↠ rev(xs) ++ [x]}}{\texttt{([] ++ [2]) ++ [1]}}\ \texttt{rev([]) ↠ []}}{\texttt{[2] ++ [1]}}\ \texttt{[] ++ xs ↠ xs}}{\texttt{[2,1]}}\quad \texttt{[x] ++ xs ↠ x:xs}$$

V A General System for Learning and Reasoning in Symbolic Domains

Paper-V presents an extended version of the formerly described Occam system, now called Occam*, and illustrates its usage by following a line of examples in the domain of elementary arithmetic.

The major improvement in the new system is the ability to learn grammar rules of a new domain, which were previously provided manually. No longer does the system need an inner (pre-programmed or pre-written) representation of the grammar of terms for any domain. Given suitable examples of terms, it can discover the general patterns for forming correct terms. The paper also provides a more precise definition of the main function responsible for learning,

i.e., the Occam function. Many small improvements were made in the computer program for efficiency and better results.

OCCAM* uses type labels to annotate its memory contents with types. Variables, which were previously just letters (*i.e.,* x, y, z), are now labeled with their type. For example, $(\mathrm{x} : \mathrm{Number})$ and $(\mathrm{y} : \mathrm{Boolean})$ are variables. Note that the basic values are still symbolic (strings), not the programming types such as integer and boolean.

Similarly, axioms contain a type that shows what domain they are useful for. Axioms are written (τ, t, t'), indicating that t is equivalent to t' in the domain τ.

OCCAM* was implemented in Haskell with around 2,500 lines of code, starting from the code of OCCAM. The program takes as input an agent and a small set of facts, and outputs a new agent with updated memory (or replaces the old agent). In the process, it also produces bounded computations of the facts.

VI Learning Propositional Logic from Scratch

OCCAM* was designed to reason in arbitrary symbolic domains, with no domain-specific knowledge built-in in the system. In the last paper, this was applied to an example domain, *i.e.,* elementary arithmetic. To further test and improve the system, it was applied to propositional logic and illustrated with examples in Paper-VI with how it could learn logical axioms from concrete examples. This resulted in further extensions and improvements in the system, especially, simultaneous learning in multiple domains.

In the previous two papers, there was a single rewrite rule that used axioms of the form $t \rightarrow t'$ to rewrite a term t to t'. This was a deep rule, such that it could rewrite exactly one occurrence of a subterm in a larger term. Semantically, this rewrite rule preserved equivalence of the two sides of an axiom (though it was never applied in reverse, to rewrite t' to t given $t \rightarrow t'$.)

In logic, however, the deep rewrite rule is insufficient. Therefore, this paper introduced a shallow rewrite rule that rewrites a term t to t', but does not rewrite any subterm.

The paper includes a set of examples to illustrate the learning process of an agent in OCCAM*. The program was evaluated by comparing its performance with average human performance on a set of problems.

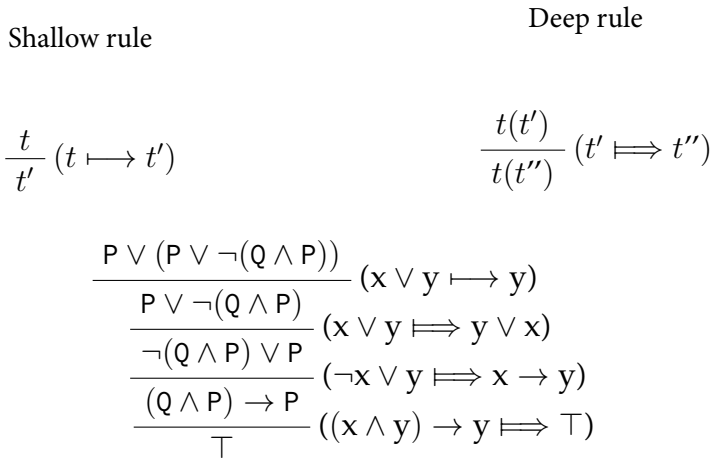

Figure 4.3: Computation of an item from the set of problems used for evaluating Occam*. This is a proof that the item is a tautology. It was automatically generated by Occam* using axioms learned from examples.

Eighty problems were used to evaluate human performance in [31]. Each problem was a question of the form "Is A a tautology?", where A was a propositional statement. Half of the problems were true and half were false, randomly presented to the participants who were undergraduate students in computer science with some tuition in logic. We took these problems and used an agent trained in Occam* to solve them. This agent was trained automatically, starting from a blank theory, and no manual changes were made to its learned theory. This agent solved 34 of the 40 tautologies (only tautologies were used for evaluation), compared to the average participant accuracy of 32.1. It would not be right to claim that this agent outperformed humans (maximum score was 35), nonetheless, the scores indicate that Occam* can compete with humans in more than one domains (including the arithmetic domain used in the previous paper).

Comments

This paper did not delve into the problem of learning a complete theory of logic. The reported theory in the appendix is probably incomplete, however no formal analysis of completeness was performed. It still remains to be checked whether an agent constructed and trained in Occam* will eventually learn a complete set of logical axioms in the domain of propositional logic. Though interesting in

itself, this question is more relevant to traditional AI where solutions to problems are often required to be globally optimal.

Tautology 70 in the appendix of the paper is incorrectly reproduced and would evaluate to a non-tautology. This however does not change the results as the correct form of the formula was used for computing the results, which is the following:

$$\neg(\neg((\mathtt{P} \rightarrow \mathtt{Q}) \wedge \mathtt{Q}) \wedge \mathtt{Q})$$

The shallow rewrite rule was introduced not for completeness but for boosting performance and solving more problems within the bounded resources. It may not be necessary for a complete theory of logic, and therefore may be redundant. In Occam*, redundancy is not addressed and redundant axioms and rules are allowed. The next paper addresses the issue of redundancy.

Other kinds of rewrite rules can also be constructed, as evident in Paper-II where a number of different rules were introduced. However, automatic learning of different types of rules is considerably more difficult. Other techniques for increasing the performance may also be possible, such as abstraction boxes, which were used in Paper-I and II. It should be noted that every method has its pros and cons and a only thorough testing can reveal these. Additionally, these techniques should be as general as possible to avoid adhoc solutions.

VII Bounded Cognitive Resources and Arbitrary Domains

One of the major problems with Occam and Occam* was that these models required a fine-tuned set of examples from which they could deduce a rule. This necessitated the use of an intelligent tutor, without which the models were unable to learn properly. Paper-VII describes a new system called Alice In Wonderland (AIW) that can learn rules from an arbitrary sequence of examples. Another important difference is that AIW is able to update the theory by not only adding new knowledge but also removing existing knowledge. Axioms are removed either due to inconsistency (wrong axioms are removed immediately) or due to overload (inefficient axioms are removed if the theory exceeds long term memory capacity).

