Ph.D. Thesis

Incremental and developmental perspectives for general-purpose learning systems

Candidate:
Fernando Martínez Plumed

Supervisors:
José Hernández Orallo,
María José Ramírez Quintana,
Cèsar Ferri Ramírez

– April 2016 –

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

This work has been partially supported by the EU (FEDER) and the Spanish MICINN, under grants TIN2010-21062-C02-02, TIN 2013-45732-C4-1-P; the REFRAME project, granted by the European Coordinated Research on Long-term Challenges in Information and Communication Sciences Technologies ERA-Net (CHIST-ERA), and funded by MINECO in Spain (PCIN-2013-037); the Generalitat Valenciana under the project PROMETEO2011/052 and the FPI-ME grant BES-2011-045099.

Author's address:

Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València
Camino de Vera, s/n
46022 Valencia
España

To my parents.

Acknowledgments

It is a pleasure to thank the many people who made this thesis possible.

First and foremost, I am indebted to my supervisors. It is difficult to overstate my gratitude to them and one simply could not wish for better supervisors. Cèsar, one of the most enthusiastic people I have ever met and an endless source of wonderful opportunities, ideas and suggestions of inestimable value. María José, whose encouragement, endless attention, sound advice and vast knowledge steered me in the right direction. Finally, Jose. I cannot stress enough my gratitude to him. Without his tremendous academic support, selfless time, infinite patience and maddening attention to detail, this thesis would not have been completed or written. Words can never express my gratitude to all of them.

I would like to thank the Extensions of Logic Programming (ELP) group, led by María Alpuente, for their support and for providing a friendly and comfortable working atmosphere. I am particularly grateful to Salvador Lucas for the confidence reposed in me. Likewise, I am also thankful to all the *Reframers* for their amiability, professionalism and scientific quality, I have learnt a lot from them. Finally, I must express my gratitude to all my (old and new) lab mates for all the good times we have shared. I had some of the best times of my life here —can't wait to see the rest of it.

Thank you to all the amazing friends I have; thanks for the meetups, parties, trips and (absurd) experiences we have lived together. You guys have made an impact greater than you will ever know!

My deepest gratitude goes to those who suffered me at those difficult times, especially Ana. Thank you for your love and patience on my moody days and for being a constant source of support.

Lastly, and most importantly, I wish to thank my parents and my sister, Ruth. I really appreciate everything you have done for me. I would be nowhere near where I am today without their love and support. They have made every opportunity I have been given in life possible and are always there for me. To them I dedicate this thesis.

Nando.
Valencia, June 2016

Abstract

The stupefying success of Artificial Intelligence (AI) for *specific* problems, from recommender systems to self-driving cars, has not yet been matched with a similar progress in *general* AI systems, coping with a variety of problems. This dissertation deals with the long-standing problem of creating more general AI systems, through the analysis of their development and the evaluation of their cognitive abilities.

Firstly, this thesis contributes with a general-purpose learning system that meets several desirable characteristics in terms of expressiveness, comprehensibility and versatility. The system works with approaches that are inherently general: inductive programming and reinforcement learning. The system does not rely on a fixed library of learning operators, but can be endowed with new ones, so being able to operate in a wide variety of contexts. This flexibility, jointly with its declarative character, makes it possible to use the system as an instrument for better understanding the role (and difficulty) of the constructs that each task requires. The learning process is also overhauled with a new developmental and lifelong approach for knowledge acquisition, consolidation and forgetting, which is necessary when bounded resources (memory and time) are considered.

Secondly, this thesis analyses whether the use of intelligence tests for AI evaluation is a much better alternative to most task-oriented evaluation approaches in AI. Accordingly, we make a review of what has been done when AI systems have been confronted against tasks taken from intelligence tests. In this regard, we scrutinise what intelligence tests measure in machines, whether they are useful to evaluate AI systems, whether they are really challenging problems, and whether they are useful to understand (human) intelligence. Finally, the analysis of the concepts of development and incremental learning in AI systems is done at the conceptual level but also through several of these intelligence tests, providing further insight for the understanding and construction of general-purpose developing AI systems.

Keywords: artificial intelligence, general-purpose learning systems, inductive programming, reinforcement learning, forgetting, task difficulty, cognitive development, evaluation of artificial systems, intelligence tests.

Resumen

El éxito abrumador de la Inteligencia Artificial (IA) en la resolución de tareas específicas (desde sistemas de recomendación hasta vehículos de conducción autónoma) no ha sido aún igualado con un avance similar en sistemas de IA de carácter más general enfocados en la resolución de una mayor variedad de tareas. Esta tesis aborda la creación de sistemas de IA de propósito general así como el análisis y evaluación tanto de su desarrollo como de sus capacidades cognitivas.

En primer lugar, esta tesis contribuye con un sistema de aprendizaje de propósito general que reúne distintas ventajas como expresividad, comprensibilidad y versatilidad. El sistema está basado en aproximaciones de carácter inherentemente general: programación inductiva y aprendizaje por refuerzo. Además, dicho sistema se basa en una biblioteca dinámica de operadores de aprendizaje por lo que es capaz de operar en una amplia variedad de contextos. Esta flexibilidad, junto con su carácter declarativo, hace que sea posible utilizar el sistema de forma instrumental con el objetivo de facilitar la comprensión de las distintas construcciones que cada tarea requiere para ser resuelta. Por último, el proceso de aprendizaje también se revisa por medio de un enfoque evolutivo e incremental de adquisición, consolidación y olvido de conocimiento, necesario cuando se trabaja con recursos limitados (memoria y tiempo).

En segundo lugar, esta tesis analiza el uso de tests de inteligencia humana para la evaluación de sistemas de IA y plantea si su uso puede constituir una alternativa válida a los enfoques actuales de evaluación de IA (más orientados a tareas). Para ello se realiza una exhaustiva revisión bibliográfica de aquellos sistemas de IA que han sido utilizados para la resolución de este tipo de problemas. Esto ha permitido analizar qué miden realmente los tests de inteligencia en los sistemas de IA, si son significativos para su evaluación, si realmente constituyen problemas complejos y, por último, si son útiles para entender la inteligencia (humana). Finalmente se analizan los conceptos de desarrollo cognitivo y aprendizaje incremental en sistemas de IA no solo a nivel conceptual, sino también por medio de estos problemas mejorando por tanto la comprensión y construcción de sistemas de propósito general evolutivos.

Palabras clave: inteligencia artificial, sistemas de aprendizaje, programación inductiva, aprendizaje por refuerzo, olvido, dificultad de las tareas, desarrollo cognitivo, evaluación de sistemas artificiales, tests de inteligencia.

Resum

L'èxit aclaparant de la Intel·ligència Artificial (IA) en la resolució de tasques específiques (des de sistemes de recomanació fins a vehicles de conducció autònoma) no ha sigut encara igualat amb un avanç similar en sistemes de IA de caràcter més general enfocats en la resolució d'una major varietat de tasques. Aquesta tesi aborda la creació de sistemes de IA de propòsit general així com l'anàlisi i avaluació tant del seu desenvolupament com de les seues capacitats cognitives.

En primer lloc, aquesta tesi contribueix amb un sistema d'aprenentatge de propòsit general que reuneix diferents avantatges com ara expressivitat, comprensibilitat i versatilitat. El sistema està basat en aproximacions de caràcter inherentment general: programació inductiva i aprenentatge per reforç. A més, el sistema utilitza una biblioteca dinàmica d'operadors d'aprenentatge pel que és capaç d'operar en una àmplia varietat de contextos. Aquesta flexibilitat, juntament amb el seu caràcter declaratiu, fa que siga possible utilitzar el sistema de forma instrumental amb l'objectiu de facilitar la comprensió de les diferents construccions que cada tasca requereix per a ser resolta. Finalment, el procés d'aprenentatge també és revisat mitjançant un enfocament evolutiu i incremental d'adquisició, consolidació i oblit de coneixement, necessari quan es treballa amb recursos limitats (memòria i temps).

En segon lloc, aquesta tesi analitza l'ús de tests d'intel·ligència humana per a l'avaluació de sistemes de IA i planteja si el seu ús pot constituir una alternativa vàlida als enfocaments actuals d'avaluació de IA (més orientats a tasques). Amb aquesta finalitat, es realitza una exhaustiva revisió bibliogràfica d'aquells sistemes de IA que han sigut utilitzats per a la resolució d'aquest tipus de problemes. Açò ha permès analitzar què mesuren realment els tests d'intel·ligència en els sistemes de IA, si són significatius per a la seua avaluació, si realment constitueixen problemes complexos i, finalment, si són útils per a entendre la intel·ligència (humana). Finalment s'analitzen els conceptes de desenvolupament cognitiu i aprenentatge incremental en sistemes de IA no solament a nivell conceptual, sinó també per mitjà d'aquests problemes millorant per tant la comprensió i construcció de sistemes de propòsit general evolutius.

Paraules clau: intel·ligència artificial, sistemes d'aprenentatge, programació inductiva, aprenentatge per reforç, oblit, dificultat de les tasques, desenvolupament cognitiu, avaluació de sistemes artificials, tests d'intel·ligència.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

There is no doubt that Artificial Intelligence (AI) is a successful discipline. Although we are far away from the intelligent systems that proliferate in science fiction books and films, the current state of research in AI can claim some impressive milestones in such complex tasks as winning against the human chess champion Garry Kasparov [CHH02], winning the *Jeopardy!* TV quiz [FBCC+10, FLB+13], wining at several Atari arcade games [MKS+13], master the ancient game of Go [SHM+16] or building SPAUN, a 2.5-million-neuron artificial model brain which mimics human behaviour [ESC+12]. AI can also solve truly awe-inspiring problems in other knowledge-intensive areas such as developing self-driving cars [Sel], adding intelligent personal assistants in smartphones (such as Apple's Siri [Sir], Google's Google Now [goo], Microsoft's Cortana [Cor] or Amazon's Alexa [Ale]) or even creating robots able to learn from YouTube videos [YLFA15].

In the light of all these astonishing achievements in recently AI research boosted by the growing power of machine learning, it is becoming increasingly clear that creating artificial intelligence is much more than "pattern matching". However, although it would be unfair to deny that some current AI systems exhibit some intelligent behaviour (especially those that incorporate some learning potential), in general terms, most AI research is focused on designing AI systems for a particular functionality or adapted for a specific problem. AI systems are able to solve a great amount of tasks without featuring intelligence and, paradoxically, this is one of the reasons of AI's success. Therefore, and thinking about the purpose of AI, we can say that AI research is better identified with Minsky's AI definition [Min82]: "[AI is] the science of making machines capable of performing tasks that would require intelligence if done by [humans]" rather than the less pragmatic McCarthy's definition [McC07] : "[AI is] the science and engineering of making intelligent machines".

Anyway, it is not the purpose of this thesis to be an in-depth debate between weak AI and strong AI since[1] both are equally important for AI. It is useful to have specialised AI systems that solve specific tasks, as well as systems that have more general abilities so that they can solve new problems they have never faced before (perhaps achieving human level intelligence). This thesis is, instead, more interested in the construction of systems able to learn automatically, not pre-programmed or without fixed handcrafted features. This a challenging issue that should, furthermore, pervade the evaluation procedures in AI where systems are usually evaluated in terms of task performance, not really in terms of intelligence. Hence, AI evaluation must necessarily be linked to the purpose of the discipline: general AI systems should require an ability-oriented evaluation in the same way that specialised AI systems should require a task-oriented evaluation [HO14a].

Therefore, in this thesis we will pay attention to both construction and evaluation aspects of general-purpose learning systems. Given the challenge, firstly, we will investigate some designing and implementation aspects involving the abilities of achieving a variety of goals (in different contexts) as well as generalizing, handling and transferring the knowledge gained. Secondly, we will also analyse and study specific ability-oriented tools to evaluate AI systems as well as their cognitive development to better understand what artificial intelligence is and, hopefully, of human intelligence as well. In the following, we will go deeper into these both issues raising several questions that we will try to answer throughout this thesis.

Regarding the former point about construction, the first question we raise is what we can do in order improve the way AI systems learn. We find the key in those methods that display learning characteristics based on the way the human mind works, i.e., *general* and *incremental* (and thus developmental) learning systems that are able to use previous knowledge and context information to deploy solutions for some unexpected situations.

Starting with *generality*, this can be stated in terms of versatility, namely, whether AI systems are specialised in solving very specific tasks or are general enough to solve a wide variety of problems—not even anticipated during design. Working towards this goal should imply AI systems to have a minimal, although extensive, set of capabilities as well as acquiring new capabilities and improve them through learning [Nil05]. This means that general-purpose

[1]While a weak AI system is focused one one narrow task, a strong AI system should be as versatile as a human at solving problems.

learning systems should not be able to rely on a fixed library of concepts or constructs, but they must be dynamic and open, thus making it possible to adapt to new problems. Furthermore, another fundamental characteristic should be borne in mind: the environment must be complex, with diverse, interacting and richly structured objects. Therefore, this sort of systems should have an appropriate handling of rich data and knowledge thus addressing objects that have more elaborate (complex) and flexible internal structure (perhaps abstracted from prior knowledge).

As we said above, generality should work together with an *incremental* or cumulative nature of the learning system in order to overcome the limits of AI approaches (e.g. upper bound accuracy and robustness using all the training data available). The question then arises as to how an AI system can accumulate knowledge (or expertise) over time thus incrementally improving its own ability to learn and solve problems, e.g., avoiding an AI system to become stagnant after a few concepts learnt because there is a deficient handling of its knowledge base. This is closely related to *The Stability-Plasticity* dilemma [CG88], a constraint for artificial and biological neural systems. The basic idea is that a learning system must be capable of learning new things (plasticity) without losing previously learnt concepts (stability). Again, incremental or cumulative knowledge acquisition has much to learn from the study of human cognition in order to overcome the previous dilemma. If we look at how humans learn, we see some similarities: humans have important working memory limitations, so complex hypotheses can only be constructed (or abstracted) over previously learnt or existing concepts [Mil56]. However, most knowledge bases (and expert systems) are to be improved in these aspects: they are still based on sets of rules over some predefined features and concepts, where an abstract and constructive learning is not fully integrated (apart from some kind of incrementally learning more rules). Basically, the constructs and elements the system deals with after a time are the same it had initially.

Therefore, the learning process cannot longer be a transformation from data to knowledge, but a transformation of knowledge (plus data) into new knowledge (Figure 1.1), i.e., retaining the results learnt in the past, abstracting complex knowledge from them (thus allowing to work in rich scenarios), focusing on what knowledge to discard and what to keep on long-term memory (thus avoiding possible information overflow and redundancy, and preserving important or frequently used knowledge), and using the knowledge to help future learning and problem solving. As a result, properly representing, revising, evaluating, organising and retrieving previous knowledge is crucial in this quest for more complex, insightful, powerful and ultimately cognitive learning

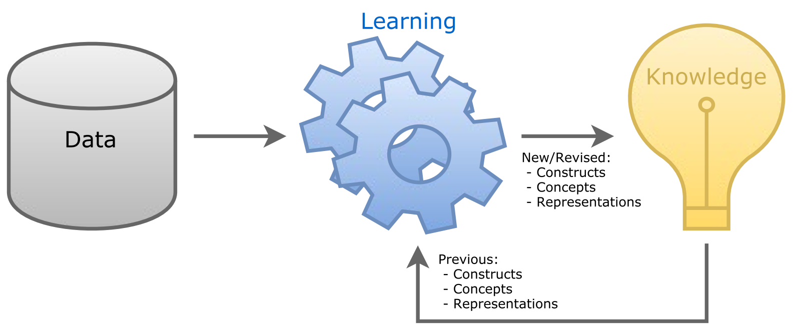


Figure 1.1: The classical learning process is revisited. New knowledge is abstracted from new data as well as from previously learnt knowledge.

approaches to make knowledge acquisition an incremental and developmental process. An important thing here is that general-purpose systems should also be able to learn from few examples, as this is the way humans handle a new task: being able to understand what to do, without further instructions. In fact, much learning in humans and many of the prospective applications that machines cannot solve today work with small data.The creation of systems capable of working in this general way is more related to the area of inductive programming [GHOK+15a], rather than other areas of AI. For that reason, this thesis vindicates more general learning frameworks as well as highlights the relevance of using inductive programming and symbolic representation languages as a learning paradigm.

Regarding the evaluation of general-purpose AI systems, this raises some questions about the appropriateness of using task-oriented evaluation tools. Several of the existing AI systems are not designed to cover one particular application but are expected to perform a variety of tasks (possibly customised by the user). Some examples of such cases are presented in cognitive robots, artificial pets, assistants, avatars, smartbots, smart houses, etc. In order to cover this wide range of (previously unseen) tasks, these systems must have some abilities such as reasoning, learning, planning, verbal skills, etc. Hence, this entails that apart from task-oriented evaluation methods we also need ability-oriented evaluation techniques. The measurement of the above cognitive abilities (and thus intelligence) in natural systems is the subject of several disciplines such as psychometrics or comparative psychology (which are based on the notion of "species") through the use of, among other tools, intelligence tests.

Since this kind of tests work reasonably well for humans, their use for evaluating machines has been suggested many times. In early AI research, the intelligence test (or IQ test) approach was considered as a useful approach for AI systems. This approach moves towards a (more general) ability-oriented categorisation of problems instead of the classical task-oriented categorisation. Although this branch of research sank into oblivion for decades, the past ten (and especially five) years have been blooming with computational models aimed at solving intelligence tests (with different purposes). However, despite this increasing trend, there has been no general account of all these works in terms of how they relate to each other and what their real achievements or purposes are. Closely linked, there is also a poor understanding about what intelligence tests measure in machines, whether they are useful to evaluate AI systems, whether they are really challenging problems, and whether they are useful to understand (human) intelligence. The key issues to understand the suitability (or unsuitability) behind the use of psychometric tests for AI evaluation will be discussed in this thesis.

By the same token, one of the early motivations and applications of intelligence tests (in humans) was the assessment of the so-called *mental age*. The progress achieved in several cognitive tests for the same individuals at different ages would give very valuable information about their cognitive development and, particularly, about the appropriate cognitive constructs needed to solve them (concept dependencies). We clearly see that intelligence tests for small children usually differ in presentation but also in the constructs that are required. One question that has caught recent interest is whether the same approach can be used to assess the cognitive development of artificial systems, namely, the process during which a natural or artificial system develops mental capabilities. In particular, can we assess whether the intelligence of an artificial cognitive system depends on the acquisition, learning or development of different operational constructs? Can we use human intelligence tests for this?

The assessment of these capabilities and concept dependencies is crucial to determine whether and how the system develops as well as it provides useful information about the elements that each problem really requires: more computational power (in terms of working memory and combinatorial search) or some basic operational constructs. In this thesis we are interested in the latter (problem-dependent characteristics). Furthermore, one can wonder whether the mechanisms underlying the behaviour of these programs are the same as or similar to the mechanisms underlying human intelligent behaviour. The question of how these abilities must be measured has been recurrent in the literature of autonomous mental development. While several specific approaches have been attempted for particular tasks, it is still relatively uncommon to use human intelligence tests for this evaluation (with some notable exceptions, e.g., [SS10, Sch13]).

1.2 Research objectives

This thesis aims at contributing how we could implement better general-purpose learning and knowledge handling tools. Furthermore, we would also like to contribute with a better understanding about what intelligence and mental development is (both in humans and AI systems) and how it can be assessed, as well as make a careful understanding of what intelligence tests offer to AI evaluation.

From the previous statement, the main objectives are:

- Propose a general-purpose declarative learning approach following some inductive programming principles (comprehensibility, expressiveness, high-order, . . .) to become a versatile tool for the analysis of learning difficulty in terms of the dependencies on the constructs and operators that have been defined in the system. Study the increase of generality and versatility through a flexible redefinition of inductive operators jointly with a reinforcement learning evaluation paradigm.

- Analyse the use of intelligence tests as an ability-oriented evaluation paradigm for AI and investigate the recent explosion of computer models addressing these kinds of tests. Not only do we aim at analysing the meaning, utility, and impact of these computer models, but also better understand what these tests measure in machines, whether they are useful to evaluate AI systems, whether they are really challenging problems, and whether they are useful to understand (human) intelligence.

- Set a parallelism between the cognitive development in humans and artificial systems through the use of a general-purpose learning approach. Explore intelligence tests as a possible way to examine concept dependencies in cognitive development in artificial systems by means of the acquisition and use of *general* mental operational constructs. Analyse and determine the complexity of this constructs.

- Implement a rule-based approach for managing, structuring, assessing and, finally, revising knowledge in incremental and lifelong learning environments. Study how we can have a precise control of how knowledge develops during learning through information theory-based principles (to characterise knowledge) and hierarchical knowledge assessment structures. Evaluate what happens when the background knowledge grows significantly. Try to overcome the problem of having bounded resources (memory and time) and support knowledge acquisition incrementally neither discarding the prior knowledge nor retraining the induced model repeatedly.

1.3 Structure of this dissertation

In the following, we briefly summarise the main contributions of the different chapters in this thesis. While the first three chapters are devoted to knowledge representation, learning and evaluation approaches in which this work has focused, the next four chapters will outline the main contributions about contruction and evaluation of general-purpose learning systems.

- Chapter 2 summarises different knowledge representation paradigms: propositional, first-order logic and functional. In this chapter we review the basic features, advantages and limitations of these paradigms, as well as the most important learning methods and techniques defined for them. We advocate for the advantages of using functional programming languages featuring powerful construction, abstraction and/or higher-order features as a representation language for general-purpose learning systems. It concludes by giving the standard notions and terminology for representing knowledge in our rule-based approaches although some further notation will be introduced "on the fly" if necessary.

- In Chapter 3 we review some of the AI areas and techniques able to work with complex data environments. From here, we have collected some ideas for developing our declarative and general-purpose learning system (called gErl). Furthermore, this chapter is devoted to those areas of AI that are more related to incremental and cognitive acquisition of knowledge in which our incremental, lifelong view of knowledge acquisition approach (called Coverage Graphs) is based.

- Chapter 4 gives a short overview of some techniques and approaches for assessing the quality of the knowledge learnt both in terms of complexity and compression, and in terms of link-based quantitative assessment measures. Both approaches have inspired the evaluation procedures of the settings developed in this thesis (gErl and Coverage Graphs) with which we address the issue of selecting, arranging and revising knowledge.

- In Chapter 5, we present and outline our general and declarative learning system gErl which can be configured with different (possibly user-defined) learning operators and where the heuristics are also learnt from previous learning processes of (similar or different) problems. Operators are applied to rules for generating new rules, which are then combined with existing or new programs. With appropriate operators, using some optimality criteria (based on coverage and simplicity) and using a reinforcement learning-based heuristic (where the application of an operator over a rule is seen as a decision problem fed by the optimality criteria) many complex problems can be solved. These results have been published in [MFHR13b, MFHR13c, MFHR13d, MFHR13a].

- Chapter 6 introduces some theory and techniques of psychological measurement (most common intelligence tests and cognitive development tests). Furthermore, we make a review of what has been done when intelligence test problems have been analysed through cognitive models or particular systems. We make a general account of all these works in terms of how they relate to each other what their real achievements are. Overall, the ultimate goal of the chapter is to understand the meaning, utility, and impact of these computer models taking intelligence tests, whether these test are useful to evaluate AI systems, and explore the progress and implications of this area of research. These results have been published in [HMS$^+$16].

- In Chapter 7, we try to better understand the role of mental capabilities in cognitive mental development and whether we can evaluate them by setting a parallelism between the concepts of fluid and crystallised intelligence in humans and artificial systems. We explore fluid intelligence test tasks as a possible way to examine the development of general intelligence in artificial systems. To meet this objective we address several common intelligence test tasks with our general learning system gErl to better understand what these IQ test tasks measure, what a system — human or artificial— requires to solve them, and whether an inability can be turned into ability through development. These results have been published in [MFHR16].

- In Chapter 8 we present an incremental, long-life view of knowledge acquisition which tries to improve task after task by determining what to keep, consolidate and forget, overcoming *the stability-plasticity* dilemma [CG88]. It is formalised as a general assessment setting for knowledge acquisition in incremental cognitive systems (called Coverage Graphs): a hierarchical rule-assessment tool driven by coverage relations and metrics based on the Minimum Message Length (MML) principle [WB68a]. The metrics are used to forget some of the worst rules and also to consolidate those selected rules that are promoted to the knowledge base. These results have been published in [MFHR14, MFHR15a, MFHR15b].

- The dissertation concludes with Chapter 9, where we summarise the main contributions of the work and outline several possible directions for further research.

Finally we add some appendices with additional information.

2

Complex data and knowledge representation

In this chapter we give a straightforward introduction to a number of paradigms for *knowledge representation* in artificial intelligence: propositional, first-order logic and functional and discuss about their strengths and limitations. These knowledge representation paradigms are usually associated with techniques and approaches for reasoning and learning, i.e.: propositional learning, Inductive Logic Programming and Inductive Functional Programming. The purpose of this chapter is to reflect the need of using expressive knowledge representation paradigms with highly expressive hypothesis languages when learning from rich data scenarios.

This chapter is organised as follows. Firstly, in Section 2.1 we motivate the problem of choosing knowledge representation paradigm in AI. Next, we briefly introduce both the propositional (Section 2.2) and first-order logic settings (Section 2.3) for knowledge representation in AI and their inherent limitations. To overcome these limitations we advocate for the use of higher-order mechanisms and types presented in the functional paradigm in Section 2.4. Finally, we summarise some notions about the functional programming language used for developing our general rule-based learning approach in Section 2.5 and we close the chapter with a brief summary and conclusions (Section 2.6).

2.1 Introduction

During our life, we are confronted with a vast amount of data that we perceive from our surroundings through, for instance, seeing, hearing, smelling, etc. Therefore, our brain is—in a way that is not very well-known yet—in charge of learning from these complex inputs in order to infer knowledge.

This ability of learning is what makes us intelligent. For this reason, machine learning is a central research topic within the field of artificial intelligence. Its main goal is to explore the construction and study of algorithms that can learn through the acquisition of knowledge in some environments and make predictions from experience in them. This knowledge (or data) required as the input of any learning system consists of descriptions of objects (measurements of some of their properties) from the universe (observations) and, in the case of supervised learning, an output value associated with the example. The encoding of this knowledge must be done in a suitable way such that it can be manipulated or understood by a computer. The learner (or learning system) is required to extract some critical features, or rules, from this data, compressing the information that the data contain into a generalisation, or hypothesis, so that correct inferences can be made on unseen observations. More formally, according with [Mit82], the inductive learning (or generalisation) problem can be defined as follows:

Definition 1 ***Given:***

1. *A language $\mathcal{L}_E$ for describing observations or evidences (*Observation Language*);*
2. *A language $\mathcal{L}_H$ for describing hypothesis imposing a bias on the form of induced rules (*Hypothesis Language*);*
3. *A deductive framework that matches generalisations to instances, i.e., a* coverage *relation for $\mathcal{L}_H$ and $\mathcal{L}_E$;*
4. *A set of positive and negative training instances of a target generalisation to be learnt.*

Determine: *a plausible solution (hypothesis) belonging to $\mathcal{L}_H$ and consistent with the given training instances*

Depending on the task of inductive learning, this definition can be extended to deal with noisy data (relaxing requirements about the quality criterion for evaluating the hypotheses), or adding some prior knowledge K (background knowledge) to bias the learning, or learn with only positive examples.

Focusing on the language in which instances and solutions are described, it should be flexible enough to abstract the essential features of observations for the particular problem and comprehensible enough to the user providing a transparent description of the problem. Unfortunately, representational power comes at a price:

- The knowledge representation language chosen to express a problem dictates what can and cannot be learnt in terms of the language's expressivity (types of problems that it can be tackled).
- The speed of learning (in terms of its efficiency) is related the language's expressiveness, namely, the richer the language is, the larger the search space.
- The transparency or understandability of the knowledge that is learnt (in terms of its explanatory power) establishes the comprehensibility for the user. A representation that is opaque to the user may allow the program to learn, but a representation that is transparent also allows the user to learn.

Therefore, the choice of a knowledge representation language depends on the nature of the problem to be solved. As we have said, learning systems are not only constrained by the available information in their environment (not everything can be explained, modelled or predicted) but also they are biased by the representation framework. For example, in the attribute-values representation language, the common data representation in AI and machine learning, relationships between objects are difficult to represent. Whereas, a more expressive language, such as first order logic, can easily be used to describe relationships. However, complex data is usually also characterised by existing relations between features and with the existence of complex structures such as lists, sets, trees, graphs, etc., so the expressive power of the knowledge representation language is even more important . The importance of the representation language in the learning process has been stated by different works [BGCL00a, Fla00, HO14b]. ,

In what follows, we will complete a rapid tour of a variety of declarative representation paradigms highlighting some of their inherent limitations, and we will advocate for the use of symbolic languages for AI, particularly featuring functional semantics and the use of notions such as higher-order and different kinds of abstractions. From our point of view, functional languages are the most appropriate when learning in complex and rich scenarios due to its highly expressibility and comprehensibility, as well as attractive knowledge manipulation mechanisms.

2.2 Propositional logic

The vast majority of machine learning systems are propositional (decision trees [Mic83], (most of) genetic algorithms [BGH89], artificial neural networks [RMPRG86], Bayesian models [Bun92], support vector machines (SVM) [Gär05], . . .), in the sense that they are only able to deal with flat, well structured data in one table. The use of propositional inputs implies that the observations are represented as attribute value pairs, or feature vectors. An *instance* is described by a fixed collection of *attributes*: $A_i, i \in 1, \ldots, n$. An attribute can either have a finite set of values (discrete) or take real numbers as values (continuous or numerical). The set of all possible unlabelled examples of an specific problem E is composed of all the elements $e = \langle v_1, v_2, ..., v_n \rangle$ with n being the number of attributes and v_i is the value of attribute A_i. A labelled dataset D is a set of examples described as pairs $\langle e, c \rangle$ where $e \in E$, $C = c_1, \ldots, c_k$ is the set of classes and $c \in C$ is one of the k possible values of attribute *Class*. This relative simple representation language has been widely employed because its high efficiency that allows problems with a great number of instances to be handled, and it is perfectly adequate for representing problems (instances) that do not contain any complex relationships.

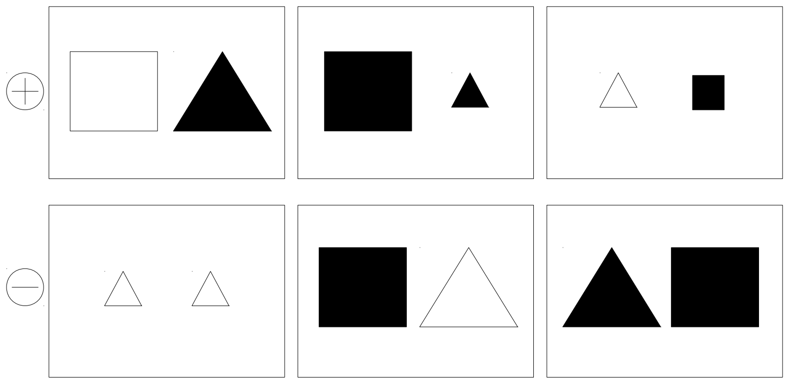


Figure 2.1: A simple binary classification problem (positive ⊕ and negative ⊖ class). Each scene corresponds with one observation or instance and can be decomposed into two objects: left-side and right-side object. Adapted from [VLDR01].

As an example of an attribute-value classification, Figure 2.1 show two sets of observations (instances) belonging to two classes (positive and negative) where the goal is to discriminate between the observations belonging to the *positive* class and those belonging to the *negative* class. Each scene can be described by a fixed number of attributes describing particular characteristics of the objects (left and right side) in each instance ($shape_{left}$, $size_{left}$, $color_{left}$, $shape_{right}$, $size_{right}$, $color_{right}$) and this information can be summarised in one table (see Table 2.1) where each tuple represents one observation and each column corresponds to one attribute. A typical (incomplete) set of classification rules to classify the instances would be:

$$\text{if } color_{right} = black \text{ and } color_{left} = white \text{ then } class = \oplus$$

$$\text{if } color_{right} = white \text{ and } shape_{left} = square \text{ then } class = \ominus$$

id	$shape_{left}$	$size_{left}$	$color_{left}$	$shape_{right}$	$size_{right}$	$color_{right}$	*class*
1	square	large	white	triangle	large	black	$\oplus$
2	square	large	black	triangle	small	black	$\oplus$
3	triangle	small	white	square	small	black	$\oplus$
4	triangle	small	white	triangle	small	black	$\ominus$
5	square	large	black	triangle	large	white	$\ominus$
6	triangle	large	black	square	large	black	$\ominus$

Table 2.1: A simple classification problem. Each scene (instance) consists of two objects (left and side shape). Each object is represented by its shape *(*square *or* triangle*, its* size *(*small *or* big*), and its* color *(*white *or* black*). Each scene is tagged/labelled with a class ($\oplus$ or $\ominus$).*

Now, consider the more complex Bongard's analogy problem in Figure 2.2 (adapted from [Bon70]) where each diagram contains a different number of geometrical objects (such as lines, points, squares, triangles, ...), each having a number of different properties (small, large, white, black, horizontal, ...) and a variable number of relations between them (inside, intersection, cross, ...). Here we can observe that, when objects are structured and consist of several related parts, we need a richer representation formalism and, therefore, the propositional representation paradigm presents several and important drawbacks when rich data comes into the picture [Fla00]:

- Propositional logic requires to select a fixed number of attributes that could appear for every problem thus limiting its expressiveness power. Not all the scenes contain the same number of objects and some of the attributes will have a *null* value. Note that each possible atomic fact requires a separate unique propositional symbol. If there are n people and m locations, representing the fact that some person moved from one location to another requires $n \cdot m^2$ separate symbols (combinatorial problem), therefore being unfeasible to handle each possible combination and, perhaps, not all the attributes are meaningful for each object. Moreover, for every available relation between objects, there should be a propositional attribute for every possible tuple of the relation. Again,

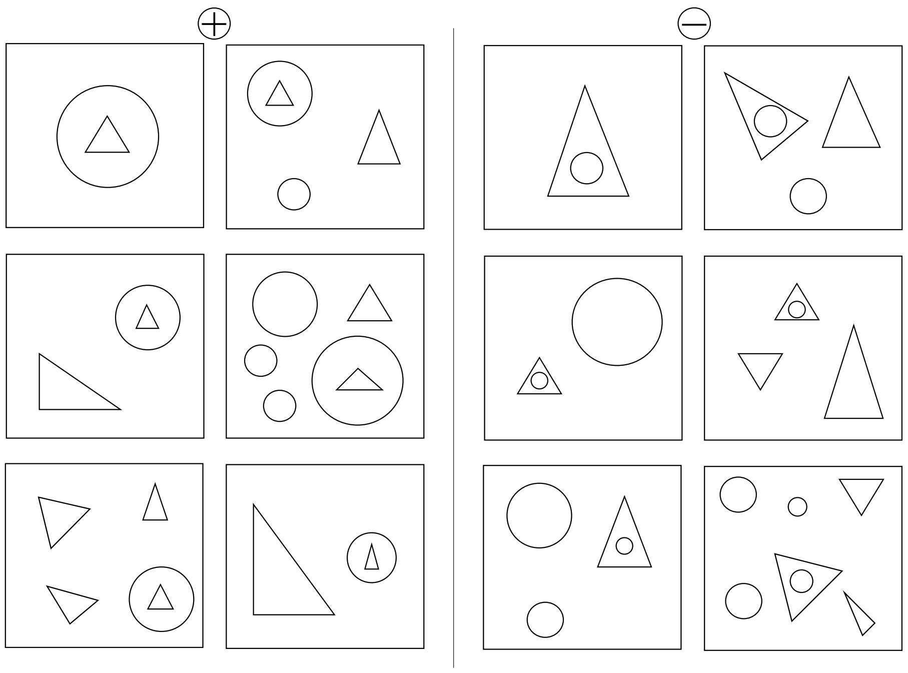


Figure 2.2: Evidence for a Bongard's analogy problem. Adapted from [Bon70]

the number of such attributes is polynomial in the number of available objects.

- One should also order the objects in the examples. Without determining the order of the objects within a scene, there is an exponential number of equivalent representations of a problem (with respect to the number of objects).

- It is not possible to use the previously-known information (background knowledge) expressed with auxiliary functions (including recursion). For instance, consider the problem of learning the product of two natural numbers. This hardly could be solved without the help of a function for the addition of natural numbers.

- Finally, another important problem is that, intrinsically, propositional models are unable to capture relational concepts, i.e., the problem of determining if a number is even or not; or the impossibility of including variables and thus express general relations among the values of the attributes, i.e., the popular problem from the UCI repository [AA07] which defines the function `monks1`, such that `monks1(_,_,_,_,1,_)` and `monks1(X,X,_,_,_,_)`, where `X` is a variable, are true.

The above limitations indicate that a more expressive and flexible representation of data and knowledge should be used.

2.3 First-order logic

First-order logic representation is an extension of the propositional logic form of knowledge representation, which includes a richer ontology of objects (terms), properties (unary predicates on terms), relations (n-ary predicates on terms) and functions (mappings from terms to other terms). By means of a formal language (Horn clauses), first-order logic express the knowledge about a certain phenomena or a certain portion of the world in a more flexible and compact way (compared to propositional logic). Furthermore, it uses quantified variables over (non-logical) objects. First-order logic allows the use of very expressive and comprehensive models for representing complex problems. In particular, the Logic Programming (LP) language Prolog, which uses first-order logic and SLD resolution (which stands for "SL resolution with Definite clauses")[KK83] as the theorem-proving strategy, has and is being one of the most widely used knowledge representation languages for complex problems in databases, medical informatics, bioinformatics research, ... Apart from its expressiveness power due to its mathematical foundation, other reasons why Prolog is suitable for the development of advanced AI systems include:

- The use of *Terms* as the single data structure to implement any other data structure;
- Simple syntax, a Prolog program is actually a collection of atoms;
- Program and data use the same syntax. Thus, we can take data as programs or programs can be data of other programs;
- Weak typing because types of variables in Prolog do not have to be declared explicitly which, although eases the programmer's task (program design decisions can be postponed to the last moment), makes it difficult to find program bugs;
- Incremental program development, where a program can be developed and tested incrementally (other language programs need to be developed almost fully before execution);

- Extensibility, where a Prolog program can be extended and modified adapting both its syntax and semantics to the needs.

Let us refer to the Bongard problem (Figure 2.2) presented in Section 2.2. In that section we illustrated the drawbacks of using propositional languages for its representation. By using Prolog as knowledge representation language, each example can be easily described by a fact, using predicates to represent the different objects, characteristics and relations. In particular, the upper left scene of the Bongard problem (Figure 2.2), which consists of a small triangle which is in a circle, can be specified as follows (the other scenes can be encoded in the same way):

```
positive :- object(o1), object(o2), circle(o1), triangle(
    o2), in(o1,o2), large(o2).
```

This representation allows us to add (if necessary) more complex relations or characteristics to the above definition (such as size, pointing,...). Alternatively, we can represent the above clause by providing identifiers for each example adding the corresponding facts from the condition part of the clause to the background theory (thus avoiding long clauses). For the above example, the following facts

```
object(e1,o1).
object(e1,o2).
circle(e1,o1).
triangle(e1,o2).
in(e1,o1,o2).
```

would be added to the background theory and the above positive instance would then be represented through the fact `positive(e1)`, where `e1` is the identifier. This is a common representation (for examples and hypotheses) employed in the Inductive Logic Programming (ILP) literature.

Continuing with the Bongard problem and the latter representation, since the goal is to discriminate between the positive and the negative scenes (instances), the following first-order hypothesis forms a solution to the learning problem (⊕ class):

```
positive(E) :- object(E,X), object(E,Y), circle(E,X),
    triangle(E,Y), in(E,Y,X).
```

which states that if there exists a circle and a triangle such that the triangle is inside the circle, the examples is of the class ⊕.

This example has shown the power of representation of first-order languages. The rich expressiveness was the motivation for many researchers to apply first-order logic language for machine learning creating a new area named Inductive Logic Programming (ILP).

2.3.1 Inductive Logic Programming

ILP [Mug99] can be seen as the dual paradigm of Logic Programing, while the latter is centred on the deduction of facts from a logic program introduced by the user, ILP describes the process of generating logic programs from facts (and probably other logic programs) introduced by the user. ILP is considered a subfield of Inductive Programming (IP) which allow to address a wider class of problems, not only the synthesis of recursive logic programs [FS08, FY99], but also inductive theory revision [RM91, DR92] and declarative program debugging [Sha83].

The beginning of ILP dates to the 1980s (with the `MIS` system of Shapiro [Sha83]) when machine learning researchers started using Horn clauses logic as their knowledge representation language with the aim of overcoming the limitations of the propositional learners and advocated for more expressive and comprehensive models. In the 1990s decade, some ILP systems were presented as general-purpose approaches to machine learning from relational data such as the systems `PROGOL`[Mug95], `GOLEM` [ALLM94] and `FOIL` [CJQ94].

ILP can be defined formally as the inference of a theory H from an evidence E (which may be only positive E^+ or both positive and negative (E^+, E^-)) possibly using a background knowledge theory K, such that the following condition hold:

1. $K \not\models E^+$ (prior necessity)
2. $\forall e^- \in E^- : K \not\models e^-$ (prior satisfiability)
3. $K \cup H \models E^+$ (posterior sufficiency)
4. $\forall e^- \in E^- : K \cup H \not\models e^-$ (posterior satisfiability)

For exploring the space of possible hypotheses, ILP relies (a) on fixed inductive operators [Mug95, MB92, MF90, ALLM94] ; (b) on traditional concept-learning techniques [CJQ94, QCJ95]; or on (c) applying traditional propositional systems [LDG91, LD93, BDR98] (we will further discuss about it in the following chapter).

Because first order logic is very expressive (turing-complete), ILP can target problems involving structured data and background knowledge. This makes ILP especially appropriate for scientific theory formation tasks where the data are structures, the model may be complex, and the comprehensibility of the generated knowledge is essential. Some of the domains where ILP has been applied include drug design for pharmaceutical purposes [KMLS92], natural language [MC95], learning of medical rules [MODS97], protein primary-secondary shape prediction [MKS92] and mutagenicity prediction [SMKS94a]. Detailed surveys of ILP are provided by [LD93, MD94]

Despite the fact that first-order logic has represented an important step beyond the representational limitations of attribute-value paradigms (it provides understandable and interpretable models and a well-understood theoretical framework for knowledge representation and reasoning), this kind of languages also has some limitations:

- Functions: functions are artificially expressed in first-order languages by using predicates. If a function is categorical and has only two possible output values, it is usually represented by predicates using the examples of one class as positive examples and the examples of the other class as negative ones. When there are more than two classes, there is only positive evidence, and each example is expressed like a fact. In these cases, the attributes and the class are only distinguished through the use of modes to guide the modelling (rule-induction) process[1]. This representation based on modes and facts is not supported by many ILP systems.

- Types: First-order languages have a limited use of types, i.e., they are usually untyped or weak typed languages. Constructor types cannot be defined naturally. The use of complex structures like sets and multisets is not direct.

- High order: Higher order is not supported by the basic first-order logic representation. This is a drawback for the learning of complex problems where higher-order features can be very useful. Instead, some ILP systems are schema-based, where these schemata cannot be learnt.

- Numerical attributes: The use of numerical types in logic languages is not direct. Therefore, the learning of problems that contain numerical attributes is not possible for most of the ILP systems unless they use some kind of constraint logic programming (CLP) scheme [JL87].

[1]Mode declarations state the target predicate to be modelled ("head" mode declarations) as well as other attributes and relationships in the data and the way these will be used in the rules to be constructed ("body" mode declarations)

2.4 Higher-order and Types

In the previous sections we have discussed the limitations of propositional and first-order logic learning approaches for knowledge representation (and learning) in complex scenarios, where not only the structure of each individual but also the treatment of types and functions are important. Although most researchers use an (untyped) first-order logic for knowledge representation (such as Prolog), we advocate for the use of type theories and higher-order mechanisms in Functional Programming (FP) as the knowledge representation formalism for declarative programming languages and learning systems. The main differences between functional and logical paradigms lie in (a) how knowledge is represented: in LP it is represented as a database of clauses (facts or rules) while in FP the individuals are represented as *equations*; and in (b) how the programs are defined: logic programming uses logic expressions (predicates which do not have a return value) and functional programming defines programs through mathematical expressions (functions).

Functional programming has its origins in λ-calculus which was defined in the 1930s by Church [Chu33] as a mathematical notation for functions, its application and recursion. Variable binding and substitution are central parts in λ-calculus: variables are bound when a function is formed by abstraction and, when a function is applied, the formal parameters are replaced by their actual counterparts by replacement (matching). From the very beginning, the functional programming community has used the elegance of higher-order functions jointly with dynamic or static typed systems, depending on the freedom given to the programmer. The functional style of programming has been growing in popularity over the last thirty years, from its beginnings in early dialects of LISP (Common Lisp or Clojure), to the up-to-date functional programming languages such as Haskell, Erlang, Clean, and the ML family of languages. Despite some differences between them, it is clear that all of them provide the following facilities (for a complete introduction to functional programming we refer the reader to [BW88, Rea89, Wik87]):

- The use of algebraic types, both basic (integers, floats, characters, strings, tuples, sets, lists, tree, graphs, . . .) and user defined (recursive) types implies that not only the representation is compact (all information about an individual is contained in one place), but it also provides a straightforward mechanism for manipulating such representations.

- This representation can be used to guide the induction of suitable definitions allowing to apply machine learning directly obtaining comprehensible hypotheses (providing insight into the nature of the data).

- The evaluation mechanism is based on matching and reduction (following ordinary mathematical practice). It can be a *strict evaluation*, where the arguments of the functions are first evaluated (before the (instantiated) function itself) and components of data types are fully evaluated on object formation; or *lazy evaluation*, where function arguments and data type components are only evaluated when their become necessary for function evaluation.

- Higher-order, where functions are first-class objects, i.e., they may be passed as arguments to and returned as results of other functions or they may form components of composite data structures and so on.

- Modularity, where the languages allow for the use of modules (set of functions) of varying degrees of complexity by means of which large systems can be developed more easily.

Therefore, functional knowledge representation languages have several advantages over other approaches. Knowledge is not only supervisable and comprehensible thus allowing for introspection (as with first-order logic), but they also facilitate the handling of complex and deep knowledge structures by using notions such as higher-order and other kinds of abstractions. The field that has worked on these ideas in the past decades is precisely Inductive Functional Programming (IFP), aimed for the synthesis of programs or algorithms having a precise control over the hypothesis spaces generated.

2.4.1 Inductive Functional Programming

Inductive Functional Programming (IFP), as a subfield of IP, is concerned with the synthesis of (recursive) functional programs or algorithms from incomplete specifications (positive and negative examples) by using no side-affects functional languages such as LISP, Miranda, ML or Haskel. The inferred program must be correct w.r.t. the provided evidence generalising it (it should be neither equivalent to it, nor inconsistent).

IFP research started in the 1970s in the seminal `THESIS` system of Summers [Sum77] and the work of Biermann [Bie78] which addressed the synthesis of linear recursive LISP programs by inferring computation traces from examples, and then using a trace-based programming method to fold these traces into a recursive program. There seemed to be some disillusion in the IFP community in the following decades ([Smi84] surveys the main results until the mid 1980s) as the research activities decreased significantly while the advent of logic programming brought a research direction in the early 1980s.

Currently, IFP has revived over the last years with analytical approaches in term-rewriting frameworks such as `IGOR I` and `IGOR II` [KS06, Kit07] based on analysis of the input-output example pairs; and by the search-based approaches `ADATE` [Ols95] and `MAGICHASKELLER` [Kat05], that generate a set of programs and selects those that satisfy the given condition. Depending on the approach followed, the algorithms behind IFP systems rely on different inductive searching methods and operators. For instance, analytical systems such as `THESIS` or `IGOR I`[Sum77, KS06] are schema-guided, while `IGOR II` [Kit07] is based on constructor-term rewriting techniques (including predicate invention). On the other hand, search based approaches such as `ADATE` [Ols95] uses evolution operators (mutation and crossover), whereas `MAGICHASKELLER` [Kat05] relies on higher-order functions (as some kind of program pattern or scheme) jointly with breadth-first search. Currently, many AI and machine learning areas have shown to be successful application niches for IFP, including knowledge acquisition [SHK09], artificial general intelligence [CKHS09], reinforcement learning and theory evaluation [HO00b], cognitive science in general [SK11], intelligent agents, games, robotics, personalisation, ambient intelligence and human interfaces.

As we have seen, there is no single prominent approach to inductive program synthesis. Instead, research is scattered over the different approaches mainly in the areas of ILP and IFP, but also in others such as ILFP (inductive logic functional programming), program synthesis, AI, cognitive science, etc. Nevertheless, IP is a research topic of crucial interest for AI in general because of its operational ability to generalize a program —containing control structures as recursion or loops— from examples. This is a challenging problem which calls for approaches going beyond the requirements of algorithms for concept learning. Pushing research forward in this area can give important insights in the nature and complexity of learning. Furthermore, IP is the right approach for a series of fundamental problems to make machines learn from experience in a more incremental and constructive way. In the following chapter we will put more emphasis on the appropriateness of IP as a learning paradigm that may facilitate the integration, modification, maintainance, application and, finally, the incremental acquisition of complex knowledge.

2.4.2 Alternatives for knowledge representation

As an alternative to the use of logic (LP) or functional (FP) paradigms for knowledge representation, the last two decades, several proposals have been proposed to amalgamate both paradigms. The need for an improved and general logic programming paradigm has led to hybrid approaches such as functional logic programming languages (FLP) [HORQ98, Fla00, FHR01, Llo01] as an extension of logic programming which increases the first order expressiveness with a rich representation (types and higher-order constructs, constraints, probabilities, etc.). On the other hand, we find the *domain-specific languages* (DSL), which are usually better suited for the application at hand. A DSL is a computer language specialised to a particular application domain in contrast to a *general-purpose language* (GPL) (e.g. Prolog, Haskell, C or Java), which is broadly applicable across domains. Therefore, DSLs allow to express complicated data structures and model a particular type of problems or solutions more clearly than with other existing languages [Gul11, LG14, PG15].

2.5 Knowledge representation in Erlang

As we have seen in the previous section, not only do functional programming languages provide a highly expressive and comprehensible knowledge representation framework, but also appropriate knowledge manipulation mechanisms such as higher-order for biasing, explicitly or implicitly, the induction of programs. For these reasons, among some others such as reflection and meta-programming, we have chosen the functional programming language Erlang [VWW96] as unique representation language for all the knowledge and development of our general-purpose learning system gErl (see Appendix A for specific advantages of using Erlang and for further information). Briefly, Erlang is a strict, dynamically typed[2] functional language that comes with built-in support for, as commented, reflection, higher-order and meta-programming, but also for lightweight concurrency, transparent distribution, hot code replacement and effortless scalability. Although Erlang is dynamically typed, it is possible to declare types and annotate functions in order to both document them and help to formalise the implicit expectations about types put in the code.

Let us see how to represent the previous Bongard problem in Erlang. We start with the appropriate type definitions. A Bongard diagram (type *diagram*) consists of a undefined list of tuples ({}) each of which consists of a *shape* (circle, triangle or a square) together with the number of times that shape occurs in the diagram. Furthermore, the *shape* data type has the tuple constructor *inside* for representing this relation between shapes. Therefore, we need to declare the types for the *diagram*, the *shape* and, finally, the *inside* relation as follow:

[2]Type checks are performed mostly at run time. It is opposed to *Static Typing*.

```
-module(Bongard).
% Syntax for type declaration: %
% -type TypeName() :: TypeDefinition. %
-type inside() :: {shape(), shape()}.
-type shape() :: circle | triangle | square | inside().
-type diagram :: list({shape(), Integer}).

% Syntax for type signature: %
% -spec FunctionName(ArgumentTypes) -> ReturnTypes. %
-spec class(diagram()) -> positive | negative
```

Code 2.1: Type definitions in Erlang for the Bongard problem 2.2

Furthermore, we can also specify the type signature of different functions in order to define the input/output arguments accepted. In the previous type specification (Algorithm 2.1) we see the type signature for the function called as `class()` which returns if the given diagram belongs to the positive or the negative class. In the specification above we say that this function accepts *diagrams* as input arguments, according to type definition of `diagram()`, and either returns the atom `positive` or `negative`. Therefore, calling, for instance, `class/1` with a wrong *diagram*, i.e., `class([{cirle},{triangle}])`, is not syntactically valid according to our specifications.

Here are a few syntactically correct examples:

```
class([{{triangle,circle},1}]) -> positive;
class([{circle,1},{triangle,1},{{triangle,circle},1}]) ->
    positive;
class([{{square,triangle},1}]) -> negative;
```

Code 2.2: Examples for the Bongard problem 2.2

Note that the examples are rules (or equations) instead of facts. As it can be seen, functions are a more natural way for representing learning problems (in this case, a classification task). Finally, if we were supposed to induce the definition for the above function, not only could we take advantage of the built-in-functions from the background user-defined functions, but also from the higher-order functions from Erlang. Rather than learning a recursive function, the IP system then only needs to pick the suitable higher-order function and instantiate it appropriately. For instance, the code for checking the class of a diagram could be:

```
class(Diagram) ->
    lists:foldr(fun({Shape,_},Class)->
        if
            Shape == {triangle,circle} -> positive;
            true -> Class
        end
    end, negative, Diagram).
```

Code 2.3: Possible solution for the Bongard problem 2.2

Here `foldr` (also known as reduce) is a specific higher-order tail recursive built-in function which calls `fun(Shape,_,Class)` (a lambda function declaration, also known as `fun` in Erlang terms) on successive elements (from right to left) of the *diagram* (list) and returns the final value of an accumulator (`Class`). If the diagram contains the relation `inside` between a triangle and a circle, the class is positive, otherwise, the class is negative.

We have employed the same example in three different frameworks: propositional, logic programming and functional programming. It has given a perspective of the limitations of each approach. The main advantage of using a representation of examples that reflects directly their real structure (by user-defined type declarations, as in the example, or by built-in language types) is that we can design learning algorithms capable of manipulating such representation directly and, furthermore, guide the hypothesis construction [BGCL00a].

In what follows, we will introduce some basic notions of the functional programming language used in our learning setting.