AIW is an extension of the Occam model with major changes and improvements, and contains more sophisticated perception, knowledge representation and reasoning methods. The main learning algorithm narrows down

from an exhaustive search of random patterns to an analytic approach of constructing axioms from examples. A more fine-grained mathematical model of inductive reasoning is constructed, with the aim of learning complete theories of arbitrary domains from a random stream of examples. Furthermore, AIW is re-written in Haskell with a graphical user interface for better accessibility (see Appendix A for further details). Paper-VII includes a short description of the model, which is currently being extended and improved with precise definitions of concepts and structures involved in induction and learning, and a larger paper is under way for submission to a scientific journal.

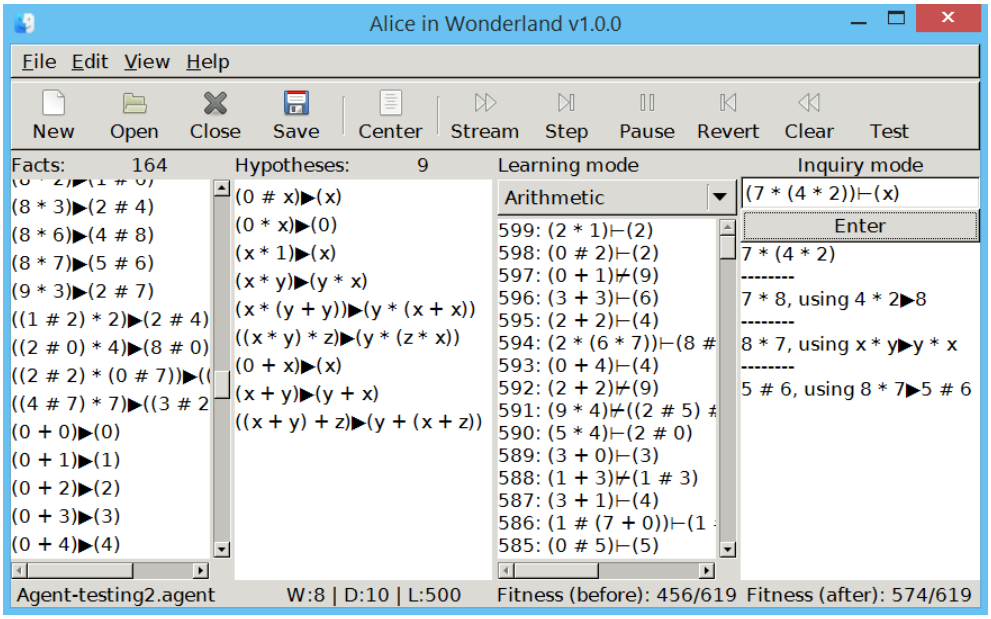

Figure 4.4: A screenshot of AIW. The theory is visible in the left most two columns: closed and open axioms are listed separately. The third column (from left) shows the random sequence of facts presented to the program (generated automatically). The right most column shows a question entered by the user and its solution generated by the program.

Comments

In AIW, redundancy of axioms in a thoery is generally disabled for improving the learning rate. With redundant axioms in the theory, deductive computations become more computationally complex due to a larger branching factor in the search algorithm. Since bounded computations are used for evaluating the simplicity and effectiveness of proposed new axioms, it is important to keep

the search complexity low for faster learning. However, it is not strictly the best option. For instance, performance of a learned theory on a set of problems will be higher if redundant axioms are allowed (leading to shorter computations). Therefore, the redundancy is introduced as a parameter in the agent that can be changed by the user.

The paper lacks a proper evaluation of the system due to insufficient space. The system was tested in a variety of different domains, and was able to learn integer arithmetic in decimal and binary systems, even and odd functions and other modulo n functions, propositional logic with and without propositional variables, quantifier-free first-order logic, quantifier-free second-order logic, basic grammar, and analogical reasoning. By learning addition first, it was even able to correctly learn modulo 3 function by acquiring the following axiom:

$$\text{mod3}(x \,\#\, y) \vdash \text{mod3}(x + y)$$

The learning rate is fast in the beginning and slows down as the theory increases in size. This is one of the major problems with AIW and program optimization is needed for higher performance. However, even with current speed, AIW learned a complete theory of arithmetic on a standard laptop computer. With suitable streams of facts, it also learned a good theory of propositional logic enough to solve many of the tautologies reported in Paper-VI. A more thorough evaluation will be reported in a future paper.

5

Discussion

This chapter discusses some of the issues related to the published papers summarized in previous chapters, mostly issues that relate to more than one papers. Important issues include the problem of vapidity in inductive reasoning, aesthetics and elegance, and design choices in programming.

5.1 The problem of vapidity

Vapidity is one of the major problems of using classical logic to characterize human reasoning [90]. Logical conclusions, when drawn mathematically and without psychological factors, are often vapid and non-interesting even if valid and correct. An example is to rewrite a proposition as a conjunction of itself some arbitrary number of times. That is, if A is true, then so is $A \wedge A \wedge A$.

This problem is faced by all the inductive models described in chapter 4. These models try to solve this problem by using a range of selection criteria to choose the best rule/axiom for the current situation from a set of numerous valid rules. These selection criteria are inspired by the philosophy of science and cognitive theories. The foremost of them is Occam's razor, which is formalized in a number of ways including the minimum length of a rule and the shortest computation a rule generates. These ways are then put in an order to form the criteria for selecting the best rule at any instance.

When confronted with a new item, AIW uses any of the two update types to keep its theory updated. If the item is incomputable with the current theory, it becomes necessary to add an axiom that can make it computable. Even if the

item is computable, AIW updates the theory with an axiom that may enhance its performance in future. This, however, can lead to vapid and non-sensible axioms to be added to the theory. To illustrate this, the axioms in Table 5.1 are all valid, yet only one of these is necessary for the commutative law.

AIW tackles this problem using a variety of approaches. One of them is the requirement of irredundance, that ensures that the theory can only be updated with non-redundant axioms, and the redundant ones can later be removed. This leads to a learned theory that contains the minimum set of axioms necessary for a complete theory of that particular domain. It ensures that only non-vapid axioms are acquired. Although many other non-vapid axioms will be ignored (for example different forms of the associativity rule), this minor loss is smaller than the accumulation of a large number of vapid axioms.