2.5.1 Basic Notation

In this subsection we briefly summarise some basic notions of the functional programming paradigm. Note that the notation used matches perfectly with the Erlang syntax. For a signature, a set of function symbols (together with their arity) Σ and a countably set of variables $\mathcal{X}$, we denote the set of all terms over Σ and $\mathcal{X}$ by $\mathcal{T}(\Sigma, \mathcal{X})$ and the (sub)set of ground (variable free) terms by $\mathcal{T}(\Sigma)$. We distinguish function symbols that denote datatype constructors ($\mathcal{C}$) from those denoting (user-)defined functions ($\mathcal{F}$). Thus $\Sigma = \mathcal{C} \cup \mathcal{F}, \mathcal{C} \cap \mathcal{F} = \emptyset$. Depending on the arity of symbols in Σ, a function is said to be a *constant* if its arity is equal to 0, otherwise it is said to be a *functor*. The set of variables occurring in a term t is denoted *Var(t)*. Following Erlang syntax, constant terms begin with a lower-case letter and variables begin with an upper-case letter. A term t is a *ground term* if $Var(t) = \varnothing$.

The equations are applied as simplification (or rewrite) rules from left to right, i.e., they form a *term rewriting system*. An equation is an expression of the form $l = r$ where l (the left hand side, *lhs*) and r (the right hand side, *rhs*) are terms. $\mathcal{R}$ denotes the space of all (conditional) functional rules ρ expressed as l [when G] $\to r$, where terms l and r are, respectively, the *lhs* and the *rhs* of ρ, and G is an ordered conjunction (comma-separated) or disjunction (semi-colon separated) of equality constraints (Boolean expressions) called guards $g_i = h_i$ with $g_i, h_i \in \mathcal{T}(\Sigma, \mathcal{X})$[3]. Code 2.4 shows an example of a conditional rule in Erlang. If $G = \varnothing$, then ρ is said to be an unconditional rule. The *lhs* l of a rule ρ has the form $F(a_1, \ldots, a_n)$, called *function head*, where $F \in \mathcal{F}$ is the defined function symbol, and arguments $a_i \in \mathcal{T}(\mathcal{X})$ are built up from constructors and variables only. We call the terms from $\mathcal{T}(\mathcal{C}, \mathcal{X})$ constructor terms. The sequence of the argument(s) a_i is called pattern. This format of rules or equations is known as constructor-based in functional languages. If we apply a defined function to ground constructor terms $F(i_1, \ldots, i_n)$, we call the i_j inputs of F.

```
adult(X) when X >= 18, X =< 288 -> True;
adult(_) -> false.
```

Code 2.4: Conditional rule in Erlang

Let $\mathcal{P} = 2^{\mathcal{R}}$ be the space of all possible functional programs (modules in Erlang) formed by sets of rules $\rho \in \mathcal{R}$. Given a program $\omega \in \mathcal{P}$, we say that term t reduces to term s with respect to ω, $t \to_\omega s$, if there exists a rule l [when G] $\to r \in \omega$ such that a subterm of t at occurrence u matches l with substitution θ, all conditions $(g_i = h_i)\theta$ hold, for each equation $l_i = r_i \in r$, $l_i\theta$ and $r_i\theta$ have the same normal form (that is, $l_i\theta \to_\omega^* \sigma$, and $r_{i_r}\theta \to_\omega^* \sigma$ and σ can not be further reduced) and s is obtained by replacing in t the subterm at occurrence u by $r\theta$.

2.6 Summary

The actual choice of knowledge representation paradigms (or languages) depends on what information is to be processed and on what functions are needed to manipulate, transform and discover knowledge from it. For a machine to perform these tasks, knowledge must be encoded in a suitable way, namely:

- The representation should adequately reflect the types of knowledge needed.
- The representation should allow new knowledge to be added and existing knowledge to be updated easily.

[3]One negative point about guards in Erlang is that they will not accept user-defined functions because of side effects.

- The representation should permit the derivation of new knowledge not explicitly represented in the knowledge base.
- The representation should promote efficient processing of the information.

Therefore, the kind of knowledge representation chosen to express a problem dictates the concepts that an algorithm can and cannot learn, in addition to its expressiveness, comprehensibility, speed of learning and efficiency.

The main motivation of this chapter has been to put emphasis on the idea that in order to learn from rich data scenarios we need to use expressive knowledge representation paradigms (for the background knowledge, evidence and hypotheses). We thus advocate for the use functional programming languages which provide not only highly expressive hypothesis languages (supporting a variety of data types with complex structure), but also higher-order mechanisms for their manipulation thus allowing learning algorithms to have precise control over the hypothesis spaces generated. An attractive feature of this approach is that the induced definitions in such hypothesis languages are comprehensible, thus providing insight into the nature of the application data. In the following chapter we will go deeper into the discussion about the techniques for learning from complex structures as well as the problem of learning from experience in incremental and constructive scenarios which entail an inductive programming approach and those which not.

3 Learning in complex and incremental scenarios

In the previous chapter we introduced some of the most used knowledge representation paradigms in AI. Furthermore, we reflected the advantages of using symbolic approaches (featuring powerful construction, abstraction and/or higher-order features) over non-symbolic approaches, especially when knowledge becomes complex, which is one of the desirable characteristics in general-purpose learning systems. Now it is the turn to revisit the issue of learning in incremental scenarios, adding to it the problem of addressing complex structured data. We suggest that inductive programming can be the right approach for that.

The chapter is organised as follows. Section 3.1 motivates the need of incremental learning approaches able to appropriately deal with complex structured knowledge. Sections 3.2 and 3.3 give a short account of, respectively, the many approaches able to deal with complex structures (input or output), and some of those that exhibit incremental, cumulative and cognitive characteristics. Finally, a summary of the chapter and a more comprehensive analysis is discussed in Section 3.4. .

3.1 Introduction

Some important challenges for AI systems lie in the construction of systems that are able to (incrementally) acquire and use both previous learnt knowledge and context information. Lifelong learning, transfer learning, meta-learning, incremental or cumulative learning are some of the approaches in which research and advances in incremental knowledge acquisition have increased steadily over the years. In fact, these approaches stand apart from classical in that they are able to retain the knowledge gained from past learning experiences and use it to aid future learning and new problem solving, i.e., once a small piece of knowledge is integrated or consolidated into the knowledge base, new inference processes can take place by using this prior knowledge. However, apart from some simple incremental learning of knowledge (e.g.,rules), most of these approaches for knowledge acquisition do not fully integrate an abstract and constructive learning: those constructs and elements the system deals with after a time are the same it had initially.

Additionally, when dealing with complex data scenarios, these approaches are limited by a still inappropriate handling of knowledge: as knowledge becomes more complex and abstract, it is no longer processed in a completely automatic way. In fact, while the capability of automatically storing and handling factual, textual, numerical and statistical information has increased exponentially over the years, this has not been the case for knowledge bases of arbitrary complexity and sophistication [HO14b]. As we saw in he previous chapter, flattened data is the most common data representation used in AI, where data is organised as a table. Each row in the table corresponds to an example (evidence or observed items) and each column represents a scalar (or numerical) attribute of the examples. It is assumed that there are no given relations between attributes (or between examples), neither complex structures (such as lists, sets, trees or graphs) can be directly represented. In the same way, the learnt knowledge usually have a predefined flat structure (even complicated or incomprehensible), thus, being unable to capture the underlying complex patterns behind the data examples. Furthermore, most AI systems and knowledge bases are still based on sets of (propositional) rules over some predefined features and concepts (although the use of ontologies have made impressive increases in this respect), e.g., as those extracted from many association rule algorithms.

More often than not, the above problems have nothing to do with large amounts of data. In fact, much learning in humans and many of the promising applications [GHOK$^+$15a] that machines cannot solve today work with small amounts data where each particular inference is not performed from a large number of examples, but just a few. Once a new example is added to the knowledge base, this may may force an increment or revision of the existing knowledge with generalisation or abstraction mechanisms. It should be noticed that this issue is closely related to the deep learning approach [Ben09] that attempt to model high-level abstractions in data by using multiple non-linear transformations and complex structures, as architectures, concepts and features are said to be hierarchical. However, the disadvantage of most deep learning approaches is that it is not clear how knowledge can be accessed, revised and integrated with other sources of knowledge due to they are based on artificial neural networks and other statistical approaches.

All of these uses (complex data and knowledge, incremental learning and comprehensibility) are basically the knowledge acquisition, handling and application problem [Kid87, ZCC+13] but where depth (and not necessary size or diversity) is the change. Therefore, much more needs to be done on knowledge acquisition and reuse. Not only do some problems require the efficient induction of small but conceptually complex programs (including different data types, structures, variables and recursion), but also crave for a cumulative and developmental nature of learning which facilitate the acquisition, integration, modification, maintainance and application of this complex knowledge, where new constructs and concepts can be developed and, ultimately, examined and evaluated. In this chapter we will give a comprehensive (nor exhaustive) review of some of the areas and techniques in AI able to tackle, firstly, complex structured data and knowledge and, secondly, continuous and incremental learning, discussing on the appropriateness of inductive programming to address the knowledge acquisition problem.

3.2 Learning from complex data

Over the last years, several AI areas have become able to (automatically) deal with rich data and knowledge representations. Rich data scenarios in AI involve predicting sophisticated and possibly complex or recursively structured objects, rather than scalar values. Focusing on the literature of inductive programming, some complex data (toy) examples have been used: the East-West trains dataset [MMPS94], mutagenesis [SMKS94b], block towers [SHK09], the rectangle problems [Ols95], Bongard problems [DR10], to name a few. In particular, there is also a wide variety of application domains containing rich or structured data including bioinformatics, natural language processing, speech recognition, and computer vision.

Regarding to knowledge, the typical example of a complex theory or model is a program or algorithm. Handling knowledge bases that are composed of programs and algorithms is not an easy task, such as software repositories (to be precise, knowledge usually has some truth connotations, while software repositories are operational). We are interested in knowledge that can be learnt semi-automatically. Examples of learning complex knowledge are any model that is able to capture the underlying patterns behind complex data examples (such as those above or others). Inductive programming (IP) [Kit10, GHOK+15a] and some related areas such as relational data mining [DL01] are arguably the oldest attempts to handle this kind of knowledge.

IP systems can be considered *general-purpose* AI systems, because any problem can be represented, preserving its structure, with the use of the Turing-complete languages underneath (as seen in the previous chapter): logic (ILP [Mug99]), functional (IFP [HKS09]) or functional-logic (IFLP [HORQ98]). IP is particularly useful when the number of examples is small but the hypothesis space is large.

The current state of IP research is constituted by two different but complementary kind of approaches (both in functional and logic program induction): *search-based* (or *generate-and-test* based) approaches, where candidate programs are generated independently from the given specification (i.e., evidence) and then tested against it thus selecting the best evaluated candidate(s) to be developed further; and *analytical* approaches, where candidate programs are constructed in an example-driven way by inspecting the examples and detecting recurrent structures in them. Analytical approaches (such as [KS06, Kit07] in IFP or [Mug95, ALLM94] in ILP) are limited to structural problems (such in list reversing), where only the structures of the arguments matters (not the content), and where the induction of problems is efficient due to a bias on the control and data flow of the synthesised programs. On the other hand, search-based approaches (such as [Ols95, Kat05] in IFP or [CJQ94, QCJ95, Qui96] in ILP) can also handle semantic problems where the values of the arguments matter (such in list sorting) due to the more general methods used. Furthermore, these two approaches can also be coupled [Fle97, Kit07].

ILP, for instance, has been found especially appropriate for scientific theory formation tasks where data are structured, models may be complex, and comprehensibility of the generated knowledge is essential. Learning systems using higher-order features (see, e.g., [Llo95, Llo99, Llo01]) were one of the first approaches to deal with complex structures, which were usually flattened in ILP. Since this approach solve defects found by first-order logic in complex structures, it is very suitable for the study of complex structure domain knowledge discovery such as structure data decision tree learning [BGCK$^+$97, Bow98, BGCL00b], nuclear learning [Mon95], genetic programming [Ken98, KGC99] drug design [KMLS92], natural language [MC95], learning of medical rules [MODS97], protein primary-secondary shape prediction [MKS92] and mutagenicity prediction [SMKS94a]. Despite the power of higher-order functions to explore complex structure, this approach has never become mainstream.

Consider that, for searching the space of possible hypotheses, almost all the IP systems and approaches rely on the use of different *fixed* inductive operators and mechanisms for arrange the hypothesis space. These operators are developed, among others, by inverting well formalised deductive rules (unification, resolution, implication) leading to bottom-up and top-down approaches, where generalisation and specialisation operators, respectively, are used [MDRP+12]. Some examples include Plotkin's *lgg* [Plo70, Plo72] operator which works well for a specific-to-general search. The ILP system Progol [Mug95] combines the Inverse Entailment with general-to-specific search through a refinement graph. The inverse operator constitutes also the base of the algorithm employed in the systems `Cigol` [MB92], `Crustacean` [MF90] and `Golem` (jointly with *lgg*) [ALLM94]. The Aleph system [Sri04] is based on Mode Direct Inverse Entailment (MDIE). The Antiunification operator is used to obtain the least general generalization of the set of the I/O examples in `IGOR II` [Kit07]. In inductive functional logic programming, the FLIP system [FHR01] includes two different operators: inverse narrowing and a consistent restricted generalisation (CRG) operator [HORQ99]. Other IP systems are based on traditional concept-learning techniques (FOIL [CJQ94, QCJ95]) or on applying traditional propositional systems like decision tree learners or rule learns after converting a (restricted) set of relational problems onto propositional problems (`Linus` [LDG91, LD93], `TILDE` [BDR98]). In any case, the set of inductive operators configures and delimits the performance of each learning system.

Additionally, we need to consider that complex structured (in the form of structured data types or structured prediction) do not entail an inductive programming approach. Kernels, distances or other notions [Gär03, EFHORQ05] can be used to convert a structured problem into a scalar feature representation (thus leading to incomprehensible patterns/models described in terms of the transformed (hyper-)space). Even the field of Statistical Relational Learning [Get07] goes beyond what inductive (logic) programming has been. Focusing only on structured outputs, the family of generic techniques for this kind of problems are generally known as *Structured Prediction* (SP) [Bak07] algorithms and include problems such as sequence labeling, parsing, collective classification, bipartite matching (word alignment in NLP or protein structure prediction in computational biology), entity detection and tracking, automatic document summarisation, machine translation and question answering. These techniques include, but are not limited to, Conditional Random Fields (CRFs) [LM01], which are an alternative extension of logistic regression (maximum entropy models) to structured outputs by using a log-linear probability function to model the conditional probability of an output y given an input x (Markov assumptions are considered in order to make inference tractable); and kernel-based approaches such as the Maximum Margin Markov Networks (M^3N) or the Support Vector Machines (SVM) for Interdependent and Structured Output spaces (SVM-ISO, also known as SVM^{struct}). The former considers the structured prediction problem as a quadratic programming problem incorporating (a) kernels (which efficiently deal with high-dimensional features) and (b) probabilistic graphical models to capture correlations in structured data. On the other hand, SVM^{struct}, which is strikingly similar to the M^3N formalism, can be seen as a SP evolution of [Gär05] (Kernels) or [EFHR06, THJA04, EFHR12]. On the other hand, *hierarchical classification* can also be viewed as a case of SP where taxonomies and hierarchies are associated with the output [KS97]. Finally, the recurrent neural networks (RNN), a special type of artificial neural networks (ANN) where connections between units form a directed cycle (thus exhibiting a dynamic temporal behavior), have also been used in SP achieving the best known results in handwriting recognition [GLF+09].

What is clear from the those previous approaches is that the number and performance of AI techniques dealing with rich, complex or structured data have considerably increased in the past decades. However, this review have only strengthened our conviction about the appropriateness of the symbolic approaches such as inductive programming to approach complex structured learning.

3.3 Incremental learning

Incremental acquisition or the reuse of previously learnt knowledge is of critical importance in the majority of knowledge-intensive application areas. Although *one-shot* learning approaches can still be improved, there exists an upper limit in terms of performance (e.g. upper bound on full-training accuracy and robustness). The question is, therefore, how we can go beyond classical algorithms to improve learning and knowledge acquisition much further. We find the answer in those methods which display human mind-based learning characteristics as the ability of learning continuously: humans retain the knowledge gained from past learning experiences and use it to aid future learning and problem solving. However, to follow this designing principle implies a well-known constraint for AI systems (mainly artificial neural networks): *The Stability-Plasticity* dilemma [CG88][1]. Although there are several AI areas where this issue of incremental learning is undertaken (as we will briefly review in the following), we will argue again that inductive program-

[1] *The Stability-Plasticity* dilemma is the problem whereby the brain learns new knowledge (plasticity) without forgetting its past knowledge (stability). The complete forgetting of previously learnt information when exposed to new information is known as *catastrophic forgetting* [MC89]

ming is also good way to address this challenge.

Starting with the area of lifelong learning [Thr96b], this is concerned with the persistent and cumulative nature of learning and, thus, it aims to design and develop AI systems that must be: (a) capable of retaining and using prior knowledge, and (b) capable of acquiring new knowledge over a series of prediction tasks, in order to improve the effectiveness (more accurate hypotheses) and efficiency (shorter training times) of learning. Although current research in lifelong learning is still in its infancy, several lifelong learning approaches have been proposed. Thrun was one of the first authors that addressed on lifelong learning from the point of view of transferring learning[2] in a series of works [TO96, MT93, Thr96a, Thr96b] but, unlike other previous transfer learning approaches (see [PY10] for a survey on transfer learning for classification, regression, and clustering problems), he used a measure of *task relatedness*, for transferring knowledge from the most related set of learning tasks. Thrun et al. [Thr96a] also worked on a lifelong learning approach called *explanation-based neural networks* where previously learnt networks are used to guide the subsequent learning of tasks.

Multiple Task Learning[3] (MTL) [Bax00, Car93, Thr96b] has also been considered to incremental learning. Motivated by the limitations of the application of MTL networks to continuous or lifelong learning systems (several outputs of the network, task relatedness measure required, contextual cues, etc.) there have been some approaches that try to overcome them. Silver et al. proposed variants of sequential learning and consolidation systems using standard back-propagation neural networks [SM02, SP04]. They introduced the *task rehearsal method* (TRM), that uses the representation of previously learnt tasks (virtual training examples generated by previous learnt hypothesis) as a source of inductive bias and overcomes the problem of retaining specific training examples of early learning. Furthermore, Silver et al. [SP06] presented the *context-sensitive* multiple task learning (*cs*MTL) , a method that uses standard back-propagation single-output neural network and additional contextual inputs (attributes to distinguish task examples) for learning multiple tasks. MTL research has also considered the use of a shared basis for all task models to improve learning over a set of tasks. By using a common basis, these approaches share information between learning tasks and account for task relatedness as the models are learnt in tandem with the basis. For instance, Ruvole et al. [ER13] proposed Efficient Lifelong Learning Algorithm

[2]The transfer of learning is the ability of a AI system to recognize and apply knowledge and skills learnt in previous domains/tasks to novel domains/tasks

[3]Multiple Task Learning is well recognized as a method of inductive transfer that is able to maximise performance across several related tasks through shared knowledge.

(ELLA) that incorporates aspects of both transfer and multi-task learning.

Learning incrementally has also be approached from the point of view of reinforcement learning. Ring in [Rin97] presented the *continual learning* theory that stands for the constant development of increasingly complex behaviors. He developed CHILD, an agent capable of *Continual*, *Hierarchical*, *Incremental Learning* and *Development* with has no ultimate, final task. CHILD combines the well-known Q-learning [WD92] algorithm with the *Temporal Transition Hierarchies* (TTH) [Rin93] learning algorithm, a constructive neural-network-based learning system that allows a learning agent to incrementally build a hierarchy of skills (new behaviours are composed from variations on old ones) based on the temporal context. Tanaka et al. [TY97, TY03b, TY03a] also proposed a lifelong ML approach for autonomous-robots (what they call Lifelong Reinforcement Learning) by treating multiple environments as multiple-tasks. In this case the tasks to be solved are similar. The systems uses the *stochastic gradient method*, a type of memory-less reinforcement learning where the policy is modeled by an artificial neural network. As the bias, the authors focused on the variable range of each weight throughout the n tasks: the average weight is used as initial bias (in each node); the dispersion of weights is used as learning bias for the learning rate (α) of each weigh.

Within the area of cognitive and brain sciences, Grossberg et al. [Gro87] proposed the *Adaptive Resonance Theory* (ART), a "cognitive and neural theory of how the brain autonomously learns to attend, categorize, recognize, and predict objects and events in a changing world", to overcome the aforementioned *Stability-Plasticity* problem of forgetting previous learnt concepts. A central feature of all ART systems is a pattern matching process that compares an external input with the internal memory of an active code thus allowing memories to change only when input from the external world is close enough to internal expectations, or when something completely new occurs (the neural network-based model is retrained to hold the a new example). This feature makes ART systems well suited to problems that require on-line learning of large and evolving databases

So far, we have seen that in the vast majority of the previous approaches, the use of neural networks seems to be preferred, as the human and animal brains are based on (natural) neural networks. In contrast, humans are good at appropriately handling prior knowledge, both in the way it develops and is refined, and in the way it is applied to new problems. In fact, in humans, difficulty depends on how unrelated or non-contextual the solution is w.r.t. previous knowledge. This way of learning incrementally is closely related to the construction and contextual application of (large repositories of) background and prior knowledge for inductive problems. These have been a key and recurrent issue in the past in areas such as incrementality (data [KW92, FHR01] and knowledge [Ols95, Sol02, Sch04, Hen10]), policy reuse [MFHR13a, MFHR13d, MFHR13c], incremental self-improvement [SZW97], function and predicate invention [MB92, Ols99, HM12, KMP98], constructive induction [Mug87], constructive reinforcement learning [HO00b], meta-knowledge [CKM+05, MS95], declarative-bias [BT08], (interactive) theory revision [RM91, DR92], theory completion [MB00], expert and knowledge-based systems [Mar13, AS10, ZCC+13], etc. Some of these areas are oldies (but goldies) in ILP, IP, program synthesis, AI, cognitive science and other areas. Many also have recently had a new revival.

While some of the latter approaches are not in the scope of inductive programming, we argue that, although clearly different to the way humans brains work, inductive programming is appropriate for incremental learning and has several advantages. First, as knowledge grows but systems do not develop natural language (non-declarative models), it becomes more and more difficult to analyse, supervise, fix and understand the knowledge of a system if it is not represented internally in an intelligible way. Using inductive programming does not ensure this introspection, but can make it possible due to the use of a rich symbolic representation that permits the use of a single language to represent background knowledge, examples and hypotheses (the latter being expressed as potentially comprehensible rules). As commented in the previous chapter, this explanatory power make it possible to extract human understandable knowledge. In contrast, those previous artificial neural network-based systems are much more difficult to interpret by humans once they begin to develop and growing possibly integrating millions of of neurons in several subsystems. For instance, by using Spaun [ESC+12], an impressive brain model based on millions of neurons that is able to do several tasks, including some list processing, it is hard to understand and really see how some problems are solved, i.e., it is difficult to check whether Spaun solves list problems by capturing the recursive concept of a list structure or because it uses some shortcuts (such as recognising the first and last element). Increasing model interpretability allows a better acceptance of the results by the domain users and, in the end, many applications require model understanding (scientific discovery, multi-agent systems, software engineering, engineering modelling...). Additionally, cognitive science, development robotics and robot programming by demonstration are also areas where inductive programming induced hypotheses are usually related to the solutions that humans would find for the same problem: we can understand the solutions reached by the system and we can see explicitly the mental constructs the system has been given [SHK09, MCSK06, Bur05b, SS12b]

Secondly, apart from the accessibility of one single language for the (end)user, knowledge can be easily inspected, revised and integrated with other sources of knowledge. Many tools for handling knowledge have been developed with symbolic or rule-based approaches, such as theory revision, consistency check, adding or removing constraints, etc. In fact, many pieces (or all) of hypotheses, data and background knowledge are comprehensible or understandable. This can hold in the short, mid and long terms for the background: we can provide start-up knowledge and, hereinafter, we can revise, evolve and fix their knowledge. This results in that incremental, cumulative or life-long learning becomes easier [HO14b]. As a successful example of this we find NELL [CBK+10], a *N*ever *E*nding *L*anguage *L*earner which uses an ILP algorithm for incrementally learning probabilistic Horn clauses. All this is basically another vindication of symbolic AI and symbolic learning.

3.4 Summary

Through the previous review we would like to criticise the widespread research bias in AI over the development of learning systems that learn from data again and again, and start completely from the scratch (without any knowledge acquisition and reuse) for each new task or problem. In fact, we wish to highlight the importance and relevance of developing incremental systems that need to acquire knowledge and keep this knowledge to improve their (posterior) learning abilities.

However, current incremental approaches have, in our opinion, two main drawbacks: (a) they lack of suitable mechanisms for dealing automatically with complex structured knowledge (both data and knowledge learnt); and (b) most of them are based on artificial neural networks and other statistical approaches, where it is not clear how knowledge can be accessed, revised and integrated with other sources of knowledge. Furthermore, catastrophic forgetting has emerged as one of the main concerns facing sequential learning problem with artificial neural networks.

We have (briefly) argued that inductive programming is a good way to address the knowledge acquisition problem. We say that, when dealing with the techniques to incrementally acquire, maintain, revise and use the learnt knowledge, symbolic knowledge representation paradigms (by using declarative programming languages featuring powerful typing systems, abstraction and/or higher-order features) have several advantages over non-symbolic ones, especially when knowledge becomes complex. Declarative approaches (such ILP, IFP or IFLP) permit the use of a single language to represent background knowledge, examples and hypotheses. Furthermore, induced hypotheses are not only comprehensible (knowledge can be inspected), but also they are usually related to the solutions that humans would find for the same problem, as the constructs that are given as background knowledge are explicit and shared by users and the inductive programming system. As a result, incremental, cumulative or lifelong learning becomes easier [HO14b]. However, creating incremental systems that need to acquire and handle complex data and knowledge and keep this knowledge to improve their learning abilities is a challenge that requires a more continuous effort and larger teams, as results can be discouraging initially: newborn baby inductive programming systems are not expected to be very useful until they can be trained and can fully develop.

4 Complexity, compression and structure in knowledge evaluation

In the previous chapters, the importance of the knowledge representation language used in AI is stated arguing that symbolic knowledge representation paradigms and symbolic learning have several advantages over non-symbolic ones, especially when dealing with complex scenarios. This makes it possible to deal with a wide variety of problems (which is the main characteristic of general-purpose learning systems). Furthermore, and following the initial motivation of designing and developing AI systems that learn as humans do, we also brushed up some of the most relevant approaches in AI to handle, firstly, complex structured knowledge and, secondly, cumulative and incremental learning. In this chapter we move to the evaluation of learnt knowledge and we review some specific techniques and approaches for assessing the quality of this knowledge in terms of information theory and the relationships between its parts. Here we consider the techniques are those in which the learning approaches developed in this thesis (particularly, their assessment criteria) have been based and, as we will see in Chapter 8, best suit the incremental learning view of knowledge acquisition.

This chapter is organised as follows. Section 4.1 introduces the basis for evaluation in predictive machine learning and motivates the use of complexity and link-based evaluation measures. Section 4.2 gives a short overview of the MML principle as an assessment and selection criterion for inductive programming. Section 4.3 introduces link analysis focusing on *object ranking* and those algorithms (HITS, PageRank and SALSA) used to exploit hyperlinks of the web to rank the websites. Finally, Section 4.4 closes the chapter with a brief summary and some conclusions.

4.1 Introduction

Mitchell [Mit97][Chapter 5] formalised the goal of predictive models in AI as the learning of a target function f by considering a space H of possible hypotheses where a hypothesis h is an approximation of f. In order to induce h, the learning algorithms employ a set of evidences (observations) formed by training instances D according to a distribution which defines the probability of encountering each instance in the space of all instances. Since it is possible to induce many and diverse hypotheses from the same evidence (by using different machine learning techniques), it is important to compute "goodness" measures to rank the hypotheses according to its "distance" with respect to f in order to know which is the best. The criterion for evaluating the "goodness" of each of the hypothesis is usually a real value function that takes as parameters the instances and/or class descriptions, and can be computed either over the sample data (*sample error*), or over the whole distribution of instances D (*true error*). The *sample error* of h with respect to a target function f and data sample S is the proportion of examples that h misclassifies, formally:

$$error_S(h) \equiv \frac{1}{n} \sum_{x \in S} \delta(f(x) \neq h(x))$$

where n is the cardinality of S and $\delta(f(x) \neq h(x))$ is 1 if $f(x) \neq h(x)$, and 0 otherwise. Meanwhile, the *true error* of hypothesis h with respect to a target function f and distribution D is the probability that h misclassifies an instance drawn at random according to D, formally:

$$error_D(h) \equiv P_{x \in D}[f(x) \neq h(x)]$$

The *sample error* is the only one possible to compute since the target function f is only known for the training sample. In order to avoid overfitting, there are many mechanisms to measure the sample error of a hypothesis with respect to an evidence. This includes *partition*, which splits the evidence into two disjoint subsets (*training* and *test* dataset), and *k-fold cross validation*, which repeatedly divide the available dataset into k disjoint subsets (the union of $k-1$ subsets for training and the remaining for testing).

However, not always there are just one plausible explanation, model or theory about a phenomenon. Consider some data D and several hypotheses over D, all of them having the same sample error (in fact, for each sample error there are infinity different hypothesis which may take this error), which of them is the most plausible hypothesis? In this case we need a different from a *black-*

box evaluation [1] criteria such as the previous sample error. Since most learning systems are inductive in principle, involving reasoning from few examples to a generalisation, and learning by induction is inherently about compressing observations (the instances) into a theory (the model), we find that making use of *Algorithmic Information Theory* (AIT) [Cha70, Kol68, Sol64b, Sol64c] criteria is an effective and unified framework for selecting the best hypothesis. AIT intuitively allows us to quantify the notion of complexity and compressibility of objects which is crucial for AI systems (and specially cognitive systems), especially when patterns need to be transmitted. Wallace and Boulton [WB68b], extend this compressibility notion into their Minimum Message Length (MML) principle, which is intended not just as a theoretical construct, but as a technique that may be deployed in practice. The MML approach defines a goodness measure for a model in which simpler theories are preferable to more complex ones, namely, it specifies that the minimal encoding in terms of length, which includes the model itself (complexity) and the objects covered by it (compression), is the best one. This approach has been used as a foundation of a selection (or optimality) criterion for our learning system gErl as we will see in Chapter 5.

Following with these notions of complexity and compression, an step beyond would be taking into account the relationships between the different pieces of knowledge learnt. This is not a black-box but a *piecewise* evaluation approach. While most learning systems rely on stored knowledge primarily in the form of unrelated rules (observations and hypotheses), these methods are not effective for a comprehensive analysis of data relations (e.g., coverage relations). In this regard, we find appropriate and inspirational those techniques of *Link Analysis* and *Web Search* used to evaluate relationships (connections) between nodes (websites) in networks of interconnected objects (WWW) in order to determine rankings between them. The analysis of relationships and information flow between individuals jointly with the idea of using a linking graph as the basis for structuring knowledge is an appealing idea for extending the MML principle to a knowledge network (in terms of coverage or generality). The metrics extracted from this knowledge network would be more flexible than actual probabilities and appropriate for AI systems that accumulate knowledge continuously as we will see in Chapter 8.

Therefore, while in the following sections we will provide an introduction to both the MML principle and link analysis (in web graphs) as a criteria

[1]In black-box evaluation we want to measure and compare AI models with different architectures and mechanisms by comparing their results independently of how they achieve them.

to rank diverse kinds of knowledge, in Chapters 5 and 8 we will see how we could use these approaches as a foundation of selection criteria and optimality assessment metrics for incremental acquired knowledge.

4.2 Complexity and compression

The relationship between complexity and compression is the foundation of both the Minimum Message Length (MML) [WB68b, BW70, BW75] and the closely related Minimum Message Length (MDL) [Ris78a]. They both are based on AIT and hence lie on the intersection of computer science and statistics. MML is and has been one of the most popular selection criterion in inductive inference (for a formal justification and its relation to Kolmogorov complexity and the related MDL principle, see [LV08b, WD99b, Wal05b]) and provides an interpretation of the Occam's Razor principle: the model generating the shortest overall message (composed by the model and the evidence concisely encoded using it) is more likely to be correct. A Bayesian interpretation of the MML principle [WB68a] asserts that the best conclusion to draw from data is the theory with the highest posterior probability or, equivalently, the theory which maximises the product of the prior probability of the theory and the probability of the data occurring in light of that theory. We quantify this immediately below.

Given a hypothesis H and an observed E (evidence), we can write the posterior probability of H given E by application of Bayes' Theorem:

$$P(H|E) = \frac{P(H) \cdot P(E|H)}{P(E)} = \frac{P(H \cap E)}{P(E)} \tag{4.1}$$

where $P(H)$ is the prior probability of H, $P(E|H)$ is the likelihood, and $P(E)$ is the probability of the evidence E. Since E and $P(E)$ are given and we need to infer H, we can regard the problem of maximising the posterior probability, $P(H|E)$, as one of choosing H such that maximises $P(H) \cdot P(E|H)$.

An information-theoretic interpretation of MML is that a given evidence E of probability $P(E)$ can be coded (e.g. by a Huffman code or by better compression codes) by a message of length $L(E) = -log_2(P(E))$ [Sha48]. Therefore, taking the negative logarithm of expression 4.1, since $-log_2(P(H) \cdot P(E|H)) = -log_2(P(H)) - log_2(P(E|H))$, maximising the posterior probability, $P(H|E)$, is equivalent to minimising $-log_2(P(H)) - log_2(P(E|H))$, the length of a two-part message which represents (describes) both the model H and data E jointly. Then, the length of a hypothesis H given a fixed evidence E denoted by $L(H|E)$ can be formally defined as:

$$L(H|E) = L(H) + L(E|H) - L(E) \tag{4.2}$$

By minimising equation 4.2 we maximise the posterior probability, which involves searching for the model that gives the shortest message.

Apart from its connection with Kolmogorov complexity and Solomonoff induction [LV08b], which gives additional support for its use, the MML principle (and the similar the *Minimum Description Length* (MDL) principle [Ris78b]) has been successfully applied in many areas of machine learning (including unsupervised classification, decision trees and graphs, DNA sequences, Bayesian networks, neural networks, image compression, image and function segmentation, etc.), inductive programming, AI and cognitive science.

4.2.1 Advantages of MML

Given that the selection of the "right hypothesis" represents an ideal scenario usually not feasible in classical learning problems, an effective approach is to use an optimality criteria in order to select an optimal or best program from all the many possible valid programs. We argue that the MML principle, as applied to the assessment of models or theories from data has several key points:

- MML is a purported truth simplicity measure of simplicity: the notion of simplicity is a virtue in scientific theories where simpler theories should be preferred to more complex ones.

- As the message length combines both a measure of complexity in the model and in the data, the MML approach provides a method for simultaneously evaluating the trade off between the two, both measured in bits, preventing over-fitting.

- In regular problems, MML estimators have good properties, such as asymptotic consistency [Wal05a] (although the amount of data increases, the estimator converges to the true value). Furthermore, it is scalable and statistically invariant (for problems where the amount of data per parameter is bounded above, MML can estimate all parameters with statistical consistency).

- MML is a method of model comparison that not only can it be used to compare models of different structure, but also it gives every model a score.

- MML can handle continuous-valued parameters by using different approximations of how accurately to state parameters in a model.

Both The MML and MDL principles helped popularise the view of inductive inference in terms of compression. Actually, this view has extended over areas such as pattern recognition, machine learning and artificial intelligence in the past decades [Wat72, VL97, CV05, SB06]

4.3 Link analysis

Structuring knowledge hierarchically in the form of a network or a graph can provide interesting knowledge about relationships between nodes, information flow between individuals or groups, generality, relative importance in network, isolation from other entities, similarity, etc. Based on a branch of mathematics called "graph theory", *link analysis* is the data analysis technique used to evaluate those relationships between entities belonging to a network or graph (a collection of entities and links between them). The domain of Link Analysis encompasses several distinct tasks determined by the different possible outcomes of analysing link data: *Link-based object classification* (LOC) is a technique used to assign class labels to entities according their link characteristics; *Link prediction* is used to extrapolate the knowledge or the patterns of links in a given network to infer novel links that are plausible, and may occur in the future; and *Link-based Object Ranking* (LOR), which associates a relative quantitative assessment with each entity using link-based measures. Object ranking is the focus of this section.

Mainly studied in the development of web search engines (that essentially are document-level ranking and retrieval engines), the aim of object ranking is to calculate the scores for the websites based on their "relevance" according to a given query and the connections through hyperlinks between websites. Object ranking has intellectual antecedents in the fields of *Social Network Analysis* and *Citation Analysis* (an area of *bibliometric* research). Since many standard documents include references or explicit citations to other previously published documents, by using this citations as links, standard corpora can be viewed as a graph which can provide interesting information about the similarity of documents and the structure of information. These disciplines seek to quantify the influence of scholarly articles by analyzing the pattern of citations amongst them, thus establishing a (semantic) similarity relationship between documents (*Co-Citation* [Sma73] or *Bibliographic Coupling* [Kes63]), or grading the importance (quality, influence) of scientific journals by measuring how often papers in the journal are cited by other scientists (*Impact factor* [G$^+$72]).

Notwithstanding, web links are a bit different than citations. Although link analysis on the Web treats the hyperlinks from one website to another as a bestowment of authority (such as citations represent the conferral of authority from one article to others), not every citation or hyperlink implies that and, thus, simply measuring the quality of a web page by the number of inlinks (citations from other pages) is not robust enough. This is because each individual link may have many possible meanings: it may be off-topic, it may convey criticism rather than endorsement, or it may be a paid advertisement. See, for instance, the link spam phenomenon which is an intent of artificially boosting the score of a target website by setting up multiple web pages pointing to it. Furthermore, the Web introduces new kinds of problems such as the dynamic and constantly-changing nature of Web content. Therefore, it is hard for search engines to automatically assess the intention of each link and it is necessary to derive useful aggregate signals for ranking from more sophisticated link analysis.

During the late 1990s, the two most famous algorithms to exploit the hyperlinks of the web to rank the pages were proposed: HITS (Hypertext Induced Topic Search) [Kle99] and PageRank [BP98]. We will see these algorithms (among others) in the following subsections.

4.3.1 Hubs and Authorities

The HITS algorithm produces two rankings of a set of pages: the *hub* ranking and the *authority* ranking. The intuitive idea behind hubs and authorities came from the way how humans analyse a search process, namely, actually understanding the search query rather than returning the exact (literal) matching. An authority is a website with many ingoing links meaning that its content is important (authoritative) for some topic and, thus, there are many other websites linking it. Other websites, known as hubs (websites that have outgoing links to other websites) serve as organisers of the information on a topic by linking to authoritative websites. Therefore, a good hub page is one that points to many good authorities, while a good authority page is one that is pointed to by many good hub pages. This define a recursive relationship between websites.

The HITS algorithm starts by collecting the set S of relevant websites for a specific query (returned by the search engine by means of information retrieval techniques) and, then, it iteratively computes a hub score and an authority score for every web page in the returned subset of websites (in an

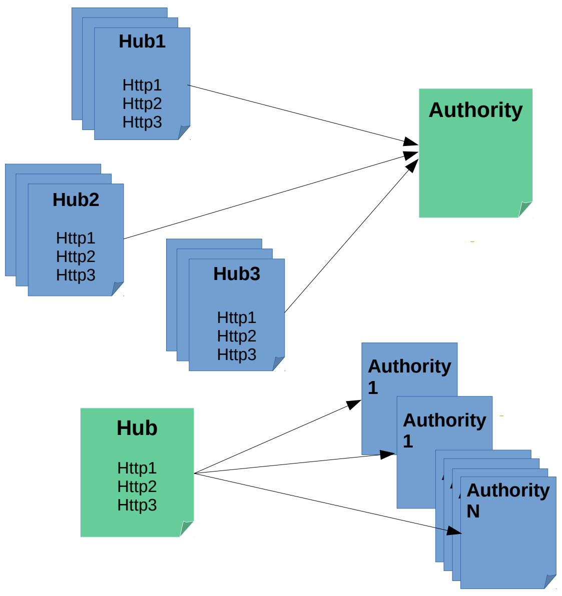


Figure 4.1: Authority and hub values are defined in terms of one another in a mutual recursion. An authority value is computed as the sum of the scaled hub values that point to that page. A hub value is the sum of the scaled authority values of the pages it points to. Some implementations also consider the relevance of the linked pages.

offline mode). Therefore, HITS does not rank the whole web but a subgraph of the web according to a query. Then HITS expands S with any website that links to a page in S and any website that is linked by pages in S. Next, HITS iterates recursively to generate a hub and an authority value for each page using the following updating formulas:

$$auth(p_i) = \sum_{j=1}^{n} hub(p_j)$$

where n is the total number of pages p_j that point to p_i $(p_j \to p_i)$.

$$hub(p_i) = \sum_{j=1}^{n} auth(p_j)$$

where n is the total number of pages p_j that are pointed by p_i $(p_i \to p_j)$. As we will see in the following section, unlike PageRank, HITS is computed at query time (online) and can therefore significantly affect the response time of a search engine. See Figure 4.2 for an example of PageRank.

While the most important feature of HITS is the mutual reinforcement relationship, its main disadvantage is that the subgraph (on which HITS is focused) is built “on the fly” and thus it is query dependent, i.e., minor changes in the web could significantly change the scores; and the response time is slow. Furthermore, it cannot detect advertisements (the exchange of links between sites full of commercial advertising sponsors can reduce the accuracy of the algorithm) and suffers from *topic drift* [LMF08], i.e, the graph could contain nodes having high authority scores for a topic unrelated to the original query.

4.3.2 PageRank

PageRank (PR) is the most important part of Google’s ranking system, a quality metric invented by Google’s founders Larry Page and Sergey Brin at Stanford University in 1995 [BP98] with the aim of bringing order to the web. It works on webgraphs: directed graph which a lognormal degree distribution, namely, there are very few large degree nodes (nodes with a large amount of incoming/outgoing links). The original idea was that a page is important if it is pointed to by other important pages: the PR score of a website is determined by summing the PRs of all pages that point to yours. PR uses a recursive scheme similar to HITS but in this case it produces and assigns a ranking (independent of a user’s query) from 0 to 10 (depending on the importance) to each hyperlinked web page within the WWW. Therefore, if an important page p points to some pages, the PR score is proportionally distributed to all its out-linked pages. Hence, a page will have a high PR score if there are many pages pointing to it, or if there are some high score pages pointing to it. Figure 4.2 shows an examples of PR scores for a simple network.

The PR score of a web page is defined in a recursive way (until convergence happens) and only depends on the PR of all the incoming pages, namely, those which have links to it. Therefore, for a page p, the higher PR score its incoming web pages have, the higher PR score the page p will obtain. PR is calculated offline (not at query time) by using the following formula:

$$PR(p_i) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j)}{L(p_j)}$$

where $p_1, p_2, \ldots p_N$ are the pages under consideration, d is residual probability (damping) usually set to d = 0.85 which is estimated from the frequency that a random web-surfer uses their browser’s bookmark feature (85% likelihood of choosing a random link from the page they are currently visiting, and a 15% likelihood of jumping to a page chosen at random from the entire web), $M(p_i)$

is the set of pages that link to p_i, $L(p_j)$ is the number of outbound links on page p_j, and N is the total number of pages. See Figure 4.2 for an example.

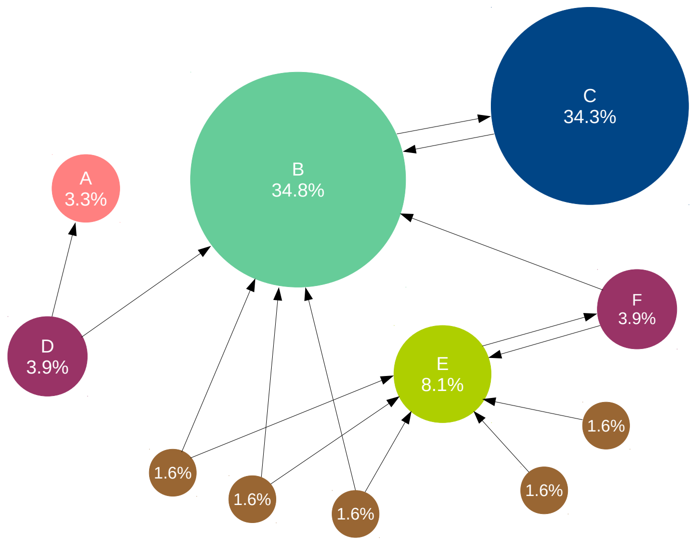


Figure 4.2: PageRanks for a simple network, expressed as percentages, where nodes represent websites and edges represent links between them. Page C has a higher PageRank than Page E, even though there are fewer links to C; the one link to C comes from an important page and hence is of high value. Without damping, all web surfers would eventually end up on Pages A, B, or C, and all other pages would have PageRank zero. In the presence of damping, Page A effectively links to all pages in the web, even though it has no outgoing links of its own. Adapted from http://commons.wikimedia.org/wiki/File:PageRanks-Example.svg

The PageRank does not require any analysis of the actual (semantic) content of the web nor user's queries. Therefore, PR is a global measure and is query independent. Furthermore, PR is robust against spam. On the other hand, the major disadvantage of PR is that it favours older pages (new pages will not have many in inbound links).

4.3.3 SALSA

Finally, we find the SALSA (Stochastic Approach for Link Structure Analysis) algorithm, a hybrid approach which combines ideas from both HITS and PAGERANK. SALSA was developed by Lempel and Moran [LM00] and is used to assign high scores to hub and authority web pages based on the quantity of hyperlinks among them. SALSA uses a bipartite hub-authority graph (like HITS) and performs a random walk (like PageRank) alternating between hubs and authorities: when it is at a node on the authority side of the bipartite graph, the algorithm randomly selects one of the incoming links and moves to a hub node; when it is at a node on the hub side, the algorithm randomly selects one of the outgoing links and moves to an authority node. Like HITS, SALSA only works in a subnetwork of pages in an online mode. The Twitter Social network uses a SALSA style algorithm to suggest the accounts to follow [GGL+13].

4.4 Summary

The objective of this chapter has been to introduce two vastly different methods for assessing the "informativeness" or "usefulness" of every single piece of knowledge learnt by an AI system in different terms: (1) the Minimum Message Length principle is able to quantitatively assess how good an induced (from examples) hypothesis is by its relationship between its complexity and its compression; and (2) the object ranking in link analysis, which, through the evaluation of the relationships (connections) in linked structures like graphs, is able to rank objects in the structure based on several factors defining their importance.

Whereas the MML criterion advocates for the model simplicity, the link analysis emphasizes relationships between data. While the former will be used as the selection criterion used in our learning approach gErl (Chapter 5), for our incremental knowledge acquisition approach (Coverage Graphs) we will see how we could go one step further by taking advantage of both techniques to assess the relevance or usefulness of an specific individual (such as a rule) based not only on the relationship between its own complexity but also on the complexity of the rest of (related) knowledge. This will lead us to an easy and general criterion to select and arrange knowledge that takes the MML as a starting criterion from which it is possible to derive new metrics to associate relative quantitative measures to individuals using the relationships between them (whichever the link between individuals is). We see this in detail in Chapter 8.

5

Towards declarative and general-purpose learning approaches

In this chapter we describe our general-purpose learning system gErl which was born as an advocacy of a more general framework for learning systems. gErl is a declarative and general-purpose rule-based learning system where learning operators (inductive mechanisms) can be defined and customised according to the problem, data representation and the way the information should be navigated. The learning process is guided by a reinforcement-based rewarding system where the application of an operator over a rule (in order to generate a new rule) is seen as a decision problem. gErl has been applied to some inductive programming problems involving deep structures and recursion. As a remark, the goal of this chapter is not to evaluate gErl, but to provide a new perspective or procedure of how through its declarative character and the use of constructs (learning operators), we can gain some insight into the characteristics and usefulness of the problems solved. In other words, gErl is not an end in itself, but an instrument for the analysis of a series of issues in this and the following chapters. Motivated also by the current increasing trend of the development of computer models aimed at solving intelligence tests (we will investigate this in Chapter 6), this versatility and expressibility features of gErl has allowed us to assess the abilities and concept dependency (learning operators) of AI systems by solving some fluid intelligence tests, as we will see in Chapter 7.

This chapter is organised as follows. Section 5.1 introduces our approach and the need of more flexible, incremental and symbolic approaches for AI. Section 5.2 introduces gErl and the way operators are expressed and applied. Section 5.3 describes the heuristics based on reinforcement learning used to guide the learning process. Finally, a more comprehensive summary and conclusions close the chapter in Section 5.4.

These results have been published in [MFHR13b, MFHR13c, MFHR13d, MFHR13a].

5.1 Introduction

As we saw in chapter 3, the number and performance of AI systems dealing with complex, structured data have considerably increased in the past decades. However, the performance of these systems is usually linked to a transformation of the feature space (possibly including the outputs as well) to a more convenient, flat, representation, which typically leads to incomprehensible patterns in terms of the transformed (hyper-)space. Alternatively, other approaches do stick to the original problem representation but rely on specialised systems with embedded operators that are only able to deal with specific types of data. Despite all these approaches and the vindication of more general frameworks, in general terms, there is no general-purpose learning systems which can deal with *all* of these problems *preserving* the problem representation, although some general frameworks are being developed within ILP (such as METAGOL [MLPTN14, ML13, CM15]) .

In this chapter we present our attempt to address some of the previous points thus following our initial goals of generality or versatility and expressiveness. For that we have developed a general-purpose rule-based learning system gErl where operators can be defined and customised for each kind of problem. While one particular problem may require generalisation operators, another problem may require operators which add recursive transformations to explore the structure of the data. A right choice of operators can embed transformations on the data but can also determine the way in which rules are generated and transformed, so leading to (apparently) different learning systems. Making the user or the problem adapt its own operators is significantly different to the use of feature transformations or specific background knowledge. In fact, it is also significantly more difficult, since operators can be very complex things and usually embed the essence of a machine learning system. A very simple operator, such as *lgg*, requires several lines of code in almost any programming language, if not more. Writing and adapting a system to a new operator is not always an easy task. As a result, having a system which can work with different kinds of operators at the same time is a challenging proposal beyond the frontiers of the state of the art in machine learning.

In addition, learning operators are tools to explore the hypothesis space. Consequently, some operators are usually associated with some heuristic strategies (e.g., generalisation operators and bottom-up strategies). By giving more freedom to the kind of operators a system can use, we lose the capacity to analyse and define particular heuristics to tame the search space. This means that heuristics must be overhauled, in terms of *decisions* about the operator that must be used at each particular state of the learning process. This will be addressed following a reinforcement learning approach [Sut98] concerned with how an AI agent ought to take actions in an environment so as to maximise some notion of cumulative reward. This approach perfectly suits our objective of developing more general and adaptive AI systems where systems are not programmed to do things, but trained to do things, e.g., in gErl, the learning agent cleverly explores and interacts the hypothesis space through the use of actions, observations and rewards.

Therefore, we propose a learning system where learning operators can be written or modified by the user. Since operators are defined as functions which transform patterns, we clearly need a language for defining operators which can integrate the representation of the examples, patterns and operators. As we have argued in Chapter 2, functional programming languages, with reflection and higher-order facilities, are appropriate for this. We also decided to use a powerful and relatively popular programming language in this family, Erlang [VWW96] (Appendix A). A not less important reason for using a functional language is that operators can be understood by the users and properly linked with the data structures used in the examples and background knowledge, so making the specification of new operators easier. The language also sets the general representation of examples as equations, patterns as rules and models as sets of rules.

Interestingly, different problems using the same operators can reuse the heuristics. Since the reinforcement system determines which rules and operators are used and how they are combined, learnt policies can be reused between similar (or totally different) tasks. The knowledge transferred between tasks can be viewed as a bias in the learning of the target task using the information learnt in the source task. In order to do that, we use an appropriate abstract feature space for describing the kinds of rules and operators that are giving good solutions (and high rewards), so this history is reused for other problems, even when the task and operators are different.

5.2 The gErl System

Our learning paradigm has drawn upon several machine learning areas and techniques which work with rich data representations and, thus, it collects ideas from all of them. Some of these areas include inductive programming, reinforcement learning, evolutionary techniques, meta-learning, or transfer learning, which have been already reviewed in Chapter 3.

gErl can be described as a flexible architecture (shown in Figure 5.1) which works with populations of rules (expressed as unconditional / conditional equations) and programs in the functional language Erlang, which *evolve* as in an evolutionary programming setting or a learning classifier system [HB00]. Operators are applied to rules for generating new rules, which are then combined with existing or new programs. gErl provides some meta-level facilities called meta-operators which allow the user to define well-known generalisation and specialisation learning operators. Therefore, with appropriate operators, using a complexity and compression (based on the MML principle explained in Chapter 4) as the foundation of an optimality criteria (which feed a rewarding module), and using a reinforcement learning-based heuristic (where the application of an operator over a rule is seen as a decision problem fed by the optimality criteria) we will eventually find some good solutions to some learning problem. As a result, this architecture can be seen as a 'meta-learning system', that is, as a 'system for writing machine learning systems' or to explore new learning operators.

Furthermore, gErl is able to take advantage of previous learning episodes (problems) in order to improve or accelerate the learning of future tasks thus following an incremental view of learning. As we will see in more detail in Section 5.3, the redefinition of what policy reuse is in this context is motivated by the abstract representation of states and actions (application of an operator over a rule) as feature vectors, and the use of a Q-matrix where rewards for each state and action combination is stored and from which a supervised model is learnt. This makes it possible to have a more flexible mapping between old and new problems, since we work with an abstraction of rules and actions.

As we have said in the previous chapter, Erlang is used as a knowledge representation functional language to represent theories and examples in an understandable way: examples as equations, patterns as rules, models as sets of rules, and, for the definition of operators. The advantages of using the same representation language has been previously shown by the fields of ILP, IFP and IFLP (except for operators). Hence, we look for a flexible language, with powerful features for defining operators and able to represent all other elements (theories and examples) in an understandable way.

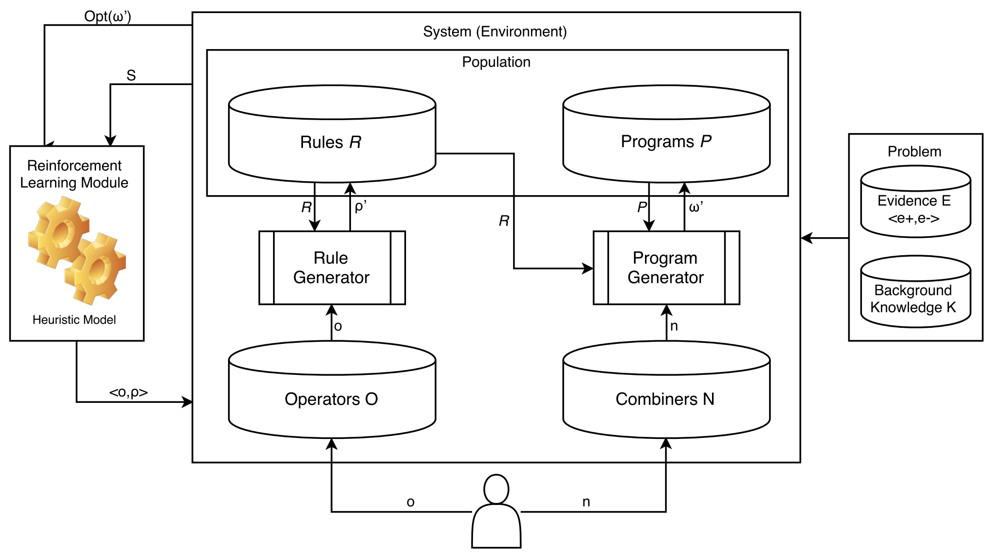


Figure 5.1: gErl's system architecture. A problem to solve is given by a set of positive and negative examples ($\langle e^+, e^- \rangle$) and a possible empty background knowledge K. There are two internal repositories containing rules (R) and programs (P). Initially, the set of rules R is populated with the positive evidence E^+ and the set of programs P is populated defining unitary programs ω from the rules of R. Both repositories are updated at each step of the algorithm: the Rule Generator *builds new rules ρ' and they are added to R while the* Program Generator *is in charge of generating new programs ω' by combining rules. The* Reinforcement Learning Module *is in charge of updating the optimality value of each new rule generated thus generating a table (Q-matrix) defining combinations of states and actions (operator applied over a rule) which will be used to select future actions to perform.*

5.2.1 Basic concepts

The core of our declarative learning system system is the mechanism for constructing (recursive) generalisation or specialization rules from a set of examples through inductive (user-defined) operators. Beforehand, we will review some classical machine learning terminology in the light of declarative learning paradigms (some of it already introduced in section 2.5). Although we use the vocabulary and notation of functional programming, this method also covers the synthesis of logic programs.

Given a subset of the functional programming language Erlang, it is used for expressing examples, patterns, operators and background knowledge, all of them in the form of functional rules. Given a set of function symbols (together with their arity) Σ (also called *signature*) and a countable set of variables $\mathcal{X}$, we denote the set of all terms over Σ and $\mathcal{X}$ by $\mathcal{T}(\Sigma, \mathcal{X})$ and the (sub)set of *ground* (variable free) terms by $\mathcal{T}(\Sigma)$. We distinguish (in Σ) function symbols that denote datatype constructors from those denoting (user-)defined functions. Thus, $\Sigma = \mathcal{C} \cup \mathcal{F}, \mathcal{C} \cap \mathcal{F} = \emptyset$ where $\mathcal{C}$ contains the constructors and $\mathcal{F}$ the defined function symbols respectively. We call terms from $\mathcal{T}(\mathcal{C}, \mathcal{X})$ constructor terms. Depending on the arity of symbols in Σ, a function is said to be a *constant* if its arity is equal to 0, otherwise it is said to be a *functor*. The set of variables occurring in a term t is denoted *Var(t)*. Hence, a term $t \in \mathcal{T}(\Sigma)$ if $Var(t) = \varnothing$. Following Erlang syntax, constant terms begin with a lower-case letter and variables begin with an upper-case letter.

$\mathcal{R}$ denotes the space of all (conditional) functional rules ρ expressed as l [when G] $\rightarrow r$, where l and r are, respectively, the left hand side (*lhs*) and the right hand side (*rhs*) of ρ, and G is an ordered conjunction (separated by comma) or disjunction (separated by semi-colon) of equality constraints (Boolean expressions) called guards $g_i = h_i$ with $g_i, h_i \in \mathcal{T}(\Sigma, \mathcal{X})$. If $G = \varnothing$, then ρ is said to be an unconditional rule. The *lhs* (l) of a rule ρ is a term of the form $F(a_1, \ldots, a_n)$, called *function head*, where $F \in \mathcal{F}$ is the name of the function symbol, and $a_i \in \mathcal{T}(\mathcal{C}, \mathcal{X})$ are built up from constructors and variables only. The sequence of the a_i is called pattern. The *rhs* (r) of a rule ρ is of the form B, r_b, being B (the body) a (possibly empty) conjunction of equations, and r_b being a term.

Let $\mathcal{P} = 2^{\mathcal{R}}$ be the space of all possible functional programs (modules in Erlang) formed by sets of rules $\rho \in \mathcal{R}$. Given a program $\omega \in \mathcal{P}$, we say that a term t reduces to term s with respect to ω, $t \rightarrow_\omega s$, if there exists a rule l [when G] $\rightarrow r \in \omega$ such that a subterm of t at occurrence u matches l with substitution θ, all conditions $(g_i = h_i)\theta$ hold, for each equation $l_i = r_i \in B \subset r$, $l_i\theta$ and $r_i\theta$ have the same normal form (that is, $l_i\theta \rightarrow^*_\omega \sigma$, and $r_{i_r}\theta \rightarrow^*_\omega \sigma$ and σ can not be further reduced) and s is obtained by replacing in t the subterm at occurrence u by $r_b\theta$.

The set of examples representing a learning problem, the evidence, is denoted as E. An example $e \in E$ is a ground equation $l \rightarrow r$ (that is, a rule without condition nor body) being r in normal form and both l and r are ground terms. We say that e is covered by a program p (denoted by $p \models e$) if r is the normal form of l with respect to p, i.e. $l \rightarrow^*_p r$.

A program $\omega \in \mathcal{P}$ is a solution of a learning problem defined by a labeled dataset E (composed as sets of positive E^+ and negative E^- examples) and a background theory K if it covers all positive examples, $K \cup p \models E^+$ (posterior sufficiency or completeness), and does not cover any negative example, $K \cup p \not\models E^-$ (posterior satisfiability or consistency). Our system has the aim of obtaining complete solutions, but their consistency is not a mandatory property, so approximate solutions are allowed. As usual, the coverage relation can also be defined in terms of the operational mechanism of the functional language.

The function $Cov^+ : 2^{\mathcal{R}} \to \mathbb{N}$ calculates the positive coverage of a program $\omega \in 2^{\mathcal{R}}$ and it is defined as $Cov^+(p) = Card(\{e \in E^+ : \omega \cup K \cup E^+ - \{e\} \models e\})$, where $Card(S)$ denotes the cardinality of the set S. Analogously, the function $Cov^- : 2^{\mathcal{R}} \to \mathbb{N}$ calculates the negative coverage of a program $p \in 2^{\mathcal{R}}$ and it is defined as $Cov^-(p) = Card(\{e \in E^- : \omega \cup K \cup E^+ - \{e\} \models e\})$. When we deal with recursive programs, we have to consider the problem of non-terminating proofs (that is, infinite sequences of rewriting steps). Note that, for proving the coverage of an example e we use the set $(E^+ - \{e\})$ as base cases for the target function (this is known as extensional coverage). Moreover, the length of the proofs are limited to a maximum number of rewriting steps.

A rule ρ can be represented as a tree, in a similar way as the usual tree representation of terms. Given a rule l [when G] $\to r$, the root of the tree represents the complete rule, and its three children represent l, G and r. Since G and r are conjunction of guards and equations (jointly with the r_b term) respectively, both nodes in the tree have as many children as components there are in the conjunctions. From here, the tree is populated following the usual tree representation of terms and equations. We call this kind of tree representation as *position trees*. Given the position tree of a rule ρ, $\mathcal{P}os(\rho)$, a *position* $p \in \mathcal{P}os(\rho)$ is a (possibly empty) sequence of natural numbers that denotes a subtree in the position tree, where Λ (the empty sequence) denotes the entire tree. The subpart of rule ρ at position $p \in \mathcal{P}os(\rho)$ is denoted as $\rho|_p$. Figure 5.2 shows the position tree of the rule

$$\rho : \texttt{member([}X\texttt{|}Y\texttt{],}Z\texttt{)[when } true\texttt{]} \to \texttt{member(}Y\texttt{,}Z\texttt{)}$$

As we can see, $\rho|_{1.1}$ is the term $[X|Y]$ and $\rho|_{3.2}$ is the term Z. It what follows, $pos(\rho) \subseteq 2^{\mathcal{P}os(\rho)}$ denote any subset of positions in the position tree $\mathcal{P}os(\rho)$.

It should be noted that the position tree of a rule ρ matches exactly with the Erlang abstract representation of the rule, namely, the *Abstract Syntax Tree* (AST) which, as explained before, is a tree representation of the abstract syntactic structure of a source rule written in Erlang. Each node of the tree denotes a component occurring in the source code. If we parse the abstract tree, we can enumerate each component in a post-order way obtaining a position tree.

Hereinafter, we will consider the following running example in order to illustrate how the system works. This example will be developed along the

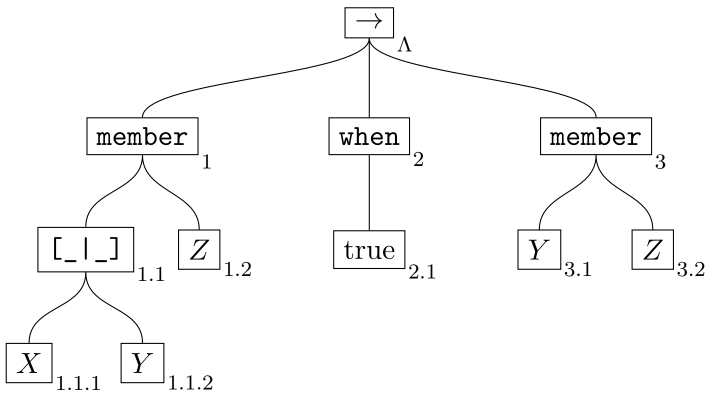


Figure 5.2: Position tree of the rule ρ : `member([X|Y],Z)[when true]` → `member(Y,Z)`. Positions (numerals) are placed at the bottom-right of each term.

following sections.

Example 2

Consider a simple recursive problem of finding the last element of a list of characters. This problem could be defined using a set of positive examples E^+ and a set of negative ones E^- (and an empty set of functions as the background knowledge K) as in Table 5.1.

id	$\mathbf{E}^+$	**id**	$\mathbf{E}^-$
1	$last([c]) \to c.$	1	$last([c]) \to b.$
2	$last([d]) \to d.$	2	$last([b]) \to l.$
3	$last([l]) \to l.$	3	$last([l]) \to c.$
4	$last([a,b,c]) \to c.$	4	$last([a,b,c]) \to a.$
5	$last([t,b,n,a,b]) \to b.$	5	$last([t,b,n,a,b]) \to t.$
6	$last([h,h,t,a,l]) \to l.$		
7	$last([a,c,b]) \to b.$		
8	$last([a,b,a,c]) \to c.$		

Table 5.1: Set of positive and negative examples provided to gErl in order to learn the recursive problem of last element of a list.

5.2.2 Operators over rules and programs

The definition of customised learning operators is one of the key concepts of our proposal. In gErl, rules R are transformed by applying a set of *operators* O. Then, an operator $o \in \mathcal{O}$ is a function $o : \mathcal{R} \rightarrow 2^{\mathcal{R}}$, where $O \subseteq \mathcal{O}$ denotes the set of operators chosen by the user for solving the problem. Roughly speaking, operators perform modifications over any of subparts of a rule in order to generalise or specialise it. The main idea is that, when the user is going to deal with a new problem, they can define their own set of *operators* (which can be selected from the set of predefined operators or can be defined by the user with the functions provided by the system) especially suited for the data structures of the problem. This feature allows our system to adapt to the problem at hand.

For defining operators, the system is equipped with meta-level facilities called *meta-operators*. A meta-operator is formally defined as

$$\mu :: (\mathcal{R} \rightarrow 2^{\mathcal{P}os}) \times (\mathcal{R} \rightarrow \mathcal{T}(\Sigma, \mathcal{X})) \rightarrow \mathcal{O}$$

which takes two functions returning a set of positions in a rule (as given by its position tree) and a term, and gives an operator. gErl provides the following two meta-operators able to define well-known generalisation and specialisation operators in inductive learning:

1. $\mu_{replace}(pos, f)$ defines an operator that, when applies to a rule ρ, replaces the subparts $\rho|_p, p \in pos(\rho)$ by $f(\rho)$. Notice that this meta-operator can be used to define both *generalisation* (replacing, for instance, a term by a variable) and *specialisation* (replacing a given term by another more specific) operators (depending on whether $f(\rho)$ is more general/specific than $\rho|_p$).

2. $\mu_{condition}(pos, f)$ defines an operator that inserts a Boolean condition $f(\rho)$ in all the positions $p \in pos(\rho)$. As gErl only allows the insertion of Boolean conditions in the guard of rules, pos is a constant function that always returns position 2. Notice that this meta-operator can be only used to define *specialisation* operators.

To simplify notation, any constant function $fun(\cdots) = const$ will be denoted as $\overline{const}$. We will see some application examples at the end of this section.

Finally, there is an internal operator called *one_step_rew* which, by default, is always included in the set O. This operator performs a step of rewriting at any position of the rhs of a rule. Our system also has a special kind of transformation $n \in N$, called *combiners*, that only apply to programs. The *Program Generator* module (Figure 5.1) applies a combiner to the last rule ρ' generated by the *Rule Generator* module and the population of programs P. Thus, a combiner $n \in N$ can be formally described as a function $n : \mathcal{P} \times \mathcal{P} \rightarrow \mathcal{P}$ that transforms programs into programs.

Example 3

Following with running example *last element of a list* in Example 2, the following step is to define appropriate operators (relying on the previous meta-operators) in order to allow the system to learn possible solutions. Since in Erlang lists can are internally dealt as *improper lists* ($[Head|Tail]$) the first pair of operators should be responsible of replace both the head and tail of the input list by a variable thus improving usability and generalisation:

$$\begin{aligned} op_1 &\equiv \quad \mu_{replace}(\overline{1.1.1}, V_{head}) \\ op_2 &\equiv \quad \mu_{replace}(\overline{1.1.2}, V_{tail}) \end{aligned}$$

Looking at the evidence in Table 5.1 and knowing that it could be useful to play with the structure of the input list, we define a new pair of operators in charge of replacing the *rhs* (equal to $\rho|_3$) of the rules by the head or the tail of the input list, namely:

$$\begin{aligned} op_3 &\equiv \quad \mu_{replace}(\overline{3}, f_h) \\ op_4 &\equiv \quad \mu_{replace}(\overline{3}, f_t) \end{aligned}$$

where $f_h(\rho) = \rho|_{1.1.1}$ and $f_t(\rho) = \rho|_{1.1.2}$.

Finally, in order to find the last element of a list we need to go through the latter until we find it. Therefore, by building recursive navigation over the input list we do not concern ourselves with having a consistent navigation (equal for all kind of arrays):

$$op_5 \equiv \quad \mu_{replace}(\overline{3}, f_{last})$$

where $f_{last}(\rho) = last(\rho|_{1.1.2})$.

Taking as an example the sequential application of some of the previous operators over, for instance, the evidence rule e_4 from Table 5.1 ($last([a, b, c]) \rightarrow c$) we will obtain:

$$\begin{aligned} op_5(op_2(op_1(e_4))) &\quad \Rightarrow \\ op_5(op_2(last([V_{head}, b, c]) \rightarrow c)) &\quad \Rightarrow \\ op_5(last([V_{head}|V_{tail}) \rightarrow c) &\quad \Rightarrow \\ last([V_{head}|V_{tail}) \rightarrow last(V_{tail}) & \end{aligned}$$

5.3 Heuristics based on reinforcement learning

Reinforcement learning [Sut98] is learning what to do so as to maximise a numerical cumulative reward signal. The learner is not told which actions to take, but instead it ought to discover which actions yield the most reward by trying them in an environment (trial and error search). Furthermore, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards (aggregated rewards). Reinforcement learning differs from standard supervised learning in that neither correct input/output examples nor the goal of the task (and the task itself) are provided, but indirectly given in the form of rewards, from which the agent can acquire its ultimate goals.

In this section we describe the reinforcement learning approach followed by gErl in order to guide the learning process. We decided on this approach since the freedom given to the user concerning the definition of their own operators implies the impossibility for us to define specific heuristics to explore the search space. This means that heuristics must be overhauled as the problem of deciding the operator that must be used (over a rule) at each particular state of the learning process.

Therefore, following a reinforcement learning approach, the population of rules at each step of the learning process can be seen as the *state* of the system and the selection of the tuple operator and rule can be seen as the *action*. However, the probably infinite number of states and actions makes the application of classical reinforcement learning algorithms not feasible. To overcome this, states and actions are represented in an abstract way using features. From here, a model-based reinforcement learning approach has been developed in order to use propositional machine learning methods for selecting the best action in each possible state of the system.

5.3.1 Optimality and stop criterion

Since the system is flexible and general in the way it represents and operates with rules, we need some general optimality criteria. First we have to select the optimal program (or a set of optimal programs, depending on the user's interests) as the solution of the learning problem. Second, we also need to feed the reward module in each step of the learning process.

The *Minimum Message Length* [WB68a] (MML), seen in Section 4.2, provides, as commented, a general way for selecting "the right hypothesis" supported by the classical view of unsupervised learning as compression: the model generating the shortest overall message (composed by the model and the evidence concisely encoded using it) is more likely to be correct. According to this philosophy, we present the simplest criterion we have essayed which is a concretisation of the MML principle. In particular, the optimality of a program ω is defined as the weighted sum of two simpler heuristics, namely, a complexity-based heuristic (which measures the complexity of ω) and a coverage heuristic (which measures how much extra information is necessary to express the evidence given the program ω)[1]:

$$Opt(\omega) = -\beta_1 \cdot L(\omega) - \beta_2 \cdot L(E|\omega) \tag{5.1}$$

Note that we use negative values due to the promotion of lower optimality values. Since programs are composed by rules, we can define the length of a program as the sum of the length of each rule belonging to ω. In particular, we can define the length of a rule ($L(\rho)$) in different ways depending on the rule representation language. For instance, if we are using logical or functional rules we could use the following approximation. Given Σ a set of m_Σ function symbols of arity ≥ 0, and $\mathcal{X}$ a set of $m_\mathcal{X}$ variables, we could define the length of a rule ρ containing n_Σ functors and $n_\mathcal{X}$ variables as

$$\begin{aligned} L(\rho) \triangleq\ & m_\Sigma \log_2(n_\Sigma + 1) \\ & + \frac{m_\mathcal{X}}{2} \log_2(n_\mathcal{X} + 1) \end{aligned} \tag{5.2}$$

Note that we promote variables over constants or functors. Therefore, we define the length of a program ω as

$$L(\omega) \triangleq \sum_{\rho \in \omega} L(\rho) \tag{5.3}$$

On the other hand, we define the covering factor w.r.t. an specified class i as the coding lengths of the instances not covered from the class i plus the cost of coding the exceptions, namely, the instances belonging to the rest of classes covered by ω:

[1]For the time being we have just considered $\beta_1 = 1$ and $\beta_2 = 1$, but these values could be parametrised for different kinds of problems.

$$L^i(E|\omega) = L(\{e \in E^i : \omega \not\models e\}) + \sum_{j \in C j \neq i} L(\{e \in E^j : \omega \models e\}) \tag{5.4}$$

In particular, since gErl accepts only learning problems E defined by positive and negative instances, equation 5.4 is amended to read as follows:

$$L(\left\langle E^+, E^- \right\rangle |p) = L(\{e \in E^+ : \omega \not\models e\}) + L(\{e \in E^- : \omega \models e\}) \tag{5.5}$$

As we have mentioned at the beginning of this section, the optimality measure is used to rank the programs generated by the system (in increasing order of their optimalities).

Finally, regarding the *stop criterion*, it can be specified by the combination of two conditions. The first one establishes that the learning process stops when the difference between the optimalities of the programs generated in the last n steps (where n is a parameter determined by the user) is less than a threshold ϵ (also determined by the user) which indicates that a better program is not likely to be found. Figure 5.3 shows the optimality of the programs generated at each step of the system for the current running example. As can be seen, the difference between the optimalities of the last generated programs maintain themselves within a tight fluctuation range, so the system stops by condition 1. The second stop condition limits the number of steps of the learning process to a maximum.

5.3.2 Reinforcement Learning problem statement

As we can see in Figure 5.1, gErl works with a set of rules $R \subseteq \mathcal{R}$, a set of programs $P \subseteq \mathcal{P}$ and a set of operators $O \subseteq \mathcal{O}$. Initially, R is populated with the positive evidence E^+ while the set of programs P is populated with as many unitary programs as rules are in R. As the process progresses, new rules and programs will be generated. First, the *Rule Generator* module (Figure 5.1) takes the operator o and the rule ρ returned by the *Reinforcement Learning Module* (policy). Then the operator is applied to the rule giving new rule ρ' which are added to R. Then, the *Program Generator* module combines ρ' (if appropriate) with an existing program to create a new program ω'(which is added to P). Therefore, in each iteration of the system, we have to select a rule and an operator to produce new rules. Depending on the problem to

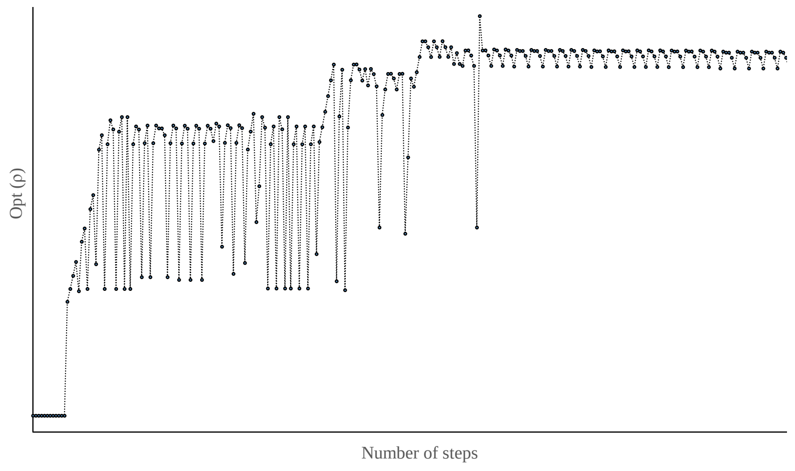


Figure 5.3: Example of the optimality values of the programs generated by gErl in successive steps (starting from 1). Notice that in final steps, the variation of the optimality is not constant but shows a stabilisation.

solve, the number of required iterations could be astronomically high. To address this issue we need a particular heuristic (policy) to tame the search space and make good decisions about the choice of rule and operator, in which the application of an operator to a rule is seen as a decision problem.

To guide the reinforcement learning process we need to describe the system in each step of the process (before and after applying an action) in terms of the quality of the system states (that is, the population of rules and programs generated until now). The infinite number of states makes their abstraction necessary. As there are infinitely many rules, we also have to use an abstraction for actions. This is done by defining an abstract representation of states and actions which constitutes the configuration of the RL problem we see next.

Formally, we define a state at each iteration or step t of the system as a tuple $\sigma_t = \langle R, P\rangle$ that represents the population of rules R and programs P in t. An action is a tuple $\langle o, \rho\rangle$ with $o \in \mathcal{O}$ and $\rho \in \mathcal{R}$ that represents the operator o to be applied to the rule ρ. Our decision problem is a four-tuple $\langle \mathcal{S}, \mathcal{A}, \tau, \psi\rangle$ where: $\mathcal{S}$ is the state space; $\mathcal{A}$ is a finite actions space ($\mathcal{A} = \mathcal{O} \times \mathcal{R}$); $\tau : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is a transition function between states and $\psi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function. These components are defined below:

- **States.** As we want to find a good solution to the learning problem, we abstract each state s_t by a tuple of features $\dot{s}_t$ from which to extract relevant information in step t (average size and optimality of the population). We denote by $\dot{\mathcal{S}}$ the set of state abstractions used to represent elements $\mathcal{S}$ in $\dot{\mathcal{S}}$. We denote by $\dot{\mathcal{S}} = \mathbb{R}^3$ the set of state abstractions used to represent elements $\mathcal{S}$ in $\dot{\mathcal{S}}$ (Table 5.2).

Feature	Description
ϕ_1	Average optimality of the population (Equation 5.1)
ϕ_2	Average size of rules (Equation 5.3)
ϕ_3	Average cardinality of programs

Table 5.2: Features ϕ_i abstracting states s_t

- **Actions.** An action is a tuple $\langle o, \rho \rangle$ where o is just an (non abstracted) operator identifier and each rule ρ is abstracted (similarly as before) by a tuple of features $\dot{\rho}$ from which we extract relevant information such as size, number of functors, constructors and variables or coverage rates. Therefore, an action is finally abstracted as a tuple of *nine* features, i.e. $\dot{\mathcal{A}} = \langle \mathbb{N}, \mathbb{R}^8 \rangle$, where the abstraction of the actions goes from $\mathcal{A} \rightarrow \dot{\mathcal{A}}$ (Table 5.3).

Feature	Description
o	Operator identifier
φ_1	Positive Coverage Rate
φ_2	Negative Coverage Rate
φ_3	Number of variables of ρ
φ_4	Number of constants of ρ
φ_5	Number of functors of ρ (arity greater than 0)
φ_6	Number of structures (lists, graphs, ...) of ρ
φ_7	Number of variables of ρ
φ_8	Is ρ recursive?

Table 5.3: Features φ_i abstracting actions $\langle o, \rho \rangle$.

- **Transitions.** Transitions are deterministic. A transition τ evolves the current sets of rules and programs by applying the operators selected (together with the rule) and the combiners.

- **Rewards.** The optimality criterion seen above (eq. 5.1) is used to feed the rewards.

At each point in time, the reinforcement learning policy π can be in one of the states $s_t \in \mathcal{S}$ and may select an action $a_t \in \mathcal{A}$ to execute. Executing such action a_t in s_t will change the state into $s_{t+1} = \tau(s_t, a_t)$, and the policy receives a reward $\psi_t = \omega(s_t, a_t)$. The policy does not know the effects of the actions, i.e. τ and ψ are not known by the policy and need to be learnt. This is the typical formulation of reinforcement learning [Sut98] but here we use features to represent the states and the actions. With all these elements, the aim of our decision process is to find a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ that maximises:

$$V^{\pi}(s_t) = \sum_{i=0}^{\infty} \gamma^i \psi_{t+i} \tag{5.6}$$

for all s_t, where $\gamma \in [0, 1]$ is the *discount parameter* which determines the importance of the future rewards ($\gamma = 0$ only considers current rewards, while $\gamma = 1$ strives for a high long-term reward).

5.3.3 Modelling the state-value function: using a regression model

Since gErl works with an abstract representation of states and actions, we use a hybrid between value-function methods (which update a state-value matrix) and model-based methods (which learn models for τ and ω) [Sut98]. The idea is to replace the *state-value* function $Q(s, a)$ of the Q-learning [WD92] (which returns quality *values*, $q \in \mathbb{R}$) by a supervised model $Q_M : \dot{\mathcal{S}} \times \dot{\mathcal{A}} \rightarrow \mathbb{R}$ that calculates the q value for each state and action, using their abstractions. gErl uses linear regression (by default, but other regression techniques could be used) for generating Q_M, which is retrained periodically from $Q(s, a)$. Then, Q_M is used to obtain the best action a_t for the state s_t as follows:

$$a_t = \arg\max_{a \in \mathcal{A}} \{Q_M(\dot{s}_t, \dot{a})\} \tag{5.7}$$

Since the rules are abstractly described using features, more than one rule can share the same description, if that happens when the system selects an action, the rule is randomly selected (among those which share the description).

In order to train the model we need to provide different states and actions as inputs, and quality values as outputs. More concretely, we use a 'matrix' Q (which is actually a table), whose rows are in $\dot{\mathcal{S}} \times \dot{\mathcal{A}} \times \mathbb{R}$ where $\dot{\mathcal{S}}$ is a tuple of state features, $\dot{\mathcal{A}}$ is the tuple of rule features and operator (both described in Section 5.3.2), and $\mathbb{R}$ is a real value for q. We set initial *q-values* (in s_0) to

$t = 0$

Id_{e^+}	e^+
1	$last([c]) \rightarrow c$
2	$last([d]) \rightarrow d$
3	$last([l]) \rightarrow l$
4	$last([a,b,c]) \rightarrow c$
5	$last([t,b,n,a,b]) \rightarrow b$
6	$last([h,h,t,a,l]) \rightarrow l$
7	$last([a,c,b]) \rightarrow b$
8	$last([a,b,a,c]) \rightarrow c$

Positive examples E^-

Id_{e^-}	e^-
1	$last([c]) \rightarrow b$
2	$last([b]) \rightarrow l$
3	$last([l]) \rightarrow c$
4	$last([a,b,c]) \rightarrow a$
5	$last([t,b,n,a,b]) \rightarrow t$

Negative examples E^-

Id_o	o
1	$\mu_{replace}\ (Rt_1, last(L_{1.1}))$
2	$\mu_{replace}\ (Rt_1, last(L_{1.2}))$
3	$\mu_{replace}\ (Rt_1, L_{1.1})$
4	$\mu_{replace}\ (Rt_1, L_{1.2})$
5	$\mu_{replace}\ (Rt_1, V_{Head}))$
6	$\mu_{replace}\ (Rt_1, V_{Tail}))$

Operators $O \in \mathcal{O}$

state (s)			action (a)										q
Φ_1	Φ_2	Φ_3	o	φ_1	φ_2	φ_3	φ_4	φ_5	φ_6	φ_7	φ_8	
-125.0	14.84	1	1	10.0	0.125	0.0	0	2	2	1	0	1
-125.0	14.84	1	1	10.0	0.125	0.0	0	2	2	1	0	1
-125.0	14.84	1	1	10.0	0.125	0.0	0	2	2	1	0	1
-125.0	14.84	1	1	16.0	0.125	0.0	0	4	2	1	0	1
-125.0	14.84	1	1	22.0	0.125	0.0	0	6	2	1	0	1
-125.0	14.84	1	1	22.0	0.125	0.0	0	6	2	1	0	1
-125.0	14.84	1	1	16.0	0.125	0.0	0	4	2	1	0	1
-125.0	14.84	1	1	19.0	0.125	0.0	0	5	2	1	0	1
-125.0	14.84	1	2	10.0	0.125	0.0	0	2	2	1	0	1
...												
-125.0	14.84	1	6	10.0	0.125	0.0	0	2	2	1	0	1

Matrix Q

id_ρ	ρ	MsgLen(ρ)	Opt(ρ)	Cov+ [ρ]	Cov- [ρ]
1	$last([c]) \rightarrow c$	10.0	-125.0	1 [1]	0 []
2	$last([d]) \rightarrow d$	10.0	-125.0	1 [2]	0 []
3	$last([l]) \rightarrow l$	10.0	-125.0	1 [3]	0 []
4	$last([a,b,c]) \rightarrow c$	16.0	-125.0	1 [4]	0 []
5	$last([t,b,n,a,b]) \rightarrow b$	22.0	-125.0	1 [5]	0 []
6	$last([h,h,t,a,l]) \rightarrow l$	22.0	-125.0	1 [6]	0 []
7	$last([a,c,b]) \rightarrow b$	16.0	-125.0	1 [7]	0 []
8	$last([a,b,a,c]) \rightarrow c$	19.0	-125.0	1 [8]	0 []

Set of rules generated $R \in \mathcal{R}$

Figure 5.4: gErl learning the last element of a list *problem: in step 0, R is initialised with the positive examples E^+ and the* Q-matrix *is initialised (for this initial state $s_{t=0}$) with all possible combination of abstract actions (operators $\times$ rules) with q values equal to 1. ϕ and φ represent the different features used to abstract the representation of states (Table 5.2) and actions (Table 5.3) respectively.) Examples E and operators O are those shown during the running example (Examples 2 and 3)*

.

have a high initial value, also known as “optimistic initial conditions” [Sut98], in order to encourage exploration: no matter what action will take place. Abusing notation, to work with Q as a function (like the original Q-matrix is used in many RL methods), we will denote by $Q[\dot{s}, \dot{a}]$ the value of q in the row of Q for that state $\dot{s}$ and action $\dot{a}$. So, Q grows in terms of the number of rows. Figure 5.4 continues with the example 2 and shows how gErl initialises the Q table and the set of rules R.

Once the system has started, at each step, Q is updated using the following formula:

$$Q[\dot{s}_t, \dot{a}_t] \leftarrow \alpha \left[\psi_{t+1} + \gamma \max_{a_{t+1}} Q_M(\dot{s}_{t+1}, \dot{a}_{t+1}) \right] + (1-\alpha) Q[\dot{s}_t, \dot{a}_t] \qquad (5.8)$$

which is a variation of the formula used in Q-learning for updating the Q-matrix where the maximum future value is given by the model. The previous formula has two parameters: the discount parameter $\gamma \in [0,1]$, and the *learning rate* α ($\alpha \in [0,1]$) which determines to what extent the newly acquired information will override the old information ($\alpha = 0$ makes the agent not to propagate anything, while $\alpha = 1$ makes the agent consider only the most recent information). By default, we set $\alpha = 0.5$ and $\gamma = 0.5$.

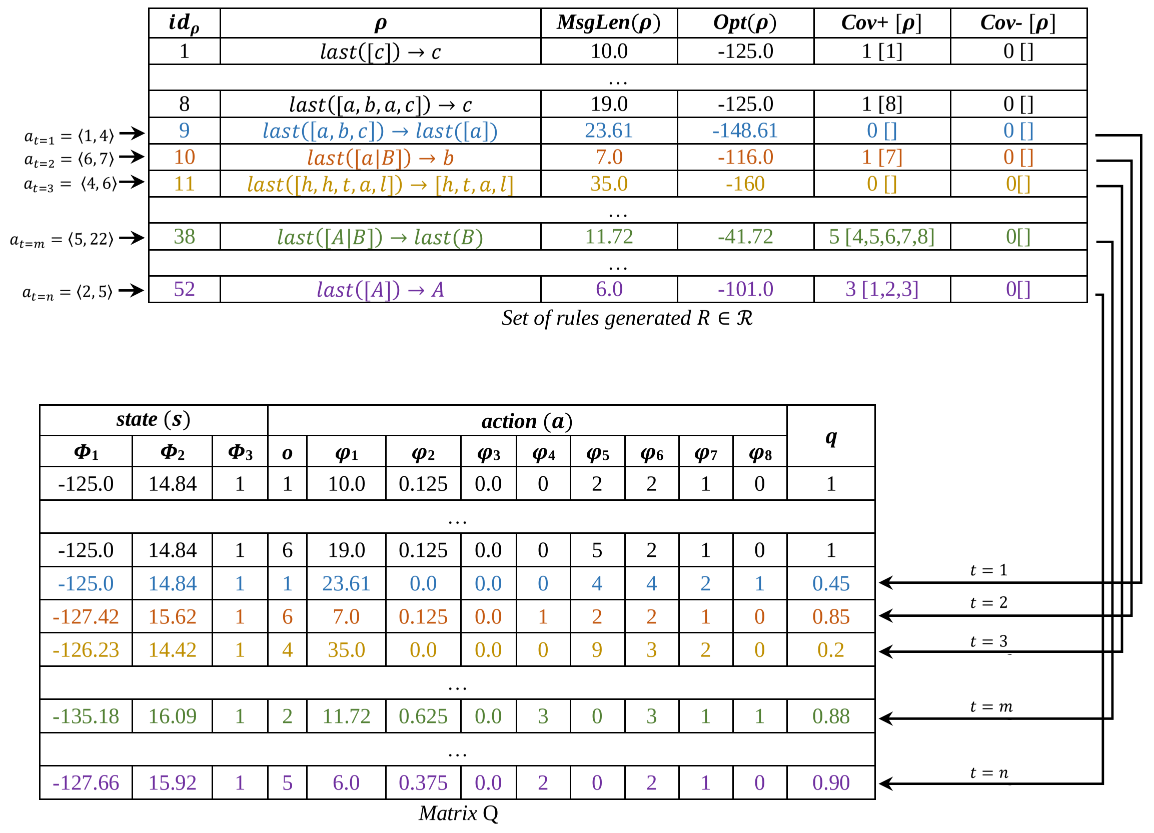
Following with Example 2 , Figure 5.4 and 5.5 show how gErl uses Q_M in order to get the best action to apply in each step of the learning process, and how the set of rules R and the Q-matrix are updated until the system reaches the *Stop Criterion*.

5.3.4 Reusing past policies

The intuition behind reusing policies comes from the simple idea that the probability that an agent behaves in a certain way should be proportional to how often this agent's behaviour has been successful in the past.

As we have seen in Section 3.3, in other TL methods the knowledge is transferred in several ways (via modifying the learning algorithm, biasing the initial action-value function, etc.) and, if the source and the target task are very different, a mapping between actions and/or states is also needed. Instead of that, in gErl the reuse of previous acquired knowledge is done in a different way, by reusing the values in the Q table.

The main reason why we can use past policies (table Q learnt in previous tasks) in order to accelerate the learning of different new tasks is due to the abstract representation of states and actions (the ϕ and φ features of $\dot{\mathcal{S}}$ and $\dot{\mathcal{A}}$ respectively) which prevents the system from starting from scratch. Actions



$a_{t=1} = \langle 1,4 \rangle$
$a_{t=2} = \langle 6,7 \rangle$
$a_{t=3} = \langle 4,6 \rangle$
$a_{t=m} = \langle 5,22 \rangle$
$a_{t=n} = \langle 2,5 \rangle$
$t = 1$
$t = 2$
$t = 3$
$t = m$
$t = n$

id_ρ	ρ	$MsgLen(\rho)$	$Opt(\rho)$	$Cov+ [\rho]$	$Cov- [\rho]$
1	$last([c]) \rightarrow c$	10.0	-125.0	1 [1]	0 []
		…			
8	$last([a,b,a,c]) \rightarrow c$	19.0	-125.0	1 [8]	0 []
9	$last([a,b,c]) \rightarrow last([a])$	23.61	-148.61	0 []	0 []
10	$last([a\|B]) \rightarrow b$	7.0	-116.0	1 [7]	0 []
11	$last([h,h,t,a,l]) \rightarrow [h,t,a,l]$	35.0	-160	0 []	0[]
		…			
38	$last([A\|B]) \rightarrow last(B)$	11.72	-41.72	5 [4,5,6,7,8]	0[]
		…			
52	$last([A]) \rightarrow A$	6.0	-101.0	3 [1,2,3]	0[]

Set of rules generated $R \in \mathcal{R}$

state (s)			action (a)									q
Φ_1	Φ_2	Φ_3	o	φ_1	φ_2	φ_3	φ_4	φ_5	φ_6	φ_7	φ_8	
-125.0	14.84	1	1	10.0	0.125	0.0	0	2	2	1	0	1
					…							
-125.0	14.84	1	6	19.0	0.125	0.0	0	5	2	1	0	1
-125.0	14.84	1	1	23.61	0.0	0.0	0	4	4	2	1	0.45
-127.42	15.62	1	6	7.0	0.125	0.0	1	2	2	1	0	0.85
-126.23	14.42	1	4	35.0	0.0	0.0	0	9	3	2	0	0.2
					…							
-135.18	16.09	1	2	11.72	0.625	0.0	3	0	3	1	1	0.88
					…							
-127.66	15.92	1	5	6.0	0.375	0.0	2	0	2	1	0	0.90

Matrix Q

Figure 5.5: After initialising the set of rules R and the Q-matrix, *in each following learning step ($t = i + 1$), gErl selects an action (a_t) (from Figure 5.4) and performing it thus generating a possible new rule that will be added to R. The* Q-matrix *is also updated with this new information (current state abstract definition ϕ_i, operator selected o and rule abstract definition φ_j). This process will continue until the until the* Stop Criterion *is reached.*

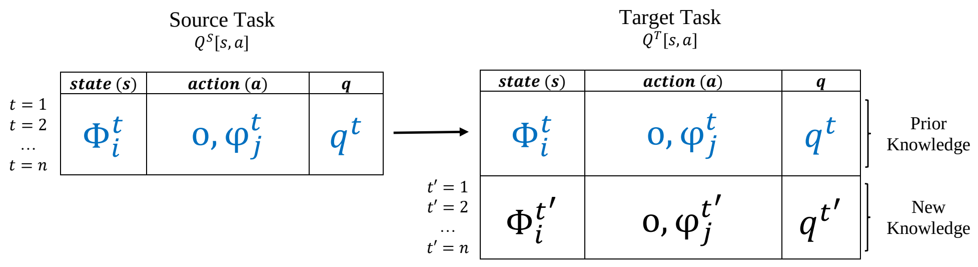


Figure 5.6: The knowledge transfer process in the gErl system where t represents the learning step and $i \in 1..3$ and $j \in 1..8$ are, respectively, the set of features indexes used to describe the states S and rules R

which are successfully applied to certain states (from the previous task) are also applied when a similar (with similar features) new state is reached. Due to this abstract representation, how different the source and target task are does not matter.

The knowledge transfer between two task (source S and target T respectively) is performed as follows: when gErl reaches the solution of a given problem (or it executes a maximum number of steps), the table Q^S (which has been filled in by the model Q_M^S and equation 5.8) is copied and transferred to a new situation. Concretely, when gErl learns the new task, Q^S is used to train a new model Q_M^T[2]. Therefore, Q^S is used from the first learning step and it is afterwards updated with the new information assimilated by the model Q_M^T. Figure 5.6 briefly describes the process of reusing a Q^S table and how it becomes a new Q^T table.

Further information showing the usefulness of the policy reuse strategy can be found in [MFHR13d, MFHR13a, MFHR13c].

5.4 Summary

The increasing interest in learning from complex data has led to a more integrated view of this area, where the same (or similar) techniques are used for a wide range of problems using different data and pattern representations. This general view has not been accompanied by general systems that address a large variety of problems representations. Most systems need to be modi-

[2]We do not transfer the Q_M^S model since it may not have been retrained with the last information added to the table Q^S (because of the periodicity of training, the generation of the latest model may not match with the stopping criterion, so there may be a bunch of information in Q which is not used to retrain Q_M^S).

fied (or became completely useless) when the original data representation and structure change. Some other approaches require the data to be flattened, sliced or migrated to hyperspaces of unintelligible fictitious features. In fact, the most general learning approach can still be found in ILP [MLPTN14] (or the more general area of inductive programming). However, each system is still specific to a set of embedded operators and heuristics.

In this chapter, we have shown that more general systems can be constructed by not only giving power to data and background knowledge representation but also to a flexible operator redefinition and the reuse of heuristics across problems and systems. This flexibility also carries a computational cost. In order to address this issue we rely on two (compatible) mechanisms: the definition of customised operators, depending on the data structures and problem at hand, done by the user, using a language for expressing operators. The other mechanism is the use of generalised heuristics, since the use of different operators precludes the system from using specialised heuristics for each of them. The choice of the right pair of operator and rule has been reframed as a decision process, as a *reinforcement learning* problem.

The definition of operators is a difficult issue, and it requires some expertise and knowledge of the functional language used to express them. Even so, the definition of heuristics is a more difficult issue that cannot be assigned to users. Therefore, not only is this a novel approach but also allow us to better understand the role of operators and heuristics in learning systems. Due to the fact that gErl provides us inspectable and intelligible knowledge (symbolic nature) as well as its already mentioned versatility, this will allow us to, as commented in the introduction, use our system as an appropriate cognitive tool to analyse several general intelligence problems in the context of cognitive development. gErl thus could help us to makes it explicitly how complex each pattern is and what constructs (learning operators) are used for each problem providing useful information about the role of the cognitive operational constructs that are needed to solve these intelligence tests. We will further discuss this in Chapter 7.

Finally, it is worth commenting that our work has several limitations. Constructing new operators is not always easy and requires expert knowledge. Despite providing some facilities (meta-operators), the inference of powerful hypotheses requires the ability of coding complex abstracts constructs as well as the use of auxiliary concepts in the background knowledge. Furthermore, we are assuming the number of examples to be a small number, otherwise the system becomes very inefficient. This is common in other inductive programming approaches where, similarly to humans learning, these systems have important limitations on working memory and the size of the data (although humans are exceptional for perception mechanisms such as vision, speech or music). Complex hypotheses can be only constructed over previously derived or existing concepts [Mil56] and the background knowledge should thus be maintained, refined and developed in the way it is applied to new problems. This has been the motivation of our incremental and lifelong learning view of knowledge acquisition that we will discuss in Chapter 8.

6 Intelligent test tasks for AI evaluation

In the early days of artificial intelligence, the intelligence test approach was considered useful not only as a tool for the study of cognitive processes and the development of new techniques, but also for the evaluation of AI systems or even as the goal for AI research. Since then, human psychometric tests have been repeatedly suggested as a much better alternative to most task-oriented evaluation approaches in AI. The question thus is whether this measurement of mental developmental capabilities lead to a feasible, practical evaluation for AI. In this in this chapter we make a revision of what has been done when intelligence test problems have been analysed through cognitive models or particular systems. We make a general account of all these works in terms of how they relate to each other and what their real achievements are. Also, we provide some insight about what intelligence tests measure in machines, whether they are useful to evaluate AI systems, whether they are really challenging problems, and whether they are useful to understand (human) intelligence. Overall, the ultimate goal of the chapter is to understand the meaning, utility, and impact of these computer models taking intelligence tests, and explore the progress and implications of this area of research.

This chapter is organised as follows. The current state of research of computational models aimed at solving intelligence test problems motivates our study in Section 6.1. In Section 6.2, some of the most common types of intelligence test problems, and what they are supposed to measure in humans, are summarised, with a more detailed exposition of each problem in the appendix. In Section 6.3, we will go chronologically through all the computer models. The results will be summarised in a table at the end of this Section. A technical analysis of the systems and the relation with the kinds of problems solved is more deeply examined in section 6.4. From the table and the technical analysis, in Section 6.5 we will address a series of questions including a discussion about the insights and implications we can infer from this study. Finally, some possible directions and perspectives of integration are discussed in Section 6.6.

These results have been published in [HMS+16].

6.1 Introduction

Artificial intelligence is typically defined as "the scientific understanding of the mechanisms underlying thought and intelligent behaviour and their embodiment in machines"[1]. Associated with the notion of natural and artificial intelligent agents is our (human-centred or anthropocentric) belief that intelligence underlies most human behaviour. The origin of AI research was based on the conviction that "intelligence is the computational part of the ability to achieve goals in the world" [McC07].

Indeed, AI research can claim some impressive milestones as we saw in the introduction of this thesis. For example, already in 1959 Arthur Samuel presented a self-learning program that could play checkers (or draughts) [Sam59]. In 2002 the 1957 prophecy of Herbert Simon that within 10 years a computer would be world's chess champion (eventually) came true when Deep Blue won against the human chess champion Garry Kasparov [CHH02]. In 2010 IBM's program Watson [FBCC+10, FLB+13] was the winner of the *Jeopardy!* TV quiz. However, one can ask whether the mechanisms underlying the behaviour of AI systems are the same as or similar to the mechanism underlying human intelligent behaviour. In fact, the success in specialised tasks is a very illustrative demonstration of the *big switch*[2] approach in AI research. If we fix the switch to one particular problem, we can devise, after several years or decades of research, a system that performs better than humans. We can even embed many specialised programs into a system and devise an automated switch. For instance, if we have a good program for checkers, a good program for chess, etc., we can devise a meta-system that is able to recognise which kind of game has to be played and switch to the appropriate program. If AI evaluation is based on specific benchmarks that are known beforehand, non-intelligent systems will be able to thrive. Game playing is a good example where a reaction against this specialisation is beginning to flourish. Since 2005 the

[1]See homepage of the Association for the Advancement of Artificial Intelligence, http://www.aaai.org.

[2]The so-called *big switch hypothesis* [EN69] was postulated as a way of achieving systems that could show performance on more complex tasks —and even on a range of tasks—, by integrating several specialised modules [Min88].

performance of game playing systems is evaluated in the game playing competition [GLP05] on a wide variety of games—some invented ones not disclosed to the participants until the competition. Consequently, in the area of game playing, systems using a big switch approach are hardly successful in contrast to systems realising general game playing algorithms.

While successfully playing games can be seen as a special manifestation of intelligent behaviour, intelligence tests assess the underlying ability to act intelligently in many different domains [Ste00]. In psychology research the classical approach to intelligence assessment is to apply psychometric tests measuring intelligence [Ste00]. Some of these tests, the so-called IQ tests, are standardised in such a way that humans can be classified as below, about, or above average intelligence. Nonetheless, there are many other human intelligence tests and cognitive tests that also measure intelligence. In addition, other similar tests from other areas were not originally targeted to humans. In what follows, for simplicity, we will use the term *intelligence tests* for all of them. The intelligence test tasks address a variety of reasoning abilities, for example, solving number series problems, detecting regularities in spatial configurations, or understanding verbal analogies. Some types of problems are rather independent of the subject's educational and cultural background, others depend on background knowledge.

As briefly commented in Chapter 1, in early AI research, the intelligence test approach was considered as a useful approach for AI programs as well; Newell argued that one of the ways artificial intelligence could be achieved was "to construct a single program that would take a standard intelligence test, say the WAIS or the Stanford-Binet" [New73b]. And indeed, as early as 1963, Evans devised an AI program that could solve geometric analogy tasks from the WAIS (Wechsler Adult Intelligence Scale) test [Eva63, Eva65] and, in the same year, Simon and Kotovsky [SK63] presented a program that could solve Thurstone letter series completion problems [TT41]. Both types of problems address the ability to identify regularities in patterns and generalise over them. This connection between inductive inference and intelligence tests was also identified early on, for instance by Blum and Blum [BB75]: "Intelligence tests occasionally require the extrapolation of an effective sequence".

After the initial interest of AI research in intelligence test problems, this branch of research sank into oblivion during the next decades. However, in the 1990s, cognitive science research recovered this line of research, and cognitive models were proposed to simulate the human cognitive processes that take place when solving inductive inference in intelligence test problems [CJS90].

In AI, forty years after the work of Evans and Simon & Kotovsky, in 2003, computer programs solving intelligence tests became of interest again.

On one hand, Sanghi and Dowe [SD03] wanted to make a conclusive point about how easy it was to make non-intelligent machines pass intelligence tests. On the other hand, Bringsjord and Schimanski aimed at resuscitating the role of psychometric tests—including not only intelligence tests but also tests about personality, artistic creativity, etc.—in AI [BS03]. They claimed that psychometric tests should not be dismissed but placed at a definitional, major role for what artificial intelligence is and proposed "psychometric artificial intelligence" (PAI) as a direction of research.

But the fact is that the past ten (and especially five) years have been bloomed with computational models aimed at solving intelligence test problems. The diversity of goals and approaches has also widened, including the use of intelligence tests for the analysis of what intelligence is, for the understanding of certain aspects of human cognition, for the evaluation of some AI techniques or systems, including robots, and, simply, to have more insights about what intelligence tests really represent. We will use the term *computer model* for all these approaches, independently of their purpose, of the employed techniques, and of the range of problems they are able to address.

In the current state of research, there exist many systems addressing different intelligence tests. Currently, there is no general framework to characterise all intelligence tests. Therefore, it is unclear whether instances of problems are members of the same or different problem classes. With the exception of those systems solving several problems, like the one of Sanghi and Dowe [SD03], current systems are based on algorithmic approaches specifically designed to solve a special class of such problems and even only one specific test. It is an open question whether a general algorithmic approach for a diverse set of intelligence tests can be designed in principle. Therefore, the composition of an inventory of problems and the proposal of some criteria to compare the different systems is a crucial step. We will be as inclusive as possible in the purpose of these systems and the area or research question that motivated the study, be it psychology, artificial intelligence, cognitive science, or robotics. We analyse all the computer models taking intelligence tests (or as many as we could find, about thirty in total), starting with Evans's ANALOGY [Eva63, Eva65] and going through to Spaun [ESC+12], a noteworthy 2.5-million-neuron artificial model brain that has received considerable interest [Yon12, Mac12]. This review will be the gateway for the posterior analysis of our own AI system gErl solving intelligence tests.

This analysis will also help us to have a better understanding of the relevance and connections of these approaches, and draw some conclusions about their usefulness. Furthermore, in order pursue the matter further, in Chapter 7 we will analyse the role of the cognitive operational constructs that are needed to solve some intelligence tests through our general-purpose learning system gErl. Overall, the main goal of the chapter is to understand the meaning, utility, and impact of these computer models taking intelligence tests, and explore the progress and implications of this area of research.

6.2 Intelligence tests tasks and similar problems

Intelligence is a trait that has been recognised in several degrees and qualities in humans, and to a lesser extent, other animals. While there are many ways of recognising, identifying, and ultimately measuring intelligence of humans, non-human animals and machines, in this chapter we will focus on computer models addressing intelligence test problems. Particularly, we will briefly review some of the tasks that appear in intelligence tests and similar tests. We refer to some of these tests as 'similar' to intelligence test because those tests do not originate from psychometrics but from AI or cognitive science, and have never been used to evaluate humans systematically. Nonetheless, they are similar (intentionally or not) to human intelligence tests, and we think it is worth taking them into consideration. Basically, the criteria for inclusion in this chapter are (1) that the tests have been developed as part of human intelligence tests or tests of mental abilities, or have been introduced in the context of cognitive systems addressing aspects of human intelligence, and (2) that the tests have been attempted by at least one computer model, as we will see in Section 6.3.

One of the motivations behind the advent of psychometrics in the late XIXth century was the common confusion between 'idiots savants' and genera lly-able individuals [CS08]. Tests to measure intelligence in human children and adults consolidated in psychology in the first half of the last century. One common psychometric approach to the evaluation of intelligence is to determine the intelligence quotient (IQ)[3], as a score obtained in a standardised test that quantifies the intelligence of a person [Ste00]. However, there are many intelligence tests that are not used for IQ estimation.

Intelligence tests are typically based on a factor analytical model with subtests for different abilities. There is no complete consensus about the number of factors and, most especially, how they are related. However, there is a certain level of agreement on the existence of specific and general factors, and also on the distinction between knowledge-independent abilities and those that require the use of knowledge (and, of course, language). In this regard, one important notion is the distinction between *fluid* and *crystallised* intelligence

[3]Psychometrics not only measures abilities (cognitive skills), but also traits (personality) and attitudes (social views and opinions).