In the case when the system is designed for a particular domain, adhoc solutions can be used to minimize vapidity. To exemplify, the model in Paper-III can possibly benefit from a weighted version of arithmetic such that multiplication weighs more than addition. As a result, arithmetic patterns will be preferred over geometric ones. This is however subject to testing and was not used to avoid adhocism.

5.2 Validation

A candidate axiom needs evidence to show that it is a valid and correct axiom, before considering it for theory update. Theoretically, the required evidence for inductive axioms is infinite (if the values are infinite, *e.g.,* in arithmetic). In practice, however, a certain amount of verification is deemed sufficient for the validity of an axiom.

Algorithm 1 summarizes the function of AIW program that validifies a candidate axiom from the set of substitutions that lead to convergence. Many different versions of this function were tried during the development. Some were practically infeasible as they required high amount of evidence, others led to false positives (wrong axioms labeled as correct). This algorithm is not included in the paper and is part of a manuscript that contains extended description of the AIW model. To illustrate this `satisfy` function, consider the following axiom:

$$(x + (y + z)) \blacktriangleright ((x + y) + z)$$

It has three variables and is the associative law in arithmetic. By limiting the possible values of variables to single digits, there are $10^3 = 1000$ substitutions for which it is valid. Finding all 1000 substitutions (for all 3-variable axioms) is hopeless on a personal computer. Instead, the program slowly accumulates the substitutions for which the axiom converges (is true). Then, Algorithm 1 applies a recursive strategy to check if the number of substitutions is sufficient for the correctness of the axiom.

Algorithm 1: Algorithm for measuring evidence of a candidate axiom for satisfiability.

```
Function: satisfy
Input: n // number of variable types in axiom, e.g., 2 in x+y=x
Input: evidence // set of substitutions for which the axiom
    converges
begin
    case n == 1
        let requiredEvidence = if concepts > 2 then 3 else 2
        return ( length(evidence) >= requiredEvidence )
    case n > 1
        let evidences' = for each value of first variable (e.g., x), get values of
        remaining variables
        let result = select ev in evidences' for which (satisfy (n-1) ev) is True
        return (length(result) >= 2)
end
```

5.3 Aesthetics and elegance

Some of the selection preferences are employed basically for aesthetic purpose, though they also help in reducing search complexity. AIW uses three variables for forming axioms, so a single axiom can come in a variety of forms. For example, commutativity of addition can be expressed in any of the forms listed in Table 5.1. All those forms are equivalent, though adding all of them to a theory is unnecessary and against the requirement of non-redundancy in the long-term memory. Furthermore, all these are of equal size. However, the notion of simplicity in the Occam's razor does not have to be numerical (size/length). Elegance is another form of simplicity and comprehensibility, and can be formalized as Occam's razor.

1	$\mathsf{x} + \mathsf{y} \blacktriangleright \mathsf{y} + \mathsf{x}$
2	$\mathsf{x} + \mathsf{z} \blacktriangleright \mathsf{z} + \mathsf{x}$
3	$\mathsf{y} + \mathsf{x} \blacktriangleright \mathsf{x} + \mathsf{y}$
4	$\mathsf{y} + \mathsf{z} \blacktriangleright \mathsf{z} + \mathsf{y}$
5	$\mathsf{z} + \mathsf{x} \blacktriangleright \mathsf{x} + \mathsf{z}$
6	$\mathsf{z} + \mathsf{y} \blacktriangleright \mathsf{y} + \mathsf{z}$

Table 5.1: Commutativity law in arithmetic

Generally, elegance is subjective. Yet there is some form that looks elegant to the most users. Subjective values are not necessarily random and can be agreed upon by a vast majority. We have seen this in the IQ tests, where answers are psychologically subjective, but a correct answer is the one considered correct by a majority of the test designers or test subjects. In mathematics as well as everyday English, variable x is often used to denote the unknown[1], and is arguably more common than letters l or e.

AIW applies the following aesthetic conditions that not only help chose an elegant version of the commutative law, but also help in reducing the complexity of search (by ignoring all other alternatives).

1. The order of variables in the left hand side should be lexicographic. *e.g.,* $\mathsf{x} + \mathsf{y}$, $\mathsf{x} + \mathsf{z}$ and $\mathsf{y} + \mathsf{z}$ are elegant, but not $\mathsf{y} + \mathsf{x}$.
2. Variables should appear consecutively. Thus, $\mathsf{x} + \mathsf{y}$ and $\mathsf{y} + \mathsf{z}$ are elegant, but not $\mathsf{x} + \mathsf{z}$.
3. The first variable must be x. Thus, $\mathsf{x} + \mathsf{y}$ is the only possible left hand side (for commutativity law).

5.4 Scientific contribution

One of the frequently posed questions I have been asked is that what scientific fields have I contributed to, and what exactly are those contributions? To satisfy such queries, I have divided the contributing papers in two parts.

[1]Variable x traces itself historically to the Arabic word "shai'a" which means something. From Arabic, this word was translated into Greek with the letter χ for phonetic reasons. Later, when translating into Latin, letter x was chosen to indicate an unknown value, probably due to its visual resemblance with χ. [91]

The first part (chapter 3) contributes mainly to mathematical logic and secondly to artificial intelligence. In mathematical logic, the papers introduce new proof formalisms that are designed with bounded cognitive resources and aimed at producing comprehensive and understandable proofs which are local and linear. The use of a term-rewriting system and a simple cognitive architecture also acts as a heuristic in automated proof generation, thus contributing to artificial intelligence and its subfield automated theorem proving.

The second part (chapter 4) consists of five papers that mainly contribute to the field of artificial general intelligence. Three of the five papers were presented in annual AGI conferences and all three received awards. AGI is a broad area with many different theories, methods and approaches towards realizing AGI systems. These papers are specific to symbolic approaches in AGI, and particularly relevant to inductive programming. Research in inductive programming focuses on automatic generation of computer programs from incomplete data (*e.g.,* instances of input-output or program traces), while the papers in this thesis focus on automatically learning symbolic domains from random examples.

5.5 Design choices

Almost all the programming work for this research was done in Haskell, which is a declarative, statically typed, purely-functional programming language. It is a popular choice for researchers who work within computational sciences, particularly in Gothenburg, Sweden. It has a number of advantages in comparison to popular programming languages such as Java and C++, and also to other functional languages such as Lisp and ML.

My choice of using Haskell for the computer implementation of our models is based on the following reasons:[2]

- Models of human cognition and problem solving are frequently programmed with functional programming techniques [92, p.172].