[Cat63]. Fluid intelligence refers to the capacity of reasoning and solving new problems, with a limited use of previously acquired knowledge. Crystallised intelligence, on the contrary, refers to the capacity of applying previous knowledge to new problems. It is important to clarify that crystallised intelligence is not the same as specialised knowledge. For instance, an 'idiot savant' is not generally able to use knowledge to generalise and relate concepts to new problems.

One inherent characteristic of intelligence tests is that they are composed of items with variable item difficulty. Item difficulty is determined by the percentage of subjects that are able to solve this item, using functional models as in Item Response Theory [Lor80, ER00]. However, this cannot explain why one item is more difficult than another. For such an explanation, the investigated system has to be modelled as an information processing system as proposed by Newell and Simon [NS63]. Taking this perspective, item difficulty can be either explained completely independent of humans by means of algorithmic information theory (AIT) [4] or based on the complexity of the cognitive processes and representations necessary to solve a test item[5]. This will be further investigated in Chapter 7.

Well known intelligence tests are Raven's Progressive Matrices (RPM), the German Intelligenz-Struktur-Test 2000 (IST-2000), and the Wechsler Intelligence Scales for Adults, School and Preschool Children (WAIS, WISC, WPPSI). There are computer models dealing with Raven's Standard Progressive Matrices (denoted by SPM), number series (Numb-S) which are present in different standard intelligence tests, such as IST-2000, verbal common-sense reasoning problems (WPPSI), and block design problems (WAIS-B). Furthermore, different tests for mental abilities were investigated with computer models: Thurstone letter series completion task (Lett-S) based on Thurstone's Theory of Primary Mental Abilities (PMA), geometric analogies (ACE-A) in American Council of Education (ACE) tests, Odd-one-out problems (OOO, used nowadays in many test batteries[6]), Bennett mechanical comprehension tests (BMCT), word analogies (SAT-A) from the Scholastic Assessment Test (SAT), and Montessori's object matching (Mont-O). Problems introduced in

[4]By using notions such as Solomonoff universal probability [Sol64a], Kolmogorov complexity [LV08a], and Wallace's Minimum Message Length (MML) [WB68b, WD99a], intelligence is seen as a special kind of information processing tool (able to determine the theoretical difficulty of any task) that can be defined and evaluated using mathematical constructs.

[5]Cognitive approaches are not meant to build systems that are able to perform well (or perfectly) in some tasks (or even in intelligence test problems), but rather that perform in the same way humans do, solving problems that humans usually solve but also failing at problems where humans fail.

[6]Such as http://www.cambridgebrainsciences.com/

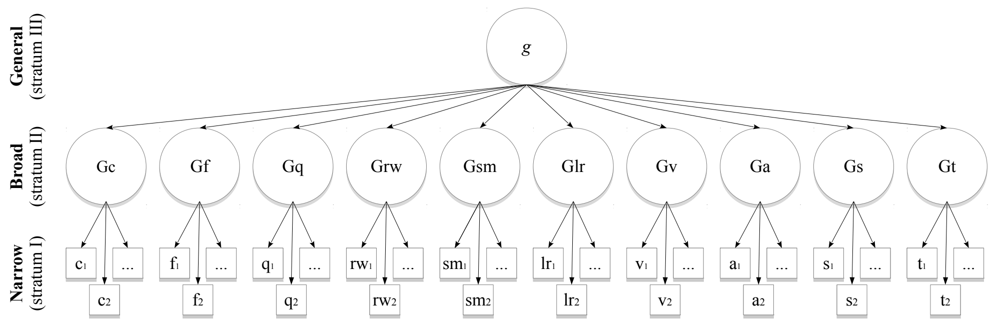


Figure 6.1: Cattell-Horn-Carroll's three stratum model. The top level represents the g-factor (a construct that is usually derived from a factorial analysis of the abilities) which is usually associated to the idea of 'general intelligence', the middle level identifies a set of broad abilities and the bottom level may include many narrow abilities. The broad abilities (second level) are Crystallised Intelligence (Gc), Fluid Intelligence (Gf), Quantitative Reasoning (Gq), Reading and Writing Ability (Grw), Short-Term Memory (Gsm), Long-Term Storage and Retrieval (Glr), Visual Processing (Gv), Auditory Processing (Ga), Processing Speed (Gs), and Decision/Reaction Time/Speed (Gt). The first level can include many more 'narrow' abilities or types of tasks.

the context of cognitive systems are Bongard's analogy problems (Bong-A), string analogies (Str-A), and Hofstadter's anagrams (Jumbles). More information on these tests is given in the appendix B. These tests represent the selection of problems covered by the computer models discussed in this chapter. We intentionally leave out related work where specific problems included in school or college tests are addressed [IGIS13, HHEK14, SHFE14, KAZB14], because the tests are not intended to evaluate general intelligence.

The selection of which tests have been attempted by computer models and which tests have not is most informative. For instance, it seems pointless to apply a memory test to a computer model, even if this kind of test is usually included in many human intelligence test batteries. Also, some verbal tests, especially those about story completion, have only been attempted by a computer model very recently because they are really challenging. Another important factor that explains why some tasks have not been used in artificial intelligence or cognitive science is due to the restricted availability of many psychometric tests. The reason behind this is mostly to prevent the task instances from becoming public, as this would allow humans to prepare and *specialise* for the tests. A third factor influencing the selection of tests is the degree and complexity of required previous knowledge. Usually tasks that are very abstract and do not require previous knowledge (aiming at measuring fluid intelligence) are preferred in the context of computer models. However, there are also computer models addressing more knowledge-intensive tests (aiming at measuring crystallised intelligence). Finally, some tests issue new editions regularly, such as the WAIS (e.g., WAIS-IV was released in 2008), and the definition of items and tasks is hence more volatile.

Because of this volatility and their extensional definition by a set of instances, the tasks that we consider are not easy to separate or characterise by a single criterion, as there is a high degree of overlaping in the processes and abilities that each of the following tasks involve. It is more convenient to analyse the abilities that each test covers. Table 6.1 shows a list of mental abilities on the rows and an identifier for each test task on the columns. We follow the list of primary mental abilities as described in [Sch10], which is mostly based on Thurstone's Mental Abilities (PMA) theory [Thu38, TT41] (see Figure 6.1). Even if there is no consensus on a list of abilities or factors, this list includes "the most important factors, in order of the proportion of individual differences explained" [Sch10]. We exclude *Associate Memory and Perceptual Speed*, as memory and speed are not very interesting from the point of view of machines. We include *Deductive Reasoning* as it was originally present in Thurstone's PMA theory.

Ability	Lett-S	Numb-S	ACE-A	SPM	WAIS-B	OOO	Bong-A	Str-A	Mont-O	Jumbles	WPPSI	BMCT	SAT-A
Verbal Comprehension											×	×	×
Spatial Orientation			×	×	×	×	×		×			×	
Inductive Reasoning	×	×	×	×	×	×	×	×	×	×			×
Number Facility		×											
Word Fluency										×	×		×
Deductive Reasoning												×	

Table 6.1: Correspondence between the tasks given in the appendix B (using the task identifiers as they appear in each subsection heading) and the mental abilities they measure according to [Sch10].

From the point of view of artificial intelligence, the list of mental abilities in Table 6.1 can be mapped to the areas in artificial intelligence. For instance, Table 6.2 shows a list of AI subdisciplines[7]. Our understanding of "knowledge representation" is not just mere representation, which obviously affects all problems, but the storage and retrieval of *previous* knowledge from knowledge bases (linguistic, common-sense, rules, constructs, etc.) and its application for new problems.

[7]These are taken from the list of topics of the Artificial Intelligence Journal (AIJ), excluding "AI and Philosophy", "Cognitive aspects of AI", "Heuristic search", "Intelligent interfaces", and "Intelligent robotics" for being metalevel, hybrid, or instrumental.

One of the first things that we observe in these two tables is that inductive reasoning is predominant in Table 6.1 while deductive reasoning is predominant in Table 6.2.

AI area	Verbal	Spatial	Inductive	Number	Word	Deductive
Automated reasoning and (deductive) reasoning						×
Commonsense reasoning	×	×				×
Constraint processing				×		×
Computer vision (and perception)		×				
Knowledge representation	×				×	×
Machine Learning			×			
Multiagent systems						
Natural Language Processing	×		×		×	
Planning and theories of action						×
Reasoning under uncertainty or imprecision						×

Table 6.2: Approximate correspondence between the mental abilities as shown in Table 6.1 and main areas in AI. The area of multiagent systems does not find a match in the list of abilities, as social abilities are not in the list. Constraint processing may only be related to the Number Facility ability when dealing with numeric constraints.

A detailed discussion of the computer models that attempt to solve the types of tasks that are found in intelligence tests seen in this section will be presented in Sections 6.3 and 6.4.

6.3 Account of computer models solving intelligence test problems

In this Section we shortly describe all the computer models that have addressed intelligence tests and related tests following a chronological order. The systems we will review address the tasks introduced in section 6.2 (Table 6.1). In order to get a comprehensive insight of many different approaches spanning over five decades, we will identify some criteria that are crucial in this understanding: year of development, range of tasks solved, purpose or intention of the study, techniques used by each model, representation of the data, performance compared to humans, kind of difficulty assessment derived from the model. These criteria are identified to characterise all the computer models we will analyse in sections 6.3 and 6.4 in a most informative way, covering the facts about what, when, why, and how, as we can see in Table 6.3. The purpose of this section is to focus on the main features and achievements for all of these models, trying to determine the values of each of them for this criteria. The values for the criteria will be summarised in Table 6.3 (where we see a row for each model (or a group of models) and the value for each criterion) and discussed in Section 6.5. Further discussion about the technical details of and the connections between these models will be given in Section 6.4. Finally, we try to put into perspective the comparison (in performance) of all these computer models.

6.3.1 Early systems: 1961–1991

The relation between artificial intelligence and psychometrics started more than fifty years ago with Evans [Eva63, Eva65] and his program ANALOGY, which was "capable of solving a wide class of the so-called 'geometric-analogy' problems ("A is to B as C is to ?") frequently encountered on intelligence tests" [Eva65] such as the ACE Tests. Evans's ANALOGY analyses the geometric figures in terms of intersections, decompositions, similarities, transformations (rotation, scaling), etc., using pattern matching.

At least for this seminal paper, it is interesting to know the reasons why Evans considered intelligence test problems were appropriate for the construction of heuristic problem-solving programs. He thought that the choice of geometric-analogy problems was suitable because: (i) "problems of this type require elaborate processing of complex line drawings", (ii) "the form of problems, [...] more speculative, [...] presents an interesting paradigm of 'reasoning by analogy' ", and (iii) "problems of this type are widely regarded as requiring a considerable degree of intelligence for their solution and in fact are used as a touchstone of intelligence in various intelligence tests" [Eva65]. The intention was then to better understand some principles of analogy and its presentation, and the problems were converted from visual to symbolic. Evans did not say that these tests could be taken as a sufficient or a necessary condition for intelligence, just that "this suggests a non-trivial aspect of any attempt to mechanize their solution". In fact, when he compared his results to humans', no mention is made of this being interpreted as ANALOGY showing intelligence.

Evans's program ANALOGY was highly acclaimed and cited in the following years, but not because of its results being comparable to humans (as this was probably considered irrelevant or anecdotal). Some people (e.g., Solomonoff [Sol66]) suggested that ANALOGY could be extended to cover a much wider range of problems, but the approach was never continued or extended.

A different, but almost simultaneous, approach to Evans's was taken by Simon and Kotovsky [SK63]. They chose "Thurstone letter series completion" tasks. The goal of the research was to understand how humans solved these kinds of problems and their difficulty, through the use of a computer model. In order to make a system that would capture the patterns, Simon and Kotovsky formulated a simple procedure for pattern descriptions using IPL-V, the Newell's information processing language V [New61]. Their work included a number of different pattern generator variants conceived to solve the series with different levels of performance (variants A to D, becoming progressively more challenging). One of the programs (variant D) was able to score better than 10 of 12 human subjects on 15 problems of the Thurstone Letter Series Completion [SK63, Table 3]. Similarly to Evans, this work did not claim that these results could lead to seeing these programs becoming closer to human intelligence.

Despite the relevance of these works, it took almost two decades to see more models. With the major goal of understanding analogy, Hofstadter developed a series of computational models, Jumbo [Hof83], and others, in the Copycat project [HM84]. Hofstadter considered analogy as key to recognition and categorisation, namely, the core of "high-level perception" and creative thought. The project and tasks were inspired by the Bongard problems. However, because of their visual difficulty, the problems were simplified and remade in a microdomain with a symbolic representation whose basic elements are letters and strings of letters: jumbles (anagrams from a given set of letters) and letter sequences analogies. The cognitive architecture and ideas used therein were very specific and generally unrelated to other techniques in AI. The architecture follows a biological metaphor in its development by using different constituent elements (long-term memory, short-term memory, and subcognitive processing mechanisms) which interplay in parallel in order to generate obvious or (in cases) creative solutions. The results were not fully compared with those of humans.

Carpenter et al. [CJS90] addressed Raven's Progressive Matrices (RPM) [RCR92]. Yet again, the goal was to better understand human intelligence and the nature of the tests. The general outline of how the model performs is divided into three main categories: (a) *perceptual analysis*, where the model encodes the information about the figures of the first pair of entries of each row, determines the correspondences between their attributes and the figures in the remaining entry are compared to obtain a pattern of pairwise similarities and differences; (b) *conceptual analysis*, induction and generalisation of the rules that account for the variation among the figures and attributes in each of the first two rows. This process is incremental and the rules are induced one by one; (c) *response generation*, the rules for the top two rows are generalised and applied to the third row to generate the solution. Carpenter et al. analysed the rules needed to solve the RPMs and identified five relations:

"constant in a row", "distribution of three values", "quantitative pairwise progression", "figure addition", and "distribution of two values". Strictly, this is not a big switch approach, but shows that the system would not work for a similar problem with different relations. As an indirect result and based on the previous relations, they produced "a pair (FAIRAVEN and BETTERAVEN) of computer simulation models that performed like the median or best college students in the sample", which were based on the analysis of eye fixations, verbal protocols, error patterns, and the identification of the previous rules.

As the reasons to choose RPM, Carpenter et al. say: "the correlation between Raven test scores and measures of intellectual achievement suggests that the underlying processes may be general, rather than specific to this one test" [CJS90]. Again, it is not argued or mentioned that these models are intelligent. Carpenter et al.'s approach does not handle re-representation or encoding of the problem. Nonetheless, for the first time, one of these cognitive models is generalised as a theory of intelligence on its own, but without any explicit reference that this could be used for machines or for artificial intelligence.

Simon et al. [SKN91] presented the cognitive computer model SC-Soar to solve series completion tasks based on the ideas of Simon and Kotovsky [SK63]. Their system was an adaptation of the cognitive architecture SOAR. In particular, SC-Soar solved letter series completion tasks by casting them as comprehension tasks so as to find relations that characterise the series. The system was able to solve ten out of fifteen series in the original set [SK63]. Although they did not make any comparison with humans, they argued that the number of decision cycles (measure of duration) that the system took to obtain a right solution provided a measure of complexity of these series that could be closely related to the difficulty of these series for the subjects. The authors also claimed that the five series SC-Soar cannot solve were the most difficult.

6.3.2 Later systems: 2003–2009

A decade later, Bringsjord and Schimanski refreshed the interest of psychometric tests in artificial intelligence [BS03]. In fact, they started by wondering "what went wrong" with Evans's ANALOGY program, as it should have been "the first system in a long-standing, comprehensive research program", but, after forty years, they found themselves "trying to start that very program" [BS03]. One of their explanations was based on the distinction between a narrow view of intelligence tests (Spearman's general intelligence g [Spe04], as illustrated by Raven's Progressive Matrices [RCR92]) in front of a broader view of intelligence tests (Thurstone's view of a range of problems [Thu38], as in WAIS [Wec81]). As an example of how to deal with WAIS (at least partially), they introduced PERI, whose name stands for "Psychometric Experimental Robotic Intelligence". PERI is based on cognitive robotics and makes no claim of being a cognitive model. It uses brute force for problem solving and space search. While it is specialised to one kind of problems (Block Design), Bringsjord and Schimanski argued that PERI was not only able to solve the particular Block Design problems in the WAIS, but any Block Design puzzle given to it.

Simultaneously with Bringsjord and Schimanski, Sanghi and Dowe [BS03, SD03] developed a computer program (without any supporting cognitive model) which was used to make a clear point about the relevance of machines passing intelligence tests. A third year undergraduate student (Sanghi) produced an obviously non-intelligent small program written in Perl which was able to pass some specific intelligence tests featuring number, letter, and word series. The authors claimed that some other kinds of more difficult intelligence tests (involving matrices, pictures, and questions) could be passed in the same way (using predefined patterns), but they were not implemented in the program. The experiment was conclusive, as nobody would assign intelligence—not even the smallest degree—to this program. This system is a clear example of the big switch approach. The code was full of *if-then-else* and *case* instructions trying to identify what kind of problem they were facing and delegate it to the appropriate module. As a result, the analysis of this work has to be done in terms of the specialisation of the techniques that were used, their performance and the range of problems, hence raising important doubts about the use of intelligence tests for evaluating AI systems.

The two previous (opposed) approaches [BS03, SD03] for solving intelligence tests led to an increasing number of works in the area (although some of them were possibly unaware of these opposed views, as they do not cite Bringsjord and Schimanski's paper or Sanghi and Dowe's paper). For instance, Tomai et al. [TLFU05] revisits Evans's problems but using a more abstract and general approach, based on general-purpose simulation models: sKEA [FU02] and the *Structure-Mapping Engine* (SME) [FFG89]. Tomai et al.'s goal was to show that through the generality of their system, they were able to solve a set of classic visual analogy problems. The results on the twenty problems used by Evans were similar to those of Evans.

A different attempt to address a psychometric test with the aim of understanding humans' cognitive mechanisms was Phaeaco [Fou06], which focussed on Bongard problems [Bon70]. Phaeaco uses a cognitive architecture with an image processing module, pattern matching techniques, long-term memory, and learning mechanisms with the aim to tell how similar such figures are and trying to emulate how humans solve these problems (hardwired mechanisms and holistic and analytic views). Phaeaco is able to solve some Bongard problems. Overall, its performance was shown to be slightly worse than humans.

Similar to Tomai et al., Lovett et al. addressed three visual problem-solving tasks: geometric analogies [LTFU09, LF12], Raven's Progressive Matrices [LFU07, LFU10], and odd-one-out intelligence tests [LLF08, LF11]. With a major goal of modeling human cognition, their models provide novel insights about which cognitive operations are easier or harder for human visual problem-solving. Lovett et al. demonstrated that qualitative spatial representations extracted by using CogSketch [FUL+08] (their sketch understanding system) and visual comparisons made by the structure mapping engine (SME) can be used to solve geometric analogy problems [TLFU05, LTFU09, LF12]. The authors combine these two models with two complementary theories of how people perform it: (1) *visual inference*, where the answer of a problem is inferred applying the differences found between the images "A" and "B" over the image "C" in order to obtain the solution "D"; and (b), *second-order comparison*, where the differences found between the images "A" and "B" are then compared to the differences between "C" and each possible answer "i", selecting the most similar one.

In order to address RPMs [LFU07, LFU10], Lovett et al. used SME (without using any processes specifically designed for the task) in a two-stage mapping process, also using CogSketch and a series of strategies based on structure-mapping techniques which are used to map the different elements in the matrix in order to obtain patterns of structural relationships (similarities and dissimilarities). Lovett et al. claimed that their model overcame the limitations of Carpenter et al.'s model [CJS90], using visual representations and task-general processes. Regarding the results, the first system scored like an average American adult, while an improved version scored "above [...] most adults" [LFU10].

Finally, in [LLF08, LF11], Lovett et al. addressed odd-one-out problems in the same way as the previous models: using SME with the analogical generalisation module known as SEQL. The results were slightly better than those of American adults [LF11, Table 3]. While these comparisons could be indicative of the achievements of these techniques, and one of the most general approaches to intelligence tests to date, we note that the goal of these works was to construct a model to better understand item difficulty.

6.3.3 Recent explosion: 2010–today

A surge of new systems has taken place since 2010. For instance, a different approach has been started by Sinapov and Sotytchev [SS10], where an intelligence test is taken by a robot in a rich sensorimotor scenario. The robot's aim was to solve personalised odd-one-out tasks. The tests were performed on six natural object categories (pop cans, plastic cups, metal objects, empty bottles, soft objects, and objects with contents) where a group of four objects (three of the same category and one outside the category) is presented. Cognitively, the model uses similarities ect. The results were not compared to humans, but the robot obtained results better than chance for all six object categories.

Another attempt to address Raven's Progressive Matrices was undertaken by McGreggor, Kunda, and Goel [MKG10, KMG10, KMG12, KMG13]. Unlike previous models focussing exclusively on modal propositional accounts, they proposed two complementary approaches that use purely iconic visual representations of test inputs (with different level of resolution): the "fractal" and an "affine" method. Both methods are used to judge the similarity between images and induce all possible transformations (selecting the best one) so as to predict an answer that will be compared to each given answer choice. The affine and fractal models were tested on all problems from the Raven's Standard and Advanced Progressive Matrices tests. Even with this original visual representation, the performance was shown to be similar to the human norm. McGreggor and Goel also applied the fractal approach to the odd-one-out problem in [MG11a, MG11b], where the system automatically adjusts the level of the resolution of the images and increases (also automatically) the complexity of the relationships (from simpler to higher-order) to resolve ambiguity. Their approach was evaluated against 2976 randomly selected odd-one-out problems from a large corpus with a span of difficulty from the very easiest (level one) up to the most difficult (level 20). From them, 1647 problems were solved and, although the performance was not compared with humans, we think that its results must be below human performance. The main goal of these works was to evaluate whether visual analogy problems (RPMs and odd-one-out problems) could be solved using visual representations.

A special journal issue on psychometric artificial intelligence [Bri11] triggered more computer models. For instance, Klenk et al. [KFTK11] challenged a new kind of test, Bennett's Mechanical Comprehension Tests [Ben69], with important content about physics and contextual information. Klenk et al. also used sKEA and SME (as [TLFU05] and [LFU07]), integrated in (and extending) the Companion Cognitive Architecture [FG97], in order to address these problems. Unlike the previous approaches, they did not make any comparison with humans.

Turney [Tur11] introduced the system PairClass as an improvement of other systems by the same author. PairClass is a system for analogy perception that recognises lexical proportional analogies (A:B::C:D, meaning “A is to B as C is to D” but with words) by using word frequency-based features vectors (searching in a corpus) and some supervised machine learning algorithms to classify word pairs. PairClass is able to solve word analogies found in the SAT college entrance exam, TOEFL (test of English as a foreign language) synonyms, ESL (English as a second language) synonyms, some synonym and antonym problems previously used in computational linguistics, similar/associated/both word pairs, and noun-modifier relations. Although only the SAT college entrance exam resembles an intelligence test (with specific knowledge), we think that the other tests are also valid as human-level tests and show that PairClass can address many kinds of linguistic problems provided it has access to a corpus.

Ruiz [Rui11] addresses the odd-one-out problems. The aim of this work was to show that a simple two-step algorithm can help to understand the relationships between symbols and the dissociation/association process (based on symbols and sets) in this kind of intelligence test problem. The results were comparable to those of humans. This has to be interpreted carefully, as instead of re-representation, the system performed a hand translation (RASCM). Also, the ad-hoc use of a Hamming distance (to find most frequent symbols) suggests that the system would need to be changed dramatically for other similar problems.

Ragni and Klein [RK11] worked on number series completion, which is very common in many intelligence tests and also occasionally in artificial intelligence [Bur05a]. They took over 50,000 number series from the Online Encyclopedia of Integer Sequences (OEIS) and applied a general method using artificial neural networks (ANNs) and dynamic learning. Although the overall results were comparable to those of humans, the error distribution was very different, probably because the way in which the problems were solved is very different to the way humans solve them.

Ragni and Neubert [RN12, RN14] presented another system for Raven’s Progressive Matrices, implemented in the cognitive architecture ACT-R. The system requires the identification of relational rules, as Carpenter did. In particular, Ragni and Neubert use five rules: constant in a row, distribution of three values, quantitative pairwise progression, figure addition, and distribution of two values. Unlike the previous work [RK11], the motivation of this work is to solve the problems in a cognitive way, i.e., explaining why the system fails (ambiguousness, objects neglected, incorrect rules applied, . . .)

	Model	Years	Range	Intention	Techniques	Performance	Difficulty
EvansGEO	Evans's ANALOGY [Eva63, Eva65]	1961–65	geometric analogy*	AI principles	own rules	global	no
Sim+LET1	Simon & Kotovsky [SK63]	1963	letter series	human cog.	list-processing	itemwise	itemwise
HofJUMBO	Hofstadter's JUMBO [Hof83, HM84]	1983	word jumbles	AI principles	own cog. arch.	no	no
HofCOPYC	Hofstadter's COPYCAT [Hof83, HM84]	1984	let. seq. analogy	AI principles	own cog. arch.	no	no
Carp+RPM	Carpenter et al. [CJS90]	1990	Raven's PMs*	human cog.	production-system	itemwise	itemwise
Sim+LET2	Simon et al. [SKN91]	1991	letter series	human cog.	own rules	global	itemwise
Bri+PERI	Bringsjord & Schimanski's PERI [BS03]	2003	psychomotor WAIS blocks*	AI evaluation	robot + own rules + NLP	global	no
San+PERL	Sanghi & Dowe [SD03]	2003	several	philosophical	own rules	global	no
Toma+GEO	Tomai et al. [TLFU05]	2005	geometric analogy	AI principles	sketches + SME	global	no
FouPHAEA	Foundalis's PHAEACO [Fou06]	2006	Bongard probs.	human cog.	own rules	itemwise	no
Lov+RPM	Lovett et al. [LFU07, LFU10]	2007–10	Raven's PMs	human cog.	sketches + SME + SEQL	itemwise	itemwise
Lov+OOO	Lovett et al. [LLF08, LF11]	2008–10	odd-one-out	human cog.	sketches + SME + SEQL	itemwise	itemwise
Lov+GEO	Lovett et al. [LTFU09, LF12]	2009–12	geometric analogy	human cog.	sketches + SME	itemwise	itemwise
Sin+ROOO	Sinapov et al. [SS10]	2010	psychomotor odd-one-out	AI principles	robot + similarities	no	itemwise
McG+RPMf	McGreggor et al. [MKG10, KMG10, KMG12]	2010–12	Raven's PMs	AI principles	similarities + fractal	global	no
McG+OOO	McGreggor & Goel [MG11a]	2011	odd-one-out	AI principles	similarities	global	no
Kle+MECH	Klenk et al. [KFTK11]	2011	Bennett's Mech.	AI evaluation	sketches + SME	no	no
TurPAIRC	Turney's PairClass [Tur11]	2011	word analogy/synonyms	AI principles	corpus + SVM + RBF	no	no
RuizOOO	Ruiz [Rui11]	2011	odd-one-out*	human cog.	similarities + clusters	itemwise	itemwise
Rag+NUMS	Ragni & Klein [RK11]	2011	number series	AI principles	ANN	global	no
Rag+RPM	Ragni & Neubert [RN12, RN14]	2012	Raven's PMs*	human cog.	ACT-R	global	itemwise
Bay+WORD	Bayoudh et al. [BPG12]	2012	word analogy	AI principles	corpus + complexity	no	no
Pra+RPMp	Prade et al. (pixels) [PR11, CPR12, PR14]	2012	Raven's PMs	AI principles	similarities	global	no
Pra+RPMf	Prade et al. (features) [PR11, CPR12, PR14]	2012	Raven's PMs*	AI principles	similarities	global	no
Eli+SPAU	Eliasmith et al.'s SPAUN [ESC$^+$12]	2012	series and analogies*	human cog.	ANN	itemwise	no
Sie+NUMS	Siebers & Schmid [SS12b]	2012	number series	AI principles	own rules	no	no
Sch+ROBO	Schenck et al. [Sch13]	2012–13	psychomotor, several	AI principles	robot + similarities	no	itemwise
McG+RPMa	McGreggor et al. (affine) [KMG12, KMG13]	2012–13	Raven's PMs	AI principles	similarities	global	no
Pra+OOO	Prade et al. [PR13, PR14]	2013	odd-one-out*	AI principles	similarities	no	no
Str+ASOL	Strannegård et al.'s ASolver[SAU13]	2013	number series	AI principles	ind. prog. (compression)	global	no
Str+SSOL	Strannegård et al.'s SeqSolver [SNSE13]	2013	number series	AI principles	ind. prog. (compression)	global	no
Str+RPM	Strannegård et al. RPMs [SCS13]	2013	Raven's PMs*	AI principles	own rules	itemwise	no
Ohl+CNET	Ohlsson et al.'s ConceptNet4 [OSTU13]	2013	verbal WPPSI	AI evaluation	NLP + knowledge base	global	no
Hof+NUMS	Kitzelmann & Schmid's IGOR [HKS14]	2014	number series	AI principles	ind. prog.	itemwise	itemwise

Table 6.3: Taxonomy of computer models solving intelligence tests. All systems are automated. The categories are the development or publication 'Years', the 'Range' of problems (an asterisk shows that the 'representation' has been transformed), the authors' 'Intention' with the model, the 'Techniques' that were used, the comparison with human 'Performance', and the analysis of item 'Difficulty'.

and why some problems are more difficult than others (number and type of rules applied, number of calls to the declarative memory, objects stored, ...). Considering the results obtained, the authors claim that the visual complexity involved in solving this kind of problems is smaller than the functional difficulty (rules to be applied). The results were compared to humans in terms of correlations for the accuracy (subjects' error rates) and to Carpenter's BETTERAVEN (where a human comparative study was made), with similar (or slightly better) score, so we can estimate the results to be around human average.

Closely related to the approach of Turney we find the work of Bayoudh et al. [BPG12], where a feature vector similar to Turney's is used to represent the concepts but with completely different semantics. The main idea is that each word in an analogical proportion problem carries an "information content" that can be formally defined via its Kolmogorov complexity K. To evaluate their approach, they used a set of 147 pairs of words coming from the SAT college entrance exam that are to be completed with another pair of words to be selected among 5 options, performing slightly better than a pure random choice.

Prade, Richard, and Correa [PR11, CPR12, PR14] developed a logical representation of the notion of "analogical proportion" (i.e., analogies of the form "A is to B as C is to D"). This logical view tries to depict how similar (*homogeneous proportions*) or dissimilar (*heterogeneous proportions*) the items in an analogical proportion are by representing them using binary or multiple-valued features. Following this perception of similarity, the authors developed a system for addressing Raven's Progressive Matrices based on solving analogical proportion equations using an extended scheme where proportion $(a, b) : f(a, b) :: (c, d) : f(c, d)$ holds for lines and proportion $(a, b) : g(a, b) :: (c, d) : g(c, d)$ for columns. Their approach may be applicable at different levels of representation: feature-based approach (propositional representation of the matrices) and pixel-based approach (bitmaps). For both approaches a simple algorithm finds both horizontal and vertical patterns (analogical proportions) between the Boolean values of the features or pixels of the pictures. The authors mainly compared their work with those which use *Structure-Mapping Engine* (SME) approaches [LFU07] claiming the simplicity and generality of their approach. The results for the pixel-based approach were worse than those for the feature-based approach (because their algorithm is unable to provide the information needed to get a proper solution of several tests), the latter being about human average.

Following the previous perception of dissimilarity between items in a formal setting of logical proportions, Prade and Richard [PR13, PR14] addressed odd-one-out problems. By using the same Boolean representation and equation-solving principle, the idea is to find the item which has most dissimilarities with regard to the rest of items for each feature: when a *homogeneous proportion* equation holds between four Boolean values, there is no intruder among them, however, if heterogeneous proportions equations hold for an item but not for the rest, an intruder is found. This approach was not evaluated.

With quite a different perspective, Eliasmith et al. [ESC$^+$12] have recently produced a 2.5-million-neuron artificial model brain (called Spaun, short for Semantic Pointer Architecture Unified Network). Since Spaun is highlighted by its functionality, attention has been drawn to its having a similar ability on certain aptitude test questions to what might be found in some humans [Yon12, Mac12]. Spaun is evaluated against eight types of problems (A0–A7). A7 is "fluid reasoning", which is arguably said to be "isomorphic to the induction problems from the Raven's Progressive Matrices (RPM) test for fluid intelligence. [...] This task requires completing patterns of the form: 1 2 3; 5 6 7; 3 4 ?" [ESC$^+$12]. Using a "match-adjusted success rate" (since Spaun must generate the correct answer, not choosing from a set of answers), and comparing to humans, the authors conclude that "Spaun is similarly good at these tasks" [ESC$^+$12].

Siebers and Schmid [SS12b] address the number series problem with a cognitive model, differently to other previous approaches for the same problem introduced before [RK11, Bur05a]. They use their own (public) series collection formed by 25,000 randomly created number series (using the basic numerical operators), for which the accuracy was 93.2%. No comparisons with humans were made.

Schenck et al., in a series of papers [SS12a, SSS12] and in Schenck's Master thesis [Sch13], address a series of tasks found in intelligence tests with an upper-torso humanoid robot performing a set of stereotyped exploratory behaviours and recording sensorimotor feedback, as in Sinapov's paper [SS10]. Three different types of problems were tried. First, some Montessori tasks [SS12a] [Sch13, chapter 4] were included in the experiments with sound cylinders, weight cylinders, pressure cylinders, and sound boxes. Second, order completion problems [SSS12] [Sch13, chapter 5] were arranged and three tasks were prepared: ordering by weight, ordering by compliance (black and white categories) and height. They posed 150 order completion tasks to the robot (3 sets of objects, 50 tasks per set). Third, matrix completion problems were arranged [Sch13, chapter 6]. They were similar to simple RPMs, where size, colour, and content were used. "The robot was presented with a set of objects arranged in a grid, with the lower-right object missing, and with a set of candidate objects". They posed "500 randomly generated matrix reasoning tasks to the robot".

Strannegård et al. [SAU13] address the number series problem by developing ASolver, a Haskell-coded anthropomorphic cognitive system based on the idea of limited working memory. The main strategy is as follows: (a) construct a term-based language that describes number sequences $(1, 2, 3, 4, \ldots$ is described as $f(n-1)+1)$; (b) define a term rewriting system to compute mathematical terms; (c) define bounded cognitive resources in terms of computations and size; (d) look for the smallest term that computes the input sequence. This model was not intended as a cognitive model but only for the purpose of problem solving. Nevertheless, it has some relations to human cognition. The same problem was undertaken by Strannegård et al. [SNSE13] with SeqSolver, which, following the same pattern discovery as in [SAU13], is based on Kolmogorov complexity for limiting the set of computations. ASolver was tested on the 11 number sequence problems of the IQ test PJP [SSF06], obtaining scores above IQ 130–140. On the other hand, SeqSolver was tested on number series from IST [ABLB01] and obtained scores of at least IQ 130.

Closely related and simultaneously, Strannegård et al. [SCS13] address Raven's progressive matrices with an anthropomorphic cognitive model in the sense that it uses certain problem solving strategies that were reported by high-achieving human solvers. Some principles are different to those of [SAU13] but the system is also based on a repertoire of patterns that are combined to reach the solution. To solve the matrices, the system relies on pattern matching and the use of the repertoire of patterns similar to those of Carpenter et al. [CJS90] (identity, one of each, numeric progression, translation, AND, OR, XOR). The system program was tested on the sets C, D, and E of Raven's Standard Progressive Matrices test and produced correct solutions for 28 of the 36 problems considered. The results are said to be roughly comparable to an IQ of 100.

The approach presented by Ohlsson et al. [OSTU13] is one of the few approaches to verbal intelligence tests. In particular, the authors proposed the use of the Wechsler Preschool and Primary Scale of Intelligence (WPPSI-III, Third Edition) to evaluate common-sense AI systems. Their method is based on the capabilities available in the commonsense knowledgebase and natural-language-processing toolkit *ConceptNet* [Sin01], an open-source project run by the MIT Common Sense Computing Initiative. The system obtained results which were comparable to results obtained by an average 4-year-old (which does not mean that the system has the verbal abilities of a 4-year-old).

Hofmann, Kitzelmann, and Schmid [HKS14] demonstrate that the inductive programming [FS08, GHOK+15b] system IGOR2 can be applied to number series problems. IGOR is a system for learning (recursive) functional programs from input/output examples, which has been applied to many problem solving domains [SK11]. The authors present an empirical evaluation with 100 number series systematically varied along three dimensions: size of starting number (small/large), type of operator ($+, \times$), and structural complexity. The latter involves variations with respect to reference (last number, second last number, third last number), linear vs. tree recursion (as for factorial vs. Fibonacci), and usage of one or more operators. These dimensions were identified from a psychological study of the difficulty of number series problems by Holzmann et al. [HPG82]. Although IGOR2 is an AI system, the authors propose that IGOR2 can also be viewed as a plausible cognitive approach to learning complex rules.

6.3.4 Comparison between systems

There are two types of comparisons that can be made with these systems. First, we can perform a *bona fide* comparison using the data from the associated papers. For instance, from all the systems in Table 6.3, there is only sufficient data in the papers to compare the *results* for those addressing RPMs. This is shown in Table 6.4. The insight from this table is narrow, for many different reasons. First, many of the systems were not designed to achieve good performance but to study the problems or the human performance on them. Secondly, they do not use the same subsets (the Set II of APM is the one for which we can compare the largest number of systems: just 5). This is sometimes caused by confidentiality agreements and copyrights. Thirdly, many systems could have been overfitted for the subset of items they used, and the results may not generalise to other items. Fourthly, we do not have information in most cases about the results for particular items, so a system may be successful on the most difficult ones, or vice versa.

A more meaningful analysis could be performed if we were able to analyse more systems and use different item sets. In order to determine whether this comparison could be possible, we performed a survey (a series of questionnaires sent by email to the systems' authors) in order to determine the availability of the systems and the input format the systems use. We complemented this information with the set of problems these systems can handle. The result is given in Figure 6.5, where we show that systems can only be compared for isolated clusters (given by the problems they can handle). An additional difficulty has been the public availability of some models, thus making their comparison impossible. Furthermore, an extra challenge lies in the input representation. Note, for instance, that solid links between systems solving RPMs (and also for those solving odd-one-out problems) indicate that, although almost all of

Model	Raven's SPM					Raven's APM	
	A (12)	B (12)	C (12)	D (12)	E (12)	Set I (12)	Set II (36)
Carp+RPM (FAIRAVEN)	-	-	-	-	-	7[a]	16[b]
Carp+RPM (BETTERAVEN)	-	-	-	-	-	7[a]	25[b]
Lov+RPM			44[c]			-	-
Rag+RPM	-	-	12	12	11	-	31
McG+RPMf	11	7	5	7	2	12	26
McG+RPMa[d]	12	11	8	1	3	7	14
Pra+RPMp	-	-	-	-	-	-	16
Pra+RPMf	-	-	-	-	-	-	16 + 16[e]
Str+RPM	-	-	-	-	-	7	16

Table 6.4: Comparison of systems addressing RPMs using the results given in the corresponding papers. The identifier (Model) corresponds to the first column in Table 6.3 Many of these systems did not have high accuracy as a goal or used different transformations of the instances. Notes: [a]Out of 7 attempted. [b]Out of 27 attempted. [c]From sets B to E. [d]Results shown for the standard *configuration. [e]The pixel-based approach was able to solve 16 problems and the feature-based approach solved 16 additional problems.*

them use propositional input representations, the dimension, number, type, syntax, and semantics of the features used vary for each model. This makes it extremely difficult to ensure equivalence between the input problems to be solved in each model. Therefore, we see that a comparison of several systems for a range of problems is not possible. It seems that apart from the RPM cluster in Figure 6.5 (for which we have already discussed the comparison in Table 6.4 there are a few models that could be compared for number series, only three of them (Rag+NUMS, San+PERL, and Sie+NUMS) share a similar representation (raw numbers with minor variations). However, there is no point in making a comparison between these three systems since their aims were completely different: the goal of Sanghi and Dowe [SD03] was not to excel in any specific task, the aim of Ragni and Klein [RK11] was to successfully solve large sets of number series from OEIS, and the goal of Siebers and Schmid [SS12b] was to get cognitive plausibility for series actually found in intelligence tests (including failure on those difficult series for which humans fail) instead of excelling on performance.

As a result, the comparison would become much more meaningful with systems that are able to handle several kinds of problems. An effort must be made towards a standardised format for problem description and the availability of a large variety of instances—so that system cannot prepare for a particular set.

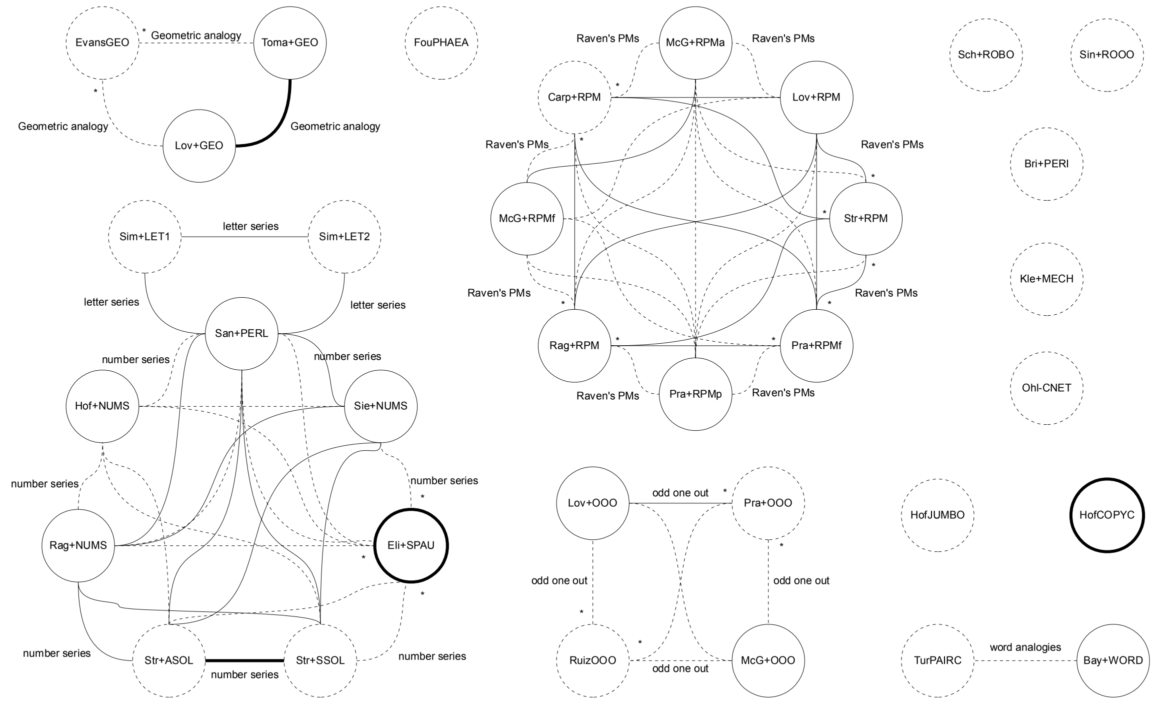


Table 6.5: Complete account of all systems and whether they can be compared and to which extent. The identifier inside the circles corresponds to the first column in Table 6.3. Dashed circles indicate that the code is not available. Solid circles show availability on demand (thin line) or publicly (thick line). Dashed edges mean that both systems are comparable in principle but they use different representations. Solid edges correspond to similar representation (thick solid means that the format is compatible). An asterisk is shown when there is a (handcrafted) preprocessing transformation on the representation. We see that problems determine the clusters, as most systems are specialised for just one problem. Note that thick solid links correspond to approaches developed (and evolved) by (almost) the same authors.

6.4 Technical analysis

In this section we present a more detailed study of the techniques that are used by the systems introduced in the previous section. We will group the systems by the techniques they used. Furthermore, we will show the evolution of the techniques and their progress and whether some kinds of techniques are more appropriate for some types of problems. Our analysis is arranged in two parts: (1) how the representation of the input problems is addressed, and (2) which techniques are used to compute the solutions.

6.4.1 Input representation techniques

Regarding the input problem's representation we find two big different approaches: (a) symbolic representations (hand-coded or not) and (b) visual representations (pixel-based or analogical visual representations). Former systems like Evans's ANALOGY [Eva63, Eva65] or those developed by Carpenter et al. [CJS90] relied on hand-coded symbolic descriptions of geometric analogies and RPMs. In particular, Evans's system made a complex preprocessing of the input problems decomposing each hand-made figure description (coordinates) into very precise internal symmetry descriptions (LISP-based representations): both a specified set of geometrical analogy relations (inside, above, . . .) and a set of "Euclidean similarity" calculations (rotations, uniform scale changes, and certain types of reflections), which were determined for each and every pair of objects by using a substantial repertoire of "analytic geometry" routines and a topological pattern matching process. By contrast, more recent works, such as those of Tomai et al. [TLFU05] and Lovett et al. [LTFU09, LF12, LLF08, LF11], are based on the process of forming high-level conceptual representations from raw data (without hand-coding). These approaches exploit qualitative visual structure computing abilities from sketching programs such as sKEA [FU02] or its successor *CogSketch* [FUL$^+$08]. Both are general-purpose architectures for conceptual sketch understanding (visual and spatial properties) that allow the generation of representations from raw input and provide a convenient platform for cognitive experiments that use visual stimuli. Specifically, in *CogSketch* the user has to label the input objects (*glyphs*) and *CogSketch* uses them to compute spatial relations (e.g., edges, groups of objects, relative position, topology, overlapping) which can lead to inferences about the spatial relationships between the content of those glyphs.

Notwithstanding, many modern computational systems keep assuming that hand-coded representations are available as inputs. Systems such as those of Ragni and Neubert [RN12, RN14] and Strannegård et al. [SCS13] continue addressing Raven's progressive matrices by generating hand-coded propositional attribute-value vectors with, respectively, a fixed set of attributes (shape, size, number of sides, width, height, colour, rotation, position, and quantity); and *vector graphics* where each matrix is encoded as vectors of attribute-value pairs using different abstraction levels (features, elements, groups, and cells) by using the XML-based specification language from Microsoft, *XAML*.

Prade, Richard, and Correa's approach [PR11, CPR12, PR14] is a special case, applicable to different levels of representation (resolution). One is a feature-based approach, where the RPM pictures are represented as vectors of Boolean features (coded manually), in some way like Ragni and Neubert's, but the features can be different for each problem and can denote, for instance, the presence of a *Big Square*, a *Small Square*, a *Circle*, or a *Cross* in a picture. A pixel-based representation is also used in a number of cases.

Clearly, a big technical advance in intelligence tests' input has been the ability of using low level image representations (bitmaps). Here we find those systems for solving RPMs of Prade, Richard, and Correa's (pixel-based approach) aforementioned [PR11, CPR12, PR14], the system of Foundalis [Fou06] for solving Bongard problems, or the (robotic or not) approaches with video input systems [BS03, SS10, SS12a, SSS12, Sch13, ESC$^+$12]. Also, the fractal and pixel-based computational models from McGreggor, Kunda, and Goel [MKG10, KMG10, KMG12, KMG13, MG11a, MG11b] work directly on visual inputs from both RPMs and odd-one-out problems without any need to extract propositional representations from them. All these techniques, computationally infeasible in previous decades (due to the hardware and software requirements), have shown that they perform surprisingly well in those kind of tasks and, thus, are valid alternatives to the verbal representations.

Finally, note that those systems addressing word, letter, or number-based tasks are able to use raw "strings" or number/letter sequences directly [SK63, SKN91, SD03, RK11, SS12b, SAU13, SNSE13], meanwhile others need specific types for input problems: Hofmann, Kitzelmann, and Schmid's system [HKS14] needs algebraic data-types for the target function together with a small set of examples; Ragni and Klein's system [RK11] and Siebers and Schmid's system [SS12b] need to transform the sequences into different train and test sets.

6.4.2 Computational techniques

A crucial difference that makes systems distinctive is not only the kind of problems they solve, but also the great deal of techniques that have been used for solving the same or different intelligence tests. This correspondence between tasks and kinds of problems is illustrated in Figure 6.2 where the the first seven kinds of techniques are general and can be applied to any type of problem. The bottom three, though, are clearly more restricted to some types of problems. While some categories can be mapped to AI areas seen in Table 6.2, and ultimately to Table 6.1, it is still difficult to tell for a particular problem which kinds of AI techniques may be more suitable. However, from the experience in other areas such as of general game playing, it is clear that the ad hoc "rule-matching" approach does not work if the system is aimed at a range of problems [GB13]. We will be develop the technical details below, attempting to clarify the evolution over the years.

Rule-matching: One of the most used techniques during the early years was what we call *rule-matching* technique. This includes, among others, the use of the authors' own code or the use of production-systems [New73a] with hand-coded production rules. In this category we include those systems such as Evans's ANALOGY, which used a substantial amount of pattern-matching techniques between pairs of figures and heuristic problem-solving mechanisms in order to generate a set of rules transforming figure A into figure B (specifying how the objects of figure A are removed, added to, or altered to generate figure B). These sets of rules are generalised and a similarity matching is carried out between figure C and each of the five answer figures given in the analogy problem to find (if possible) a correct answer. Another example is [CJS90], where Carpenter et al. used their own production system CAPS [JC87] (for Concurrent, Activation-Based Production System), a collection of procedural hand-coded *if-then-else* rules (including the five rules discovered needed to solve the RPMs) specifying what symbolic manipulation should be made when a given information condition arises in working memory. Unlike conventional production systems, (a) CAPS allows for parallel execution of all the productions whose conditions are satisfied, and (b) instead of having knowledge elements either present or absent from working memory, they can have varying degrees of activation. As commented in the previous section, Carpenter et al. produced a pair (FAIRAVEN and BETTERAVEN) of computer simulation models. The BETTERAVEN model differs from FAIRAVEN in two major ways: BETTERAVEN has the ability to induce more abstract relations than FAIRAVEN, and it has the ability to manage a larger set of goals in working memory and hence can solve more complex problems.

Currently, several computational systems still use their own *if-then-else* patterns to pass some specific intelligence tests. This is the case with Sanghi and Dowe's program written in Perl (960 lines of code) [SD03], which, having access to a word-list of approximately 25,000 words and using some predefined *if-then-else* patterns, was able to pass some specific intelligence tests featuring number, letter, and word series. Furthermore, some other systems use hand-coded variations of those patterns found by Carpenter. Strannegård et al. [SCS13] also relied on a repertoire of patterns similar to the one of Carpenter et al. [CJS90], which can operate with one or more abstraction levels—attributes (features), elements, groups, and cells—and can be processed row or column-wise in order to obtain a prediction value for the solution cell.

Cognitive architectures: On the other hand, Ragni and Neubert [RN12, RN14] also relied on the use of some predefined hand-coded rules or patterns for solving RPMs. However, in their case, these rules were implemented using the well-known *cognitive architecture* ACT-R [And96], a cognitive architecture for simulating and understanding human cognition implemented as a production system, where each symbolic processes is controlled by subsymbolic equations (responsible for most learning processes) which estimate (and then decide) the relative cost and benefit associated with their execution. Their system also requires the identification of relational rules, as Carpenter did.

Former systems such as those of Hofstadter also relied on cognitive architectures. In this case, Hofstadter developed its own cognitive architecture in the Copycat project [HM84]. Particularly, *Jumbo* [Hof83] and *Copycat* [HM84] were developed for solving different letter-based intelligence tests and, furthermore, both are composed of three constituent elements following a biological metaphor. Jumbo's architecture, used to make plausible anagrams from a given set of letters, is composed by: the *chunkabet*, storing a database of pondered chunks (small sequences of letters); the *cytoplasm*, modelling the working memory and containing partial associations of letters; and the *coderack*, formed by *codelets* (fragments of code) that run in parallel (based on the current state of the *chunkabet* and *cytoplasm*) and can perform modifications over the structures in the *chunkabet* and *cytoplasm*. On the other hand, Copycat's architecture, which is similar to Jumbo's, was used for solving letter sequence analogies. Its main components are: the *slipnet*, the *workspace*, and the *coderack*. The *slipnet* models the long-term memory in humans by a semantic network composed of nodes that represent permanent concepts about the letter-string world (*sameness*, *leftmost*, *opposite*, ...) and their weighted relations (strength of associations between concepts). In total there are more than 60 such concepts. Both the relevance of the concepts (related to their activation) and the distance of the links between them change dynamically depending on CopyCat's perspective on the given problem. On the other hand, the *workspace* is the site of subcognitive processing activity that is in charge of modelling the short-term memory where partial structures can be formed (single letters, descriptions, groups, bonds, and bridges). These are temporary combinations built up entirely from *Slipnet* concepts, e.g., *"jjfg"* may be described as the *sameness* group *"jj"* and the *successor group*, consisting of the letters *"f"* and *"g"*. The third main component of the architecture is the *coderack*, which is equal to Jumbo's but based on the current state of the *slipnet* and *workspace*. *Codelets* in the *coderack* examine in parallel the letters of an analogy problem trying to build a coherent set of structures around them and possibly chunking them together into groups based on a common relationship, representing a particular interpretation of the problem. Since analogy problems can have many interpretations (giving rise to a vast space of potential configurations in the *workspace*), Copycat runs according to the *parallel terraced scan*—introduced by Jumbo—that executes several probabilistically selected possible processes in parallel.

Simon et al. [SKN91] used the cognitive computer model SC-Soar, an adaptation of the cognitive architecture SOAR [LNR87] for solving letter series completion. Based on a production system, the SOAR architecture formulates the task in a *problem space* in which different operators are selectively applied to bring the system gradually closer to its goal state. In each decision cycle SOAR brings different pieces of knowledge from its long term recognition memory to the working memory and decides which action to be taken. In particular, SC-Soar used letter series completion tasks and solved them by casting them as comprehension tasks: comprehension operators are applied in order to find relationships between items reaching different states (which encode the knowledge about the relations that characterise the series) for a goal state.

Foundalis's Phaeaco [Fou06] is another cognitive architecture for visual pattern recognition and abstraction. Phaeaco, which creates abstract representations of geometric figures received as input (pixels), uses three mechanisms for solving Bongard's problems. The first mechanism is the *hardwired recall*, related to those problems that can be solved by means of mechanisms hardwired in the human brain. Phaeaco employs a mechanism ("zero variance"), very different from the one humans use, as a first attempt to solve a Bongard problem. If this first stage fails, Phaeaco enters its *holistic view*, related to the initial strategy of conducting a panoramic overview of the problem for a brief period of time in an attempt to see apparent differences. Finally, if the previous stages fail, the systems performs the *analytic view*, where individual images are selected and reexamined in an effort to come up with fresh ideas due to Phaeaco's nondeterministic nature.