- All of the models in this thesis incorporate algebraic types and mathematical structures that are easier to implement using Haskell.

[2]In addition to the listed reasons, I chose Haskell because it receives considerable support at the Chalmers University of Technology and the University of Gothenburg, including a number of elementary and advanced courses at the department of computer science.

- Haskell is reputed for saving programmer time more than computational complexity. As our models were continually evolving and changing, a substantial effort went into rewriting the code of a program to match the ongoing changes, sometimes to the extent that the whole program had to be rewritten. This would have required significantly more programmer time if non-functional languages were used.

It is noteworthy that a number of other cognitive architectures are also written in functional programming languages. ACT-R's primary implementation is in Common Lisp. CHREST allows specific models to be written in Lisp. Epic, CHREST, FORR, GLAIR and REM have a Lisp version besides other implementations.[3]

5.6 Conclusion

Automated theorem provers are invaluable tools for mathematicians and scientists. Inductive reasoning techniques can even automate scientific discovery. Inductive programming will eventually enable computers to write their own programs and to rewrite existing programs into more efficient and verified programs. However, generalized domain-independent symbolic reasoning is still a major problem.

The model of bounded cognitive resources has proved valuable in the construction of automated proof generators. The proofs produced in such systems are linear and local, and are more comprehensible than proofs generated through classical methods. The search complexity is also reduced by the use of such resource bounds.

Bounded computations are also valuable to define the simplicity of axioms, that in turn leads to a better inductive learning of logical axioms. The papers in the second part of this thesis present models of inductive reasoning that use bounded computations and other simplicity measures (the Occam's razor) to find the best solution for any inductive problem.

Inductive reasoning has been a hard problem in computer science and AI. This thesis, as well as many new methods for inductive reasoning in computer science, show that it is not an impossible task. We already have automated space vehicles exploring the solar system. We are now moving towards driver-less vehicles and pilot-less airplanes in near future, with several other applications of

[3] `http://bicasociety.org/cogarch/architectures.htm`, accesed 2015-09-18

AI in real life. Thus, I believe that we will have computers doing programming, formal reasoning and scientific discovery for us in not-so-distant future, and research like the current thesis is very valuable to that goal.

Predicting specific timelines for major technological advances such as self-improving robots, superintelligence and singularity is beyond the scope of this thesis. However, the social and scientific change is not just moving ahead, but accelerating [93]. Thus, creating artificial systems with general intelligence is no more a question of possibility, or a question of how, but a question of when.

Future prospects

Some possible ways to improve the models of this thesis include (i) introducing a more complex cognitive model and (ii) using probabilistic methods and algorithms for reducing complexity with negligible loss of correctness of solutions. The models in this thesis are deterministic, which can be coupled with probabilistic methods to produce hybrid models of reasoning.

Further research in this area can possibly lead to a universal reasoner that is able to learn and reason in more complex symbolic systems such as higher order logics and programming languages. Inductive programming techniques can benefit from the body of knowledge collected in this thesis.

A

Software

This chapter describes the computer programs that are outcomes of my research work in addition to the attached papers. These programs are reported in the research papers, however a full account of these programs was not produced earlier and is now reported here. All these programs were coded by this author independently; design issues were decided collaboratively with Claes Strannegård. Brief instructions for how to compile are provided, together with example usage.

The programs listed below were all coded in Haskell with Haskell Platform, and compiled with the GHC compiler. All the programs use parallel processing and require the `-threaded` compiler option for enabling parallel computations. The `-O` option enables optimization within reasonable time.

```
# ghc programname.hs -threaded -O
```

To run a program with parallel processing, the Haskell runtime system needs to know how many logical processors are available to the program. This is achieved by providing runtime system (RTS) parameters. For example, the following command line can be used to run a program on 4-core processor (or 4 virtual processors).

```
# programname.exe [arguments] +RTS -N4 -RTS
```

For further information on compiling Haskell programs, consult the GHC User's Guide [94].

Some of the programs use extended parsing of algebraic expressions and other structures. The parsing modules for these programs were automatically

generated by BNF Converter, a compiler construction tool that can generate parser code from an abstract syntax [95], in conjunction with `Alex` [96] and `Happy` [97].

Following Haskell libraries have been used in the computer programs mentioned earlier. These libraries may have additional dependencies which must be installed. The `cabal` tool takes care of the dependencies.

1. `ansi-terminal`: Simple ANSI terminal support.
2. `array`: Mutable and immutable arrays.
3. `astar`: General A* search algorithm, used to search for the shortest bounded computations.
4. `base`: Basic libraries that include data types, control structures (*e.g.,* Monads, Arrows [98]) and System (console IO).
5. `containers`: Assorted concrete container types (maps, sets, trees, graph).
6. `deepseq`: For fully evaluating data structures.
7. `directory`: Library for directory handling.
8. `gtk`: Binding to the Gtk+ graphical user interface library.
9. `haskell-src`: Manipulating Haskell source code.
10. `monad-parallel`: Parallel execution of monadic computations.
11. `mtl`: Monad transformer library.
12. `parallel`: Parallel programming library.
13. `PSQueue`: Priority Search Queue.
14. `QuickCheck`: Automatic testing of Haskell programs.
15. `random`: Random number library.
16. `text`: An efficient packed Unicode text type.
17. `time`: A time library.
18. `transformers`: Concrete functor and monad transformers.
19. `utf8-string`: Support for reading and writing UTF8-encoded strings.

A.1 FOLP

FOLP is a computer program that proves truth or falsity of a first-order sentence in a finite directed graph with colored nodes. Parsing functionality was automatically generated with BNF Converter.

FOLP was written in Haskell and compiled on a Windows 64-bit platform with GHC compiler. It comprises approximately 2,000 lines of code including comments and blank lines (excluding the automatically generated code).

A small program of 300 lines was written to generate random problems for the experiment (`Folpgenerator.hs`).

The program does not use many standard libraries from the Haskell repertoire as much of the algorithmic functionality is designed by the author. Apart from the `base` library, it uses `parallel` for parallel processing and `deepseq` for fully evaluating data structures.

FOLP uses a number of arguments, indicated in the following line:

```
# folp.exe "formula" axioms rules models model (T | NT) SM WM
```

where,

formula is a first-order logic formula.
axioms is a file containing the set of axioms to use.
rules is a file containing rules.
models is a file containing description of the models.
model is the model number to be used.
(T | NT) proof system to use: T specifies truth and NT specifies falsity.
SM is the size of the sensory memory, *e.g.,* 5.
WM is the working memory capacity, *e.g.,* 8.

The program also accepts following optional arguments:

--**length** to search for proofs of minimum length (default).
--**size** to search for proofs of minimum size.
--**weight** to search for proofs of minimum weighted length.
--**latex** to produce latex code of the proof.

The program can be downloaded from GitHub from the following URL:

`https://github.com/arnizamani/FOLP`

A.2 DLvalidity

DLvalidity is a program written in Haskell that automatically generates shortest explanations of justifications in description logic. Its formal model is described in Paper-II, where it was used to solve the problems used in the psychological experiment. Parsing functions were generated automatically with BNF Converter using an abstract syntax of description logic and rewrite rules.

The program uses the following standard Haskell libraries: `astar`, `base`, `containers`, `directory`, `monad-parallel`, `parallel`, `time`, `transformers`, `utf8-string`. The source code can be downloaded from GitHub from the following URL: `https://github.com/arnizamani/DLvalidity`

The program accepts the following arguments:

```
# DLvalidity.exe problem output width depth [{start} {goal}]
```

where,

problem a file that contains the logical justification as a small set of axioms.
output filename to store results.
width maximum width of a proof step, *e.g.,* 8.
depth maximum length of a proof, *e.g.,* 10. This can be iteratively increased if proofs are not found at smaller lengths. The program gets slower with increasing depth.
{start} (optional) the set of axioms to prove. Default is $\{C1 -> C2\}$.
{goal} (optional) the set of axioms to look for. Default is {} (to prove that {start} is true).

A.3 SeqSolver

SeqSolver version 5.0 is developed in Haskell and has been compiled and tested on Windows 64-bit platform. Parsing functionality was automatically generated with BNF Converter using an abstract syntax of arithmetic expressions.

Following Haskell libraries were used in the program: `array`, `astar`, `base`, `containers`, `directory`, `mtl`, `QuickCheck`, `PSQueue`.

SeqSolver program contains two modules, SeqSolver5.hs and Sequence5.hs, and an additional set of modules for bounded computations in arithmetic, under the Amath folder (included in the source code). The source can be downloaded from GitHub from the following URL:

`https://github.com/arnizamani/SeqSolver`

The program accepts a number of arguments:

```
# SeqSolver5.exe expression Min-len Max-len dm-file pm-file WM
```

where,

expression is a sequence of at least three integers, *e.g.,* [1,2,3].

Min-len is the minimum length of the pattern, usually 3.

Max-len is the maximum length of the pattern and can be up to 8. More than this will render the program extremely slow and the patterns found might be cognitively implausible.

dm-file contains declarative memory contents.

pm-file contains rules and user-defined functions.

WM is the working memory capacity, *e.g.,* 8.

The `dm-file` contains arithmetic tabular facts in the format `a * b = c`. Facts must be separated by a semicolon. `C`-style single line comments can be used to hide some lines from the program.

The `pm-file` contains a list of rules the program is allowed to use. Rules are defined in the program code, however this file can enable or disable certain rules. Additionally, it contains simple user-defined functions that can be used in the patterns to reduce pattern size (*e.g.,* `sqr(x)` instead of `x*x`). These functions can also be used to utilize memorized sequences into the program (such as digits of Pi and prime numbers). The syntax is similar to the `C` language functions. A single parameter is used to indicate to the program whether a rule/function is enabled or not (0 for disabled, 1 for enabled). Below is a glimpse from the rules file.

```
// built-in rules
Expand(1) { }
Recall(1) { }
// user-defined rules
AdditionCommutativity(1)
  { x + y = y + x ; }
Square(1)
  { Sqr(x) = x * x ; }
// memorized sequences
Pi(0) {
   Pi(0) = 3 ;
   Pi(1) = 1 ;
   Pi(2) = 4 ;
   Pi(3) = 1 ;
   Pi(4) = 5 ;
   Pi(5) = 9 ;
   Pi(6) = 2 ;
```

```
    Pi(7) = 6 ;
}
```

A.4 Occam*

Occam* (and its predecessor Occam) is written in Haskell. It reads and writes Haskell source code (for agent memory) using the `haskell-src` library. It has been compiled and tested on Windows 64-bit platform.

Following Haskell libraries are used in Occam*: `astar`, `base`, `containers`, `directory`, `haskell-src`, `parallel`, `time`.

Occam* source code contains the following modules.

- OccamStar.hs the main module
- Interpreter.hs functions for deductive computations
- Haskell.hs Haskell source code interpreting functions
- Parsing.hs parsing functions
- Instances.hs data types and class instances
- Niz.hs general purpose helper functions not available in standard Haskell

To create a new agent, Occam* is run with the following syntax:

```
# Occam.exe -create AgentFile
```

To run the program on an agent (`AgentFile`) with a new problem (`IPFile`), following syntax is used:

```
# Occam.exe -run AgentFile IPFile
```

Occam* can be downloaded from GitHub from the following URL:

`https://github.com/arnizamani/OccamStar`

A.5 Alice In Wonderland

AIW is a major extension to Occam* and, apart from the changes in reasoning model, the user interface is fully changed to GUI. The program is coded entirely in Haskell with approximately 5,000 lines of code. It has been compiled and tested on Windows 8 64-bit with Haskell Platform. Gtk+ was used as the graphics library together with Haskell bindings to it (`gtk` library).

The program AIW was coded by me together with Claes Strannegård. I would also like to credit Ulf Persson for his contributions to its development.

Following Haskell libraries were used in the program AIW: `ansi-terminal`, `array`, `astar`, `base`, `containers`, `directory`, `gtk`, `monad-parallel`, `mtl`, `parallel`, `QuickCheck`, `random`, `text`, `time`. Source code can be downloaded from GitHub from the following URL: `https://github.com/arnizamani/aiw`

The program code contains the following modules.

- Alice.hs the main module containing the graphical user interface
- AliceLib.hs functions implementing the computational model
- Instances.hs data structures and class instances
- Parsing.hs parsing functions
- Interpreter.hs functions for bounded computations
- Domains.hs functions for generating random streams of facts for various domains

The program AIW is graphical and needs the `-optl-mwindows` compiler option to hide the command line window when the program is run. Compilation has been tested using GHC version 7.8.3.

```
# ghc Alice.hs -threaded -O -optl-mwindows
```

AIW uses parallel processing and might not function properly without it. To run it, use the following command syntax:

```
# Alice.exe +RTS -N4 -RTS
```

Bibliography

[1] C. Strannegård, F. Engström, A. R. Nizamani, L. Rips, Reasoning about truth in first-order logic, Journal of Logic, Language and Information 22 (1) (2013) 115–137.