Simulation models: Notwithstanding the previous techniques, the best-known approach to analogy-making is the use of *simulation models* and, in particular, the use of the *Structure-mapping Engine* (SME) [FFG89]. SME is a program for analysing analogical processing which compares the similarities and differences between objects. SME has been a significant advance facilitating a lightweight high-level visual matching solving analogy problems. The SME is based on Gentner's structure-mapping theory [Gen83] and operates over symbolic representations (entities, attributes, and relations) and thus, it is used together with sketch understanding systems (such as sKea or CogSketch) using the qualitative spatial representations returned by the latter. SME takes as input two propositional descriptions, a *base* and a *target*, and produces a set of mappings between the pictures aligning their common relational structure. Each mapping consist of: 1) correspondences linking items in the base and target (commonalities in the representations and corresponding objects in two pictures); 2) a structural evaluation score which provides an indication of match quality (thus giving an overall similarity between the pictures); and 3) *candidate inferences* which identify particular differences between pictures. Furthermore, SME accepts input constraints to bias the mappings.

This combination of sketch understanding systems and a structure-mapping engine has been used for solving a wide range of intelligence tests: Tomai et al. [TLFU05] revisited Evans's geometric analogy problems. Lovett et al. addressed geometric analogies [LTFU09, LF12], Raven's Progressive Matrices [LFU07, LFU10] and odd-one-out problems [LLF08, LF11]. For solving both RPMs and odd-one-out problems, Lovett's models also perform a generalisation of the patterns founded using SEQL [KFGQ00], a model of analogical generalisation (describing what is common between two structural representations of objects) through a process of *progressive abstraction* [GL02] built upon SME. The generalisation can then be compared to new objects: each given solution in a RPM problem is inserted into the matrix, returning the one with the closest matching structural relationship, or, each individual image in an odd-one-out problem can be compared to the generalisation thus returning the noticeably least similar as the solution. Furthermore, since SME automatically figures out what kinds of things can be matched [FGMF98], this combination of qualitative representations and structure-mapping engine has also been used (all together with the *Companion Cognitive Architecture* to perform qualitative reasoning) to solve Bennett's Mechanical Comprehension Tests [KFTK11], demonstrating its generality.

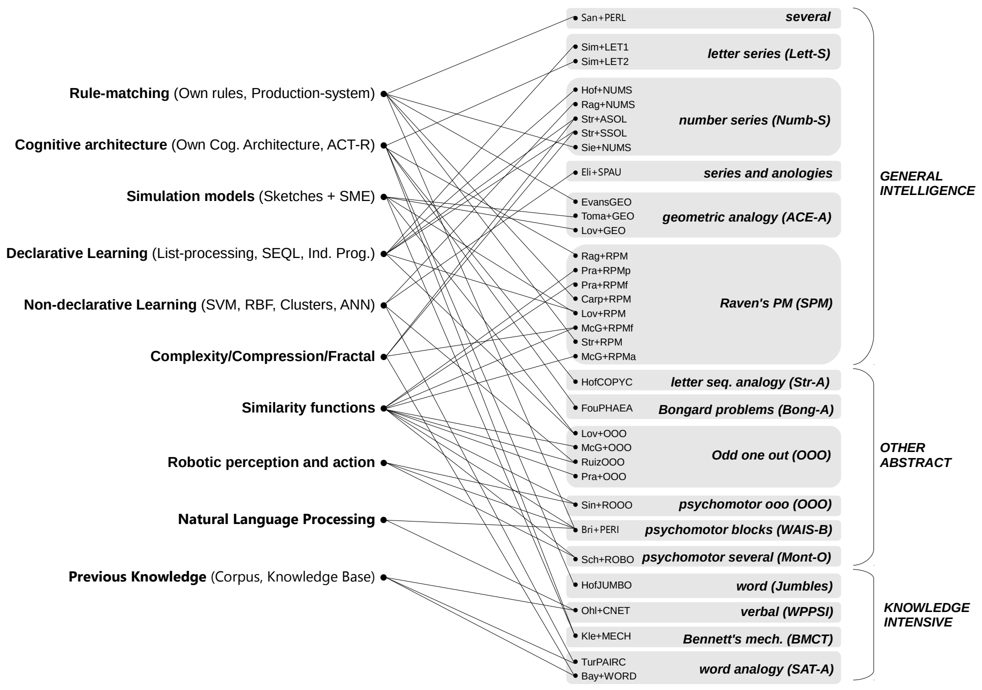


Figure 6.2: Correspondence between kinds of techniques and problems. Compare with tables 6.1 and 6.2. The id of the computer models can be found on the first column of Table 6.3.

Declarative Learning: Techniques related to *declarative learning* such as list processing mechanisms or inductive programming [FS08, GHOK$^+$15b] have also been widely used for solving letter and number series intelligence tests. Simon and Kotovsky [SK63] proposed a list processing procedure to induce pattern descriptions from segments using IPL-V, Newell's information processing language V [New61]. The main task of this procedure is to seek for initial conditions, periodicity and relations in the given sequence, and to arrange them in the corresponding pattern. Simon and Kotovsky also developed an extremely simple IPL-V based procedure capable of generating sequences from pattern descriptions by executing elementary list processes called for by the descriptions.

Likewise, Siebers and Schmid [SS12b], Strannegård et al. [SAU13, SNSE13] and Hofmann, Kitzelmann and Schmid [HKS14] demonstrate that the number series problem can be addressed with the use of inductive programming and bounded cognitive resources (use of heuristics emulating certain limitations of human cognition). Siebers and Schmid [SS12b] use basic numerical operators (addition, subtraction, division, multiplication, and exponentiation) to construct the patterns that lead to extrapolate the sequences. Following a human-like strategy, the hypothesis formation is guided by an analytical strategy.

Strannegård et al. [SAU13] developed two Haskell-coded anthropomorphic cognitive systems based also on the idea of limited working memory: ASolver and SeqSolver. Both systems have a repertoire of predefined patterns (syntactic expressions defining sequences), mathematical operators ($+$, $-$, $*$, $\div$) and simplifications (odd, even, exception, last) used to describe number sequences; and a limited set of computations with bounded cognitive resources (SeqSolver based on Kolmogorov complexity for limiting the set of computations), that are used to decode patterns (rejecting solution candidates which are too computationally demanding), thus turning them into number sequences.

Hofmann, Kitzelmann, and Schmid [HKS14] used their inductive programming system IGOR2, a system for learning (recursive) functional programs from input/output examples. IGOR2 can be applied to number series problems using different example presentation: a list of the initial elements of the series as input and the next element as output, a position as input and an enumeration of the series up to this position as output, or a position as input and the value at this position as output.

Non-declarative Learning: In addition, some systems use general machine learning techniques, what we have called *non-declarative learning*. In this group of techniques we find the large-scale model of the functioning brain developed by Eliasmith et al. [ESC+12]. It is remarkable that the system learns to recognise the problems (with digit recognition) and also writes the answer with a robotic arm, which gives the system an even more impressive look. In Spaun, the tasks are learnt, which is clearly different to a hardwired big switch approach. Nonetheless, the re-representation that is made here is different to the one needed in many intelligence test tasks with visual presentation, such as RPM, since the components of each item are digits, unlike the original RPMs.

Ragni and Klein [RK11], for their part, worked on number series completion applying a general method using artificial neural networks (ANNs) and dynamic learning where the learning rate, the number of input nodes, the number of hidden nodes, and the number of training iterations are manually varied in order to allow for a comparison of the different ANNs. Like other previous techniques, most models based on non-declarative techniques need to rely on other mechanisms or techniques shown on Figure 6.2. Turney's PairClass [Tur11] uses a standard supervised machine learning algorithm: a sequential minimal optimisation (SMO) support vector machine (SVM) with a radial basis function (RBF) kernel from the machine learning library WEKA to classify word pairs according to their semantic relations. PairClass represents the semantic relations between two words using a high-dimensional feature vector, in which the elements are based on frequencies of patterns in a corpus—consisting of a large corpus of plain text (280GB), gathered by a web crawler and retrieved by the search engine Wumpus [BC05]—obtained using templates.

Ruiz [Rui11] addresses odd-one-out problems, combining a clustering technique with a similarity function. Ruiz uses the so-called "Ruiz-Absolute Scale of Complexity Management" (RASCM) in order to code or discretise the objects as character strings. For instance, a problem composed of two circles and a square is represented as (A;A;B). In order to represent objects with several features, other letters are used for each object, such as (AC;AD;AE;BF;AG), so representing four circles and a square with different sizes. Once this transformation is made, the problems are addressed by a two-step clustering algorithm that computes the average Hamming distance of the sets in the first step, and, for those misclassified items, it makes a second recoding so that the most frequent symbol within that set becomes A, the second most frequent symbol becomes B, and so on. For example, the three following sets: (AAAD, BBBE, CCDE) become respectively: (AAAB, AAAB, AABC),which clearly unveil AABC as the least similar.

Complexity, Compression or Fractals: Among the approaches based on *complexity, compression or fractal* techniques, Bayoudh et al. [BPG12] uses the relationship between Kolmogorov complexity and the universal distribution, and searching on a structured text corpus (i.e., the Web), is able to address word analogies. In more detail, the authors use the Kolmogorov complexity as a measure of the quantity of information needed to go from the word w_1 to the word w_2, namely, to define agreement and disagreement between concepts. The estimation for a given pair of words $<w_1, w_2>$ is calculated as the log inverse of the words' frequencies within a given corpus.

McGreggor, Kunda, and Goel [MKG10, KMG10, KMG12, KMG13, MG11a, MG11b] proposed two different image resolution-based methods (the "fractal" and a "affine" method) for solving RPMs and odd-one-out intelligence test problems. Both methods compare images under a variety of transformations (image rotations and mirrors, scaling, addition, and subtraction), and must judge the similarity between images (based upon features which arise from them) to determine similarity for each possible answer. The main difference is the interpretation of what constitutes a feature in each model: (a) in the fractal method, the representation of an image is a set of fractal codes which compactly describe the geometric alteration and colourisation of fragments (sets of points) of the source image that will transform into the target image, and the features are derived from these representations (which will be used to determine the similarity for each possible answer across all the relationships present in the problem); (b) in the affine method, a feature is defined to be a single greyscale pixel with each pixel associated with a single intensity value. Both approaches induce all possible transformations for the matrix, both row-wise and column-wise and select the best one according to some measure of fitness. The model applies this transformation to the incomplete row/column to predict an answer which will be compared to each given answer choice according to a similarity measure.

Similarity functions: The above combines a fractal approach with similarity. *Similarity functions* have also been in a wide range of computer models. Prade, Richard, and Correa [PR11, CPR12, PR14], following a logical view of similarity, developed a system for addressing Raven's Progressive Matrices based on solving analogical proportion equations. As seen before (Section 6.4.1), their approach may be applicable at different levels of representation. Their approach starts with the truth table of the analogical proportion (i.e., 0000 or 0101 satisfy an analogical proportion, but 1000 or 1011 do not satisfy it), where each Boolean digit represents the absence or not of a property. The equation-solving principle is used to build the missing item D (rather than selecting it from a set of potential candidates) in an incomplete proportion involving A, B, C. Furthermore, following the logical view of dissimilarity and by using the same Boolean representation and equation-solving principle, Prade and Richard [PR13, PR14] developed an approach to address the odd-one-out problem. The idea is to find the item which has most dissimilarities with respect to the rest of items for each feature.

Robotic perception and action: Unimaginable in the early years, we find *robotic* approaches developed to solve several intelligence test problems (such as WAIS Block problems [BS03], Montessori tasks and RPMs [SS12a, Sch13], or odd-one-out problems [SS10]) making use of an upper-torso humanoid robot with auditory, proprioceptive, and visual sensory abilities. Apart from the robotic abilities, other rule-matching or similarity-based techniques are needed. As an example of how to deal with WAIS (at least partially), Bringsjord and Schimanski's PERI [BS03] is capable of logic/reasoning, vision, physical manipulation through a five-degree-of-freedom vertically articulated robotic arm, speech, and hearing. At the core of PERI we find a complex Lisp program and an associated "Scorbot Advanced Control Language Library". Meanwhile, Sinapov and Sotytchev [SS10] present a framework that allows the robot to interact with a set of fifty household objects (cups, bottles, and toys) with the aim of solving personalised odd-one-out tasks. The robot explores each object to detect many of its physical properties (auditory and proprioceptive sensory feedback) to infer the pairwise similarity matrix for the objects that will be used to select the most dissimilar object in the tests. Finally, Schenck et al. [SS12a, SSS12, Sch13] address Montessori tasks and RPMs with an upper-torso humanoid robot, as in Sinapov's paper [SS10]. The robot grounded its representation for the different objects in each task in terms of the auditory and proprioceptive outcomes that they produced in response to a series of exploratory behaviours. Combining information from each sensorimotor context and using similarity measures, the robot was able to estimate the perceptual distance (similarity score) on a given set of objects.

Natural Language Processing: On the other hand, some systems use *natural language processing* methods based on keywords and statistics in order to try to understand basic facts and queries. Systems like the open-source crowd-sourced knowledge base *ConceptNet* [Sin01] support many practical textual-reasoning tasks.

Ohlsson's system [OSTU13] uses ConceptNet tools to map the input words to concepts in the system, Python algorithms to wire the different parts of the system, and the *AnalogySpace*, a concise version of the large common-sense knowledge base of ConceptNet, which is queried to obtain relations (between concepts). For a question such as, "Where can you find a penguin?", their system will query its knowledge base for the words "find" and "penguin".

Previous Knowledge Finally, Turney's PairClass [Tur11], Ohlsson's CNET [OSTU13] and Bayoudh's system seen above use *previous knowledge* (e.g., Corpus or knowledge bases) more intensively, in integration with other techniques.

6.4.3 Progress in techniques

In the case of intelligence test tasks, we have seen in Sections 6.3 and 6.4, that some improvement from the early systems exists but it is not clear how much of it is due to the use of better techniques or just due to more computational power (note that there are no time comparisons between current and previous models). However, we see that only a few systems have reused the same techniques, and in many cases it is because there are common authors. Also, even for those cases where the techniques are similar, the range of tasks solved is different. In fact, some works we have seen in Section 6.3 do not even cite all the relevant literature for the same task, which clearly limits the transfer between systems. The lack of correspondence that we observed between Tables 6.1 and 6.2 and the techniques that we see in Table 6.3 and Figure 6.2 must be an indication that something is wrong here. Of course there may be many interpretations for all this. First, there is a possibility that AI techniques are not appropriate (or too specific) to solve these problems. Second, there is the possibility that many of these works were not able to adapt AI techniques more smoothly. But there is also the possibility that AI has not considered these problems interesting enough, or simply, as mentioned above, that the systems have not been designed in many cases to excel in the results, but to better understand the nature of the problems and how humans solve them. Actually, this was the motivation behind one of the systems, Sanghi and Dowe's, [SD03], which showed that if the mere goal is to score well in these tests, one can do a very simple switch approach (with no AI techniques whatsoever).

Also, if we compare Tables 6.1 and 6.2 we see a mismatch between deduction and induction. This may suggest that inductive reasoning is still the area where more progress is required. In fact, for most approaches the system does not learn to solve the problems but it is programmed to solve the problems. In other words, *the task is hard-coded into the program* and it can be easier to become 'superhuman' in many specific tasks, as happens with chess, draughts, some kinds of planning, and many other tasks. But humans are not programmed to do intelligence tests. Only for a few cases (such as Spaun, but with many limitations and just some kinds of tasks), the system is *trained* to solve intelligence test tasks. We think that this is one of the lessons learnt in AI as stated in the introduction of this thesis (Chapter 1): the ultimate goal of AI is to make systems that learn to solve new tasks they have never seen before.

As a result, there has been limited feedback between the systems, and the progress is caused by the painstaking integration of more methods from AI and some incremental development for some particular techniques. However, the techniques that originated from these systems have had very limited impact on mainstream AI.

6.5 Discussion

After giving a historical overview of the systems in Section 6.3 and a discussion of the techniques of the systems in Section 6.4 we will now discuss some of the key questions raised in the introductory chapter of this thesis (Chapter 1) in order to understand the meaning, usefulness, and impact of those computer models taking intelligence tests. We will focus on the criteria used to characterise the different systems in Table 6.3 that will be crucial in the understanding of what intelligence tests measure in machines, whether they are useful to evaluate AI systems, whether they are really challenging problems, and whether they are useful to understand (human) intelligence.

Firstly, the **years** criterion gives us a chronological perspective that can show trends about the interest in computer models solving intelligence test problems. An overview of the table indicates that most approaches are very recent. Is it an indication of relevance? According to the venues, we see that they go from mainstream AI to cognitive science, or even psychology, and some of them are in leading conferences and journals in these areas or even in interdisciplinary general outlets. According to the **intention** criterion, it seems that most approaches aim at unveiling general (artificial) intelligence principles in ways that are not necessarily connected to the way humans solve these tests. This suggests that this is attracting more interest in artificial intelligence and cognitive science than in psychology.

What about the use of these tests for AI evaluation? Are they becoming more common? It has been recently argued—from human intelligence researchers—that intelligence tests are the right tool to evaluate AI systems. In February 2011, Douglas K. Detterman, the editor-in-chief of *Intelligence* wrote an editorial after seeing how successful IBM's program Watson [FBCC^{+}10] (the recent winner of the *Jeopardy!* TV quiz show at the time) had been (at least as a way to acknowledge the progress of artificial intelligence for lay people). As Watson was clearly unable to do other tasks and clearly non-intelligent, Detterman claimed that AI systems should be better measured by classical intelligence tests. The challenge was set in this way [Det11]: "I, the editorial board of *Intelligence*, and members of the International Society for Intelligence Research will develop a unique battery of intelligence tests that would be administered to that computer and would result in an actual IQ score". It is important to note that Detterman's challenge had two levels: the first level allowed the challenge designers to look at the (kinds of) tests beforehand, and construct their system according to this information, while the second (true) level allowed the committee to use any possible test not previously disclosed to the designer or the system. This is a very appropriate distinction in the context of this chapter, since in the previous sections we have seen systems that could come close to the first level, but there is no attempt that could come close to the second level.

Nonetheless, we do not see that artificial intelligence has changed its evaluation protocols following this increase of models taking intelligence tests (the only exceptions are the works by Sinapov [GSSS12], Sotytchev [SS10], and Schenck et al. [SS12a, SSS12, Sch13] for robots and model brains such as Spaun [ESC+12]). But overall, Detterman's challenge has had almost no impact in AI. We see that these models are becoming more common and widespread as testbeds for experimentation, but not so as regular tools for AI evaluation.

In order for intelligence tests to be useful evaluation tools for AI, several things must be considered. Instead of a collection of problems, a better approach may be a collection of instance generators, by integrating some of the existing ones we saw in Section 6.3 and developing new ones. The collection must be large, in order to avoid the big switch approach applied to a small repertoire (or ranges as in Table 6.3). Moreover, brand-new problems could be generated by the combination of existing ones or by the development of more abstract problem generators (instead of instance generators). Different presentations and difficulty levels should be explored. The categories and overlaps between problems could be assessed via theoretical models, instead of using factor analysis as in psychometrics. In other words, a theoretical alternative to the classification of mental abilities—as presented in Tables 6.1 and 6.2—should be endeavoured (see [HOD10, DHO14].

The **range** specifies the kinds of tests addressed by the models. This criterion is fundamental to understand whether computer models are specialised or general. We can see that computer models usually address one problem and the most common one has been RPMs. An asterisk is shown in this column to indicate that the representation of the problem has been transformed. This is very common for RPMs. Clearly, we need to understand where the difficulties of these problems lie and why some problems are more difficult than others. As we can see on Table 6.3, very few approaches address more than one kind of test. Actually, the more specific a test is the easier it is to develop specific solutions. The key issue is to consider a greater diversity of problems. As we can see on Table 6.3, very few approaches address more than one kind of test. Actually, the more specific a test is the easier it is to develop specific solutions. Likewise, some problems that were very 'challenging' (e.g., chess) for machines are now excelled at by specialised programs. In addition, 'representation' is also crucial, as some problems involving complex pattern recognition (those involving images) need some processing. In fact, the same problem (e.g., RPM) is very challenging if it is presented visually—as it is presented for humans—but it is not so if translated to an appropriate symbolic representation. Nevertheless, we can still specialise a system to do the re-representation for just one task. Diversity and representation are hence crucial to distinguish between specific and general AI systems. The challenging goal should be to develop a single general system that is able to solve *all* of them *as they are*.

The **intention** criterion specifies the purpose of the study: to better understand human cognition, to understand general principles of intelligence, to propose a metric for AI evaluation (i.e., psychometric AI), or to make a philosophical/epistemological question (human cognition, AI principles, AI evaluation, philosophical). This provides key information to truly understand the model achievements. The intention behind the models is usually 'AI principles' or 'human cognition', whereas 'evaluation' is not very common. Some of the works featuring computer models have tried to mimic how humans solve these problems. Nonetheless, many anthropomorphic models have succeeded and failed on the same problem items that humans do, but some other non-anthropomorphic approaches have also shown some degree of coincidence with humans, even if the mechanism to solve the items was completely different. Also, very good results have been obtained with techniques that are clearly different from those that humans use. Interestingly, when a problem is challenging for humans but not so for machines, this is very explanatory. For instance, machines are much better at arithmetic than humans. Nonetheless, we can see this from a different view. As we are using human tests, why are these kinds of problems (and not others) useful to measure human abilities? Why are Raven progressive matrices different from chess or multiplication? The use of intelligence tests for machines gives very insightful information about what intelligence tests measure and what they do not and, ultimately, about what characterises intelligence in humans.

Most interestingly, using computer models for intelligence tests can elucidate the relation between different tasks, as the factorial analysis in psychometrics reflects how abilities (or tasks) are correlated for humans. However, psychometrics says nothing about how abilities are correlated in principle, in a computational way. For instance, psychometrics may have found that the g factor is correlated to many other abilities. But does this happen for machines? Can we implement models with good results on tests measuring g and very poor results on the other tests? This seems to be possible, as some of the results shown in Section 6.3 suggest. In fact, this even sheds more doubts about the validity of comprehensive tests such as WAIS for machines, because they are inspired or derived from the knowledge about factorial analysis of human abilities in the past century. Nonetheless this can also be seen as an opportunity of computer models, AI, and other approaches based on information theory, to help improving intelligence tests.

Overall, some of these models have been useful to provide insights and valuable information about how human cognition works. This is especially the case when there is a coincidence of results between a model and humans even if the model was not conceived to follow exactly what humans do. Nonetheless, a systematic disagreement in results or ability correlations may also be very informative. In fact, these studies can be very useful to better understand what intelligence tests really measure and to better understand the correlations between the abilities found in humans. We will further discuss this in the following chapter.

The **techniques** criterion shows which techniques are used by each model, which is relevant to see the role and progress of AI techniques or other ad-hoc techniques. There is a wide variety in the **techniques** used, from more ad-hoc to more general AI techniques. As was discussed in Section 6.4, many models use specific techniques, either by developing new techniques from scratch or by performing a very particular adaptation of existing techniques. In fact, only a few common AI techniques (see column 'techniques' in Table 6.3) are used, mostly from machine learning, pattern recognition, automated reasoning, and natural language processing (apart from some other more miscellaneous techniques such as fractals and ad-hoc rules). It is interesting that a few cases have developed new techniques. If we look at Table 6.1 and especially Table 6.2, we see a correspondence between AI subdisciplines and cognitive abilities. However, if we look at the models and the techniques through Table 6.3 and Figure 6.5, it seems that there is no clear correspondence between the kind of problem and the techniques from AI that have been used to solve it. There is again more prevalence of techniques related to inductive reasoning, but some specialised techniques have also been used in some problems to circumvent this need. Some approaches (many using their 'own rules') are actually variants of a big switch. We also see that some perception techniques are used for the representation mapping when the problems are originally presented in a visual way. These techniques are of course unrelated to whether the task is inductive or deductive, but are just necessary to process the visual tasks. As a consequence, there is some correspondence between techniques and abilities but the alignment is far from perfect, due to the specialisation, as many systems are still more task-oriented than ability-oriented.

Performance specifies what the kind of comparison with humans was made while **difficulty** specifies the kind of difficulty assessment derived from the model. For these two last criteria, we specify whether it is at the test level or item by item (values can be *no*, *global*, or *itemwise*). There is also a huge diversity in whether performance and difficulty are assessed. We need to be clear that focussing on the overall results of a computer model and comparing them with the results of humans (column 'performance' on Table 6.3) is not very informative about how challenging the problem is. Humans are general-purpose systems and it is not fair to compare them with some systems that are only able to solve one problem—even if the problem comes from an intelligence test. Furthermore, many of these intelligence test problems have been developed for humans, and hence it can be unfair to evaluate AI systems limitations with anthropocentric measures.

Nonetheless, some of the works perform an interesting analysis in terms of difficulty. The purpose is to determine what instances are more difficult, but this is not very related to how challenging the problem is. In fact, focussing on the most difficult problems may even make the system more specialised to the intelligence test task at hand. Some of the previous works have studied whether difficulty is related to the size of the working memory, the size of the pattern, the number of elements that need to be combined or retrieved from background knowledge [SK63, CJS90, SAU13, SNSE13] or the operational constructs needed to solve this problems,as we will also see in the following chapter. These notions of difficulty are much more general and can work independently of the problem and the representation. One way or the other, there seems to be an agreement that there will be an increasing number of machines in the near future which show a range of cognitive abilities, and that we will require evaluation mechanisms for them.

6.6 Summary

In this chapter we take a look at all of the computer models taking IQ tests (about thirty in total), starting with Evans's ANALOGY [Eva63, Eva65] and going through to Spaun [ESC$^+$12]. The analysis in this chapter was motivated by an observed explosion of the number of papers featuring computer models addressing intelligence test problems. We wanted to investigate whether this increase was casual or was motivated by an increasing need of these tests and the computer models solving them. When we began our investigation we soon realised that computer models addressing intelligence tests have different purposes and applications: to advance AI by the use of challenging problems (this is the Psychometric AI approach), to use them for the evaluation of AI systems, to better understand intelligence tests and what they measure (including item difficulty), and, finally, to better understand what (human) intelligence is.

Therefore, the analysis is not restricted to performing a survey of all these models. Through a comprehensive account of the models we derive a set of criteria that help us to characterise each system: their relationships, the range of intelligence test tasks they address, the purpose of the models, how general or specialised these models are, the AI techniques they use in each case, their comparison with human performance, and their evaluation of item difficulty. We aim at understanding the meaning, utility, and impact of these computer models taking intelligence tests, and explore the progress and implications of this area of research. Furthermore, this analysis help us to have a better understanding of the relevance and (the limited) connections of these approaches, and draw some conclusions about their usefulness.

Finally, what has been discussed in this chapter confirms our statement posed in the introduction of this thesis: we have seen that even for supposedly general tasks that are designed for evaluation, many approaches have the (understandable) tendency to specialise to the task and hard-wire parts (or most) of the solution. This is yet another indication of a discipline like AI that is being extremely successful in task-specific applications—from playing chess to driving a car—but yet of limited success in general-purpose systems. In a nutshell, we could say that AI has become a big switch discipline. This problem is being recognised in AI itself, as several benchmarks and competitions are now aiming at more general classes of problems (e.g., the general game playing competition [GLP05] or the proposal of a *Turing championship* [You15]).

7 Concept dependencies of intelligent systems

Apart from assessing several factors of human intelligence, some of the early motivations and applications of intelligence tests was the assessment of the so-called *mental age*. The progression in several cognitive tests for the same subjects at different ages would give very valuable information about their cognitive development and, particularly, about how humans gradually come to acquire, construct, and use knowledge to solve them. We understand this knowledge as specific operational cognitive constructs (sometime referred as *concepts*) defining different mental capabilities. The question which we pose in this chapter is whether the same approach can be used to assess the cognitive development of artificial systems. In particular, we want to know whether the intelligence of an artificial system depends on the acquisition, learning or development of different operational constructs and whether we can use human intelligence tests for this. With this is mind, in this chapter, we address several intelligence tests (IQ tests) problems with our general-purpose learning system gErl (presented in Chapter 5) as a tool to better understand the role of the cognitive operational constructs that are needed to solve these intelligence test problems.

The chapter is organised as follows. After introducing and motivating our approach in Section 7.1, Section 7.2 briefly overviews the use of cognitive tests and their variants (according to age and set of abilities), mostly focusing on the evolution of scores with (mental) age and their relation to human cognitive development. Their application to evaluate artificial cognitive systems is also discussed, and the need for a better assessment of their difficulty in terms of the operational constructs and search space that are involved. Section 7.3 discusses how cognitive tests can be analysed proposing gErl as an appropriate kind of tool for this. Section 7.4 applies gErl to several odd-one-out problems, Raven's progressive matrices and Thurstone letter series, analyses the required cognitive operational constructs and the complexity of the found patterns. Section 7.5 makes an overall analysis of the findings over the previous three problems and its implications in the issue of concept dependencies in cognitive development assessment. Finally, a comprehensive summary closes the chapter in Section 7.6.

These results have been published in [MFHR16].

7.1 Introduction

Humans undergo a cognitive development that starts with important neurological transformations even before birth and lasts during their whole life span. This cognitive development is also accompanied by an uneven variation of a range of cognitive capabilities. In order to assess this, many cognitive tests have been specifically devised for different capabilities and age ranges. In fact, one of the applications of these tests (among many others) is the assessment of children's cognition and learning, in order to spot development problems or to identify specially talented individuals. The notion of 'mental age', for instance, was introduced by one of the fathers of psychometrics, Alfred Binet, to compare retarded children with the normal development of children of their age. Nowadays, the notion of a *single* mental age is more elaborate, as many abilities are known to develop at different age intervals.

As we have already seen in the previous chapter, this idea of using tests, in the form of a set of tasks or exercises, is also becoming more and more common for the evaluation of *artificial* cognitive systems or architectures. For example, in the last decade we have seen many proposals: the *cognitive decathlon* [MJMH07, Mue08, SJT08, CG03], the *staged developmental test* [Kee10], the *intelligence tests for robots* [SS10, Sch13], the so-called *psychometric AI* [BS03, Bri11], the *universal anytime intelligence tests* [HOD10] and others [AL03, Lan11]. Many of them are inspired by human mental development tests such as Kuhlmann's test [Kuh39] and Griffith's mental development scale (a test) "covering the locomotor, personal-social, hearing and speech, eye and hand, and performance areas" [Gri54], and also by intelligence tests.

This approach is becoming now crucial in AI, robotics and cognitive science as the Turing Test [Tur50a] has been left as a philosophical rather than a practical test [HO00a]. Similarly, we also have the more classical evaluation of AI systems using specific testbeds, such as maze problems, games such as chess, pattern recognition problems, robot navigation, etc. However, it is more and more manifest that the success or failure at some specific tasks is not well correlated to the degree of intelligence or cognitive development of a system, a phenomenon that is well illustrated by the problems we can find in CAPTCHAs [vABL04, VAMM$^+$08]. In the end, it is clear that one goal is to solve a specific (application) problem and a very different goal is to devise a cognitive system that can solve many different problems. Interestingly, it is not very likely that such an artificial cognitive system could have a high degree of intelligence from the beginning. In other words, it is *potential* intelligence [HOD13] rather than *actual* intelligence what these systems should have originally.

It is then understandable that cognitive science and artificial intelligence are becoming more interested in tests that evaluate general abilities instead of success on a particular set of tasks. In development robotics "the notion of task-independence" [OK07] is related to "psychometrics" and "general intelligence": "developmental robots shall not be programmed to achieve a prespecified practical task". The choice of intelligence tests to evaluate cognitive development is consistent to one observed phenomenon in human intelligence: many of its underlying abilities increase during childhood and youth, stabilise for several decades and then slowly decline while ageing. However, is this really the consequence of a cognitive development or is it a result of some neurophysiological changes in the brain affecting its efficiency? In fact, once we distinguish between fluid intelligence and crystallised intelligence (Section 6.2), should not fluid intelligence be constant while crystallised intelligence increases through cognitive development?

We can rephrase the previous question more specifically. Are the increasingly better results in IQ tests from infancy to early adulthood explained by the development of new *cognitive operational constructs* that are useful (or even necessary) to solve the tests? Or is it because those taking the tests develop increasing better combinatorial search power? What about artificial systems where the computational power is constant? Is their intelligence expected to remain constant?

In order to shed some light on these questions, we set a parallelism between the concepts of fluid and crystallised intelligence in humans and artificial systems by exploring how a general learning system (without a proper cognitive development) addresses several fluid intelligence tests as a possible way to examine the development of general intelligence in artificial systems. With the goal of getting more insight from this experiment, we will use a learning system that uses intelligible ways of expressing background knowledge (to make the required cognitive operational constructs explicit) and the extracted patterns for each exercise. In particular, we will use the system gErl (Chapter 5) to solve some prototypical fluid intelligence test tasks: odd-one-out problems, Raven's progressive matrices and Thurstone letter series. The presentation of the problems will be set as bare as possible, in order to eliminate the influence of pattern recognition issues that may interfere in our analysis. While gErl is the first general-purpose learning system that is able to score average human results in *several* IQ test tasks[1] (see Chapter 6), we clearly make the point of how misleading a superficial *score* comparison is. Rather, we use the system to better understand what these IQ test tasks measure and what a system —human or artificial— requires to solve them, and whether an inability can be turned into an ability through development. Therefore, we will pay special attention on what makes some of the instances in each category harder than others: is it because the search space is larger or because they need more cognitive constructs? In other words, we distinguish two previously conflated aspects in item difficulty: the mental operational constructs that each task requires and the combinatorial problem of combining these constructs to find the solution.

It is important to note that gErl is not a cognitive model and it is not inspired by how humans solve problems or how their development takes place. It is orthogonal to them and this is precisely what will allow us to determine those features that are dependent on the problem and not on the system. Also, we do not want to show how good or bad gErl is at solving these IQ test problems. Actually, what we want to show is that, through the use of a general learning tool that uses intelligible patterns, we can better understand their difficulty and the role of the cognitive constructs that are required in the process. This is possible because of the generality of the system, as other systems that are made on purpose for solving IQ tests may lead to misleading interpretations as they are relatively easy to create with many built-in constructs and may have a strong bias in their scaling of problem difficulty (as we also saw Chapter 6). From this analysis using a general system for several IQ test problems, we will not settle the question of how much important cognitive development is for scoring better in general intelligence tests for natural and artificial cognitive systems, but we will provide a new perspective and procedure to investigate this question. The take-away message is the understanding of the appropriateness and care of using these and other general intelligence tests to evaluate the mental development of artificial systems.

[1]The system presented by Shangi and Dowe [SD03], although scored well in several IQtests, it is not a general system but an adhoc small program.

7.2 General intelligence evaluation during cognitive development

As human capabilities improve with age, many evaluation procedures use several sets of exercises of different types and difficulty in order to accommodate for the subject's expected cognitive development (such as the Wechsler Intelligence Scale tests [Wec58, KL06] seen in the previous chapter). In this and other tests we have a broad range of abilities, so we can give specific assessments of what components of intelligence are more or less developed in a particular individual, as well as their improvement in time. For instance, the results on verbal tests are expected to improve with years, as language must be acquired during early childhood and is consolidated for many years even after. However, it is not clear why other abilities, especially the more abstract abilities related to abstract reasoning and inductive inference, have a similar behaviour.

This is closely related to Cattell's concepts of fluid and crystallised intelligence (Section 6.2), where the former is the ability of solving problems independently of previously acquired knowledge, while the latter is the ability of correctly finding and applying the given knowledge to a particular problem. Again, it is easy to understand why crystallised intelligence increases with cognitive development (with still moderate increments until the age of 40 or 50, see 7.1). However, why does fluid intelligence grow until the age of 20? One main hypothesis is that the brain develops until that age, so the increase in fluid intelligence has a neuropsychological explanation [Eps79]. However, another possible explanation is that while the brain reaches its final size and connections very early, some *cognitive constructs* (such as identity, difference, size, order, counting, symmetry, logic, quantification, (re)iteration, recursion, etc.) are also useful in fluid intelligence and need to be acquired during a long period of time. These two hypotheses are also closely linked to the nature-vs-nurture dilemma, as fluid intelligence in adults can be well predicted from early fluid intelligence [NBBJ$^+$96], showing a strong genetic basis, but contrasting with the diminished performance in fluid intelligence tests for children with a poorly-stimulating environment or education. Instead of two alternative hypotheses, the question is rather to establish what the role and relevance are of both an internally-driven cognitive development and an externally-enhanced cognitive development. In fact, the picture becomes more complicated, as some recent experimental evidence has shown that fluid intelligence can be increased by cognitive exercising [Ste08]. The area of autonomous mental development requires, thus, better measurement techniques and devices to de-

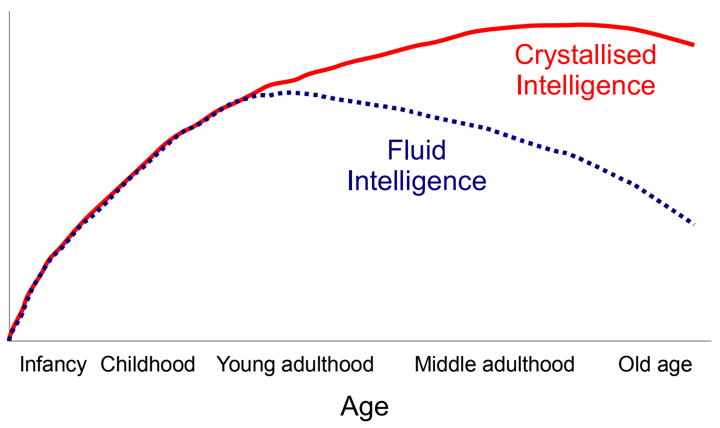


Figure 7.1: A figurative representation of how fluid and crystallised intelligence "display different life-span developmental trajectories" [Bal87, Fig.1]. Can we expect (and measure) the same evolution of general intelligence for artificial cognitive systems? What is the role of cognitive operational constructs in this evolution?

termine whether and how systems develop [OK07], and to tell between what a system has been programmed to do and what abilities the system has really developed. As an interdisciplinary area, the understanding and adaptation of measurement devices from humans to artificial cognitive systems may have an important impact in how the goals and progress in the field are understood, strengthening the connections and cross-citations with other disciplines and a source of new research questions.

As mentioned in the introduction an in the previous chapter, the use of intelligence test problems for the evaluation of artificial cognitive systems has increased in the past decades. On one hand, we have those who advocate for the use of human cognitive tests for machines directly. This idea is behind the psychometric AI proposal [BS03, Bri11] (see Section 6.3), where sets of tests for all possible human capabilities should be used to evaluate artificial cognitive systems. This view is consistent with an editorial [Det11] by Douglas K. Detterman (already commented in Section 6.5), the editor-in-chief of *Intelligence*, motivated by the success of IBM's program Watson [FBCC+10] on the *Jeopardy!* TV quiz show. Detterman challenged Watson and other artificial systems to pass a battery of human intelligence tests. For instance, Spaun (see Section 6.3), a 2.5-million-neuron model of the brain has been able to "reproduce the largest amount of functionality and behaviour" [Yon12] by their performance on a "diversity of tasks" [ESC+12], where some of them are very similar to intelligence test tasks, such as serial working memory, counting, number series, etc. Despite this pullulation and advocacy of intelligence test tasks for evaluating artificial cognitive systems, some criticisms have been raised as whether the resulting scores can be meaningful [DHO12], starting with [SD03], who devised a very small and ad-hoc program that was able to score relatively well on many human IQ tests.

That does not mean that cognitive tests for humans need to be discarded for artificial systems, but that a proper selection and a careful analysis of results may be required in each case, as we stated in the previous chapter. In fact, some approaches to evaluate *artificial* cognitive systems or architectures have *adapted* existing tests or have been inspired by them, such as the *cognitive decathlon* [MJMH07, Mue08, SJT08, CG03], a set of tests designed to evaluate the performance of truly intelligent agents in a variety of situations that cover a core set of cognitive, perceptual, and motor skills typical for a two-year-old human child; the *staged developmental test* [Kee10], which states that if we are to judge intelligent machines by our own standards of intelligence, then we can enrich the process of AI development by a staged approach to cognitive development similar to those of Piaget [Pia64]; the *intelligence tests for robots* [SS10, Sch13], where intelligence test are taken by a robot in a rich sensorimotor scenario (see Section 6.3); and others [AL03, Lan11]. Many of them are inspired by human cognitive development tests such as Kuhlmann's test [Kuh39], which consists of eight separate subtests administered at different difficulty levels, involving nonverbal (problem solving, picture and number patterns, proportions and symmetry, . . .) and verbal tests (scrambled words, scrambled sentences, ordering, visual clues, . . .); and Griffith's mental development scale (a test) "covering the locomotor, personal-social, hearing and speech, eye and hand, and performance areas" [Gri54], and also by intelligence tests.

Many tasks that comprise development tests are associated with specific achievements or functions [AHK$^+$09], such as smiling, reacting to voice, grasping objects, following a gaze, walking, etc., rather than general abilities, and are clearly more prone to hard-wiring specific circuits and mechanisms from the very system design. We are then more interested in purely cognitive abilities appearing in these tests, such as the more abstract tasks that are found in intelligence tests. While the analysis of both crystallised and fluid intelligence tests would be interesting, crystallised intelligence tests can be more easily contaminated by the existence of predefined structures and task-specific problems. Also, most crystallised intelligence problems are verbal, which limits the application of these tests to systems with a limited or no comprehension of natural language. In fact, Watson, although not tested with a crystallised intelligence test, is able to access millions of gigabytes of textual information to answer tricky natural language questions that are very hard for most humans. In fact, adapting Watson to achieve some good scores on crystallised intelligence tests would not be difficult, even if Watson is still a very specialised system without a real intelligence.

Fluid intelligence tests have several advantages for evaluating cognitive development as most of them do not require natural language and also because they are commonly represented in very abstract terms, so it seems possible to analyse and ultimately understand what processes and constructs may be needed to solve them and what their actual difficulty is. If we look at the Wechsler's WPPSI, WISC and WAIS mentioned in Section 7.1, they usually measure fluid intelligence on the performance scale and crystallised intelligence on the verbal scale. If we focus on non-linguistic abstract problems (usually found under the categories of working memory and perceptual organisation), and exclude memory problems because of lack of a suitable machine evaluation, we find matrix reasoning tasks, problems about categories and similarity (picture concepts and symbol search) and letter-number sequencing (only present in WISC and WAIS). These tasks usually feature a combination of inductive inference and abstract reasoning and are among those with highest g loadings, namely, they highly correlate with the g-factor[2], so an individual's performance at one type of cognitive task tends to be comparable to their performance at other kinds of cognitive tasks. This is particularly interesting, as the g factor accounts for an important fraction of the variation of other cognitive abilities, and it seems to be higher for individuals with lower ability levels and children [TD09].

From the previous analysis, we will focus on fluid intelligence tests with high g loading. In order to make experiments more insightful and easy to replicate we will select problems that are easily accessible, either because they are on the open domain or because they have been well studied by previous works. For instance, odd-one-out problems have been chosen as very relevant for cognitive development [SS10], as they are closely related to the capability of forming categories and distinguishing between them [GSSS12]. Raven's Progressive Matrices (RPM) [RCR92] have some of the highest g loadings of standard cognitive tests. Finally, Thurstone's letter series [TT41] are a well-known instance of series problems, which are regularly found in most general intelligence tests.

It is important to note, however, that the use of any cognitive test to

[2]The g factor (short for "general factor") is a construct developed in *psychometrics* (theory and practice of psychological measurement) investigations that is usually derived from a factorial analysis of the cognitive abilities, and is usually associated to the idea of *general intelligence*.

evaluate cognitive development is based on the assumption that the difficult items in these tests require a more advanced cognitive development than the simpler ones. Otherwise, we could use the adult versions for all. Nonetheless, it is not clear whether this difficulty comes from a higher search power when looking at the combinatorial possibilities or it is because some difficult problems require some cognitive operational constructs. With the term 'cognitive operational construct' we do not refer to very elaborated linguistic concepts, neither do we refer to knowledge facts, but rather about elementary *operational* concepts such as identity, difference, order, counting, logic, etc. Many of these constructs are specifically addressed in children education curricula.

Circumscribing our study to fluid intelligence tests frames the question in terms of the evolution of the development and evolution of general intelligence, as opposed to the mere acquisition of factual knowledge, from which general intelligence (including crystallised intelligence) should, therefore, not benefit (e.g., "how many protons a hydrogen atom" has is not usually found in a culture-fair test, as could be answered by any idiot savant or a web searching engine). This analysis of general intelligence tests removes many other sources of contamination and renders the question in a more pristine way. Also, it links cognitive development to an increase in general intelligence and suggests a possible path to develop and study how artificial cognitive systems could be created. We do not expect a general cognitive system to be highly intelligent from its very creation, neither to be highly specialised, but rather to have very low levels of intelligence and being mostly useless in the beginning. In other words, development robotics and other areas of artificial (general) intelligence would aim instead at developing potential intelligent systems, whose general intelligence could gradually increase with the interaction with the world, as it happens with humans (up to the limit of the computational resources of the system).

Analysing these questions by administering some tests to humans and by asking them what constructs and patterns they have used and found respectively is a traditional approach for the analysis of human cognitive development. This provides a subjective and anthropomorphic view, and it is also contaminated about many other issues taking place during human development, including physical, biological and social effects. Nonetheless, it is still a useful approach. However, when the goal is to assess the development of artificial cognitive systems, we need more general approaches for a theoretical or experimental analysis of intelligence test problems. In fact, there have been other works in the past where IQ tests, development tests or other kinds of cognitive tests are undertaken by artificial systems. The goal of the study is not to evaluate these systems, but to better understand how the tests work and what they measure. In fact, this analysis of tests should have really *preceded* their use over artificial (and natural) systems. As most studies so far have been performed at the experimental level, the artificial system that is chosen to analyse the test is basically seen as *a cognitive tool*, sometimes in the form of a cognitive model (which is supposed to help understand how cognition works) or as a merely instrumental tool.

7.3 Analysing the tests: specific cognitive models and general tools

Although already introduced in Chapter 6 (Section 6.3), let us briefly recapitulate what has been done when intelligence tests have been analysed experimentally through cognitive models or particular systems, but focusing on our previous problem selection. For a extensive overview of the IQ tests problems we refer the reader to the Appendix B (although in the following section we will recap those selected).

Starting with letter series, we need to go back to the 1960s when [SK63] undertook "Thurstone Letter Series Completion" tasks, with the aim of better understanding how humans solved these kinds of problems and their difficulty (analysed in terms of memory requirements), through the use of a computer model (again with a symbolic representation) that could generate series.

Raven's Progressive Matrices (RPM) were investigated by [CJS90] with two computer simulation models ($FAIRAVEN$ and $BETTERAVEN$). Yet again, the goal was to better understand human intelligence and the nature of the tests. These systems included five relational rules (constant in a row, distribution of three values, quantitative pairwise progression, figure addition and distribution of two values) emulating the cognitive operational constructs that are needed to solve this intelligence test problem, and several attributes to describe the information about the images. [LFU07, LFU10] developed an RPM solver based on Carpenter's work by using structure mapping (identifying commonalities and differences in non-preprocessed images) and analogical generalisation. The authors claimed that their model overcame the limitations of [CJS90]'s model, using visual representations and task-general processes. RPMs were also analysed by [RN12] by using the cognitive architecture ACT-R [And96] and a rule identification mechanism. The system requires the identification of relational rules, as Carpenter did, and a description of the geometric objects of the input matrices using attributes. A different attempt to address RPMs was undertaken by [MKG10] and [KMG10, KMG12] by capturing similarities directly between images (without any conversion to symbolic representation) and induce transformations. [PR11] and [CPR12] also developed a system for solving RPMs based on an "analogical proportion" view applicable at different levels of representation: feature-based, and pixel-based. Finally, [SCS13] used an anthropomorphic cognitive model (in the sense that it uses certain problem solving strategies that were reported by high-achieving human solvers) and a repertoire of patterns for RPMs.

Odd-one-out problems have also been subject of several studies, mainly by those authors that already addressed RPMs. Some have focused on the visual problem (analogical approaches), with the use of fractals and similarities [MG11a] in the same way they addressed RPMs [MKG10, CPR12]. Another analogical approach was carried out by [LLF08], who used spatial representations with structure mapping with the aim of constructing a model to better understand item difficulty (also in a similar way to their attempts to address RPMs). In a different way, some other systems have assumed some previous feature transformation [Rui11], called the "Ruiz-Absolute Scale of Complexity Management" (R-ASCM) where the objects in each item of an example are coded ("discretised") as letter sequences and the problems are solved by using clustering with some similarity measures as required constructs.

As discussed in the previous chapter (Section 6.3.4), this collection of systems and approaches are difficult to compare, as the cognitive principles, languages and goals of each work are very different. In fact, any assessment of difficulty has to be taken very carefully, as a specific method solving one type of problem is relatively easy to build [SD03, DHO12] and may have a strong bias in their scaling of problem difficulty. In brief, most of the previous systems and cognitive models have been defined on purpose for one problem, usually with some strong assumptions about how the problems are solved by humans. On many occasions, when the model is not anthropomorphic, the patterns cannot be properly investigated as the systems that are used do not yield comprehensible patterns (as usually happens with, e.g., neural networks, although *constructive* neural networks have been found to be insightful for the problem of psychological development [Shu12]. However, most importantly, none of them has focused on the issue of item difficulty in terms of both the combinatorial search space of each item and the required operational constructs under the context of cognitive development. Item difficulty is crucial in order to set different test scales for different stages of development. Note that it is not very meaningful if we define problem difficulty as the average result of a population of artificial cognitive systems (mimicking what is done with human population, item response theory and IQ normalisation), as the sample of systems would be completely arbitrary. So, in order to assess difficulty we need system generality and intelligibility. More precisely, we need a cognitive tool that meets the following properties:

- The system must have a general purpose. It cannot be defined ad-hoc for intelligence tests, as this may lead to specialisation to one or more tests [SD03], and the results would not be meaningful [DHO12]. The use of inductive programming systems for cognitive modelling tool has been recently shown to be effective by [SK11].
- Because of the above design generality, the system must *learn* to solve the problems. In other words, it has to be a *learning* system, which must be guided by a rewarding system or the level of success on several examples.
- The system has to be explicit in what constructs and operators must be given as background knowledge when addressing each problem. These operational constructs must also be intelligible.
- The system must produce intelligible solutions, such that their patterns can be compared to those usually extracted by humans, and their structure can be evaluated in terms of size and number of constructs.

The above properties suggest the use of declarative learning systems, i.e., inductive programming systems [Kit10, SK11, FS08]. We will use the declarative learning system gErl (described in Chapter 5), as a cognitive tool that fulfils the above properties. Note that a symbolic system can also be emergent (compatible in the sense of [Wen12]) if new constructed concepts can be incorporated or created by the system. We will use gErl to analyse several general intelligence problems in sections 7.4 and 7.5, discussing on the identification of what makes each instance easy or hard, and the proportion of this difficulty in terms of the pattern size and the constructs that are involved.

7.4 Developmental analysis of IQ test problems

In this section we analyse several general intelligence tests problems, as chosen in section 7.3: odd-one-out, Raven's matrices and letter series by using the gErl system. For a further description description of these and others IQ tests, we refer the reader to Appendix B.

7.4.1 Odd-one-out problems

The odd-one-out problems are focused on geometry and spatial understanding, where the goal is to spot the most dissimilar object from the rest. They were first introduced by [ZHH74, ZHE80] as non-verbal test and was used in the study of comparative animal learning ([ZHH74] assessed pigeon's intelligence), where some species appear to learn this kind of problems readly, slowly or nothing at all. The oddity task has been also used for cross-cultural testing to probe the conceptual primitives of geometry in the Mundurukú, an isolated Amazonian indigenous group, for which [DIPS06] designed a visual oddity task (further information in Appendix B). Figure 7.2 shows three examples of odd-one-out problems of increasing complexity. Typically, the presented items can vary along one dimension (e.g., shape, size, quantity).

Problem representation and constructs

In order to focus on the core of the problem and not on the visual recognition issues we use the abstract representation R-ASCM, introduced by [Rui11] and seen in Section 7.3. Remember that in R-ASCM, for instance, an item composed by two circles and a square is represented as (A, A, B). We will also use the same 35 examples in [Rui11], with the R-ASCM coding, as shown in Table 7.1. To process the examples with gErl, we represent each set of items as a list of lists. This will form the *lhs* of each example (that in gErl is an equation $lhs \to rhs$). And its corresponding *rhs* will be a number indicating the position of the item in the list that is the odd one. For instance, example number 3 in Figure 7.1 is represented as:

```
ooo([[a,a,a], [a,a,b], [a,a,c]]) -> 1
```

Next we need to define appropriate operators, both to navigate the structure and to apply local or global changes to the rules. As in [Rui11], we will use some constructs to analyse these lists. In our case, we identify three types of constructs:

- **Differences between items**: As used in [Rui11], we incorporate the notion of difference between items, by adding the function `hamming`, which represents the average Hamming Distance[3] between objects. This measure is calculated for every item inside an example and refers to the average distance of a given item to all the other items within an example. For instance, given the items of the example #3 in Figure 7.2 ($[[a, a, a], [a, a, b], [a, a, c]]$), if we apply this construct over each item we will obtain, for all items, a result equal to 1.

[3]The Hamming distance between two strings of equal length is the number of positions at which the corresponding symbols are different.