URL http://dx.doi.org/10.1007/s10849-012-9168-y

[2] F. Engström, A. R. Nizamani, C. Strannegård, Generating Comprehensible Explanations in Description Logic, in: M. Bienvenu, M. Ortiz, R. Rosati, M. Simkus (Eds.), Informal Proceedings of the 27th International Workshop on Description Logics, Vol. 1193 of CEUR Workshop Proceedings, Sun SITE Central Europe, 2014, p. 530–542.

URL http://ceur-ws.org/Vol-1193/paper_17.pdf

[3] C. Strannegård, A. R. Nizamani, A. Sjöberg, F. Engström, Bounded Kolmogorov Complexity Based on Cognitive Models, in: K.-U. Kühnberger, S. Rudolph, P. Wang (Eds.), Artificial General Intelligence, Vol. 7999 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2013, p. 130–139.

URL http://dx.doi.org/10.1007/978-3-642-39521-5_14

[4] C. Strannegård, A. R. Nizamani, F. Engström, O. Häggström, Symbolic Reasoning with Bounded Cognitive Resources, in: P. Bello, M. Gaurini, M. McShane, B. Scassellati (Eds.), 36th Annual Conference of the Cognitive Science Society, Cognitive Science Society, Austin, Texas, 2014, p. 1539–1544.

URL https://mindmodeling.org/cogsci2014/papers/269/

[5] C. Strannegård, A. R. Nizamani, U. Persson, A General System for Learning and Reasoning in Symbolic Domains, in: B. Goertzel, L. Orseau, J. Snaider (Eds.), Artificial General Intelligence, Vol. 8598 of Lecture Notes in Computer Science, Springer International Publishing, 2014, p. 174–185.

URL http://dx.doi.org/10.1007/978-3-319-09274-4_17

[6] A. R. Nizamani, C. Strannegård, Learning propositional logic from scratch, The 28th Annual Workshop of the Swedish Artificial Intelligence Society, SAIS (2014).

[7] A. R. Nizamani, J. Juel, U. Persson, C. Strannegård, Bounded Cognitive Resources and Arbitrary Domains, in: J. Bieger, B. Goertzel, A. Potapov (Eds.), Artificial General Intelligence, Vol. 9205 of Lecture Notes in Computer Science, Springer International Publishing, 2015, p. 166–176.

URL http://dx.doi.org/10.1007/978-3-319-21365-1_18

[8] J. Allwood, Meaning potentials and context: some consequences for the analysis of variation in meaning, Cognitive approaches to lexical semantics 23 (2003) 29.

[9] Wikipedia, Intelligence — wikipedia, the free encyclopedia, [Online; accessed 21-August-2015] (2015).

URL https://en.wikipedia.org/wiki/Intelligence

[10] G. Roth, U. Dicke, Evolution of the brain and intelligence, Trends in Cognitive Sciences 9 (5) (2005) 250–257.

URL http://www.sciencedirect.com/science/article/pii/S1364661305000823

[11] B. Anderson, The g factor in non-human animals, The nature of intelligence (2000) 79–95.

[12] A. M. Turing, Computing machinery and intelligence, Mind (1950) 433–460.

[13] S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice-Hall, 1995.

[14] P. Wang, B. Goertzel, Theoretical Foundations of Artificial General Intelligence, Atlantis Press, 2012.

[15] N. L. Cassimatis, Artificial intelligence and cognitive modeling have the same problem, in: Theoretical Foundations of Artificial General Intelligence, Springer, 2012, p. 11–24.

[16] A. Newell, Unified theories of cognition, Harvard University Press, 1994.

[17] J. R. Anderson, C. J. Lebiere, The atomic components of thought, Psychology Press, New York, 2014.

[18] J. E. Laird, A. Newell, P. S. Rosenbloom, Soar: An architecture for general intelligence, Artificial Intelligence 33 (1) (1987) 1–64.

URL http://www.sciencedirect.com/science/article/pii/0004370287900506

[19] W. Pei, From NARS to a thinking machine, in: Advances in Artificial General Intelligence: Concepts, Architectures and Algorithms: Proceedings of the AGI Workshop 2006, Vol. 157, IOS Press, Amsterdam, 2007, p. 75–93.

[20] A. D. De Groot, F. Gobet, R. W. Jongman, Perception and memory in chess: Studies in the heuristics of the professional eye, Van Gorcum & Co, 1996.

[21] J. Bach, The MicroPsi Agent Architecture, in: Proceedings of ICCM-5, international conference on cognitive modeling, Bamberg, Germany, Citeseer, 2003, p. 15–20.

[22] C. S. Peirce, Deduction, induction, and hypothesis, Popular Science Monthly 13 (1878) 470–482.

[23] J. E. Adler, L. J. Rips, Reasoning: Studies of human inference and its foundations, Cambridge University Press, 2008.

[24] A. R. Nizamani, Anthropomorphic Proof System for First-Order Logic, Master's thesis, Department of Applied Information Technology, Chalmers University of Technology, Göteborg, Sweden (2010).

[25] R. C. Atkinson, R. M. Shiffrin, Human memory: A proposed system and its control processes, The psychology of learning and motivation 2 (1968) 89–195.

[26] N. J. Holt, A. Bremner, E. Sutherland, M. Vliek, M. Passer, R. Smith, Psychology: The science of mind and behaviour, McGraw-Hill, 2012.

[27] G. A. Miller, The magical number seven, plus or minus two: some limits on our capacity for processing information., Psychological review 63 (2) (1956) 81.

[28] E. L. Kaufman, M. Lord, T. Reese, J. Volkmann, The discrimination of visual number, The American journal of psychology (1949) 498–525.

[29] C. Strannegård, Anthropomorphic Artificial Intelligence, in: F. Larsson (Ed.), Kapten Mnemos kolumbarium, Vol. 33 of Philosophical Communications, Department of Philosophy, University of Gothenburg, 2005.

[30] C. Strannegård, A proof system for modeling reasoning processes in propositional logic, The Bulletin of Symbolic Logic.

[31] C. Strannegård, S. Ulfsbäcker, D. Hedqvist, T. Gärling, Reasoning Processes in Propositional Logic, Journal of Logic, Language and Information 19 (3) (2010) 283–314.

URL http://dx.doi.org/10.1007/s10849-009-9102-0

[32] D. Hedqvist, Human reasoning in propositional logic, Master's thesis, Chalmers University of Technology, Göteborg, Sweden (2007).

[33] C. Strannegård, Goal-Driven Reasoning in First-Order Logic, Philosophical Communications, Department of Philosophy, University of Gothenburg (2006) 261–274.

[34] C. Strannegård, M. Amirghasemi, S. Ulfsbäcker, An anthropomorphic method for number sequence problems, Cognitive Systems Research 22–23 (0) (2013) 27–34.

URL http://www.sciencedirect.com/science/article/pii/S1389041712000356

[35] M. Amirghasemi, A General Method to Solve Numerical Sequence Puzzles, Master's thesis, Department of Applied Information Technology, Chalmers University, Göteborg, Sweden (2011).

[36] C. Strannegård, S. Cirillo, V. Ström, An anthropomorphic method for progressive matrix problems, Cognitive Systems Research 22–23 (0) (2013) 35–46.

URL http://www.sciencedirect.com/science/article/pii/S1389041712000435

[37] S. Cirillo, V. Ström, An Anthropomorphic Solver for Raven's Progressive Matrices, Master's thesis, Department of Applied Information Technology, Chalmers University, Göteborg, Sweden (2010).

[38] L. J. Rips, Logical Approaches to Human Deductive Reasoning, in: J. E. Adler, L. J. Rips (Eds.), Reasoning, Cambridge University Press, 2008, p. 187–205, cambridge Books Online.

URL http://dx.doi.org/10.1017/CBO9780511814273.011

[39] L. J. Rips, The Psychology of Proof: Deductive Reasoning in Human Thinking, MIT Press, 1994.

[40] P. N. Johnson-Laird, Mental Models and Deductive Reasoning, in: J. E. Adler, L. J. Rips (Eds.), Reasoning, Cambridge University Press, 2008, p. 206–222, cambridge Books Online.

URL http://dx.doi.org/10.1017/CBO9780511814273.012

[41] R. Sun, The CLARION cognitive architecture: Extending cognitive modeling to social simulation, Cognition and multi-agent interaction (2006) 79–99.

[42] N. L. Cassimatis, Polyscheme: A cognitive architecture for intergrating multiple representation and inference schemes, Ph.D. thesis, Massachusetts Institute of Technology (2001).

[43] J. Pearl, Probabilistic reasoning in intelligent systems: networks of plausible inference, Morgan Kaufmann, 2014.

[44] E. Gradel, Guarded Fragments of First-Order Logic: A Perspective for New Description Logics?, in: Proceedings of 1998 International Workshop on Description Logics, Vol. 11 of CEUR Workshop Proceedings, Sun SITE Central Europe, 1998.

[45] D. L. McGuinness, F. Van Harmelen, et al., OWL web ontology language overview, W3C recommendation 10 (10) (2004) 2004.

[46] I. Horrocks, et al., DAML+OIL: A Description Logic for the Semantic Web, IEEE Data Eng. Bull. 25 (1) (2002) 4–9.

[47] D. Tsarkov, I. Horrocks, FaCT++ Description Logic Reasoner: System Description, in: U. Furbach, N. Shankar (Eds.), Automated Reasoning, Vol. 4130 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2006, p. 292–297.

URL http://dx.doi.org/10.1007/11814771_26

[48] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, Y. Katz, Pellet: A practical OWL-DL reasoner, Web Semantics: Science, Services and Agents on the World Wide Web 5 (2) (2007) 51–53, software Engineering and the Semantic Web.

URL http://www.sciencedirect.com/science/article/pii/S1570826807000169

[49] G. Gentzen, Untersuchungen über das logische Schließen. I, Mathematische Zeitschrift 39 (1) (1935) 176–210.

URL http://dx.doi.org/10.1007/BF01201353

[50] G. Gentzen, Untersuchungen über das logische Schließen. II, Mathematische Zeitschrift 39 (1) (1935) 405–431.

URL http://dx.doi.org/10.1007/BF01201363

[51] S. Negri, J. von Plato, Sequent calculus in natural deduction style, The Journal of Symbolic Logic 66 (04) (2001) 1803–1816.

[52] R. M. Smullyan, First–order logic, Dover Publications, New York, 1995.

[53] A. J. Robinson, A. Voronkov, Handbook of automated reasoning, Vol. 2, Elsevier, 2001.

[54] M. Sheeran, G. Stålmarck, A tutorial on Stålmarck's proof procedure for propositional logic, in: Formal Methods in Computer-Aided Design, Springer, 1998, p. 82–99.

[55] Z. G. Szabó, Compositionality, in: E. N. Zalta (Ed.), The Stanford Encyclopedia of Philosophy, fall 2013 Edition, 2013.

URL http://plato.stanford.edu/archives/fall2013/entries/compositionality/

[56] F. Pelletier, The Principle of Semantic Compositionality, Topoi 13 (1) (1994) 11–24.

URL http://dx.doi.org/10.1007/BF00763644

[57] A. Baker, Simplicity, in: E. N. Zalta (Ed.), The Stanford Encyclopedia of Philosophy, fall 2013 Edition, 2013.

URL http://plato.stanford.edu/archives/fall2013/entries/simplicity/

[58] L. K. Nash, The Nature of the Natural Sciences, Little, Brown, and Company, Boston, MA, 1963.

[59] P. Domingos, The role of occam's razor in knowledge discovery, Data mining and knowledge discovery 3 (4) (1999) 409–425.

[60] R. Solomonoff, A formal theory of inductive inference. part i, Information and Control 7 (1) (1964) 1 – 22.

URL http://www.sciencedirect.com/science/article/pii/S0019995864902232

[61] R. Solomonoff, A formal theory of inductive inference. part ii, Information and Control 7 (2) (1964) 224 – 254.

URL http://www.sciencedirect.com/science/article/pii/S0019995864901317

[62] M. Li, P. M. Vitányi, An introduction to Kolmogorov complexity and its applications, Springer Science & Business Media, 2009.

[63] M. Hutter, Algorithmic information theory, Scholarpedia 2 (3) (2007) 2519.

[64] C. Pennachin, B. Goertzel, Contemporary Approaches to Artificial General Intelligence, in: B. Goertzel, C. Pennachin (Eds.), Artificial General Intelligence, Cognitive Technologies, Springer Berlin Heidelberg, 2007, p. 1–30.

URL http://dx.doi.org/10.1007/978-3-540-68677-4_1

[65] B. Goertzel, Artificial General Intelligence: Concept, State of the Art, and Future Prospects, Journal of Artificial General Intelligence 5 (1) (2014) 1–48.