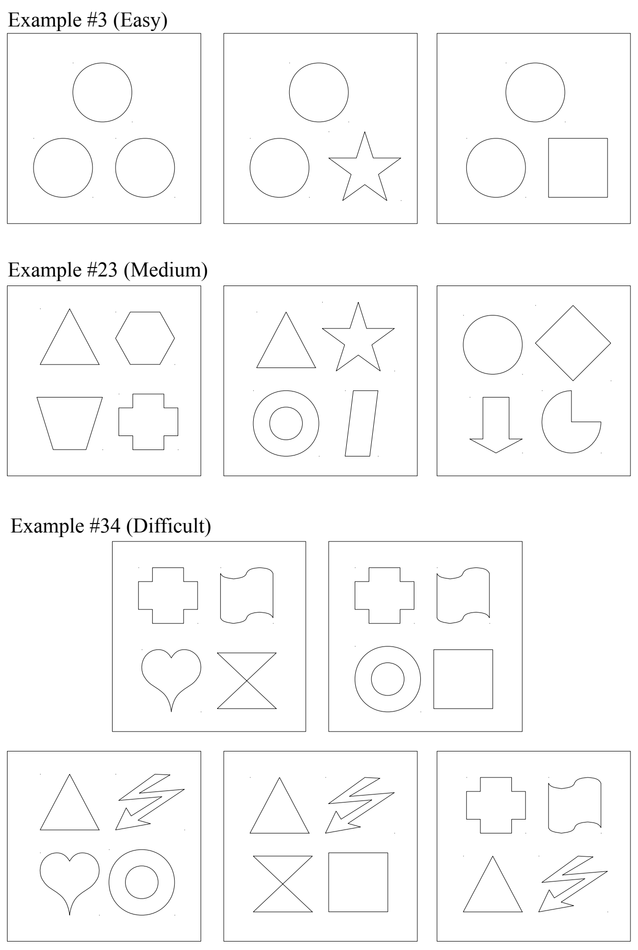


Figure 7.2: Examples 3, 23 and 34 from the 35 original odd-one-out examples in [Rui11] (adapted and redrawn), sorted by human-subjective complexity (from easy to difficult).

# Example	Item 1	Item 2	Item 3	Item 4	Item 5
01	AAA	AAA	ABB		
02	AAA	AAA	BCD		
03	AAA	AAB	AAC		
04	AAA	ABB	ABB		
05	AAA	BBB	ABC		
06	AAA	BCD	EFG		
07	AAA	BBC	CCB		
08	AAB	AAB	ABC		
09	AAB	AAC	DEF		
10	AAB	ABB	EFG		
11	ABC	ABC	ABD		
12	AAB	ABB	ABC		
13	ABC	ADE	FGH		
14	AAAA	BBDE	CCFG		
15	AAAA	AABB	AACC		
16	AAAD	BBEF	CCGH		
17	AABB	AABB	ABCD		
18	AABC	AACD	ABCD		
19	AAAB	BBBD	CCCE		
20	ABCD	ABCD	ABCE		
21	ABCD	ABCE	ABFG		
22	AABC	BBAC	CCAF		
23	ABCD	AEFG	HIJK		
24	AAAA	AAAA	BBBB	BBBB	CCCC
25	AAAD	AAAE	BBBF	BBBG	CCCH
26	AABB	BBCC	AADD	DDCC	EEFF
27	AAEF	BBGH	CCIJ	DDKL	ABCD
28	AAAE	BBBF	CCGH	DDIJ	ABCD
29	AAAE	BBBF	CCGH	DDIJ	AABB
30	AAAB	BBBF	CCGH	DDIJ	AABB
31	AABB	BBCC	AADD	DDCC	AAEE
32	ABCD	BCDE	CDEF	DEFG	FGAB
33	ACDE	AFGH	BIJK	BLMN	OPQR
34	ABEF	ABGH	CDEG	CDFH	ABCD
35	ACDE	AFGH	BIJK	BLMN	ABOP

Table 7.1: 35 examples of R-ASCM abstract representation (solutions in grey) from [Rui11]. For simplicity and space, letters are used here instead of the figural symbols of the real test. Both the items in each example and the coded figures in each item are sorted alphabeticaly. The first letters of the alphabet correspond to the most frequently used symbols in an item (e.g., A is more frequently used than B, B than C, and C than D).

- **Differences inside items**: In addition, we also include another construct about item diversity, which counts the number of different objects inside an item (`diffObj`). For instance, taking the previous example into account, the diversity for the first item is equal to 1 while for the next two items is equal to 2.

- **The notion of an element that is distinct**: Finally, this concept must actually be given in order to solve these problems, as it is usually explained in the test instructions. It is extremely related to the previous constructs since, once we have applied any of the previous ones to one example, we have to select the odd one.

Since these functions have to be applied to a list, we define operators op_1 and op_2 with the help of the higher-order function *map* that Erlang provides in order to introduce the two first constructs.

$$\begin{aligned} op_1 &\equiv \mu_{replace}(\overline{3}, f_{hamming}) \\ op_2 &\equiv \mu_{replace}(\overline{3}, f_{diffObj}) \end{aligned}$$

where $f_{hamming}(\rho) = \texttt{map}(\texttt{hamming}, \rho|_{1.1})$ and $f_{diffObj}(\rho) = \texttt{map}(\texttt{diffObj}, \rho|_{1.1})$, and $\overline{3}$ is a constant function returning the position 3, namely, the *rhs* of the input rule over which the operator will be applied[4] (this has been explained in Section 5.2). This meta-operator ($\mu_{replace}$) is thus used to define an operator in charge of replacing the *rhs* of the input rule by the defined functions. Since the previous operators return a list with the values for each item in each example, we must let the system apply the function `distinct`, which selects the different item (if exists) in a list. Hence, the operator op_3 can be defined as

$$op_3 \equiv \mu_{replace}(\overline{3}, f_{distinct})$$

where $f_{distinct}(\rho) = \texttt{distinct}(\rho|_3)$.

Finally, we need a way of generalising the examples. This is performed by introducing variables at each possible position.

$$op_4 \equiv \mu_{replace}(pos_{list}, \overline{V_{lists}})$$

where we use now the function $pos_{list}(\rho)$ (which is not a constant) returning all the positions in ρ where we can find a list (1.1, 3.2 and 3.1.2) and $\overline{V_{lists}}$ is a constant function returning a list variable.

[4]Following the usual representation of (functional) rules as trees, each sub-part (term) of a rule is denoted by a natural number that represents the position of such term in the tree.

The abstract idea of generalisation through the use of variables appears as an operator, but it is not considered a cognitive construct here as it is rather a way of representing that the function *ooo* can be applied to any list.

Results

Table 7.2 shows the two hypotheses (rules) with highest optimality found by gErl (h_1 and h_2). Taking the first rule, 28 of 35 (80%) examples are solved (being on a par with an average human adult). Regarding the second rule, it solves 17 of 35 examples. Note that some examples are covered by both rules. Example number 31 is not covered by any rule because it exhibits other more complex properties, not captured by the constructs. 7.2 also shows the results provided by Ruiz. Ruiz defines a 2-step clustering algorithm. In the first step, 28 of 35 (80%) examples are solved, exactly the same number that gErl covers with its first rule. The second step consists in taking those examples that were misclassified (*using* the classes of the test set), and recode them using what he calls the *Structural Hamming Distance* (SDM). Note that gErl's approach is more general as we do not have to recode misclassified examples. Also, the `diffObj` construct is much simpler than the SDM. In any case, it is not our goal to compare which system is best (in fact, we mostly follow Ruiz's approach, as we use his representation and the Hamming distance function), but to see what constructs are used in each case.

Example \ Approach		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	Total
gErl	$h_1^{*^1}$	●	●		●	●		●	●	●	●	●		●				●	●	●	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	28
	$h_2^{*^2}$	●	●	●	●		●	●	●	●	●		●		●	●	●	●	●									●	●								17
Ruiz	1^{st}*step*	●	●		●	●		●	●	●	●	●		●				●	●	●	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	28
	2^{nd}*step*			●									●		●	●	●																				5

$*^1$ h_1: `ooo(`V_{lists}`)`$\rightarrow$ `distinct(map(hamming,` V_{lists}`)`
$*^2$ h_2: `ooo(`V_{lists}`)`$\rightarrow$ `distinct(map(diffObj,` V_{lists}`)`

Table 7.2: Odd-one-out results for both gErl and [Rui11]'s approach. Filled dots shows those examples from Table 7.1 solved. h_1 refers to the hypothesis using (among others) the Hamming construct (h_1: `ooo(`V_{lists}`)` $\rightarrow$ `distinct(map(hamming,`V_{lists}`))`), whereas h_2 refers to the hypothesis using the `diffObj` construct (h_2: `ooo(`V_{lists}`)` $\rightarrow$ `distinct(map(diffObj,`V_{lists}`))`)

Table 7.2 is very informative as we can separate those examples (28, actually) that can be solved with the notion of difference between same-length lists (i.e., the Hamming distance, which implies some kind of alignment) and those examples (17, actually) that can just be solved with an internal account of diversity of each item. Finally, only one example requires some other constructs. Apart from this case, we see that the pattern complexity is similar in both categories. At least in gErl, both rules have the same structure and number of constructors. In fact, the number of steps the system takes is similar (227 and 230). As a result, we can say that this is a clear example of problems where two categories can be established by the constructs they require and not because the search space is significantly higher for one problem or the other. In other words, each category will only be solved when the construct is available (or can be discovered) at a given stage of cognitive development, and does not require any exceptional combinatorial or brain power. Performance differences in humans must be then attributed to the complexity of the problems that can be built. Namely, the more symbols are included in the sets, the more advanced mathematical properties could appear (see, for instance, item 31 in 7.1, which cannot be solved with any of the construct used). However, mathematical abilities (prime, odd or even numbers, squares, . . .) are not measures of fluid intelligence but of crystallised ability, so this mathematical knowledge is beyond the scope of this chapter.

7.4.2 Raven's Progressive Matrices

Raven's Progressive Matrices (RPM) [RC96] consist of a pattern or a set of items where a missing part or item has to be guessed. The most typical case is a 3×3 grid where a figure is placed at each of the nine positions except the bottom-right cell, which is empty. Eight possible choices ('distractors') to fill in the gap are displayed at the bottom, as illustrated in Figure 7.3. There is a logical relation between the figures, which can be seen either horizontally (rows) or vertically (columns).

There are three different sets of RPM for participants of different IQ ranges or different abilities [RC96]: the original Standard Progressive Matrices (SPM), the Coloured Progressive Matrices (CPM) for children aged 5 through 11 years-of-age, the elderly, or people with learning difficulties, and the Advanced Progressive Matrices (APM), developed to assess individuals of above-average intelligence. We will just work with SPM, which consist of 5 sets with 12 items in each set —60 items in total. Sets A and B do not use a 3×3 grid structure, so we will use sets C, D and E as they share the same structure (although the difficulty varies). The items were reconstructed using information from [Rav01].

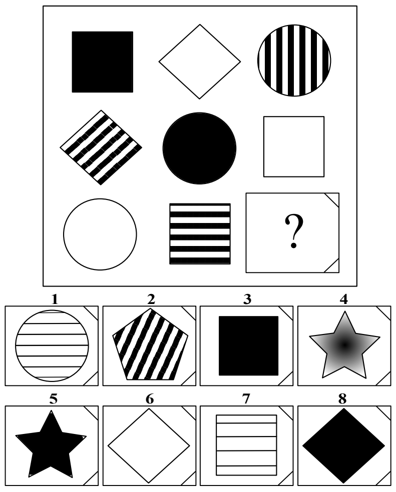


Figure 7.3: A geometrical reasoning problem similar to Raven's Progressive Matrices. The solution is no. 8 (for copyright reasons, the illustrations in this paper do not depict original RPM problems, but constructed equivalents).

Problem representation and constructs

Given an example of one kind of Raven's Progressive Matrices (as the one shown in Figure 7.3), the task of gErl will be to guess the pattern and, then, to apply it to infer the solution. In other words, gErl tries to build the solution rather than choose among the collection of 8 possible choices (1 solution and 7 distractors) under the matrix (which are not given to gErl). So, in this way, the problem formulation is slightly more difficult than the original one.

In order to represent RPM in gErl, a feature-based coding similar to that of [RN12] is used. Additionally, a list-based coding is used to represent cells, rows, and matrices. Every figure inside a cell is abstractly represented as a tuple of features:

$$\langle shape, size, quantity, position, type \rangle$$

Every cell is represented as a list of figures. Every row is represented as a lists of cells, and, finally, every Raven's matrix as a list of rows.

Since every single Raven's matrix is a problem itself (each matrix shows a different pattern), we need a way to generate several instances in order to make learning possible, by taking the most information from each matrix. To do that, each matrix is decomposed into several sub-matrices (the number depends on the problem) as we can see in Figure 7.4. The last row/column of the original matrix cannot be used to generate any training instance since it contains the gap to be filled in. However, they will be used to create two *test* cases (row and column). Note that the output element must obviously be the same, but the vertical and horizontal patterns may be different. If that happens, the system selects the best program with respect to the row training instances to be applied to the row test instance, and the best program with respect to the column training instances to be applied to the column test instance.

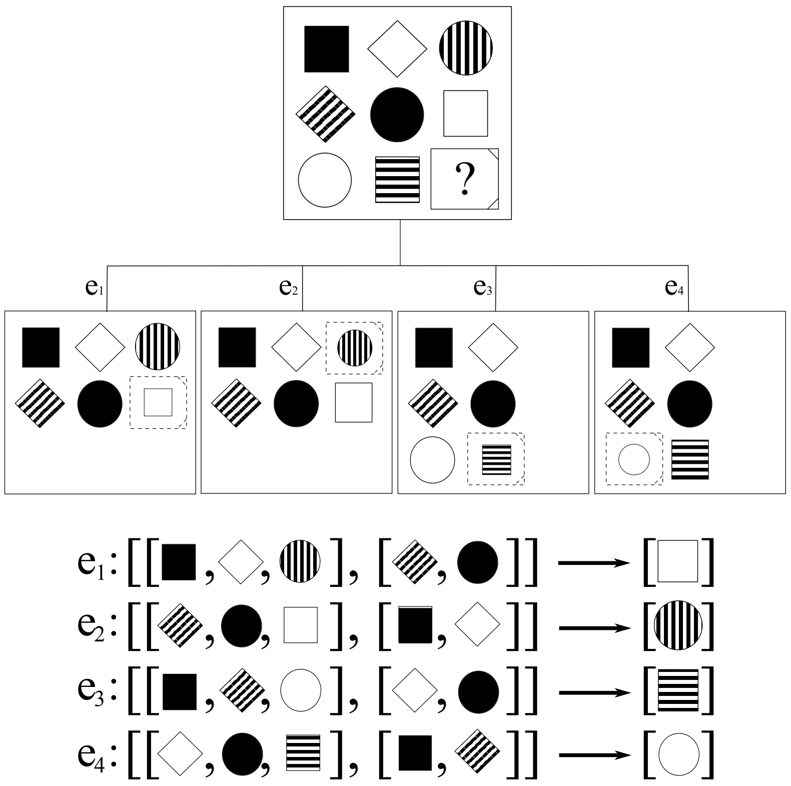


Figure 7.4: (Top) Raven's matrix decomposition example: e is decomposed as a four new examples (e_1, e_2, e_3 and e_4). (Bottom) List-based representation of the training instances generated (bottom).

For instance, in gErl, the example e_1 from Figure 7.4 is represented as follows, where `none` represents the absence of a particular feature, and the special variable `_` matches anything, and never gets bound:

```
raven([[[<square,big,1,__,black>],
[<diamond,big,1,none,white>],
[<circle,big,1,none,striped>]],
[[<diamond,big,1,none,striped>],
[<circle,big,1,none,black>]]]) ->
[<square,big,1,none,white>].
```

To solve RPMs, it is necessary to determine the relations between the objects in a row or column. [CJS90] identified five 'relations':

- **Identity (equality)**: is a particular attribute of different objects remaining constant in a row? For instance, in Figure 7.5, the attribute *shape* (square) remains constant in every row/column. We will use the function `ident`.
- **Ternary distribution (difference)**: is an attribute of the objects in a row/column always differing (3 distinct values)? For instance, in Figure 7.5, the attribute *type* differs in matrix a. We will use the function `dist3` for this.
- **Progression**: are the values of an attribute in an increasing or decreasing sequence in all rows? This rule is seen in matrix *b* (Figure 7.5), where the attribute for the object's position turns 45 degrees in each cell. We will use the function `prog` for this.
- **Addition (OR)**: is each object in the third row/column appearing in any of the first two rows/columns? An example is depicted in matrix *c* (Figure 7.5). We will use the function `addition` for this.
- **Binary distribution (XOR)**: are there exactly two equal values and one differing value of an attribute in the rows/columns? This rule is seen in matrix *d* in Figure 7.5. We will use the function `dist2` for this.

While these *relations* were expressed in terms of solutions, here they are just included as operational constructs and implemented as complex functions in gErl, i.e., each construct goes through each cell (in a complete row/column) looking for a specific descriptive feature and returning a solution for the construct.Since we are only interested in knowing which of those relations are required to solve each RPM problem, and knowing the abstract mental operations they represent, the low-level programming details of these previous functions are beyond the scope of this chapter.

To solve the selected 36 matrices from SPM, we need a way to apply the five *relations* (functions) to the different attributes of the figures. The

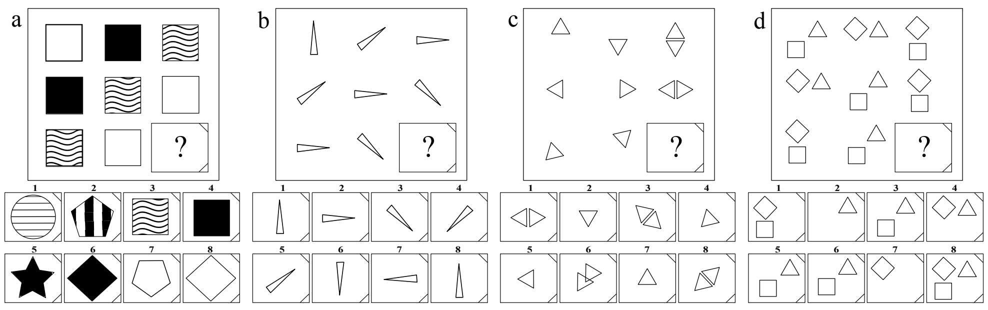


Figure 7.5: Problems illustrating several 'relations' in RPM problems (redrawn and adapted from [RN12]). For matrix a, the ternary distribution *is required. The solution is no. 4. Matrix b requires* progression*. The solution is no. 6. For matrix c,* addition *is required. The solution is no. 8. Matrix d requires* binary distribution*. The solution is no. 4.*

meta-operator $\mu_{replace}$ does this perfectly by defining operators following this scheme:

$$\mu_{replace}(pos_F, f_{Rel})$$

where $pos_F(\rho)$ returns all positions where feature F (shape, size, quantity, position or type) appears in the rhs of rule ρ and $f_{Rel}(\rho)$ applies the relation function Rel (`ident`, `dist3`, `prog`, `addition`, `dist2`) to $\rho|_p, p \in pos_F(\rho)$.

We will obtain as many operators as kinds of attributes multiplied by the number of relations (5 attributes $\times$ 5 relations, 25 operators). For instance, the operators that apply *ident* to the five possible attributes will be defined as:

$$\begin{array}{llll} op_1 \equiv & \mu_{replace}(pos_{shape}, f_{id}) & op_2 \equiv & \mu_{replace}(pos_{size}, f_{id}) \\ op_3 \equiv & \mu_{replace}(pos_{quantity}, f_{id}) & op_4 \equiv & \mu_{replace}(pos_{position}, f_{id}) \\ op_5 \equiv & \mu_{replace}(pos_{type}, f_{id}) & & \end{array}$$

where $f_{id}(\rho) = \texttt{ident}(\rho|_p), p \in pos_F$. Below we show an example of application of the operator op_3, which is in charge of applying the relation `ident` over the feature *quantity*, to a rule (where some other operators have been already applied):

$op_3\big(\texttt{raven}(V_{matrix}) \rightarrow$ `[<dist3(shape),ident(size),1,none,dist3(type)>]`$\big) \Rightarrow$
$\quad\texttt{raven}(V_{matrix}) \rightarrow$ `[<dist3(shape),ident(size),ident(quantity),none,dist3(type)>]`

where V_{matrix} is a matrix variable. This suggests that we also need a generalisation operator for input lists (as we did in the odd-one-out problem):

$\mu_{replace}(\overline{1.1}, \overline{V_{matrix}})$. If we apply the previous rule learnt by the system to the test row/column in Figure 7.3, the different functions applied over the particular features will be in charge of returning the correct values (e.g. `dist3(shape)` will return `diamond` since the `circle` and `square` shapes have been already used), thus returning the following cell as solution:

```
[<diamond,big,1,none,black>]
```

which covers the correct solution (no. 8).

Results

Id	Solution	Steps	E^+	$\lvert o_{app} \rvert$	Diff
C01	raven(V) → [⟨ident(shape), none, none, none, none⟩]	37	3	2	-2.6
C02	raven(V) → [⟨ident(shape), prog(size), none, none, none⟩]	99	4	3	-3.0
C03	raven(V) → [⟨ident(shape), prog(size), none, none, none⟩]	99	4	3	-3.3
C04	raven(V) → [⟨ident(shape), prog(size), none, none, none⟩]	111	4	3	-2.2
C05	raven(V) → [⟨ident(shape), none, prog(quantity), prog(position), none⟩]	131	4	4	-3.5
C06	raven(V) → [⟨ident(shape), prog(size), none, none, none⟩]	88	4	3	-1.5
C07	raven(V) → [⟨ident(shape), none, none, prog(position), none⟩]	81	4	3	-2.7
C08	raven(V) → [⟨ident(shape), none, prog(quantity), none, none⟩]	79	4	3	-0.8
C09	raven(V) → [⟨ident(shape), none, none, prog(position), none⟩]	91	4	3	-1.6
C10	raven(V) → [⟨ident(shape), none, none, prog(position), none⟩]	91	4	3	-0.5
C11	raven(V) → [⟨ident(shape), none, prog(quantity), none, none⟩]	81	4	3	-0.5
C12	raven(V) → [⟨ident(shape), none, none, prog(position), none⟩]	83	4	3	1.2
D01	raven(V) → [⟨ident(shape), none, none, none, ident(type)⟩]	75	4	3	-2.8
D02	raven(V) → [⟨dist3(shape), none, none, none, none⟩]	69	4	2	-2.3
D03	raven(V) → [⟨dist3(shape), none, none, none, none⟩]	71	4	2	-2.3
D04	raven(V) → [⟨ident(shape), none, none, none, dist3(type)⟩]	94	6	3	-2.1
D05	raven(V) → [⟨ident(shape), none, none, none, dist3(type)⟩]	96	6	3	-2.6
D06	raven(V) → [⟨ident(shape), none, none, none, dist3(type)⟩]	93	6	3	-2.6
D07	raven(V) → [⟨dist3(shape), none, none, none, dist3(type)⟩]	106	6	3	-2.1
D08	raven(V) → [⟨dist3(shape), none, none, none, dist3(type)⟩]	91	6	3	-2.0
D09	raven(V) → [⟨dist3(shape), none, none, none, dist3(type)⟩]	104	6	3	-1.5
D10	raven(V) → [⟨ident(shape), none, none, none, dist3(type)⟩]	93	6	3	-1.4
D11	raven(V) → [⟨ident(shape), none, dist3(quantity), none, dist3(type)⟩]	146	6	4	1.1
D12	raven(V) → [⟨dist3(shape), none, none, none, dist3(type)⟩]	106	6	3	1.8
E01	raven(V) → [⟨addition(shape), none, none, none, none⟩]	61	4	2	-1.5
E02	raven(V) → [⟨addition(shape), none, none, none, none⟩]	55	4	2	-1.0
E03	raven(V) → [⟨addition(shape), none, none, none, ident(type)⟩]	99	6	3	-1.3
E04	raven(V) → [⟨dist2(shape), none, none, none, none⟩]	63	4	2	-0.6
E05	raven(V) → [⟨dist2(shape), none, none, none, none⟩]	60	4	2	-0.7
E06	raven(V) → [⟨dist2(shape), none, none, none, none⟩]	61	4	2	-0.4
E07	raven(V) → [⟨dist2(shape), none, none, none, none⟩]	77	4	2	0.9
E08	raven(V) → [⟨dist2(shape), none, none, none, none⟩]	99	4	3	2.9
E09	raven(V) → [⟨dist2(shape), none, none, none, dist3(type)⟩]	100	4	3	1.5
E10	raven(V) → [⟨dist2(shape), none, none, none, none⟩]	60	4	2	0.6
E11	raven(V) → [⟨dist2(shape), none, none, none, none⟩]	65	4	2	0.7
E12	-	-	4	-	1.6

Table 7.3: Solutions returned, steps needed, number of examples (E^+) that are derived from each problem and the number of different operators $|o_{app}|$ that are applied in order to get the solution in gErl (as a measure of the complexity of the solution) for Raven's SPMs sets C, D & E. The last column, taken from [Geo08] shows the results of the b parameter (difficulty) of an IRT model of human performance on these sets. The last example (E12) is not solved by gErl.

As we see in Table 7.3, gErl is able to solve 35 out of the 36 problems (12 of 12 in sets C and D, and 11 of 12 in set E). If we take a look at the length and complexity of the solution, we see that the number of steps is usually larger. However, we see no clear correspondence between pattern size and the difficulty humans find on these problems. Hence, the time that gErl requires to solve a problem is explained by the requirement of relations needed to solve it: the more relations needed, the longer it takes to obtain a solution pattern. Otherwise, as we have said, this is not well correlated with the difficulty found by humans, for which the difficulty lies on the complexity of the relation to apply. In fact, some of the examples in set E have a small solution with a small number of steps, but their difficulty for humans is high. However, if we take a look at the constructs we see that most of the examples in set C use the `ident` and the `prog` constructs, which yields easy problems for humans in general. The use of `dist3` does not seem to be a big challenge for most humans either. However, the use of `dist2` seems to be responsible of the higher difficulty for set E, except for the three cases that only use `addition`, which look easier for humans.

It seems that the operative constructs play an important role in how difficult these problems are for humans, with `ident` and `prog` being constructs that most individuals have and can use, `dist3` and `addition` being intermediate and `dist2` being a construct that seems to be accessible or be developed by some individuals, in the line of the study carried out by [CJS90], where the author claims that the problem difficulty not only appears when the number of figures per cell is not constant but also occurs in problems containing a distribution-of-two-values (`dist2`) relation, as well as figure addition (`addition`) and difference (`dist3`). Finally, there are some cases that could be explained by a combination of pattern size and constructs, such as the extra difficulty of D11. Also, there are some cases that show a strange result, such as D12, as it has the same pattern as other matrices but the difficulty humans find here is much higher. We have to say that we have not analysed the distractors (how good and plausible the other 7 incorrect given options are, which are given to humans but not to gErl). They may play a contaminating role here as well as some issues about the visual shape and type that have not been considered.

7.4.3 Letter series completion problems

Thurstone et al. [TT41] introduced the letter series completion problems as part of some test batteries. These problems were developed to assess "reasoning ability". The goal of the letter series problems is to identify the following letter in a series (see Figure 7.6) from five letter choices. Further information

1.	cdcdcdcd_
2.	aaabbbcccdd_
3.	atbataatbat_
4.	abmcdmefmghm_
5.	defgefghfghi_
6.	qxapxbqxa_
7.	aducuaeuabuafua_
8.	mabmbcmcdm_
9.	urtustuttu_
10.	abyabxabwab_
11.	rscdstdetuef_
12.	npaoqapraqsa_
13.	wxaxybyzczadab_
14.	jkqrklrslmst_
15.	pononmnmlmlk_

Figure 7.6: 15 letter series completion test problems from [SK63].

in Appendix B. To solve a letter series, an abstract pattern has to be identified which captures the regularity of the sequence. The correct answer can be generated by applying this pattern. As typical for induction, there is no generally acceptable concept of correctness. For example, a person might continue a sequence just with some constant letter. Correctness in the context of induction problems typically presupposes that there is one continuation which is most plausible with respect to the given regularity. Typically, problems are carefully constructed in such a way that there is a unique solution. However, there is also research on ambiguous problems, for example in the domain of letter string analogies [Hof08].

Problem representation and constructs

Once again, the first step to deal with Thurstone's letter series problems is to code the examples as equations in order to be correctly addressed by gErl. Each letter series (*lhs* of the equations) will be coded as a list of characters (or strings). The *rhs* will be the character that follows the series. Below we can see the representation of example 1 in Figure 7.6:

$$\mathsf{e}_1 : \texttt{thurstone}(\text{“}\textit{cdcdcdcd}\text{”}) \rightarrow \text{“}\textit{c}\text{”}$$

Since each letter series is a problem itself, we need to provide the system with more than one training instance as we did for the RPM problems. We do that by decomposing each initial example (the input letter series) into several letter series of increasing length. For instance, from the previous example $\mathtt{e}_1$ we create the following training instances:

$$\begin{aligned}
&\mathtt{e}_{1,1} : \mathtt{thurstone}(\text{“}cd\text{”}) \rightarrow \text{“}c\text{”}\\
&\mathtt{e}_{1,2} : \mathtt{thurstone}(\text{“}cdc\text{”}) \rightarrow \text{“}d\text{”}\\
&\mathtt{e}_{1,3} : \mathtt{thurstone}(\text{“}cdcd\text{”}) \rightarrow \text{“}c\text{”}\\
&\mathtt{e}_{1,4} : \mathtt{thurstone}(\text{“}cdcdc\text{”}) \rightarrow \text{“}d\text{”}\\
&\mathtt{e}_{1,5} : \mathtt{thurstone}(\text{“}cdcdcd\text{”}) \rightarrow \text{“}c\text{”}\\
&\mathtt{e}_{1,6} : \mathtt{thurstone}(\text{“}cdcdcdc\text{”}) \rightarrow \text{“}d\text{”}\\
&\mathtt{e}_{1,7} : \mathtt{thurstone}(\text{“}cdcdcdcd\text{”}) \rightarrow \text{“}c\text{”}
\end{aligned}$$

In order to determine the set of constructs, we follow the ideas about the basic “subroutines” from [SK63, KS73] to explain human behaviour in these letter series problem tasks:

- **Sequence**: The problem is a series and needs to be extrapolated: the pattern discovered (symbolic structures built from the vocabulary of such a language) is held in order to continue the generation of the letter series.
- **Letter identity and alphabet order**: It is assumed that the subjects must know the English alphabet (backwards and forward) and are told that it is circular (‘a’ follows ‘z’). That means that they have to know the concepts of letter *identity*, and the *next* and *previous* letters.
- **Periodicity**: Cyclical patterns (or iterations) may be required, e.g., to repeat the list *at* in order to produce *atatatat*. . . . Finding the length and the positions of where the repetitions start is one of the difficulties of the problem, such as in the series *atbataatbat_*: we can mark it off in segments of length 3 (*ata*, *atb*, *ata* and *at_*). Here we observe that the first and the second position of each segment are occupied, respectively, by an *a* and a *t* which is a symple cycle of *a’s* and *t’s* (as previously), and the third position is occupied by the cycle *ba ba* . . . .
- **Composition**: The assembly of two previous components must be needed in order to generate the entire series.

For our purposes, only the first two subroutines will be taken into account (the last two subroutines have nothing to do with learning aspects but with implementation aspects of the system in [KS73]). Therefore, we need some operators in order to (a) work with letter sequences (strings), (b) establish interletter relations (alphabet order) and, finally, (c) deal with the composition (finding regularly occurring breaks in a given series). As sequences are represented with strings, we need some string operators (a) in gErl:

$$\mu_{replace}(\overline{3}, f_{list_L})$$
$$\mu_{replace}(\overline{3}, f_{list_R})$$

to replace the rhs of the rules (position 3) by an application of any of the (Erlang) built-in functions f_{list} that works with lists: `head` (which returns the first element of the list), `tail` (which returns the list without the first element), `last` (which returns the last element of the list) and `init` (which returns the list without the last element). This can be applied either over the input ($\rho|_{1.1}$) letter series string ($f_{list_L}(\rho) = f_{list}(\rho|_{1.1})$) or over the rhs ($\rho|_3$) of ρ ($f_{list_R}(\rho) = f_{list}(\rho|_3)$). Note that this latter function allows the system to construct rules whose rhs are the result of successive applications of different functions since it takes whatever there is in the rhs of ρ as input of the new function to allocate there. This will also make possible to deal with periodicity.

For handling the alphabet order (b) we can also use the meta-operator $\mu_{replace}$, instantiated as:

$$\mu_{replace}(\overline{3}, f_{order})$$

where f_{order} is an alphabet order function, either `previous` or `next` of a specific letter. Note that `next`(“z”) = “a” and `previous`(“a”) = “z”.

Finally, in order to deal with the composition (c), we use the meta-operator $\mu_{condition}$ to generate operators in charge of inserting Boolean conditions (guards) to the rules. In this way, some parts of the sequences are handled differently. In order to distinguish different parts of the sequence we use the position relative to the length of the list. This makes it possible to learn problems with more than one pattern (for instance “abxcdx", where the following letter is “x” if the position of the missing letter is a multiple of 3, and the application of `next` to the last letter, otherwise). So we used the following two conditions.

$$length(L)\ mod\ \varphi = 0$$
$$length(L)\ mod\ \varphi \neq 0$$

where the period $\varphi \in \{2, 3, \ldots\}$. Both conditions are defined in the background knowledge as several functions mod_φ and not_mod_φ (respectively), which apply over the input letter sequence parameter in the lhs of the rules. Hence, the operators are defined as

$$\mu_{condition}(\overline{2}, mod_{\varphi})$$
$$\mu_{condition}(\overline{2}, not\_mod_{\varphi})$$

where $mod_{\varphi}(\rho) = length(\rho|_{1.1})\ mod\ \varphi = 0$ and $not\_mod_{\varphi}(\rho) = length(\rho|_{1.1})\ mod\ \varphi \neq 0$, generating as many operators as periods φ we have.

As we did for the RPM problems, we also need a generalisation operator that will be applied to generalise the input attribute, $\mu_{replace}(\overline{1.1}, \overline{V_{string}})$.

The following example illustrates how the operators are applied to solve the letter series problem $\mathtt{e}_{1,3}$:

$$\begin{aligned} op_3(op_2(op_1(\texttt{thurstone}(\text{“}cdcd\text{”}) \to \text{“}c\text{”}))) &\Rightarrow \\ op_3(op_2(\texttt{thurstone}(V_{string}) \to \text{“}c\text{”})) &\Rightarrow \\ op_3(\texttt{thurstone}(V_{string}) \to \texttt{init}(V_{string})) &\Rightarrow \\ \texttt{thurstone}(V_{string}) \to \texttt{last}(\texttt{init}(V_{string})) & \end{aligned}$$

where

$$\begin{aligned} op_1 &\equiv \mu_{replace}(\overline{1.1}, \overline{V_{String}}) \\ op_2 &\equiv \mu_{replace}(\overline{3}, f_{init_L}) \\ op_3 &\equiv \mu_{replace}(\overline{3}, f_{last_R}) \end{aligned}$$

It is easy to see that the example $\mathtt{e}_{1,3}$ follows a regular pattern just alternating the characters “c” and “d”, so the last rule obtained $\texttt{thurstone}(V_{string}) \to \texttt{last}(\texttt{init}(V_{string}))$ returns the right solution whatever the input is (it covers all seven training instances generated from example $\mathtt{e}_{1,3}$). For instance, if $V_{string} =$ “$cdcdc$”, then $\texttt{init}(V_{string}) =$ “$cdcd$” and $\texttt{last}(\texttt{init}(V_{string}) =$ “d”, which is the correct and general solution for all letter series that follow the same pattern as the previous example.

Results

gErl has been tested on the same 15 problems of the Thurstone Letter Series Completion task (Figure 7.6) from [SK63]. With the operators that were provided, gErl learns 14 of the 15 test sequences, as shown in Table 7.4. As we see in this table, the size of the solution (represented by $|o_{app}|$ or their sum when there are two rules) is not related to the difficulty humans find in these exercises (DiffH). In fact, there is no clear trend or correlation between any pair of the three last columns. If we take a look at the constructs, we see that all solutions use string operator constructs, such as *init* and *last*. Only problems 1, 3, 6 do not use the letter order (*next* and *previous*), which may explain why they are easy for humans. The use of composition does not seem to add too much complexity to humans, as problem 4, for instance, is easy for humans, and problems 8 and 10 are not very difficult. Finally, one of the most difficult problems for humans (15) is relatively short and simple in its functional representation. In fact, there seems to be some contamination in the degree of difficulty for humans, depending on which letters in the alphabet are used in each problem. It seems that problems that have patterns involving the first letters of the alphabet are easier for humans than those involving other letters. This possible explanation is not reflected by any of the computer programs used by Simon and Kotovsky, or by the way the problems are presented to gErl.

7.5 Discussion

In the previous section we have seen how several intelligence test problems are addressed by a general-purpose learning system, which uses a declarative, rule-based representation language for examples, patterns and operators. This representation language and the generality of the learning process (which does not have any hard-wired operator) make it explicitly how complex each pattern is and what operators are used for each problem. This provides useful information about the elements that each problem really requires: more computational power (in terms of working memory and combinatorial search) or some basic operational constructs. Many previous works in the literature that have analysed the complexity of intelligence test problems using computational models have focused on the former. Here, we are interested in the latter.

It should be noted that, when we analyse and determine the complexity of intelligence test problems based on problem-dependent characteristics rather than user-dependent ones, we should ensure that we generate and test all possible solution candidates (program outputs in order of their complexities) for a given problem until the minimal one (the shortest) is returned, while keeping the execution time of the solution under some reasonable terms. Actually, we consider a relation between space or length of the solution (L) and its execution time (T) as follows: $LT = L + \log(T)$. This is closely related to a Levin search [Lev73, Lev84] (see [LV08b] for an overview), which, for a broad class of search problems, can be shown to be optimal with respect to total search time (leaving aside a constant factor independent of the problem size). Despite this strong result, in practice, Levin search will fail to solve problems whose solutions all have a high value of LT, which is highly dominated by L. Because of this infeasibility of Levin search, we need to use an approximation. gErl is an approximation of this search. This, of course, might lead to a solution that is not the optimal in terms of LT, thus giving an overestimation of the difficulty

Id	Problem	Solution	Steps	E^+	$\lvert o_{app} \rvert$	DiffH	DiffC
1.	cdcdcdcd_	$\texttt{thurstone}(V) \to \texttt{last}(\texttt{init}(V))$	42	5	3	0.03	0.00
2.	aaabbbcccdd_	$\texttt{thurstone}(V) \to \texttt{next}(\texttt{init}(\texttt{init}(V)))$	91	9	4	0.08	0.25
3.	atbataatbat_	$\texttt{thurstone}(V) \to \texttt{last}(\texttt{init}(\texttt{init}(\texttt{init}(\texttt{init}(\texttt{init}(V))))))$	131	7	7	0.11	0.75
4.	abmcdmefmghm_	$\texttt{thurstone}(V)$ when $\texttt{length}(V)$ mod $3 = 0 \to \texttt{last}(\texttt{init}(\texttt{init}(V)))$	174	11	5	0.13	0.25
		$\texttt{thurstone}(V)$ when $\texttt{length}(V)$ mod $3 \neq 0 \to \texttt{next}(\texttt{next}(\texttt{init}(\texttt{init}(V))))$	174		6		
5.	defgefghfghi_	$\texttt{thurstone}(V) \to \texttt{next}(\texttt{init}(\texttt{init}(\texttt{init}(V))))$	105	8	5	0.40	0.75
6.	qxapxbqxa_	$\texttt{thurstone}(V) \to \texttt{last}(\texttt{init}(\texttt{init}(\texttt{init}(\texttt{init}(\texttt{init}(V))))))$	129	7	7	0.29	0.00
7.	aducuaeuabuafua_	-	-	-	-	0.59	1.00
8.	mabmbcmcdm_	$\texttt{thurstone}(V)$ when $\texttt{length}(V)$ mod $3 = 0 \to \texttt{last}(\texttt{init}(\texttt{init}(V)))$	165	8	5	0.27	0.25
		$\texttt{thurstone}(V)$ when $\texttt{length}(V)$ mod $3 \neq 0 \to \texttt{next}(\texttt{init}(\texttt{init}(V)))$	141		5		
9.	turtustuttu_	$\texttt{thurstone}(V)$ when $\texttt{length}(V)$ mod $3 = 0 \to \texttt{next}(\texttt{init}(\texttt{init}(V)))$	192	9	5	0.39	0.25
		$\texttt{thurstone}(V)$ when $\texttt{length}(V)$ mod $3 \neq 0 \to \texttt{last}(\texttt{init}(\texttt{init}(V)))$	185		5		
10.	abyabxabwab_	$\texttt{thurstone}(V)$ when $\texttt{length}(V)$ mod $3 = 0 \to \texttt{previous}(\texttt{init}(\texttt{init}(V)))$	182	9	5	0.24	0.50
		$\texttt{thurstone}(V)$ when $\texttt{length}(V)$ mod $3 \neq 0 \to \texttt{last}(\texttt{init}(\texttt{init}(V)))$	171		5		
11.	rscdstdetuef_	$\texttt{thurstone}(V) \to \texttt{next}(\texttt{init}(\texttt{init}(\texttt{init}(\texttt{init}(V)))))$	154	9	6	0.46	0.75
12.	npaoqapraqsa_	$\texttt{thurstone}(V)$ when $\texttt{length}(V)$ mod $3 = 0 \to \texttt{last}(\texttt{init}(\texttt{init}(V)))$	123	8	5	0.40	0.75
		$\texttt{thurstone}(V)$ when $\texttt{length}(V)$ mod $3 \neq 0 \to \texttt{next}(\texttt{init}(\texttt{init}(V)))$	141		5		
13.	wxaxybyzczadab_	$\texttt{thurstone}(V) \to \texttt{next}(\texttt{init}(\texttt{init}(V)))$	99	9	4	0.37	0.75
14.	jkqrklrslmst_	$\texttt{thurstone}(V) \to \texttt{next}(\texttt{init}(\texttt{init}(\texttt{init}(V))))$	103	9	5	0.32	0.75
15.	pononmnmlmlk_	$\texttt{thurstone}(V) \to \texttt{previous}(\texttt{init}(\texttt{init}(V)))$	112	9	5	0.51	0.75

Table 7.4: Solutions returned by gErl and steps needed to learn the series completion test problems from [SK63]. We see that the solutions can extrapolate the sequences indefinitely. Except for problem 7 all extrapolations are correct. We also show the number of examples (E^+) that are derived from each problem and the number of operators $\lvert o_{app} \rvert$ that are applied to get the solution in gErl (as a measure of the complexity of the solution). The last two columns (DiffH and DiffC) show the difficulty (on a scale between 0 and 1) as the proportion of humans that missed the solutions (among the 12+67 cases in [SK63, Table 4]) and among the 4 computer programs in [SK63, Table 3], respectively.

of the search problem.

A related issue is whether the solutions are efficient or get more and more efficient after use. This is not related to fluid intelligence but rather to crystallised intelligence. Given a solution or policy to address a problem (e.g., multiplication), the repeated use of the algorithm will lead to 'routinisation', i.e., a phenomenon in humans and non-human animals by which those mechanisms that have been so well mastered and regularly replicated no longer require mindful manipulation. A possible future work would be to analyse that, instead of the optimal solution in terms of *LT* (which is usually approximated as the shortest solution in the gErl representation), some program transformation techniques could be used to render it more efficient (but semantically equivalent). For instance, gErl may find that the solution for a problem is to sort a sequence of letters. That solution could be expressed with a very short, but inefficient, sorting algorithm. A further improvement —similar to a routinisation— could be to transform the sorting algorithm into another more efficient one.

The problems seen in the previous section feature the following operational constructs (in parenthesis I, II, II referring to odd-one-out, RPM and letter series problems respectively): differences intra and inter items (in I, II), the notion of being distinct (I, II), the concept of identity (I, II, III), the ideas of combination/addition (I, II, III), the sense of sequence and progression (II, III), the notion of order (III) and the notion of periodicity or repetition (III). It seems that if a system lacks many of these concepts, these problems become irresolvable (unless the system can *invent* or *discover* all these concepts). In fact, gErl is not able to solve them when we remove the constructs. On the other hand, if we artificially provide these constructs, even if the system, such as gErl, does not have a real cognitive development or physical embodiment, we can get excellent results.

A very different thing is how much time gErl takes to solve the problem *when it can solve the problem.* For instance, the Pearson correlation coefficient between the number of different operators $|o_{app}|$ (a measure of the complexity of the pattern) and the time gErl takes (in number of steps) is 0.907 for the RPM problems and 0.944 for the letter completion problems[5]. In other words, the time a problem requires does correlate with the combinatorial problem of using the operational constructs, but solving it or not correctly may mostly depend on the constructs. This suggests that many studies about intelligence tests are conflating two components of the difficulty of a problem: whether

[5]Since we only have two odd-one-out problem types and actually just two different learning problems, we do not have a meaningful value for the correlation in this case.

we have the pieces and how many pieces we need to combine. In fact, in cognitive development we are interested in the set of elements and constructs that evolves with time rather that the computational ability of combining them.

This leads to several observations. If we focus on the evaluation of cognitive development in humans, we see clearly that intelligence tests for small children usually differ in presentation but also in the constructs that are required. For instance, even the easy RPM matrices of the standard series for adult humans may be impossible for children under 6. Note that we are not referring here to psychomotor abilities but pure fluid ability, where the concepts of identity, difference, order, repetition, etc., may still be under development. Naturally, some of these constructs are acquired by interacting with the world and with the use of language, as many of them are not innate. As these cognitive constructs are useful in a wide variety of problems, fluid intelligence (and not only crystallised intelligence) increases when these constructs are developed or learnt.

If we focus on the evaluation of cognitive development in artificial systems, this work confirms that looking at the score of a (non-human) system on a human intelligence test can be very misleading, as already anticipated a decade ago by [SD03] and seen in Chapter 6. Unlike the other computer models solving IQ tests viewed in Section 7.3, gErl is the first system (apart from [SD03]) that is able to deal with more than one IQ test. But, unlike [SD03], the system has not been designed on purpose for intelligence tests, but rather as a general learning system. In other words, gErl *learns* to solve the problems. For instance, for the RPM problems, gErl had 59/60 hits on sets C, D & E. As humans who performed well on these sets typically got a perfect score in the easier sets A and B [RC96, Table SPM2], we *could* infer that gErl is in the 95^{th} percentile for American adults (IQ: 140), according to the 1993 norms [RC96]. This is not very meaningful, as we know that gErl has some learning abilities, but it is *not* really intelligent. We can always find explanations in the way the problems are represented (where the complex visual representation is simplified), but our results suggest that a better explanation is just to look at the constructs that are provided to the system.

Nonetheless, it has to be said that if we provide many constructs in a huge background knowledge base, the difficulty would then lie in designing the system in such a way that it can choose among them in an efficient way, one of the most challenging problems in AI and machine learning today. This suggests that artificial cognitive systems, if their computational power remains the same (no change of hardware or basic algorithms), should not become faster with time for those problems they were able to solve in previous cognitive stages.

In fact, it may still happen that efficiency can be degraded if the system has a much larger knowledge base so that retrieving the appropriate constructs becomes more difficult. This of course assumes that we evaluate the systems without over-training on the particular tasks, as the system can just change its contextual priorities if some kinds of exercises have been presented to the system immediately before the evaluation[6].

In general terms, for both humans and machines, are these (and other) human intelligence tests useful to evaluate cognitive development? As a take-away message, we do think that some of them *can* be useful. For the three kinds of problems we have seen here, the odd-one-out problem will have less interest than the two other problems, as we can only evaluate two particular constructs with the odd-one-out problem, and the constructs are relatively elaborated. In contrast, the other two types of problems (RPM and letter series) have a diversity of constructs. In order to make these intelligence tests more useful we need to create categories about the constructs they use. These categories can be created (less subjectively) by the use of declarative learning systems such as gErl. Also, we have to be very careful to separate success/failure with speed of resolution in the analysis of results, and the evolution of both speed and success for the whole cognitive life of the system. Let us remember that gErl is not a cognitive model and it is not based in how humans learn or develop (we will see our own developmental view for knowledge acquisition in the following chapter). When comparing systems (humans, machines or hybrid), we can potentially discover whether they share the same constructs or not, and identify whether the difference in speed and success is because a higher or lower computational power or the disposition and better handling of cognitive operational constructs.

7.6 Summary

In this chapter, we address several IQ tests problems (odd-one-out problems, Raven's Progressive Matrices and Thurstone's letter series) with a general-purpose learning system, the inductive programming system gErl, which is not particularly designed on purpose to solve intelligence tests. The goal is

[6]It is very difficult to expect that an underdeveloped subject (e.g., a child) could ever guess the solution for some of the problems that require complex operational concepts. This is related to the notion of 'spontaneous overtraining', that is able to get adult performance in preschoolers. It is also relevant to consider the studies about the blocking of some constructs, which can make adults perform like preschoolers [SS06]. One way of obtaining a similar result with artificial systems would be by giving or training the subject with the necessary constructs on one case and forbidding (or making forget) the constructs on the other case.

not to evaluate gErl but to use it as a tool to better understand the role of the mental operational constructs that are needed to solve these intelligence test problems. From here, we gain some insights into the characteristics and usefulness of these tests and how careful we need to be when applying human test problems to assess the abilities and mental development of robots and other artificial cognitive systems. The process and outcome of our study are as follows:

- We set a parallelism between the concepts of fluid and crystallised intelligence in humans and artificial systems.
- We explore fluid intelligence test tasks as a possible way to examine concept dependencies for the cognitive development of general intelligence in artificial systems.
- We use a general system, gErl, with intelligible representation, to examine three different IQ test problems: odd-one-out problems, Raven's Progressive Matrices and Thurstone's letter series.
- While gErl is, to our knowledge, the first general system that is able to score average human results in several IQ test tasks, we clearly make the point of how misleading a superficial *score* comparison is.
- Rather, we use the system to better understand what these IQ test tasks measure and what a system —human or artificial— requires to solve them, and whether an inability can be turned into ability through development.
- We distinguish two previously conflated aspects in item difficulty: the mental operational constructs that each task requires and the combinatorial problem of combining these constructs to find the solution.
- The take-away message is the appropriateness and care of using these and other general intelligence tests to evaluate concept dependencies of artificial systems.

There are of course some shortcomings about the use of human intelligence tests for machines, as already pointed out in [DHO12], but most criticisms are related to an anthropocentric choice of the exercises and an anthropocentric determination of their difficulty and scales. We have seen that through the use of general declarative systems where constructs, patterns and examples are explicit and intelligible, we can find non-anthropocentric criteria about what exercises we choose and what constructs and computational effort they require. This indicates that many tasks that have been devised in traditional intelligence tests can be *reused*. This is a compatible alternative to the conception of more principled and better grounded tests based on a mathematical basis [HO00a, LH07, HOD10] for both actual [HODHL14] and potential [HOD13] capabilities.

Other limitations are intrinsic to the work in this chapter. More tests could be analysed and other (preferably declarative) systems could be used to see whether the identification of required constructs is convergent with this work. Also, the ability of acquiring constructs (and consolidating them for future use or forgetting those useless) has not been analysed in this work and it is a key issue in any cognitive development. In the following chapter we will provide an insight about how this could be done.

In concluding, the study in this chapter supports the assumption that, even for fluid intelligence tests, the difficult items require a more advanced cognitive development than the simpler ones. This study also encourages the use of general intelligence tests for the evaluation of cognitive development of humans and machines, but with extra care when evaluating the latter. In contrast, many doubts have been raised about the use for artificial systems of the difficulty gradation for humans. Similarity, and the stage arrangement used for humans may not match with that of artificial systems, as the pace and sequence of acquisition of constructs may differ dramatically between humans and artificial cognitive systems.

8 Forgetting and consolidation in knowledge acquisition

In the previous chapter we saw how the cognitive development of intelligent systems —human or artificial— is explained by the development of cognitive capabilities (cognitive operational constructs) that are necessary to solve specific tasks. Although we used gErl in order to better understand the difficulty and the role of the cognitive constructs that are required in the process of learning a new task, the system has not a developmental nature: it has to be explicit in what constructs and operators must be given as background knowledge when addressing each problem. The ability of acquiring, consolidating and forgetting constructs fulfils the initially stated goal of properly representing, revising, evaluating, organising and retrieving knowledge in the quest for more developmental and general-purpose approaches in AI. In this chapter we present an incremental, lifelong view of knowledge acquisition which tries to improve task after task by determining what to keep, what to consolidate and what to forget. This approach is based in two characteristic features of intelligence that are essential for knowledge development: forgetting and consolidation. Both play an important role in knowledge bases and learning systems to avoid possible information overflow and redundancy. They are also useful to preserve and strengthen important or frequently used rules and remove (or forget) useless ones. The final aim is to overcome *The Stability-Plasticity* dilemma [CG88].

This chapter is organised as follows. Section 8.1 motivates this adaptive, developmental and cognitive view of knowledge acquisition and reviews what has been achieved in different AI areas related to this topic. Section 8.2 introduces the notion of Coverage Graphs, which defines the setting for a knowledge base. Over this Coverage Graphs, we are able to introduce an adaptation of the MML principle and related metrics in Section 8.3. Section 8.4 deals with knowledge structuring, how rules are forgotten, promoted and demoted. In Section 8.5 we include several experiments that illustrate how knowledge consolidation and forgetting works in practice. Finally, Section 8.6 closes the chapter with a brief summary and discussion.

These results have been published in [MFHR14, MFHR15a, MFHR15b].

8.1 Introduction

Memory in artificial systems is usually understood as an *encode-store-recall* structure where learnt knowledge is placed, remaining static until recall. However, memory (as storage) should be understood as an active cognitive component [WBB11], where the process of knowledge acquisition (automated process of abstracting knowledge from facts and other knowledge) cannot be understood as a naive accumulation of what is being learnt. New knowledge must be checked to see whether it is redundant, irrelevant or inconsistent with old one, and whether it may be built upon previously learnt knowledge. We argue that artificial intelligent systems should be developed for this purpose. This leads us to *the stability-plasticity* dilemma [CG88]. The basic idea is that an adaptive cognitive system must be capable of learning new things (plasticity) without losing previously consolidated learnt concepts (stability). This has been a designing principle within the perspective of neural computation over the last thirty years. Some of the proposed solutions include: (a) dual-memory systems simulating the presence of short and long-term memory [Fre97, AR97], and (b) cognitive architectures such as the *Adaptive Resonance Theory* (ART [Gro13]) emulating how the brain processes information. In both cases, those approaches can only incorporate new knowledge, without the ability of reorganisation or forgetting obsolete knowledge.

Therefore, with the purpose of overcoming the above dilemma and following the goals about developmental knowledge acquisition stated in the introduction of this thesis, we have conducted an approach able to (a) support incrementally knowledge acquisition without the need to have (one-shot) models discarded and retrained repeatedly (which is not cost-effective), (b) integrate inductive and deductive reasoning algorithms for such a goal and guided by knowledge evaluation metrics (thus having the knowledge integrated in the system rather than being an adjunct storage system), and, finally, (c) focus on relevant knowledge (or discard what is not) by the use of memory-based mechanisms that simplify the learning of new knowledge. These principles, being the starting point for a general framework for knowledge acquisition, aim at generating adaptive behaviour in intelligent learning systems based on previously acquired knowledge.

A crucial aspect relies on *theory and knowledge evaluation*. When the theory or hypothesis is considered as a whole and separated from the evidence, we have many well-founded proposals, such as the MML principle or the similar (but posterior) MDL principle seen in Chapter 4 (Section 4.2). However, for knowledge integration and consolidation it is necessary to assess each part of the theory since different parts of the theory can have different degrees of validity, probability or reinforcement [HO00b, HOGV00]. But even in this case, there is still a separation between knowledge and evidence. It would be meaningful to fully integrate knowledge and evidence into a hierarchical assessment structure (from specific facts to more abstract rules). The perspective of a network or hierarchy of nodes that get support from other nodes is more common in the area of link analysis (as we also saw in Chapter 4, Section 4.3) , or in infometrics.

8.1.1 Forgetting

Properly revising, evaluating, organising and retrieving knowledge in AI systems have much to learn from the study of human cognition and behaviour [RV08, EWB14, HT12, LCA14]. Cognitive factors, responsible for generating intelligent and adaptive behaviour, need to receive full consideration in order to improve current AI systems (not intelligent in human terms). In particular, there is a characteristic feature of intelligence that is essential for knowledge development: *forgetting*. Remembering absolutely everything prevents from having abstract thought (the process of generalisation). Forgetting can refer to a complete and irreversible elimination of significant old knowledge while learning new one; or it can denote that new learnt knowledge is not always kept in the working memory but abstractly encoded by identifying their relation to abstract concepts already present in the knowledge base. This latter definition is the desired one: forgetting should exist in knowledge bases and learning systems to avoid possible information overflow and redundancy, and in order to preserve and strengthen important or frequently used rules and remove (or forget) useless ones.

The ability to focus on what knowledge to discard is becoming more relevant not only in cognitive science and neuroscience [Qui12], but also in artificial intelligence (e.g., reasoning, planning, decision making). Usually considered in biology to be a combination of *decay* (to a lesser extent) and a proactive and retroactive *interference* (to a major extent) [Wix04], forgetting has been frequently used in AI systems because of performance and space constraints [NTHW10, AZM09]. Also known as *variable elimination*, forgetting has been widely investigated in the context of classical logic [LR94, LL03, LM10] and developed under the notion of logical equivalence. A similar approach but for reasoning from inconsistent propositional bases is proposed in [LM10]. Recently, the concept of forgetting has been widespread in other non-classical logical systems such as in logic programs [ZF06, EW08, WZZZ12] where a semantic forgetting is used to develop a number of criteria for forgetting atoms; in modal logic [ZZ09, SSLZ09, LW11], in description logics (DLs) [WWTP10] and in planning [EF07]. Note that the forgetting mechanisms used in the previous approaches are based either on decay or relevance-related measures rather than interference. This is due to the symbolic nature for representing the information and the encapsulation property of symbols. This makes a scalar comparison (i.e. determination of the degree of overlap between discrete components) difficult [WBB11]. We argue that this could be overcome by means of hierarchical assessment structures of knowledge (as commented before) where relations between different individuals (pieces of knowledge) may be better understood and measured.

Closely related to the above concept we found *memory consolidation*, namely, the neurological process of converting information from short-term memory into long-term memory. Some studies about episodic memory in humans [Sha01, tag08] claim that memory traces in the hippocampus are not permanent and are occasionally transferred to neocortical areas in the brain through a consolidation processes. Recent cognitive models of memory have given great importance to consolidation procedures [DS10].

The development of a new learning system for knowledge acquisition that is meant to be cumulative is not an easy task. As commented before. the need of making general principles that were available for any AI system (not only gErl) motivated the approach we present in this chapter. Indeed, we take a most general approach by considering that we start with an off-the-shelf *inductive engine* (e.g., a rule learning algorithm, an ILP system or an IP system) and an off-the-shelf *deductive engine* (e.g., a coverage checker, an automated deduction system or a declarative programming language) and, over them, we construct a lifelong knowledge acquisition system where the working memory follows the division imposed on human memory which separates processes for short and long-term recall [Heb49]. Several issues have to be addressed:

1. The inductive engine can generate many possible hypotheses and patterns. Once brought to the working memory (short-term memory system) we require metrics to evaluate how these hypotheses behave and how they are related to previous knowledge. Also, at any time new evidence can be added to the working space.

2. As working memory and computational time are limited, we need a forgetting criterion to discard some rules which are considered irrelevant (or redundant) in terms of informativeness. Forgetting should, therefore, follow a *proactive interference* principle (rather than decay or relevance) where new information may be more likely to be forgotten because of already existing information covering or overlapping the former.

3. The deductive engine checks the coverage of each hypothesis independently, using the background or consolidated knowledge as auxiliary rules, but not other working rules. As a result, only when new knowledge is consolidated (long-term memory system) we can use it for new problems or for more difficult examples of the same problem. This means that deduction is "modulo the background knowledge".

4. The promotion of rules into consolidated knowledge must avoid unnecessarily large knowledge bases and the consolidation of rules that are useless, too preliminary or inconsistent. That is, rules must be promoted and demoted to keep a powerful, but still manageable knowledge base.

The idea of a hierarchical knowledge arrangement (what we call Coverage Graphs) is used as the basis for structuring knowledge (and, thus, their relations) and is delegated to the deductive engine (the Coverage Graphs will be defined in the following section). The generation of new rules is delegated to the inductive engine. The crucial part is the definition of appropriate metrics to guide the way knowledge develops. Note that the use of both linking structures with complexity and compression metrics helps determine the degree of overlap between individuals. Therefore, the existing information can prevent the storage (consolidation) of already covered (overlapped) information.

8.2 Coverage graph

We consider that 'rules' are used for expressing examples, hypotheses and background knowledge. Rules are denoted by lower case Greek letters where, regarding examples, $class(\rho) = c$, $c \in C$ and C is the set of classes, such as $\{\text{false}, \text{true}\}$. The set of all possible rules is denoted by $\mathcal{R}$, where $W \subset \mathcal{R}$ is the working space or memory, and $K \subset \mathcal{R}$ is the background or consolidated knowledge base.

Rules are presented as vertexes or nodes V (and we refer them indistinctly) in a directed acyclic graph $G(V, A)$ (which is the DAG representation of a specific working space) we call *coverage graph* because the directed edges A

represent the coverage relation between the different rules. We denote an edge from node μ to node ν as $\mu \rightarrow \nu$. For simplicity, the *coverage graphs* do not include the edges for the transitive closure of the covering relation, i.e., if a node μ covers nodes ν and γ, but ν also covers γ, only the edges $\mu \rightarrow \nu$ and $\nu \rightarrow \gamma$ are included in the graph.

A rule ρ_a is covered by another rule ρ_b if $(K \cup \rho_b) \models \rho_a$. The precise understanding of the semantic consequence operator $\models$ will depend on the rule representation language used and the deductive engine. Hence, given an edge $\mu \rightarrow \nu$, we say that ν is directly covered by μ using K. The set of ancestors and successors of a node ν are defined as $anc(\nu) = \{\mu | \mu \rightarrow \nu\}$ and $suc(\mu) = \{\nu | \mu \rightarrow \nu\}$ (respectively). Also, we distinguish two subsets of nodes: $leaves$, nodes without successors ($|suc(\nu)| = 0$), where $leaves_c$ denotes the set of $leaves$ of class c; and $roots$, nodes without ancestors ($|anc(\nu)| = 0$).

Figure 8.1 shows an example of *Coverage Graph* of a well-known ILP problem [MD94]: the family relationship. In this problem, the task is to define the target relation $daughter(X, Y)$, which states that person X is daughter of person Y. W consists of three positive examples (rules 1, 2 and 5), two negative ones (rules 3 and 4), and seven selected rules that try to generalise and solve the problem (Table 8.1 right), whereas K is composed of the relations *female* and *parent* (Table 8.1 left). Note that the rules in K have not been included in the graph for clarity, although they belong to the initial "consolidated knowledge".

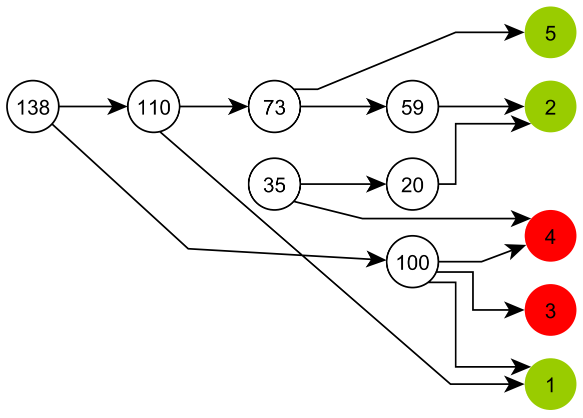


Figure 8.1: Coverage Graph *of the* family relations *problem. Green and red nodes refer to positive and negative examples respectively. The graph shows rule IDs according to Table 8.1.*

Background Knowledge		**Rules**	
ID	Rule	ID	Rule
k1	parent(ann, mary).	1	daughter(mary,ann).
k2	parent(ann, tom).	2	daughter(eve,tom).
k3	parent(tom, eve).	3	daughter(tom,ann).
k4	parent(tom, ian).	4	daughter(eve,ann).
k5	female(ann).	5	daughter(cris,tom).
k6	female(mary).	100	daughter(X,Y):- female(Y),parent(Y,mary).
k7	female(eve).	59	daughter(eve,tom):- female(eve),parent(tom,eve).
		20	daughter(eve,tom):- female(eve).
		35	daughter(eve,Y):- female(eve).
		73	daughter(X,tom):- female(X),parent(tom,X).
		110	daughter(X,Y):- female(X),parent(Y,X).
		138	daughter(V,W):- female(X),parent(Y,Z).

Table 8.1: Left: Background Knowledge for the family relations *problem. Right: Rules of this problem in* Prolog *notation.*

8.3 Basic Metrics for Acquired Knowledge Assessment

In order to select and arrange the set of rules in the working space, various measures of usefulness, relevance and consistency have to be derived from the *coverage graph*. Based on the idea that the relevance or usefulness of a rule can be stated by the relationship between its own complexity and the complexity of the rules it covers, a general criterion such as the *Minimum Message Length* [WB68a] can be used as a starting criterion from which to derive new metrics. For a motivation we refer the reader to the Chapter 4.

8.3.1 MML goes hierarchical: Support

Let us first restate some of the main points of the *Minimum Message Length* presented in Chapter 4 (Section 4.2) that will be useful to understand how we have adapted it to our Coverage Graphs. As we saw, MML is one of the most popular selection criterion in inductive inference where the model generating the shortest overall message (composed by the model and the evidence concisely encoded using it) is more likely to be correct. MML can be re-stated in a Bayesian form [WB68a] (Equation 4.1) which, in turn, by taking negative logarithms could be defined as the sum of three simple heuristics (Equation 4.2): a complexity-based heuristic (which measures the complexity of the hypothesis H), a coverage heuristic (which measures how much extra information is necessary to express the evidence E given the hypothesis H) and the length of the evidence ($L(E)$), equal for all competing hypotheses.

$$L(H|E) = L(H) + L(E|H) - L(E) \qquad \text{(4.2 revisited)}$$

By minimising equation 4.2 we maximise the posterior probability which involves searching for the model that gives the shortest message. To our knowledge, the MML principle has always been applied to select between hypotheses with respect to some given evidence. In our case, we have a coverage graph where rules cover other rules, so they become H and E at the same time. In a way, what we need is a hierarchical MML application. Having this in mind, the MML principle can be adapted to be used in this approach with the following considerations: instead of measuring the length of a hypothesis H given fixed evidence E, what we want to measure is the length of each rule ρ in W with respect to the rest of rules in W (which includes examples and hypotheses) because ρ can model not only examples, but also other rules. Therefore, $L(\rho|W)$ is defined as the sum of the length of ρ ($L(\rho)$), and the length necessary to express the rules in $\{W - \rho\}$ not modelled by ρ ($L(W|\rho)$), minus the length of the total rules in W ($L(W)$). Formally:

$$L(\rho|W) = L(\rho) + L(W|\rho) - L(W) \tag{8.1}$$

Apparently, it just seems a notational change wrt. equation 4.2. This is only true for the first term, which is estimated in the same way as the original MML principle. The term $L(\rho)$ can be defined in different ways depending on the rule representation language. For instance, if we are using logical or functional rules (as in the family example), we could use the following approximation. Given Σ a set of m_Σ functor symbols of arity ≥ 0, and $\mathcal{X}$ a set of $m_\mathcal{X}$ variables, we could define the length of a rule ρ containing n_Σ functors and $n_\mathcal{X}$ variables as

$$L(\rho) \triangleq m_\Sigma \log_2(n_\Sigma + 1) + \frac{m_\mathcal{X}}{2} \log_2(n_\mathcal{X} + 1) \tag{8.2}$$

Note that we promote variables over constants or functors.

Table 8.2 shows the length in bits and the class for the rules in the graph of Figure 8.1.

Following with the equation 8.1, we are going to reunderstand the terms $L(W|\rho) - L(w)$ to be adapted to *coverage graphs* and multiclass settings. Roughly speaking, these terms capture the "net profit" of the rules both in terms of support or coverage (length in bits of the rules covered). More formally, we define the support of a rule $\rho \in W$ as:

ID	$L(\rho)$	c
1	17.844	$\oplus$
2	17.844	$\oplus$
3	17.844	$\ominus$
4	17.844	$\ominus$
5	17.844	$\oplus$
100	11.977	
59	20.036	
20	11.591	
35	9.284	
73	13.114	
110	9.962	
138	12.462	

Table 8.2: Length and class c for the rules on the right side of Table 8.1. It should be noticed that only those rules representing the evidence have class values.

$$S(\rho, W) \triangleq L(\rho) - L(\rho|W) = L(W) - L(W|\rho) \tag{8.3}$$

where $L(W) - L(W|\rho)$ represents the coverage of a rule ρ expressed in bits, that is, the length of all the rules in W minus the length of the rules not covered by ρ. Therefore, the support of a rule ρ represents the length of the rules it covers:

$$S(\rho, W) = \sum_{\nu:\rho\models\nu} L(\nu) \tag{8.4}$$

leading to an alternative expression for $L(\rho|W)$ (equation 8.1) in terms of support:

$$L(\rho|W) = -S(\rho, W) + L(\rho) \tag{8.5}$$

which establishes that maximising $S(\rho, W)$ and minimising $L(\rho)$ we minimise $L(\rho|W)$ which involves searching for the rule ρ that covers the maximum number of rules and has the lowest length.

The following step is to adapt equation 8.5 to be used in *coverage graphs* that does not explicitly include the edges for the transitivity of the coverage relation. In order to consider the upwards propagation, only the *leaves* will have an initial support value which is equal to its length in bits, and the rest of nodes will distribute it recursively by propagating this support. Thus, the new support ($S'(\rho, W)$) adapted to work on *coverage graphs* is defined as:

$$S'(\rho, W) \triangleq \begin{cases} L(\rho) & \text{if } \rho \in \textit{leaves} \\ \sum\limits_{\nu \in suc(\rho)} S'(\nu, W) & \text{otherwise} \end{cases} \tag{8.6}$$

In order to avoid the scenario where the less grounded (upper) nodes get higher and higher support values, the support measure is required to satisfy a *conservative* condition. This property is somehow related to the law of conservation of energy, implying that at any node in a coverage graph, the sum of the total support flowing into that node is equal to the sum of the total support flowing out of that node.

Now, to make S' conservative we need to divide the support coming from the outcoming of a specific node ν by $|anc(\nu)|$ in order to equally distribute the support of ν between all of its ancestors.

Therefore, the new formula used to calculate the support of a rule ($\dot{S}(\rho, W)$) is defined to be equal to:

$$\dot{S}(\rho, W) \triangleq \begin{cases} L(\rho) & \text{if } \rho \in \textit{leaves} \\ \sum\limits_{\nu \in suc(\rho)} \frac{\dot{S}(\nu, W)}{|anc(\nu)|} & \text{otherwise} \end{cases} \tag{8.7}$$

and leading to an expression for $L(\rho|W)$ (8.5) in terms of this conservative support:

$$\dot{L}(\rho|W) = -\dot{S}(\rho, W) + L(\rho) \tag{8.8}$$

Equation 8.7 now accomplishes the mandatory conservative condition which could be stated such as the support of a node (which depends on its successors) has to be always entirely allocated in its ancestors together with the support inherited from other covered nodes (see Figure 8.2).

This implies (but not vice versa) that the total sum of the support of the *leaves* in the *coverage graph* is equal to the total sum of the support at the *root* nodes. Namely:

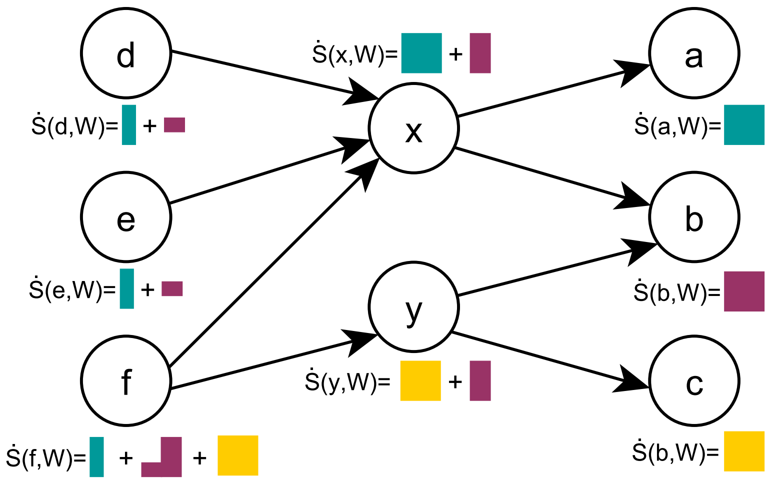


Figure 8.2: Graphical representation of the flow of the support by using equation 8.7 through the coverage graph*: the support of each* leave *node is always allocated in the* roots*. Therefore, the total support of the* leaves *is equal to the total support in the* roots

$$\sum_{\mu \in leaves} \dot{S}(\mu, W) = \sum_{\nu \in roots} \dot{S}(\nu, W) \tag{8.9}$$

For each *leaf* in the *coverage graph* we have n different paths whereby the support flows upwards to *root* nodes. Whenever a path is forked (an ancestor is found), its support is always divided by the number of the outgoing paths (incoming edges), having the ancestors an equally part of the support and thus having the roots a proportion of the original support of the leaves transitively covered by them. Therefore, if we assume that the total support at the roots is different from the total support at the *leaves*, it means that an external transfer of support (which comes from or goes to other sources) has happened. However, accordingly to equation 8.7, this is not possible and, therefore, the total sum of the support at the *roots* always remains constant and equal to the total support at the *leaf* nodes (see Figure 8.2).

Example 4

Viewed through the example in Figure 8.2 and accordingly to the equation 8.7 we have that the support of the *root* nodes is

$$\dot{S}(d,W) = \dot{S}(e,W) = \frac{\dot{S}(x,W)}{3}, \dot{S}(f,W) = \frac{\dot{S}(x,W)}{3} + \dot{S}(Y,W),$$

where

$$\dot{S}(x,W) = \dot{S}(a,W) + \frac{\dot{S}(b,W)}{2}, \dot{S}(y,W) = \frac{\dot{S}(b,W)}{2} + \dot{S}(c),$$

and

$$\dot{S}(a,W) = L(a), \dot{S}(b,W) = L(b), \dot{S}(c,W) = L(c)$$

thus making the following equations true (accordingly to the formula 8.7):

$$\underbrace{\dot{S}(a,W) + \frac{\dot{S}(b,W)}{2}}_{\dot{S}(x,W)} - \underbrace{\dot{S}(d,W)}_{\frac{\dot{S}(x,W)}{3}} - \underbrace{\dot{S}(e,W)}_{\frac{\dot{S}(x,W)}{3}} - \underbrace{(\dot{S}(f,W) - \frac{\dot{S}(y,W)}{2})}_{\frac{\dot{S}(x,W)}{3}} = 0$$

$$\underbrace{\dot{S}(c,W) + \frac{\dot{S}(b,W)}{2}}_{\dot{S}(y,W)} - \underbrace{(\dot{S}(f,W) - \frac{\dot{S}(x,W)}{3})}_{\dot{S}(y,W)} = 0$$

and, also, being the total support at leaf nodes $(L(a) + L(b) + L(c))$ equal to the total support at root nodes (equation 8.9):

$$\begin{aligned}
\dot{S}(d,W) + \dot{S}(e,W) + \dot{S}(f,W) &= (\frac{\dot{S}(x,W)}{3}) + (\frac{\dot{S}(x,W)}{3}) + (\frac{\dot{S}(x,W)}{3} + \dot{S}(y,W)) \\
&= S(x,W) + \dot{S}(y,W) \\
&= (\dot{S}(a,W) + \frac{\dot{S}(b,W)}{2}) + (\frac{\dot{S}(b,W)}{2} + \dot{S}(c,W)) \\
&= \dot{S}(a,W) + \dot{S}(b,W) + \dot{S}(c,W)
\end{aligned}$$

Finally, we need to take into account that, since the working space W can accommodate examples of different classes, we need our metric to distinguish between them and, hence, there are as many support values for each node as many different classes there are in the working space, each one holding the *conservative* property and formally defined to be equal to:

$$\dot{S}_c(\rho, W) \triangleq \begin{cases} L(\rho), & \text{if } \rho \in leaves_c \\ \sum\limits_{\nu \in suc(\rho)} \frac{\dot{S}_c(\nu, W)}{|anc(\nu)|} & \text{otherwise} \end{cases} \tag{8.10}$$

with Equation 8.8 being defined for classes as follow:

$$-\dot{L}_c(\rho|W) = \dot{S}_c(\rho, W) - L(\rho) \tag{8.11}$$

The value of $\dot{L}_c$ is interpreted as the hierarchical version of the MML principle, with $\dot{L}_c$ being the lower the better (and obviously $-\dot{L}_c$ the higher the better).

Following with the *Family* example, Table 8.3 shows the support and the negative form of $L(\rho|W)$ (for each class) of the rules in the graph in Figure 8.1.

ID	$L(\rho)$	c	$\dot{S}_+$	$\dot{S}_-$	$-\dot{L}_+$	$-\dot{L}_-$
1	17.844	$\oplus$	17.844	0.0	0.0	-17.844
2	17.844	$\oplus$	17.844	0.0	0.0	-17.844
3	17.844	$\ominus$	0.0	17.844	-17.844	0.0
4	17.844	$\ominus$	0.0	17.844	-17.844	0.0
5	17.844	$\oplus$	17.844	0.0	0.0	-17.844
100	11.977		8.922	26.766	-3.0549	14.788
59	18.791		8.922	0.0	-11.114	-20.036
20	11.591		8.922	0.0	-2.668	-11.591
35	9.284		8.922	8.922	-0.362	-0.362
73	13.114		26.766	0.0	13.651	-13.114
110	9.962		35.688	0.0	25.726	-9.962
138	12.462		44.61	26.766	32.147	14.303

Table 8.3: $\dot{S}$ and $-\dot{L}(\rho|W)$ values (both for the + and − classes) for the rules on the right side of Table 8.1. Taking a look at the table, we cannot decide which is the best rule in global terms: we can only establish a ranking per classes (by using the support values) without taking into account any other information.

8.3.2 Optimality

By using the support as the sole criterion to rank the rules in W is useful provided there are only rules belonging to one class. However, when there are more than one class in W, we need to consider the purity or confidence of the rules. In the same spirit of the MML principle, we define the optimality as the difference between the cost of coding a rule following equation 8.11 for a specific class and the cost of coding the exceptions, i.e.,: the support of the rules covered that belong to the other classes. We use a factor β indicating the relevance of rules being as pure as possible. Formally:

$$opt_c(\rho, W) \triangleq -\beta \cdot \dot{L}_c(\rho|W) - (1-\beta) \cdot \sum_{\substack{c' \in C \\ c' \neq c}} \dot{S}_{c'}(\rho, W) \tag{8.12}$$

leading to a *generic* optimality of a rule as:

$$opt(\rho, W) \triangleq \max_{c \in C}(opt_c(\rho, W)) \tag{8.13}$$

Following with the *family* example, Table 8.4 shows the optimality values per class (the generic optimality in bold) for the rules in the graph of Figure 8.1 using $\beta = 0.5$. According to these values, rule 110 is the most significant rule, as it can be easily viewed in the *coverage graph* because it covers all the positive examples and no negative one.

8.4 Structuring knowledge: forgetting, promotion and demotion

In our setting, rules are repeatedly generated by the inductive engine and added to the working space W. As an answer to the possible never-ending growth of W, it is necessary to have mechanisms for forgetting or revising useless pieces of acquired knowledge as we already said in the introduction of this chapter. Using the metrics we have just introduced, we need a mechanism to discard those rules that are not useful, are inconsistent or do not get enough support.

8.4.1 Forgetting mechanism

The optimality of a rule ρ is a core metric to determine its usefulness, but it is also important to see whether ρ could be considered superfluous because it

ID	$L(\rho)$	c	$\dot{S}_+$	$\dot{S}_-$	$-\dot{L}_+$	$-\dot{L}_-$	opt_+	opt_-
1	17.844	$\oplus$	17.844	0.0	0.0	-17.844	**0.0**	-17.844
2	17.844	$\oplus$	17.844	0.0	0.0	-17.844	**0.0**	-17.844
3	17.844	$\ominus$	0.0	17.844	-17.844	0.0	-17.844	**0.0**
4	17.844	$\ominus$	0.0	17.844	-17.844	0.0	-17.844	**0.0**
5	17.844	$\oplus$	17.844	0.0	0.0		0.0	-17.844
100	11.977		8.922	26.766	-3.0549	14.788	-14.91	**2.933**
59	18.791		8.922	0.0	-11.114	-20.036	**-5.557**	-14.479
20	11.591		8.922	0.0	-2.668	-11.591	**-1.334**	-10.256
35	9.284		8.922	8.922	-0.362	-0.362	**-4.642**	**-4.642**
73	13.114		26.766	0.0	13.651	-13.114	**6.825**	-19.939
110	9.962		35.688	0.0	25.726	-9.962	**12.863**	-22.825
138	12.462		44.61	26.766	32.147	14.303	**2.69**	-15.153

Table 8.4: Optimality values (both for the + and − classes) for the rules on the right side of Table 8.1. Bold values indicates the generic optimality (equation 8.13). Ranking the rules by optimality we see that the best rule is 110.

is covered (transitive or directly) by another rule of higher optimality. If it is the case, ρ is mostly redundant and it could be discarded safely. This idea leads to the following definition for the permanence of a rule:

$$perm_c(\rho, W) \triangleq opt_c(\rho) - \max(0, \max_{\nu:\nu\models\rho} opt_c(\nu)) \tag{8.14}$$

and for its *generic* permanence:

$$perm(\rho, W) \triangleq \max_{c\in C}(perm_c(\rho, W)) \tag{8.15}$$

The lower the value of permanence a rule has, the higher the odds it has to be forgotten.

When we perform a forgetting step, the coverage graph is affected and coverages are also affected. In order to keep as much information about the past support, each rule is provided with a trace of its old support. In cognitive systems this is associated to notions such as the preservation of belief and trust even if we forget the particular cases that gave support to a given statement. Therefore, the forgetting mechanism will work as follows:

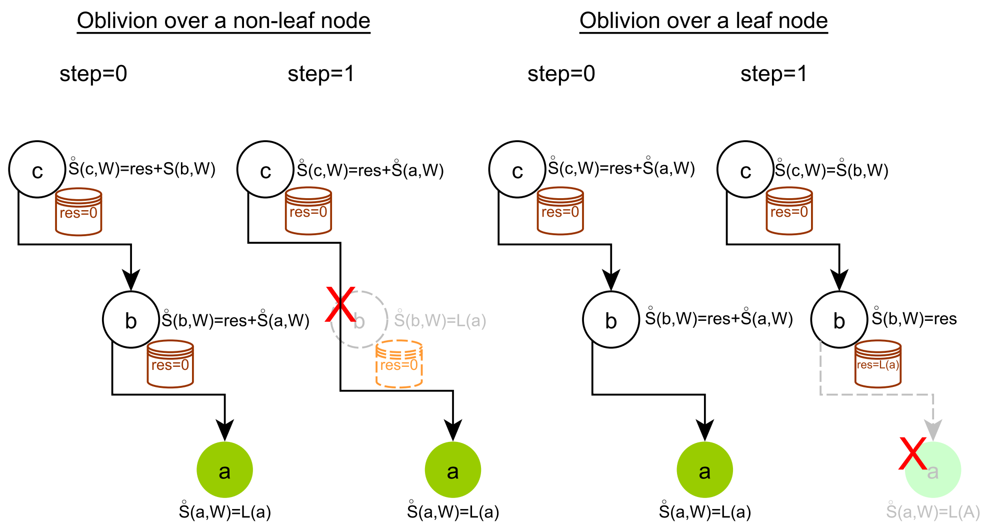


Figure 8.3: (Left) forgetting case (*a*)*: an internal node is forgotten (graphically represented with a red cross). (Right) forgetting case* (*b*)*: a* leaf *node is forgotten. Brown data storage cylinders graphically represent the concept of "residual" that collects the support (for each class) of forgotten* leaf *nodes.*

1. If a non-*leaf* node is selected to be forgotten, the support of its successors has to be re-distributed among their ancestors and the ancestors of the forgotten node (see Figure 8.3 (left)).

2. In case there is a forgetting step that removes a leaf node, its support has to be equally distributed among the rules that cover it which inherit it as their "residual" support value associated to each class c (res_c) (see Figure 8.3 (right)).

Hence, the equation 8.10 is modified to include the residual:

$$\mathring{S}_c(\rho|W) \triangleq \begin{cases} L(\rho) & \text{if } \rho \in leaves_c \\ res_c + \sum\limits_{v \in suc(\rho)} \frac{\mathring{S}_c(v,W)}{|anc(v)|} & \text{otherwise} \end{cases} \tag{8.16}$$

where res_c is initially set as 0. For each forgetting step, the support of forgotten nodes is distributed among the outcoming nodes increasing their res_c value, but if the last forgetting step removes a node without ancestor nor successors and a non-zero res_c, this value cannot be distributed and, therefore, is lost. This results in a decrease of the total support of the graph: although the support will remain conservative, the total amount will be lower than the total support of the *coverage graph* before the forgetting steps. Consequently, in the end some rules may have an under-estimated support value in terms of how many rules (of different classes) cover (see Figure 8.4).

In order to clarify how this mechanism works we illustrate this with the *Family* example. Figure 8.5 shows the evolution of the *coverage graph* in Figure 8.1 and its measures (see Table 8.5) through nine consecutive forgetting steps, where the rule with lowest permanence is forgotten in each step (shown with a grey square). For instance, in step 1, we see that rule number 59 is redundant because it is covered by a more significant rule (with ID 110), and it has the lowest value of permanence (see Table 8.5 (step 1)). Thus, rule 59 is forgotten, the *coverage graph* is redrawn (see Figure 8.5 (step 2)) and the metrics are recalculated if necessary (see Table 8.5 (step 2)). In step 2 (and other steps where a leaf node is deleted), its support is distributed equally among its ancestors and this distributed support becomes part of their residual or intrinsic support (res_c).

In this example, we have forgotten one rule at a time, but the actual pace and number of rules to forget can be tuned to the purpose of the system.

8.4.2 Consolidated knowledge: promotion and demotion

Finally, some of the rules with good indicators in the working space have to be eventually promoted to consolidated knowledge (or *belief*). This has to be a careful process, as the consolidated knowledge will be used by the deductive engine to calculate coverage. This means that an inconsistent rule that is promoted to the consolidated knowledge may have important consequences on the behaviour of the system.

The promotion function can be tuned for the application at hand, but a general choice is to use a threshold θ_p on the optimality to consolidate or promote a rule to a *belief* status in B.

When a rule is promoted to consolidated knowledge, it cannot be target of the forgetting mechanism and, hence, be forgotten. However, it may happen that this rule can be eventually removed from the consolidated knowledge. Therefore, the promotion system is mirrored by a demotion system, with the use of another threshold θ_d. The original background knowledge (B_0) cannot be demoted (and forgotten).

In the Example in Figure 8.5 in case we would establish θ_p equal to the average optimality of all the rules in the working space, all the rules that exceed this average value will be consolidated to the consolidated knowledge base

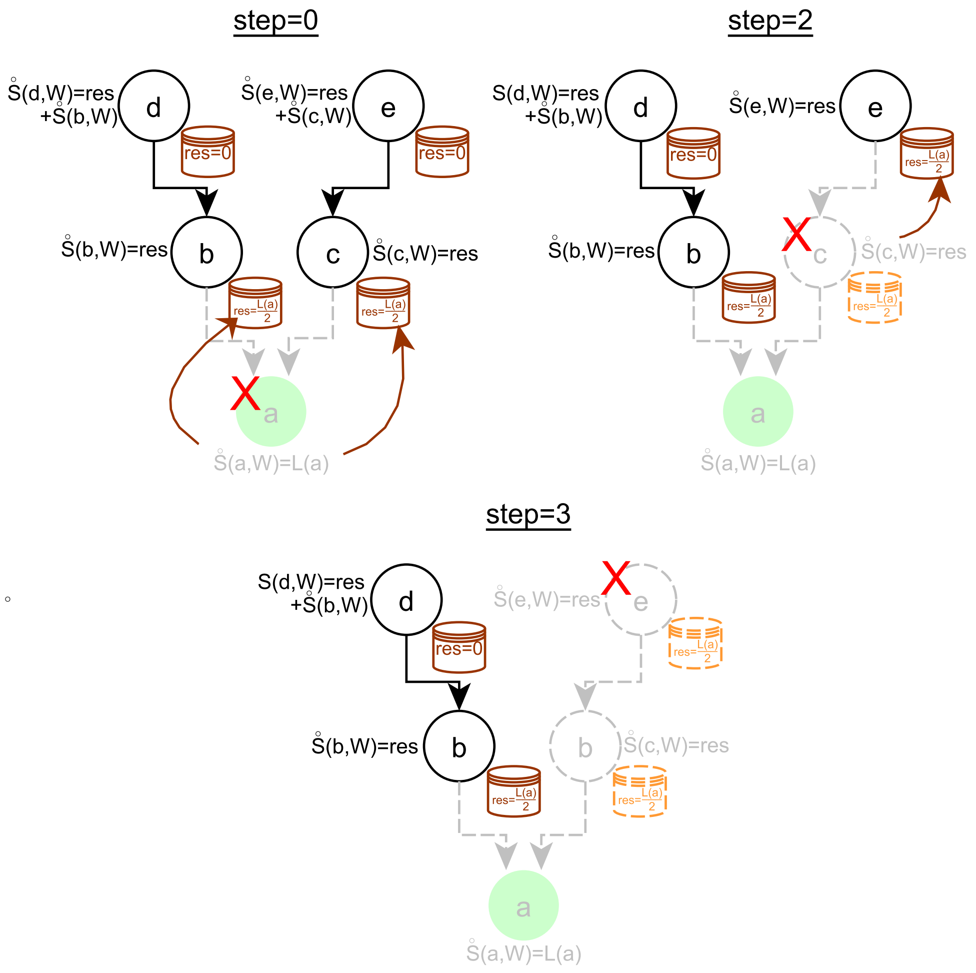


Figure 8.4: Forgetting mechanism performed over a complete branch. The conservative property over the support measure occurs in all steps, but the initial amount of support at step = 0 ($\sum_{\mu \in leaves} \mathring{S}(\mu, W) = L(A)$*) has been reduced at the last step = 4 (* $\sum_{\mu \in leaves} \mathring{S}(\mu, W) = \frac{L(A)}{2}$*) due to the forgetting mechanism.*

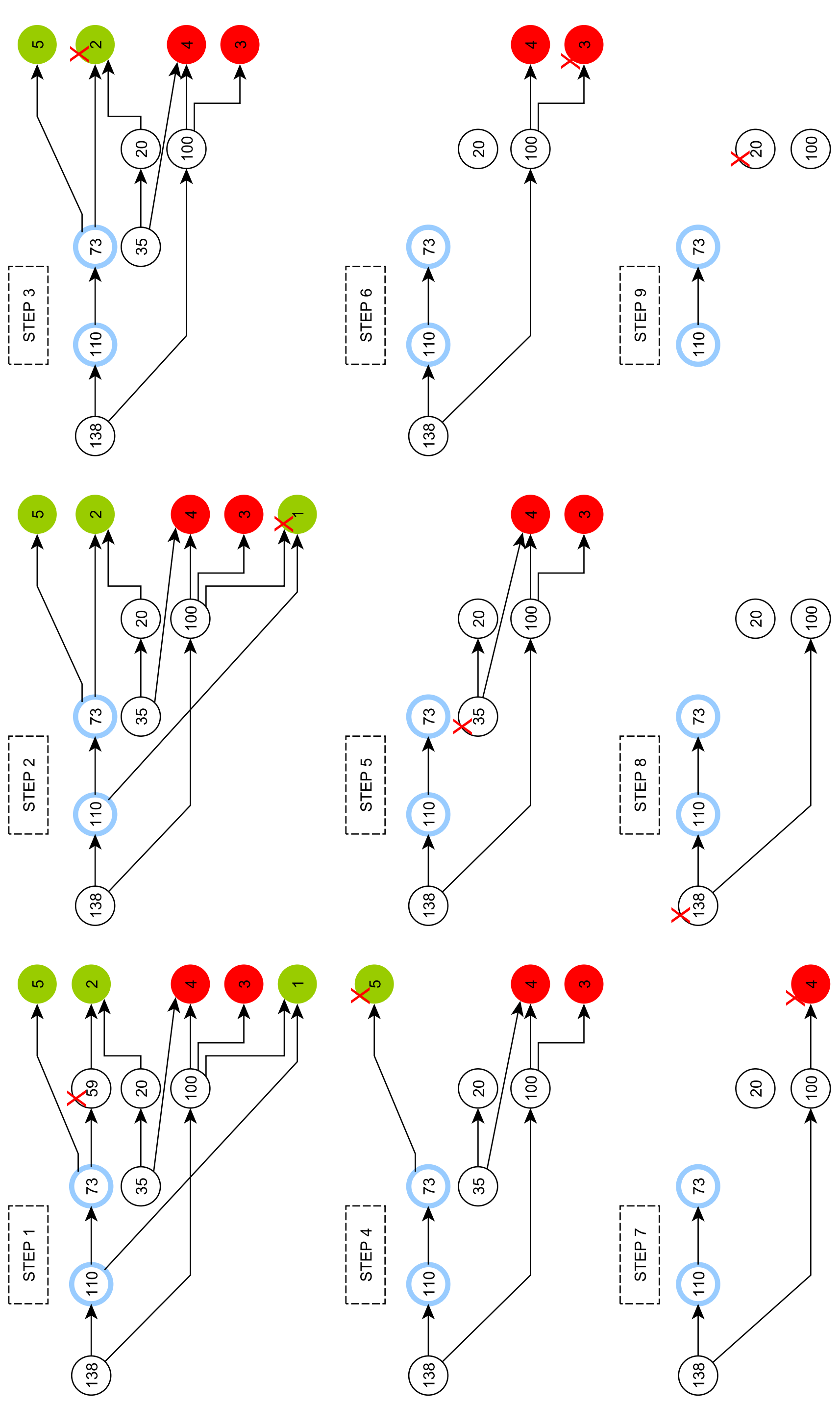


Figure 8.5: Coverage Graph of the family problem. Green and red nodes refer to positive and negative examples respectively. Nodes with a red cross represent the candidate rules to be forgotten. Nodes with a thick blue square represent those rules that have been consolidated.

		STEP 1							STEP 2							STEP 3						
ID	$L(\rho)$	$\dot{S}_+$	$\dot{S}_-$	$-\dot{L}_+$	$-\dot{L}_-$	Opt_+	Opt_-	$Perm$	$\dot{S}_+$	$\dot{S}_-$	$-\dot{L}_+$	$-\dot{L}_-$	Opt_+	Opt_-	$Perm$	$\dot{S}_+$	$\dot{S}_-$	$-\dot{L}_+$	$-\dot{L}_-$	Opt_+	Opt_-	$Perm$
1	17.844	17.844	0.0	0.0	-17.844	**0.0**	-17.844	-12.863	17.844	0.0	0.0	-17.844	**0.0**	-17.844	-12.863							
2	17.844	17.844	0.0	0.0	-17.844	**0.0**	-17.844	-12.863	17.844	0.0	0.0	-17.844	**0.0**	-17.844	-12.863	17.844	0.0	0.0	-17.844	**0.0**	-17.844	-12.863
3	17.844	0.0	17.844	-17.844	0.0	-17.844	**0.0**	-2.933	0.0	17.844	-17.844	0.0	-17.844	**0.0**	-2.933	0.0	17.844	-17.844	0.0	-17.844	**0.0**	-2.933
4	17.844	0.0	17.844	-17.844	0.0	-17.844	**0.0**	-2.933	0.0	17.844	-17.844	0.0	-17.844	**0.0**	-2.933	0.0	17.844	-17.844	0.0	-17.844	**0.0**	-2.933
5	17.844	17.844	0.0	0.0		**0.0**	-17.844	-12.863	17.844	0.0	0.0		**0.0**	-17.844	-12.863	17.844	0.0	0.0		**0.0**	-17.844	-12.863
100	11.977	8.922	26.766	-3.054	14.788	-14.91	**2.933**	2.933	8.922	26.766	-3.054	14.788	-14.91	**2.933**	2.933	8.922	26.766	-3.054	14.788	-14.91	**2.933**	2.933
59	20.036	8.922	0.0	-11.114	-20.036	**-5.557**	-14.479	-14.479														
20	11.591	8.922	0.0	-2.668	-11.591	**-1.334**	-10.256	-1.334	8.922	0.0	-2.668	-11.591	**-1.334**	-10.256	-1.334	8.922	0.0	-2.668	-11.591	**-1.334**	-10.256	-1.334
35	9.284	8.922	8.922	-0.362	-0.362	**-4.642**	**-4.642**	-4.642	8.922	8.922	-0.362	-0.362	**-4.642**	**-4.642**	-4.642	8.922	8.922	-0.362	-0.362	**-4.642**	**-4.642**	-4.642
73	13.114	26.766	0.0	13.651	-13.114	**6.825**	-19.939	-6.037	26.766	0.0	13.651	-13.114	**6.825**	-19.939	-6.037	26.766	0.0	13.651	-13.114	**6.825**	-19.939	-6.037
110	9.962	35.688	0.0	25.726	-9.962	**12.863**	-22.825	10.173	35.688	0.0	25.726	-9.962	**12.863**	-22.825	10.173	35.688	0.0	25.726	-9.962	**12.863**	-22.825	10.173
138	12.462	44.61	26.766	32.147	14.304	**2.69**	-15.153	2.69	44.61	26.766	32.147	14.304	**2.69**	-15.153	2.69	44.61	26.766	32.147	14.304	**2.69**	-15.153	2.69

		STEP 4							STEP 5							STEP 6						
ID	$L(\rho)$	$\dot{S}_+$	$\dot{S}_-$	$-\dot{L}_+$	$-\dot{L}_-$	Opt_+	Opt_-	$Perm$	$\dot{S}_+$	$\dot{S}_-$	$-\dot{L}_+$	$-\dot{L}_-$	Opt_+	Opt_-	$Perm$	$\dot{S}_+$	$\dot{S}_-$	$-\dot{L}_+$	$-\dot{L}_-$	Opt_+	Opt_-	$Perm$
1	17.844																					
2	17.844																					
3	17.844	0.0	17.844	-17.844	0.0	-17.844	**0.0**	-2.933	0.0	17.844	-17.844	0.0	-17.844	**0.0**	-2.933	0.0	17.844	-17.844	0.0	-17.844	**0.0**	-7.394
4	17.844	0.0	17.844	-17.844	0.0	-17.844	**0.0**	-2.933	0.0	17.844	-17.844	0.0	-17.844	**0.0**	-2.933	0.0	17.844	-17.844	0.0	-17.844	**0.0**	-7.394
5	17.844	17.844	0.0	0.0		**0.0**	-17.844	-12.863														
100	11.977	8.922	26.766	-3.054	14.788	-14.91	**2.933**	2.933	8.922	26.766	-3.054	14.788	-14.91	**2.933**	2.933	8.922	26.766	-3.054	14.788	-19.371	**7.394**	7.394
59	20.036																					
20	11.591	8.922	0.0	-2.668	-11.591	**-1.334**	-10.256	-1.334	8.922	0.0	-2.668	-11.591	**-1.334**	-10.256	-1.334	8.922	0.0	-2.668	-11.591	**-1.334**	-10.256	-1.334
35	9.284	8.922	8.922	-0.362	-0.362	**-4.642**	**-4.642**	-4.642	8.922	8.922	-0.362	-0.362	**-4.642**	**-4.642**	-4.642							
73	13.114	26.766	0.0	13.651	-13.114	**6.825**	-19.939	-6.037	26.766	0.0	13.651	-13.114	**6.825**	-19.939	-6.037	26.766	0.0	13.651	-13.114	**6.825**	-19.939	-6.037
110	9.962	35.688	0.0	25.726	-9.962	**12.863**	-22.825	10.173	35.688	0.0	25.726	-9.962	**12.863**	-22.825	10.173	35.688	0.0	25.726	-9.962	**12.863**	-22.825	12.863
138	12.462	44.61	26.766	32.147	14.304	**2.69**	-15.153	2.69	44.61	26.766	32.147	14.304	**2.69**	-15.153	2.69	44.61	26.766	32.147	14.304	**-1.77**	-10.691	-1.77

		STEP 4							STEP 5							STEP 6						
ID	$L(\rho)$	$\dot{S}_+$	$\dot{S}_-$	$-\dot{L}_+$	$-\dot{L}_-$	Opt_+	Opt_-	$Perm$	$\dot{S}_+$	$\dot{S}_-$	$-\dot{L}_+$	$-\dot{L}_-$	Opt_+	Opt_-	$Perm$	$\dot{S}_+$	$\dot{S}_-$	$-\dot{L}_+$	$-\dot{L}_-$	Opt_+	Opt_-	$Perm$
1	17.844																					
2	17.844																					
3	17.844																					
4	17.844	0.0	17.844	-17.844	0.0	-17.844	**0.0**	-7.394														
5	17.844																					
100	11.977	8.922	26.766	-3.054	14.788	-19.371	**7.394**	7.394	8.922	26.766	-3.054	14.788	-19.371	**7.394**	7.394	8.922	26.766	-3.054	14.788	-19.371	**7.394**	7.394
59	20.036																					
20	11.591	8.922	0.0	-2.668	-11.591	**-1.334**	-10.256	-1.334	8.922	0.0	-2.668	-11.591	**-1.334**	-10.256	-1.334	8.922	0.0	-2.668	-11.591	**-1.334**	-10.256	-1.334
35	9.284																					
73	13.114	26.766	0.0	13.651	-13.114	**6.825**	-19.939	-6.037	26.766	0.0	13.651	-13.114	**6.825**	-19.939	-6.037	26.766	0.0	13.651	-13.114	**6.825**	-19.939	-6.037
110	9.962	35.688	0.0	25.726	-9.962	**12.863**	-22.825	12.863	35.688	0.0	25.726	-9.962	**12.863**	-22.825	12.863	35.688	0.0	25.726	-9.962	**12.863**	-22.825	12.863
138	12.462	44.61	26.766	32.147	14.304	**-1.77**	-10.691	-1.77	44.61	26.766	32.147	14.304	**-1.77**	-10.691	-1.77							

Table 8.5: Metrics for the family problem in nine steps of forgetting. Identifiers refer to the rules in Table 8.1. Rows filled in blue refers to those rules which have been consolidated. Rows filled in orange (non-consolidated rules with lowest perm) refer to the rule candidate to be forgotten.

(rules 110 and 73). Any rule that is consolidated cannot be target of the forgetting mechanism until it is demoted to the working space again (considering a demoting threshold θ_d equal to θ_p). Thus, in Table 8.5 (step 5), rule 73 has the lowest permanence value ($perm(73) = -6.037$) but 35 ($perm(35) = -4.642$) would be forgotten instead, because the former is a consolidated rule.

8.5 Experiments

As mentioned in section 8.1, one of the issues in many cognitive systems (especially connexionistic, either artificial or biological) is the Stability-Plasticity dilemma. We claim that our approach is able to address this issue in a lifelong or incremental learning process. For this purpose, we have conducted an experimental evaluation to explore the following questions: (a) is it possible to gradually generate a large repository of consolidated knowledge assessing the usefulness of the rules? (b) is our approach able to forget or revise the existing knowledge in order to generate a rich and reusable knowledge base? and (c) how are the process and the resulting knowledge structure understood in terms of cognitive systems that must acquire and develop knowledge incrementally? We want to illustrate these features in one single domain (chess). The ultimate goal of these experiments is to see whether the framework is general enough to work with off-the-shelf inductive and deductive engines, to better understand how the metrics and procedures work, and finding whether they may require some tuning or improvement to the framework before addressing other problems.

8.5.1 Methodology

We will focus on the problem of learning the rules of chess by observation. In particular, we focus on learning a model of legal moves of different pieces from a set of legal and illegal move examples (extracted from [MBHMM89]). In our framework, the legal moves are the positive examples and the illegal moves the negative ones (so we have two classes). Each example represents a move of a specific piece on an empty board. Therefore, a move is represented by a triple from the domain $Piece \times Pos \times Pos$, where the second and third components represent, respectively, the piece's initial position and its destination on a chessboard. Positions are represented by a tuple from the domain $File \times Rank$ where files (a-h) stand for columns and ranks (1-8) stand for rows. For instance, Figure 8.6 illustrates all the possible moves of the knight from a specific initial position (k) to several other positions (k'). We will use a Prolog notation (see Table 8.6).

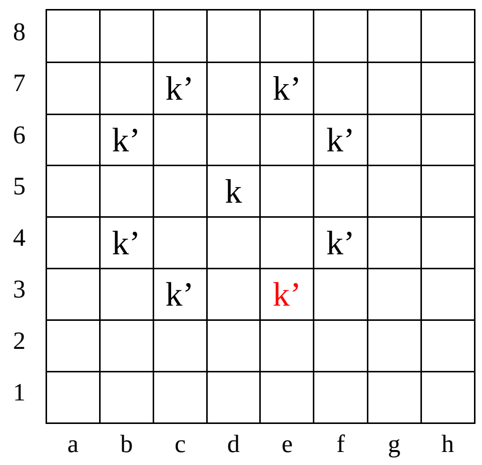


Figure 8.6: Possible moves of the knight from position (d,5). The particular legal move from k to k′ will be represented as `move(knight,pos(d,5),pos(e,3))`.

The only background predicate used is the absolute difference, *diff(X, Y)*, that calculates the distance between X and Y, where both X and Y can be ranks or files (see Table 8.6).

ID	Rule	ID	Rule
K1	project(a,1).	**K11**	rdiff(Rank1,Rank2,Diff) :- rank(Rank1), rank(Rank2), Diff1 is Rank1-Rank2, abs(Diff1,Diff).
K2	project(b,2).		
K3	project(c,3).		
K4	project(d,4).		
K5	project(e,5).	**K12**	fdiff(File1,File2,Diff) :- file(File1), file(File2), project(File1,Rank1), project(File2,Rank2), Diff1 is Rank1-Rank2, abs(Diff1,Diff).
K6	project(f,6).		
K7	project(g,7).		
K8	project(h,8).		
K9	abs(X,X) :- X>=0.		
K10	abs(X,Y) :- X<0, Y is -X.		

Table 8.6: Background knowledge for the chess problem.

The challenge we would like to face is knowledge discovery and acquisition in a progressive way from examples provided incrementally. A random set of chess moves from all chess pieces in the game except the pawn (rook, bishop, knight, queen and king) is given. This includes positive and negative examples (28 and 12 examples respectively). We also consider that an inductive engine is generating rules during the whole process (according to the working space and using the consolidated knowledge as background knowledge) and they are arriving to the system in a random order as well. In our case, we have taken the rules generated by the ILP system Progol [Mug95] (60 in total). How many examples and rules are given for each step of the system is defined following a geometric distribution. Formally, the probability that k examples (and similarly for rules) are given is $Pr(X = k) = (1 - p)^{k-1} \cdot p$ where k is $1, 2, 3, \ldots$ and p is the probability of success (we set it to 0.5). In order to better mimic a situation where the inductive engine can produce rules it has already generated (as otherwise we would need to keep trace of all this), it is more realistic to use this distribution with replacement. Similarly, as the same move can appear repeatedly in chess, we have also considered replacement for the set of examples.

Subjectively, in this experiment, we have set the consolidation criterion with a threshold of optimality greater than the average of the optimality value of the rules in W (provided that it is above the average optimality of the evidence[1]), namely,

$$opt(\rho, W) > max(0, \frac{\sum_{\nu \in W} opt(\nu, W)}{|W|}) \tag{8.17}$$

Furthermore, since we want the consolidated knowledge to represent legal chess moves, we have set the β parameter equal to 0.1 in Equation 8.12 with the aim of penalising those rules that are not pure.

8.5.2 Consolidation without forgetting

In a first experiment we try to show what would happen without applying the forgetting mechanism and check whether the MML-based measures work successfully for knowledge acquisition: are the final consolidated knowledge useful to solve the problem given the evidence? Figure 8.7 shows the evolution of the learning process during 500 steps. As no rules are forgotten, the rule population (dashed brown line) reaches its maximum value (100) and it stagnates ignoring any new evidence which arrives to the system (because they are already placed in W) from step 180 onwards. In this case we have assumed that all the evidence of the chess problem can be allocated in W, however it could be the case that all knowledge of a problem will not fit into W (memory

[1]It has no sense, for this problem, to consolidate rules with a optimality value lower than the optimality in average of the examples.

restrictions) thus collapsing with no improvement. The same applies to both the average optimality of all rules (dashed blue line) and the consolidated ones (dashed green line) which, since no more new rules are allocated into W, no further learning or knowledge improvement can take place. Table 8.7 shows the consolidated rules at step 500 where we can see that they almost represent all the legal chess moves (only two movements of the knight are missing in this set) and there is only one rule ($x20$) which, despite representing a legal move, does not completely generalise the movement of the piece (king). This is a good result as the working space is large enough to accommodate all these rules (and many other less significant rules). See Table C.1 (in Appendix C) and Figure C.1 for all the rules in W at step 500 and their coverage relations, respectively. The conclusion we can draw from these results is that the metrics used to measure the usefulness of the rules provide a guarantee of promoting those rules that, having the maximum compression, best describe the problem.