[66] J. Hawkins, D. George, Hierarchical Temporal Memory: Concepts, Theory and Terminology, Tech. rep., Technical report, Numenta (2006).

[67] H. De Garis, J. Y. Tang, Z. Huang, L. Bai, C. Chen, S. Chen, J. Guo, X. Tan, H. Tian, X. Tian, et al., The China-Brain Project: Building China's Artificial Brain Using an Evolved Neural Net Module Approach, Frontiers in artificial intelligence and applications 171 (2008) 107.

[68] P. C. Treleavenand, I. G. Lima, Japan's fifth generation computer systems.

[69] M. Hutter, Universal algorithmic intelligence: A mathematical top→down approach, in: B. Goertzel, C. Pennachin (Eds.), Artificial General Intelligence, Cognitive Technologies, Springer, Berlin, 2007, pp. 227–290.

URL http://www.hutter1.net/ai/aixigentle.htm

[70] J. Schmidhuber, Gödel Machines: Fully Self-referential Optimal Universal Self-improvers, in: B. Goertzel, C. Pennachin (Eds.), Artificial General Intelligence, Cognitive Technologies, Springer Berlin Heidelberg, 2007, pp. 199–226.

URL http://dx.doi.org/10.1007/978-3-540-68677-4_7

[71] M. Genesereth, N. Love, B. Pell, General game playing: Overview of the AAAI competition, AI magazine 26 (2) (2005) 62–73.

[72] L. M. Benitez, General game playing, Sistemas Inteligentes: Reportes Finales Ago-Dic 2013 (2013) 161–167.

[73] E. Kitzelmann, R. Plasmeijer, Approaches and Applications of Inductive Programming, Springer, 2010.

[74] R. Olsson, Inductive functional programming using incremental program transformation, Artificial Intelligence 74 (1) (1995) 55–81.

[75] E. Kitzelmann, Analytical inductive functional programming, in: M. Hanus (Ed.), Logic-Based Program Synthesis and Transformation, Vol. 5438 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2009, p. 87–102.

URL http://dx.doi.org/10.1007/978-3-642-00515-2_7

[76] S. Katayama, Magichaskeller: System demonstration, in: Proceedings of AAIP 2011 4th International Workshop on Approaches and Applications of Inductive Programming, Citeseer, 2011, p. 63.

[77] S. Katayama, Recent Improvements of MagicHaskeller, in: U. Schmid, E. Kitzelmann, R. Plasmeijer (Eds.), Approaches and Applications of Inductive Programming, Vol. 5812 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2010, p. 174–193.

URL http://dx.doi.org/10.1007/978-3-642-11931-6_9

[78] G. J. Hitch, A. D. Baddeley, Verbal reasoning and working memory, Quarterly Journal of Experimental Psychology 28 (4) (1976) 603–621.

URL http://dx.doi.org/10.1080/14640747608400587

[79] S. E. Newstead, P. Bradon, S. J. Handley, I. Dennis, J. S. B. T. Evans, Predicting the difficulty of complex logical reasoning problems, Thinking & Reasoning 12 (1) (2006) 62–90.

URL http://dx.doi.org/10.1080/13546780542000023

[80] J. S. B. T. Evans, S. E. Newstead, R. M. J. Byrne, Human reasoning: The psychology of deduction, Psychology Press, 1993.

[81] M. Horridge, S. Bail, B. Parsia, U. Sattler, Toward cognitive support for OWL justifications, Knowledge-Based Systems 53 (2013) 66–79.

URL http://www.sciencedirect.com/science/article/pii/S0950705113002578

[82] J. C. Raven, Standard progressive matrices sets A, B, C, D and E, Oxford Psychologists Press Ltd., 1990.

[83] P. Rådet, PA Number series, Stockholm, Sweden, 1969.

[84] A. Sjöberg, S. Sjöberg, K. Forssén, Predicting Job Performance, Assessio International, Stockholm, 2006.

[85] R. Amthauer, B. Brocke, D. Liepmann, A. Beauducel, Intelligenz-Struktur-Test 2000: IST 2000, Hogrefe & Huber, 1999.

[86] Maple User Manual, Maplesoft, a division of Waterloo Maple Inc., Toronto, 2005–2015.

[87] Mathematica, Version 8.0, Wolfram Research, Inc., Champaign, Illinois, 2010.

[88] Wolfram Alpha, published electronically at http://wolframalpha.com (2012).

[89] N. J. A. Sloane, Online Encyclopedia of Integer Sequences, published electronically at http://oeis.org (2005).

[90] P. Johnson-Laird, S. S. Khemlani, G. P. Goodwin, Logic, probability, and human reasoning, Trends in cognitive sciences 19 (4) (2015) 201–214.

[91] T. Moore, Why is 'x' the unknown?, presented in the Ted conference TED2012, 2012.

URL https://www.ted.com/talks/terry_moore_why_is_x_the_unknown

[92] B. Anderson, Computational Neuroscience and Cognitive Modelling: a student's introduction to methods and procedures, SAGE Publications, 2014.

[93] R. Kurzweil, The age of spiritual machines: When computers exceed human intelligence, Penguin, 2000.

[94] GHC Team, The Glorious Glasgow Haskell Compilation System User's Guide (2015).

[95] A. Ranta, Implementing Programming Languages: An Introduction to Compilers and Interpreters, College Publications, London, 2012.

[96] C. Dornan, I. Jones, S. Marlow, Alex User Guide, 2003.

[97] S. Marlow, A. Gill, Happy User Guide, 2009.

[98] J. Hughes, Generalising monads to arrows, Science of Computer Programming 37 (1–3) (2000) 67–111.

URL http://www.sciencedirect.com/science/article/pii/S0167642399000234

Index

Today's computers are only as smart as the programmer who wrote their software. But what if computers can write their own programs and robots can learn to do several tasks? The scientific field that explores such systems is called artificial general intelligence, which aims to arm computers with general intelligence so they can create new and better intelligence.

This thesis explores computational models of general intelligence, in particular learning and reasoning, in symbolic areas such as numbers and logic. Some models produce computations or proofs that are easily comprehensible and accessible to humans. Others can learn any kind of symbolic systems without any prior knowledge about it, and even outperform humans in some of them. The models utilize two important scientific discoveries: that the memory systems of humans are severely bounded, and that people prefer simplicity when choosing between patterns. The thesis ends with a model called Alice In Wonderland, which learns about new unseen worlds (symbolic domains) from random encounters.

UNIVERSITY OF
GOTHENBURG

ISBN 978-91-982069-8-2