ID	Rule	$L(\rho)$	$\dot{S}_+$	$\dot{S}_-$	$-\dot{L}_+$	$-\dot{L}_-$	Opt_+	Opt_-	*Perm*
r15	move(rook,pos(A,B),pos(A,C)).	22.133	49.594	0	27.461	-22.133	**2.746**	-46.847	2.746
q19	move(queen,pos(A,B),pos(C,B))	13.214	27.052	0	13.838	-13.214	**1.383**	-25.668	1.383
q12	move(queen,pos(A,C),pos(A,D))	13.214	27.052	0	13.838	-13.214	**1.383**	-25.668	1.383
r16	move(rook,pos(A,B),pos(C,B)).	20.455	33.815	0	13.36	-20.455	**1.336**	-32.479	1.336
x18	move(king,pos(A,B),pos(C,D)) :- rdiff(B,D,1), fdiff(A,C,1).	34.275	40.578	0	6.303	-34.275	**0.63**	-39.947	0.63
x20	move(king,pos(A,B),pos(A,C)) :- rdiff(B,D,1).	22.918	27.052	0	4.134	-22.918	**0.413**	-26.638	0.413
x13	move(king,pos(A,B),pos(C,B)) :- fdiff(A,C,1).	22.918	27.052	0	4.134	-22.918	**0.413**	-26.638	0.413
q23	move(queen,pos(A,B),pos(C,D)) :- rdiff(B,D,E),fdiff(A,C,E).	28.993	32.462	0	3.469	-28.993	**0.346**	-32.115	0.346
b10	move(bishop,pos(A,B),pos(C,D)) :- rdiff(B,D,E), fdiff(A,C,E).	24.534	26.3	0	1.766	-24.534	**0.176**	-26.123	0.176

Table 8.7: Consolidated rules and metrics for the chess problem without *the forgetting mechanism at step* 500*. IDs in bold represent those rules that perfectly generalise the legal moves of the chess pieces.*

8.5.3 Consolidation with forgetting

After that, we repeat the same experiment, but using the forgetting mechanism. This tries to represent a situation where we have bounded resources, in this case a more limited working space, so it is necessary to forget rules in order to allocate new ones. What we want to show is that if our approach is able to find a solution to a certain problem without the use of the forgetting mechanism, a suitable (and possibly better) solution to the problem should exist having bounded resources and by using the forgetting mechanism. In order to do that, we have executed several configurations with varying maximum number of rules in the working space ($|W| \in \{(20, 30, 40, 50, 60, 70, 80, 90)\}$) and every time the limit is exceeded the forgetting process is launched, for-

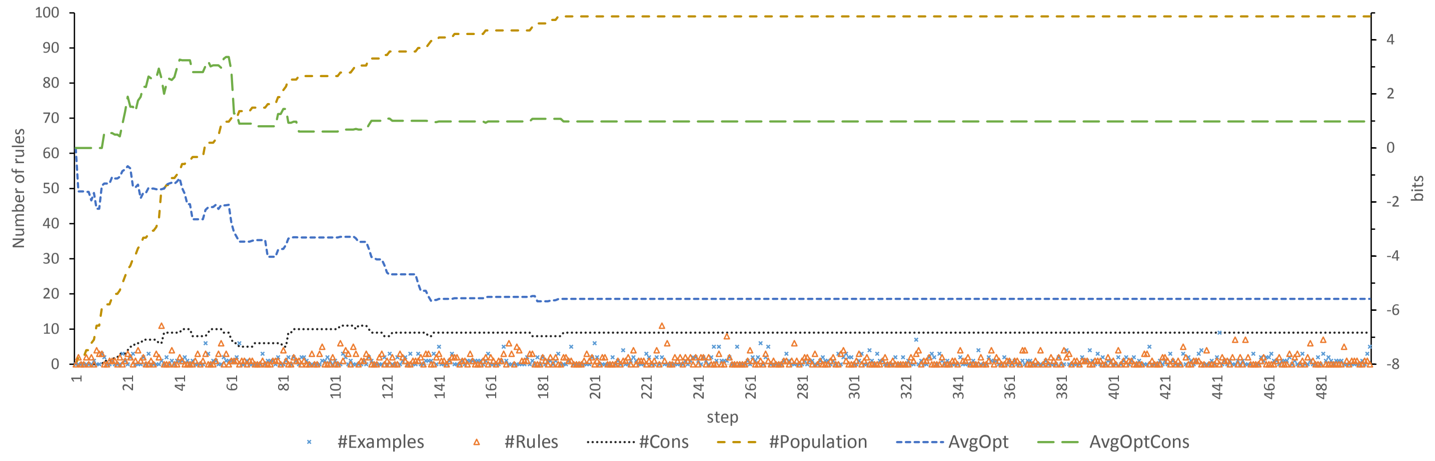


Figure 8.7: Evolution of some indicators for the chess problem without *the forgetting mechanism:* #Examples *and* #Rules *show the examples that arrive and the rules that are generated by the inductive engine for each step, #Cons shows how many rules there are in the consolidated knowledge (initially the background knowledge) and* #Population *shows the total number of rules (magnitudes shown on the left y-axis).* AvgOpt *and* AvgOptCons *show, respectively, the average optimality for all rules and the average optimality for all consolidated rules (magnitudes shown on the right y-axis). This figure shows how, after the working space is filled with all the evidence and generalised rules, the metrics become stable.*

		ρ																														
|*W*|	**Ob (%)**	**x13**	**x18**	**r15**	**r16**	**q12**	**q19**	**q23**	**b10**	**k22**	**k24**	x20	q11	q20	q21	q24	r7	r8	r9	r10	r14	r17	b5	b12	k11	k12	k14	k16	k17	k19	k25	x14
	25	1	4	5	3	6	7	5	8	5	3	7										1		1		1				1		1
20	50		1	4	3	6	3	3	6	3	1	4									2	2				1						1
	75			3	3	5	1	1	3	2		3					1					2			1	2						1
	25	3	10	6	4	9	9	9	10	9	8	9							1		1						1					
30	50	2	4	4	3	5	8	8	8	7	2	5									1											
	75	1	1	2	2	5	6	3	5	3	1	6			1						1	1										
	25	6	10	8	5	10	9	10	10	7	9	9																				
40	50	5	9	6	6	10	8	10	10	8	7	9							1													
	75	3	3	5	4	9	5	5	5	5	3	7											1			3		1				
	25	9	10	9	7	10	10	10	10	9	10	10	1																			
50	50	8	10	9	7	10	9	10	10	8	10	10	1																			
	75	5	7	6	6	9	6	8	10	7	5	9				1																
	25	9	10	8	7	10	10	10	10	10	10	10																				
60	50	10	10	9	8	10	10	10	10	10	10	10																				
	75	7	10	9	6	10	10	10	10	9	8	9									1				1		1					2
	25	9	10	7	5	10	10	10	10	10	10	2	3												4							
70	50	8	10	9	7	10	10	10	10	10	10	10	2												1							
	75	6	7	9	6	10	10	10	10	8	8	9																				2
	25	9	10	6	5	10	10	10	10	10	10	5	2	2																		
80	50	10	10	10	7	10	10	10	10	10	10	10	3	3						2	4											
	75	8	10	10	8	10	10	10	10	10	10	10								3	4											
	25	10	10	10	8	10	10	9	10	10	9	10	5												7							
90	50	10	10	10	8	10	10	10	10	10	10	10	4								4		4		6						1	
	75	10	10	10	9	10	10	10	10	10	10	10	4							2	4											
100	0	10	10	10	10	10	10	10	10			10																				

Table 8.8: Heat map showing the percentage of times a rule has been consolidated for each different configuration (rows: maximum number of rules, 20-100, percentage of rules forgotten, 25%-75%). Each cell represents 10 repetitions. The latter row ($|W| = 100$*) represents the reference solution, namely, the solution obtained by the experiment without forgetting (previous section). Rules (*ρ*) in bold (columns) are those rules that belong to the solution of the problem. As it can be seen, even with very limited resources, the consolidated knowledge improves the reference solution.*

getting up to 25%, 50% or 75% of the most meaningless rules (those with the lowest *perm* value). Each different configuration has been launched 10 times, hence, there are 240 executions in total.

Table 8.8 is a *Heat map* showing, for each possible configuration ($|W| \times forgetting(\%)$) how many times a specific rule appears in the consolidated knowledge in 10 repetitions, from white (0 times), light yellow (1 time) to dark green (10 times). Rules that are not represented in the Heat Map is because they have not been consolidated at any time. Knowing that the consolidated rules by the first experiment (Table 8.7) are those represented in the bottom row ($|W| = 100$), it is easy to see that not only the set of consolidated rules almost always includes the reference solution (even with very limited resources), but also the forgetting criterion allows the system to include those rules that perfectly generalise the moves of the king (rules in bold). The rest of rules included in the consolidated set in each experiment also generalise different movements of the pieces and, in some cases, they could disappear from this set by using a more restrictive consolidation criterion (i.e., by using the average of the optimality plus n times its standard deviation).

In order to compare both experiments, Figure 8.8 shows the evolution of the system during 500 steps for one representative setting of the 24 configurations (maximum number of rules equals to 60 and up to 50% of rules forgotten in each forgetting step). Now, the variations in the amount of consolidated rules (dotted black line) and rules in the working space (dashed brown line) allow us to observe how the forgetting mechanism works (every 30 steps approximately). Table 8.9 presents the consolidated rules at the final step (500) (see Table C.2 in Appendix C for all the rules in W at step 500). In this case, this set perfectly generalises all the legal moves of all the chess pieces. The system has reached a stable situation in which the number of consolidated rules (dotted black line) remains almost constant from step 250. The average optimality of both the consolidated rules (dashed green line) and all the rules (dashed blue line) have an increasing trend due to the distribution with replacement used to populate the working space. The appearance of new rules in the system or the execution of the forgetting mechanism mainly affect the average optimality of W (dashed blue line): every time it runs, the working space is cleaned of useless rules which strongly affects the metrics of the rules in W (and to a lesser extent to the consolidated set of rules (green line)) that have to be recalculated. Compared with the former experiment, the number of rules in W has been reduced (with one order of magnitude (10x) speedup in execution) obtaining a better set of consolidated knowledge: it includes all the rules that solve the chess problem, including the two legal moves of the knight, rules $k22$ and $k24$, which were missing from the consolidated knowledge in the first experiment.

8.5.4 Incremental knowledge acquisition

Finally, one last experiment tries to show the capability of our approach for the incremental learning of new knowledge from previously consolidated concepts. This experiment is divided in two phases: in the first one we have only taken rules and examples of moves of the rook and bishop chess pieces (15 and 30 rules respectively) providing the system with them in the same way as in the previous experiment. The consolidation criterion has not been changed, but the maximum number of rules in the working space has been established to

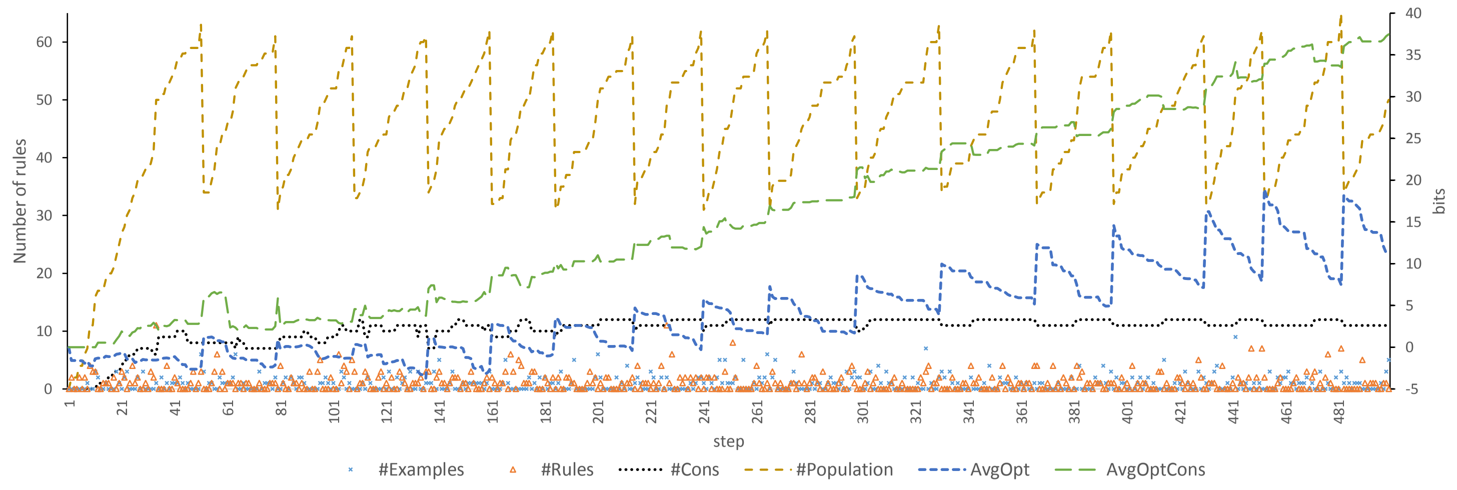


Figure 8.8: Evolution of the same indicators as in Figure 8.7 for the chess problem with *the forgetting mechanism (for a configuration with maximum number of rules 60 and up to 50% of rules forgotten for each forgetting step). Now we see a bumpier picture, where the forgetting mechanism takes place every 30 steps approximately.*

ID	Rule	$L(\rho)$	$\dot{S}_+$	$\dot{S}_-$	$-\dot{L}_+$	$-\dot{L}_-$	Opt_+	Opt_-	*Perm*
x18	move(king,pos(A,B),pos(C,D)) :- rdiff(B,D,1), fdiff(A,C,1).	34.275	662.774	0	628.499	-34.275	**62.849**	-599.924	-22.681
x13	move(king,pos(A,B),pos(C,B)) :- fdiff(A,C,1).	22.918	459.884	0	436.966	-22.918	**43.696**	-416.187	-41.834
k22	move(knight,pos(A,B),pos(C,D)) :- rdiff(B,D,2), fdiff(A,C,1).	34.275	446.358	0	412.083	-34.275	**41.208**	-405.149	29.394
q23	move(queen,pos(A,B),pos(C,D)) :- rdiff(B,D,E),fdiff(A,C,E).	28.993	437.34	0	408.347	-28.993	**40.834**	-396.505	-1.513
x20	move(king,pos(A,B),pos(A,C)) :- rdiff(B,D,1).	22.918	392.254	0	369.336	-22.918	**36.933**	-355.32	8.116
r15	move(rook,pos(A,B),pos(A,C)).	22.133	389.999	0	367.866	-22.133	**36.786**	-353.212	6.602
q19	move(queen,pos(A,B),pos(C,B))	13.214	365.202	0	351.988	-13.214	**35.198**	-330.003	-7.149
k24	move(knight,pos(A,B),pos(C,D)) :- rdiff(B,D,1), fdiff(A,C,2).	34.275	369.71	0	335.435	-34.275	**33.543**	-336.166	25.561
q12	move(queen,pos(A,C),pos(A,D))	13.214	311.098	0	297.884	-13.214	**29.788**	-281.309	-12.559
r16	move(rook,pos(A,B),pos(C,B)).	20.455	284.046	0	263.591	-20.455	**26.359**	-257.686	-3.824
b10	move(bishop,pos(A,B),pos(C,D)) :- rdiff(B,D,E), fdiff(A,C,E).	24.534	275.028	0	250.494	-24.534	**25.049**	-249.978	12.731

Table 8.9: Consolidated rules and metrics (as in Table 8.7) for the chess problem with *the forgetting mechanism at step* 500 *(for a configuration with maximum number of rules 60 and up to 50% of rules forgotten for each forgetting step). IDs in bold represent those rules that perfectly generalise the legal moves of the chess pieces.*

15 (in order to allow the forgetting mechanism to work) and the percentage of meaningless rules that are forgotten for each forgetting process up to 25%, due to the smaller size of the working set. In Table 8.10 we can see the set of consolidated rules after 100 steps. This set contains the rules that perfectly generalise all the legal moves of the rook and the bishop. In the first 100 steps of Figure 8.9 we can see how the forgetting and consolidation mechanisms work. This time, due to the lower maximum number of rules allowed in the working space, the lower percentage of rules forgotten and the geometric distribution used to provide the rules, the forgetting mechanism runs here every few steps, showing non-constant sawtooth-like wave ramps for the number of rules in the working system (dashed brown). However, the number of consolidated rules remains constant almost from step 45 to the end of this stage (100).

In the second phase, we provided the system with a new set of rules and examples (10 and 20 rules respectively) only representing moves of the queen chess piece. Apart from using the background knowledge that is provided initially, it should also be possible at this point to use the previously learned moves of the rook and the bishop in order to express the moves of the queen. This is what the inductive engine can take advantage of. Table 8.11 shows the set of consolidated rules which contains the previously consolidated rules that generalise the legal moves of the rook and bishop, and a new set of rules that represents the legal moves of the queen. This latter set includes a pair of rules ($q29$ and $q25$) that use the rook and bishop rules and represent all the possible moves of the queen piece: $q25$ which covers both the horizontal and vertical

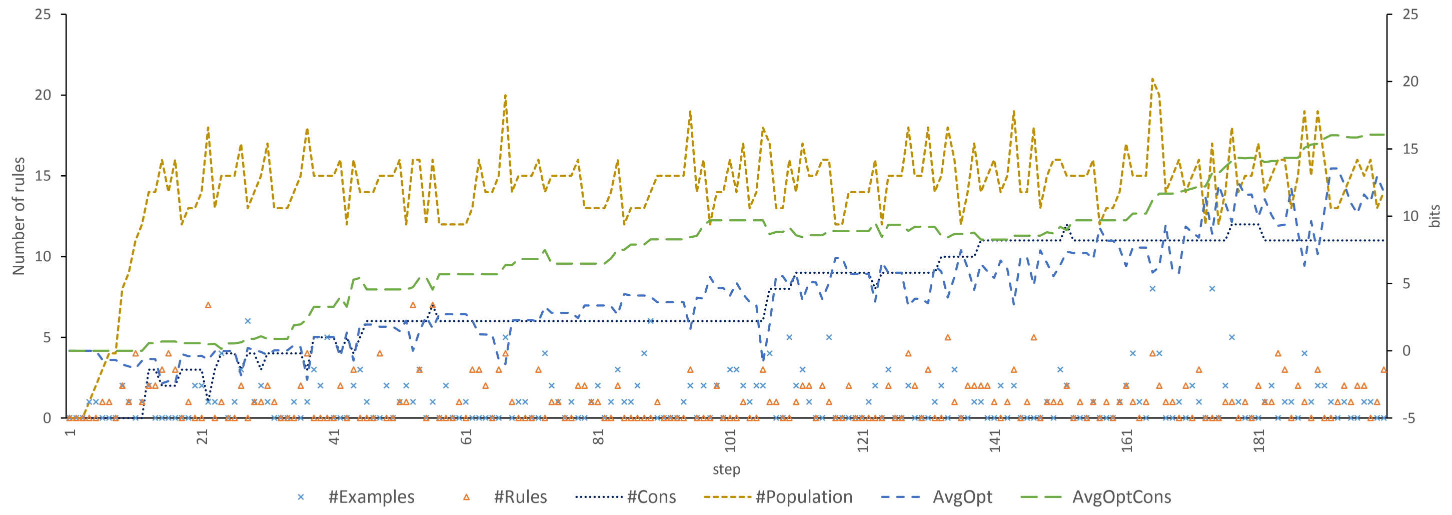


Figure 8.9: Evolution of the same indicators as in Figure 8.7 for the incremental chess problem (rook and bishop moves in the first 100 steps, and queen moves in the following 100 steps) with the forgetting mechanism. We see a non-constant sawtooth-like picture for the number of rules in the working space where the forgetting mechanism takes place every little number of steps due to the small amount of rules allowed and the low percentage of rules forgotten in every forgetting step. Nonetheless, the consolidated rules became constant in each different learning process.

moves of the queen; and $q29$ which covers the diagonal movement. The second half of Figure 8.9 (from step 100) shows how the forgetting mechanism runs even more frequently than previously (dashed brown) due to the increment of consolidated rules (that cannot be targeted by forgetting). Again, the number of consolidated rules (dotted black line) remains constant most of the time (from step 140 to step 200).

ID	Rule	$L(\rho)$	$\dot{S}_+$	$\dot{S}_-$	$-\dot{L}_+$	$-\dot{L}_-$	Opt_+	Opt_-	*Perm*
b10	move(bishop,pos(A,B),pos(C,D)) :- rdiff(B,D,E), fdiff(A,C,E).	24,534	590,635	0	566,101	-24,534	**56,61**	-534,024	24,904
r15	move(rook,pos(A,B),pos(A,C)).	22,133	277,282	0	255,149	-22,133	**25,514**	-251,767	25,514
r16	move(rook,pos(A,B),pos(C,B)).	20,455	223,178	0	202,723	-20,455	**20,272**	-202,905	20,272
r7	move(rook,pos(A,2),pos(B,2)).	19,718	175,838	0	156,12	-19,718	**15,612**	-160,226	-4,659
r14	move(rook,pos(A,2),pos(C,D)).	21,133	162,311	0	141,178	-21,133	**14,117**	-148,193	14,117
r9	move(rook,pos(a,B),pos(h,B)).	19,133	121,733	0	102,6	-19,133	**10,26**	-111,473	-10,011

Table 8.10: Consolidated rules and metrics (as in Table 8.7) for the chess problem (rook + bishop moves) at step 100. *All rook and bishop legal moves are covered by these rules and no better rules can be obtained.*

ID	Rule	$L(\rho)$	$\dot{S}_+$	$\dot{S}_-$	$-\dot{L}_+$	$-\dot{L}_-$	Opt_+	Opt_-	*Perm*
b10	move(bishop,pos(A,B),pos(C,D)) :- rdiff(B,D,E), fdiff(A,C,E).	34,275	590,635	0	566,101	-24,534	**56,61**	-534,024	56,61
q29	move(queen,pos(A,B),pos(C,D)) :- move(bishop,pos(A,B),pos(C,D))	34,275	432,832	0	405,116	-27,716	**40,511**	-392,32	40,511
q25	move(queen,pos(A,B),pos(C,D)) :- move(rook,pos(A,B),pos(C,D))	22,133	417,051	0	389,335	-27,716	**38,933**	-378,117	38,933
r15	move(rook,pos(A,B),pos(A,C)).	22,918	277,282	0	255,149	-22,133	**25,514**	-251,767	25,514
r16	move(rook,pos(A,B),pos(C,B)).	34,275	223,178	0	202,723	-20,455	**20,272**	-202,905	20,272
q12	move(queen,pos(A,C),pos(A,D))	13,214	173,583	0	160	-13,214	**16,036**	-157,546	16,036
r7	move(rook,pos(A,2),pos(B,2)).	28,993	175,838	0	156,12	-19,718	**15,612**	-160,226	-4,659
r14	move(rook,pos(A,2),pos(C,D)).	22,918	162,311	0	141	-21,133	**14,117**	-148,193	14,117
r9	move(rook,pos(a,B),pos(h,B)).	24,534	121,733	0	102,6	-19,133	**10,26**	-111,473	-10,011

Table 8.11: Consolidated rules and metrics (as in Table 8.7) for the chess problem (queen moves) at step 200 *(the* 100 *firsts steps for learning the rook and bishop moves, and the* 100 *following steps for learning the moves of the queen). All legal moves of the queen are covered by taking advantage of previously learnt moves of the rook and bishop, whose legal moves are also covered by this set.*

8.6 Summary

Learning a set of rules from data is nowadays a well-known problem for which many approaches exist, from data science to robotics. However, the use of background knowledge and the consolidation of new knowledge is one of the conspicuous problems in the understanding and creation of cognitive systems, and the management of more lifelong learning and knowledge acquisition systems. The organisation of complex knowledge structures in terms of Coverage Graphs allows for a straightforward and principled approach to knowledge acquisition, consolidation (promotion), revision (demotion) and forgetting. All this can be analysed at a meta-level, with the use of off-the-shelf deductive and inductive engines in charge of, respectively, establishing the relations between the different rules and the generation of new rules. This modularity, and the ability of dealing with declarative knowledge bases (logical, functional, algebraic, equational, grammatical, etc.) opens up a range of applications in learning, knowledge acquisition, developmental cognition, expert systems and other intelligent systems that are meant to have a non-ephemeral life. The main contributions of this chapter are:

- The first extension of the MML principle to a knowledge network (in the form of coverage graph). While the MML principle has a Bayesian inspiration, the metrics are more flexible than actual probabilities, stauncher when pieces of the working space are removed, and can be combined into metrics for different processes.

- We show that the development of a formal epistemological setting to realise how knowledge can be acquired, supports a constructive and developmental knowledge acquisition processes. In particular, we have seen how the forgetting criterion is not only necessary when the working space is finite but it can even be beneficial.

- Our approach is parametrisable to other cognitive or intelligent systems, as it works at a meta-level and is independent of the actual deductive and inductive mechanisms that are used underneath.

- The nonmonoticity problem of knowledge acquisition and revision is approached in a more lightweight and robust way, and the system can cope with redundancy, inconsistency or even uncertainty produced by conflict resolutions or complex semantic artefacts.

- *The stability-plasticity dilemma* has been addressed efficiently.

In an effort to facilitate an understanding of whether our approach is able to effectively and incrementally grow a knowledge base by using appropriate evaluation metrics and useful cognitive abilities for addressing the knowledge acquired, we have performed some experiments over a well-known scientific domain, the chess problem. As we have said, the ultimate goal is not to validate the approach but to provide some insight into both its generality, efficiency and the much-needed use of forgetting and consolidation cognitive procedures in incremental and developmental approaches for knowledge discovery.

9 Conclusions and Future Work

In this final chapter, we summarise the main contributions of this thesis and we point out several directions for further work.

9.1 Conclusions

When we talk about AI—the traditional and philosophical sense of AI—what comes into mind is the use of computer models to help us understand "intelligence" (more explicitly, "how intelligence works") and to figure out ways to make computers exhibit intelligent behaviour [Tur50b]. However, and although this stance prevailed in the beginning of the AI research, the AI community soon became more interested in the development of practical applications for solving particular tasks with no intention whatsoever of featuring intelligence. Up to date, the vast majority of the computer models are mindless rule-followers or cleverly written computer program doing statistical calculations and making predictions based on them. If we accept them as intelligent, what did that say about human intelligence? However, what it would mean for a computer to behave in an intelligent way? This thesis states that the answer lies in the construction of systems that go beyond task specific scenarios into more general-purpose ones.

Given the above challenge, we have strived to characterise a series of human intelligence attributes (incremental, developmental and lifelong learning) and cognitive-oriented procedures that, combined with the use of symbolic AI and symbolic learning, have helped us to develop a general-purpose learning approaches as well as a knowledge handling and assessment tools. Furthermore, we have analysed the use of more ability-oriented evaluation techniques for AI (such as intelligence tests) which has allowed us to have a better understanding about what intelligence and mental development is (both in humans and AI systems) and how it can be assessed. In what follows we will review what has been done with regard to the goals presented in the introduction of this thesis.

The first goal advocated for the construction of a general-purpose declarative learning approach. For that we outlined several desirable characteristics in terms of expressiveness and generality or versatility. In this regard, we have also seen that learning from small data is interesting and useful, and also that it can be difficult if the data is complex, especially if we have an expressive language and a rich deep knowledge. Taken all of this into account we decided to start working with approaches that are inherently general. We reflected that symbolic knowledge representation approaches (featuring higher-order functions, types and powerful abstraction mechanisms) and symbolic learning make it possible to not only deal with rich data environments, but also the emergence of powerful and complex constructs. Furthermore, constructs are understandable and facilitate to have an insight into the nature of the application data. Given the flexibility of using a wide variety of inductive operators (not relying thus in a fixed library of constructs and concepts in the background knowledge) we have seen how a learning system is able to operate in a wide variety of contexts. Finally, we have also adopt the idea that for the construction of more general and adaptive AI systems, it is useful to use approaches that are not explicitly programmed to achieve goals, but trained to do things. In this way we have seen that a reinforcement learning approach is appropriate for this purpose.

The second goal has to do with ability-oriented evaluation aspects of general-purpose learning systems. We have seen how the use of intelligence tests has become widespread as testbeds for experimentation, but not so as regular tools for AI evaluation. However, the use of intelligence tests for AI evaluation has provided very insightful information about what intelligence tests measure and what they do not and, ultimately, about what characterises intelligence in humans. Additionally, we have witnessed that even for supposedly general tasks that are designed for evaluation, many approaches have the (understandable) tendency to specialise to the task and hard-wire parts (or most) of the solution. Therefore, with the goal of making intelligence tests useful evaluation tools for AI, this thesis has claimed that several things must be considered. This includes, among other characteristics, a broader possible range and large number of (non public) tests, the generation of brand-new problems thus trying to be as unexpected as possible, the use of different presentations and difficulty levels, etc.

The third goal was related to the concept dependencies of AI systems and was motivated by both previous goals. We have concluded that the evaluation of a general-purpose learning with intelligible inductive operators by using intelligence tests can be useful to examine concept dependencies (mental operational constructs) in the cognitive development of artificial systems. That has opened a new way of looking at artificial cognitive development. We concluded that a superficial score comparison is misleading (the system is not really intelligent), but this kind of evaluation provides useful information about the constructs that each problem really requires. We also suggested that when comparing systems (humans, machines or hybrid), we can potentially discover whether they share the same constructs or not, and identify whether the difference in speed and success is because a higher or lower computational power or the disposition and better handling of cognitive operational constructs.

The final goal put emphasis on the idea that learning and knowledge acquisition in general-purpose systems should follow a cumulative and developmental nature. Again we claimed that properly representing, revising, evaluating, organising and retrieving knowledge is key to this end. In this sense we have presented a parametrisable approach which is able to deal with several types of declarative knowledge bases. From our studies we have concluded that the use of complex knowledge assessment structures jointly with information theory-based principles (to characterise knowledge) allow for a straightforward and principled approach to knowledge acquisition, consolidation (promotion), revision (demotion) and forgetting. Furthermore, we have seen how the forgetting criterion is not only necessary when the working space is finite (bounded resources) but it can even be beneficial. In summary, the problem of knowledge acquisition (and thus the *stability-plasticity dilemma*) has been addressed efficiently in a lightweight and robust way.

9.1.1 Contributions

In short, the main contributions of this thesis are:

1. **gErl as a general and declarative rule-based learning system**
   Vindicating the use of symbolic knowledge representation paradigms and symbolic learning approaches, we have shown that more general systems can be constructed by not only giving power to data and background knowledge representation but also to a flexible operator redefinition and the reuse of heuristics across problems and systems. This flexibility also carries a computational cost. In order to address this issue we rely on two compatible mechanisms. The first is the definition of customised operators, depending on the data structures and problem at hand, done by the user, using a functional language for expressing operators. The second mechanism is the use of generalised heuristics, since the use of

different operators precludes the system from using specialised heuristics for each of them. The choice of the right pair of operator and rule has been reframed as a decision process, as a *reinforcement learning* problem. Therefore, not only is this a novel approach, but also allows us to better understand the role of operators and heuristics in machine learning. By performing a series of illustrative experiments with our latest system version we have seen where the flexibility stands out, since gErl is able to solve a wide range of problems (from recursive ones to several IQ tests).

2. **Analysis of Intelligence tests for AI evaluation**
We have analysed over 30 papers featuring computer models addressing intelligence test problems, thus providing relevant insights for psychometrics, cognitive science, and artificial intelligence. We have studied and characterised each system by their relationships, the range of intelligence test tasks they address, the purpose of the models, how general or specialised these models are, the AI techniques they use in each case, their comparison with human performance, and their evaluation of item difficulty. Through this analysis we have realised that those systems have different purposes and applications: to advance AI by the use of challenging problems (this is the Psychometric AI approach), to use intelligence test for the evaluation of AI systems, to better understand intelligence tests and what they measure (including item difficulty), and, finally, to better understand what human intelligence is. Furthermore, we have seen that these systems systematically ignore results and ideas already present in previous related approaches specialising to the task and, therefore, losing the opportunity to understand what a computer model passing an intelligence test really means. Our aim here has also been both to encourage any future computer model taking intelligence tests to link with and build upon previous research, and to contribute to a more widespread realisation that more general classes of problems are needed when constructing new benchmarks for AI evaluation.

3. **Concept dependencies in artificial systems**
We have focused on whether we can assess the concept dependencies of an artificial cognitive system during learning, and whether we can use human intelligence tests for this. We have seen how several intelligence test problems (odd-one-out problems, Raven's Progressive Matrices and Thurstone's letter series) are addressed by our general-purpose learning system gErl, which, although lacks any mental epigenetic development

and physical embodiment and it is not particularly designed on purpose to solve intelligence tests, is able to perform relatively well for this kind of tests. gErl makes it explicitly how complex each pattern is and what operators are used for each problem due its symbolic and declarative nature: rule-based representation language for examples, patterns and operators. This provides useful information about the role of the cognitive operational constructs that are needed to solve a problem or task. Therefore, the goal has not been to to evaluate gErl but to use it as a tool to gain some insights into the characteristics and usefulness of these tests and how careful we need to be when applying human test problems to assess the abilities and cognitive development other AI systems. We do think that, in general terms, for both humans and machines, human intelligence tests are useful to evaluate cognitive development through the diversity of cognitive operational constructs required, therefore supporting the assumption that, even for fluid intelligence tests, the difficult items require a more advanced cognitive development than the simpler ones.

4. **Developmental and lifelong view of knowledge acquisition**
We have devised the problem of development as a knowledge acquisition approach that is incremental and cumulative, where new learnt knowledge should be checked to see whether it can be considered redundant, irrelevant or inconsistent with the old one, and whether it may be built upon previously acquired knowledge. Therefore, we have presented an incremental, lifelong view of knowledge acquisition which tries to improve task after task by determining what to keep, what to consolidate and what to forget. This approach is designed to combine any rule-based inductive engine with a deductive engine (is, therefore, parametrisable to other cognitive or intelligent systems) and integrates them into a lifelong learner through the use of a hierarchical knowledge assessment structure (Coverage Graphs) and by introducing several MML-based [WB68a] metrics. Therefore, given a lifelong learning problem, our approach is able to discover and develop knowledge incrementally by means of assessing the usefulness of the rules and gradually generating a large repository of consolidated knowledge where the knowledge is revised in order to generate a rich and reusable knowledge base. Particularly, we have analysed how appropriate these cognitive mechanisms are in order to deal with declarative knowledge bases in intelligent systems that are meant to have a non-ephimeral life. This complex knowledge organisation and assessment mechanisms allows for a straightforward and principled approach

to knowledge acquisition, consolidation (promotion), revision (demotion) and forgetting. We, then, show that the development of a formal epistemology to support knowledge discovery, in terms of how the knowledge can be acquired and justified, supports a constructive and developmental knowledge acquisition processes. Furthermore, the nonmonoticity problem of knowledge acquisition and revision has been approached in a more lightweight and robust way, and the system can cope with redundancy and even inconsistency without heavy conflict resolutions or complex semantic artefacts. Finally, we claim that our approach is a favourable compromise to *The Stability-Plasticity* dilemma [CG88].

Summing up, this dissertation represents one step forward in the hard and long pursuit of making more general AI systems and fostering less customary (and challenging) ability-oriented evaluation approach. Comprehensibility, expressiveness, higher-order features, incrementality and developmental knowledge discovery are all desirable features for this general-purpose AI development, apart from the requirement of accurate, effective and meaningful ability-oriented ways for evaluating its progress. For this purpose we have integrated different topics both within and outside AI, such as machine learning, inductive programming, reinforcement learning, cognitive science and psychometrics. From a methodological point of view, we have considered some conceptual developments with systematic empirical evidence.

9.2 Future Work

There are several interesting future lines of research. Let us enumerate some of them:

1. **gErl as a testbed for general-purpose learning research**
   Overall, we are conscious that our general-purpose learning system gErl entails some risks, since a general system which can be instantiated to behave virtually like any other system by a proper choice of operators is an ambitious goal. We think that for complex problems that cannot be solved by the system with its predefined operators, the system can be used to investigate which operators are more suitable. In more general terms, this can be used as a system testbed, where we can learn and discover some new properties, limitations and principles for more general machine learning systems that can be used in the future. There are also many other things to explore in the context of gErl. Regarding policy reuse for transferring learning between problems, we think that a way

(or measure) of similarity between problems would help us to better understand when the system is able to detect these similarities, from the point of view of better assessing the achievements of the system. Finally, while we have focused on the system gErl, we think that many of the principles could also be applied to other kinds of systems, most especially learning classifier systems, reinforcement learning and other evolutionary techniques.

2. **Further approaches for acquisition, structuring and development of operational cognitive constructs**
   We have seen that through the use of our general declarative system gErl, where constructs, patterns and examples are explicit and intelligible, we can find non-anthropocentric criteria about what human intelligence tests for machines choose and what constructs and computational effort they require. However, our study has several intrinsic limitations. Other (perhaps non-anthropocentric) tests could be analysed and other (preferably declarative) learning systems could be used to see whether the identification of required constructs is convergent with our study. Furthermore, much more effort should be placed to derive tests that can evaluate what happens when the background knowledge grows significantly, namely, how concepts, constructs and policies are structured and organised as a whole, how this knowledge relates to the notion of difficulty and, finally, how subjects effectively generalise similar concepts.

3. **Adaptability and modularity of the Coverage Graphs**
   Given that the Coverage Graphs approach is independent of the actual deductive and inductive mechanisms that are used underneath, we may consider many avenues of future work. We plan to apply this setting to some other tasks, by using the same or other deductive and inductive engines, and keep on with the integration into our learning system gErl (we show some preliminary results for a single letter series completion problem in Appendix C) thus making it explicit a constructive and developmental nature of the knowledge learnt. Furthermore, it is also of interest the application of the principles used (MML evaluations and knowledge handling mechanisms) in other sort of AI systems (such as decision support systems) in order to help them make better decisions based on the best available data.

4. **Benchmarks and metrics in AI evaluation**
   It seems clear that the use of one single intelligence test that is known a priori is mostly useless for evaluation, as we can specialise a system

to solve that problem. The systems we have seen in Section 6.3 are, in general, only able to deal with one kind of test. A very ambitious goal would be, therefore, to create a repository or generator of all these intelligence test problems, thus following the Detterman claim about that AI needed a "battery of intelligence tests" [Det11]. We know that many intelligence tests are not publicly available, and many of the approaches we have surveyed here have used alternative formulations because of this. It would be very useful for AI to arrange these problems, record the results of computer models and humans over them and organise competitions. Therefore, the construction of a repository using some human intelligence tests (such as PEBL), despite its limitations, could be a seed for a more proper universal benchmark in the future. This would be beneficial (at least at this moment and in the near future), as several computer models could work with the same task instances, using the same presentation, as well a previously fixed difficulty assessment. The repository could also record some previous results and the kind of abilities they usually represent in humans. This repository could also integrate some other measurement tools from AI that try to be general and domain-independent [BNVB12]. Of course, the development of an evaluation testbed for AI (or universally) need to meet certain conditions: the benchmark should be broad (including a wide range of tests), standard (using some kind of general protocol for inputs and outputs), characterised (accompanied with a catalogue of information about their difficulty, the abilities they cover, etc.), available (on a web or problem library) and renewed (new items are generated or disclosed so that systems cannot rote-learn them).

As a final remark, this thesis bring to light that there is still a huge margin of improvement in the way general-purpose AI systems are devised and built as well as evaluated. This work can serve as a comprehensive basis for new research.

Bibliography

[AA07] D.J. Newman A. Asuncion. UCI machine learning repository, 2007. 16

[ABLB99] R. F. Amthauer, B. Brocke, D. Liepmann, and A. Beauducel. *Intelligenz-Struktur-Test 2000: IST 2000.* Hogrefe & Huber, 1999. 246

[ABLB01] Rudolf Amthauer, Burkhard Brocke, Detlev Liepmann, and André Beauducel. Intelligenz-struktur-test 2000 r. *Göttingen: Hogrefe*, 2, 2001. 94

[AHK$^+$09] Minoru Asada, Koh Hosoda, Yasuo Kuniyoshi, Hiroshi Ishiguro, Toshio Inui, Yuichiro Yoshikawa, Masaki Ogino, and Chisato Yoshida. Cognitive developmental robotics: A survey. *Autonomous Mental Development, IEEE Transactions on*, 1(1):12–34, 2009. 123

[AL03] J.R Anderson and C. Lebiere. The Newell test for a theory of cognition. *Behavioral and Brain Sciences*, 26(5):587–601, 2003. 118, 123

[Ale] Amazon's Alexa. https://developer.amazon.com/public/solutions/alexa. 1

[ALLM94] DavidW. Aha, Stephane Lapointe, CharlesX. Ling, and Stan Matwin. Inverting implication with small training sets. In Francesco Bergadano and Luc De Raedt, editors, *Machine Learning: ECML-94*, volume 784 of *Lecture Notes in Computer Science*, pages 29–48. Springer Berlin Heidelberg, 1994. 19, 32, 33

[And96] John R Anderson. ACT: A simple theory of complex cognition. *American Psychologist*, 51(4):355–365, April 1996. 101, 126

[AR97] Bernard Ans and StÃľphane Rousset. Avoiding catastrophic forgetting by coupling two reverberating neural networks. *Comptes Rendus de l'AcadÃľmie des Sciences - Series {III} - Sciences de la Vie*, 320(12):989 – 997, 1997. 154

[AS10] Rajendra Akerkar and Priti Sajja. *Knowledge-based systems.* Jones & Bartlett Publishers, 2010. 37

[AVW92] J. L. Armstrong, S.R. Virding, and M. C. Williams. Use of prolog for developing a new programming language, 1992. 236

[AZM09] Fady Alnajjar, Indra Bin Mohd Zin, and Kazuyuki Murase. A hierarchical autonomous robot controller for learning and memory: adaptation in a dynamic environment. *Adaptive Behavior*, 17(3):179–196, 2009. 155

[Bak07] Gökhan Bakir. *Predicting structured data.* MIT press, 2007. 33

[Bal87] Paul B Baltes. Theoretical propositions of life-span developmental psychology: On the dynamics between growth and decline. *Developmental psychology*, 23(5):611, 1987. 122

[Bax00] Jonathan Baxter. A model of inductive bias learning. *Journal of Artificial Intelligence Research*, 12:149–198, 2000. 35

[BB75] L. Blum and M. Blum. Toward a mathematical theory of inductive inference. *Information and Control*, 28(2):125–155, 1975. 77

[BC05] Stefan Büttcher and Charles LA Clarke. Efficiency vs. effectiveness in terabyte-scale information retrieval. In *TREC*, 2005. 106

[BDR98] Hendrik Blockeel and Luc De Raedt. Top-down induction of first-order logical decision trees. *Artificial intelligence*, 101(1):285–297, 1998. 19, 33

[Ben69] G. K. Bennett. *Bennett mechanical comprehension test.* Psychological Corporation, 1969. 89, 253

[Ben09] Yoshua Bengio. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009. 30

[BGCK$^+$97] AF Bowers, C Giraud-Carrier, C Kennedy, JW Lloyd, and R MacKinney-Romero. A framework for higher-order inductive machine learning. In *Proceedings of CompulogNet Area Meeting on Computational Logic and Machine Learning" P. Flach, Ed*, page 19, 1997. 32

[BGCL00a] Antony F Bowers, Christophe Giraud-Carrier, and John W Lloyd. Classification of individuals with complex structure. In *ICML*, pages 81–88. Citeseer, 2000. 13, 26

[BGCL00b] Antony F Bowers, Christophe Giraud-Carrier, and John W Lloyd. Classification of individuals with complex structure. In *ICML*, pages 81–88. Citeseer, 2000. 32

[BGH89] L. B. Booker, D. E. Goldberg, and J. H. Holland. Classifier systems and genetic algorithms. *Artif. Intell.*, 40(1-3):235–282, September 1989. 13

[Bie78] Alan W Biermann. The inference of regular lisp programs from examples. *IEEE transactions on Systems, Man, and Cybernetics*, 8(8):585–600, 1978. 22

[BNVB12] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment. *J. Artificial Intelligence Res*, 47:253–279, 2012. 194

[Bon70] M. M. Bongard. *Pattern Recognition*. Spartan Books, 1970. 15, 16, 87, 250

[Bow98] AF Bowers. Early experiments with a higher-order decision-tree learner. In *Proceedings of the COMPULOGNet Area Meeting on Computational Logic and Machine Learning*, page 42, 1998. 32

[BP98] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Comput. Netw. ISDN Syst.*, 30(1-7):107–117, April 1998. 47, 49

[BPG12] M. Bayoudh, H. Prade, and Richard G. Evaluation of analogical proportions through kolmogorov complexity. *Knowledge-Based Systems*, 29(0):20–30, 2012. Artificial Intelligence 2010. 91, 92, 106

[Bri11] S. Bringsjord. Psychometric artificial intelligence. *Journal of Experimental & Theoretical Artificial Intelligence*, 23(3):271–277, 2011. 89, 118, 122

[BS03] S. Bringsjord and B. Schimanski. What is artificial intelligence? Psychometric AI as an answer. In *International Joint Confer-*

ence on Artificial Intelligence, pages 887–893, 2003. 78, 86, 87, 91, 99, 107, 108, 118, 122

[BT08] Will Bridewell and Ljupčo Todorovski. Learning declarative bias. In *Inductive Logic Programming*, pages 63–77. Springer, 2008. 37

[Bun92] Wray Lindsay Buntine. *A theory of learning classification rules.* PhD thesis, Citeseer, 1992. 14

[Bur05a] J. Burghardt. E-generalization using grammars. *Artificial Intelligence*, 165(1):1–35, 2005. 90, 93

[Bur05b] Jochen Burghardt. E-generalization using grammars. *Artificial intelligence*, 165(1):1–35, 2005. 38

[BW70] DM Boulton and Chris S. Wallace. A program for numerical classification. *The Computer Journal*, 13(1):63–69, 1970. 44

[BW75] DM Boulton and Chris S. Wallace. An information measure for single link classification. *The Computer Journal*, 18(3):236–238, 1975. 44

[BW88] Richard Bird and Philip Wadler. *An Introduction to Functional Programming.* Prentice Hall International (UK) Ltd., Hertfordshire, UK, UK, 1988. 21

[Car93] Richard Caruana. Multitask learning: A knowledge-based source of inductive bias. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 41–48. Morgan Kaufmann, 1993. 35

[Cat63] R. B. Cattell. Theory of fluid and crystallized intelligence: A critical experiment. *Journal of educational psychology*, 54(1–22):1, 1963. 80

[CBK$^+$10] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. Hruschka Jr., and T. M. Mitchell. Toward an architecture for never-ending language learning. In *Proc. of the 24th Conference on Artificial Intelligence*, pages 1306–1313, 2010. 38

[CG88] G.A. Carpenter and S. Grossberg. The art of adaptive pattern recognition by a self-organizing neural network. *Computer*, 21(3):77–88, 1988. 3, 8, 34, 153, 154, 192

[CG03] F. Calvo-Garzón. Nonclassical connectionism should enter the decathlon. *Behavioral and Brain Sciences*, 26(05):603–604, 2003. 118, 123

[Cha70] Gregory J Chaitin. On the difficulty of computations. *Information Theory, IEEE Transactions on*, 16(1):5–9, 1970. 43

[CHH02] M. Campbell, A. J. Hoane, and F. Hsu. Deep Blue. *Artificial Intelligence*, 134(1-2):57–83, 2002. 1, 76

[Chu33] A. Church. A set of postulates for the foundation of logic part ii. *Annals of Mathematics*, 34(2):839–864, 1933. 21

[CJQ94] R Mike Cameron-Jones and J Ross Quinlan. Efficient top-down induction of logic programs. *ACM Sigart Bulletin*, 5(1):33–42, 1994. 19, 32, 33

[CJS90] P. A. Carpenter, M. A. Just, and P. Shell. What one intelligence test measures: A theoretical account of the processing in the Raven Progressive Matrices test. *Psychological review*, 97:404–431, 1990. 77, 85, 86, 88, 91, 94, 98, 100, 101, 114, 126, 137, 140

[CKHS09] Neil Crossley, Emanuel Kitzelmann, Martin Hofmann, and Ute Schmid. Combining analytical and evolutionary inductive programming. *Second Conference on Artificial General Intelligence*, pages 19–24, 2009. 23

[CKM$^+$05] John Cabral, Robert C Kahlert, Cynthia Matuszek, Michael Witbrock, and Brett Summers. Converting semantic meta-knowledge into inductive bias. In *Inductive Logic Programming*, pages 38–50. Springer, 2005. 37

[CM15] Andrew Cropper and Stephen H Muggleton. Learning efficient logical robot strategies involving composable objects. In *Proceedings of the 24th international joint conference on Artificial Intelligence*, 2015. 54

[Cor] Microsoft's Cortana. http://www.microsoft.com/en-us/mobile/campaign-cortana/. 1

[CPR12] W. Correa, H. Prade, and G. Richard. When intelligence is just a matter of copying. In *Proc. 20th Europ. Conf. on Artificial*

Intelligence, Montpellier, Aug, pages 27–31, 2012. 91, 92, 99, 107, 127

[CS08] A. T. Cianciolo and R. J. Sternberg. *Intelligence: A brief history*. John Wiley & Sons, 2008. 79

[CV05] Rudi Cilibrasi and Paul Vitanyi. Clustering by compression. *Information Theory, IEEE Transactions on*, 51(4):1523–1545, 2005. 46

[Det11] D. K. Detterman. A challenge to Watson. *Intelligence*, 39(2-3):77–78, 2011. 110, 122, 194

[DHO12] D. L. Dowe and J. Hernández-Orallo. IQ tests are not for machines, yet. *Intelligence*, 40(2):77–81, 2012. 123, 127, 128, 150

[DHO14] David L Dowe and José Hernández-Orallo. How universal can an intelligence test be? *Adaptive Behavior*, 22(1):51–69, 2014. 111

[DIPS06] S. Dehaene, V. Izard, P. Pica, and E. Spelke. Core Knowledge of Geometry in an Amazonian Indigene Group. *Science*, 311(5759):381–384, January 2006. 129, 249

[DL01] S. Džeroski and N. Lavrac, editors. *Relational Data Mining*. Springer-Verlag, 2001. 31

[DR92] Luc De Raedt. *Interactive Theory Revision: An Inductive Logic Programming Approach*. Academic Press Ltd., London, UK, UK, 1992. 19, 37

[DR10] Luc De Raedt. Inductive logic programming. In *Encyclopedia of machine learning*, pages 529–537. Springer, 2010. 31

[DS10] S. Della Sala. *Forgetting*. Psychology Press, 2010. 156

[EF07] Esra Erdem and Paolo Ferraris. Forgetting actions in domain descriptions. In *Proc. of the 22nd AAAI Conference on Artificial Intelligence*, pages 409–414, 2007. 156

[EFHORQ05] Vicent Estruch, Cesar Ferri, José Hernández-Orallo, and Marıa-José Ramırez-Quintana. A survey of (pseudo-distance) functions for structured-data. *III Taller de Mineria de Datos y Aprendizaje (TAMIDA 2005)*, pages 233–242, 2005. 33

[EFHR06] V. Estruch, C. Ferri, J. Hernández-Orallo, and M. J. Ramírez-Quintana. Similarity functions for structured data. An application to decision trees. *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial*, 10(29):109–121, 2006. 34

[EFHR12] V. Estruch., C. Ferri, J. Hernández-Orallo, and M. J. Ramírez-Quintana. Bridging the Gap between Distance and Generalisation. *Computational Intelligence*, pages no–no, 2012. 34

[EN69] George W Ernst and Allen Newell. *GPS: A case study in generality and problem solving*. Academic Pr, 1969. 76

[Eps79] Herman T Epstein. Correlated brain and intelligence development in humans. *Development and evolution of brain size: Behavioral implications, chapter 6 in Development and Evolution of Brain Size: Behavioral Implications, edited by Martine Hahn*, pages 111–131, 1979. 121

[ER00] S. E. Embretson and S. P. Reise. *Item response theory for psychologists*. L. Erlbaum, 2000. 80

[ER13] Eric Eaton and Paul L. Ruvolo. Ella: An efficient lifelong learning algorithm. In *ICML*, volume 28, pages 507–515, 2013. 35

[ESC+12] C. Eliasmith, T. C. Stewart, X. Choo, T. Bekolay, T. DeWolf, C. Tang, and D. Rasmussen. A large-scale model of the functioning brain. *Science*, 338(6111):1202–1205, 2012. 1, 37, 78, 91, 93, 99, 105, 111, 114, 122

[Eva63] T. Evans. *A heuristic program of solving geometric analogy problems*. PhD thesis, Mass. Inst. Tech., Cambridge, Mass., U.S.A., 1963. Also available from AF Cambridge Research Lab, Hanscom AFB, Bedford, Mass., U.S.A.: Data Sciences Lab, Phys and Math Sci Res Paper 64, Project 4641. 77, 78, 84, 91, 98, 114

[Eva65] T. Evans. A heuristic program to solve geometric-analogy problems. In *Proc. SJCC*, volume 25, pages 327–339, 1965. vol. 25. 77, 78, 84, 91, 98, 114, 247

[EW08] Thomas Eiter and Kewen Wang. Semantic forgetting in answer set programming. *Artificial Intelligence*, 172(14):1644 – 1672, 2008. 156

[EWB14] Mehmet Dinçer Erbas, Alan F. T. Winfield, and Larry Bull. Embodied imitation-enhanced reinforcement learning in multi-agent systems. *Adaptive Behaviour*, 22(1):31–50, 2014. 155

[FBCC$^+$10] D. Ferrucci, E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. A. Kalyanpur, A. Lally, J.W. Murdock, E. Nyberg, J. Prager, et al. Building Watson: An overview of the DeepQA project. *AI Magazine*, 31(3):59–79, 2010. 1, 76, 110, 122

[FFG89] B. Falkenhainer, K. D. Forbus, and D. Gentner. The structure-mapping engine: Algorithm and examples. *Artificial Intelligence*, 41:1–63, 1989. 87, 103

[FG97] K. D. Forbus and D. Gentner. Qualitative mental models: Simulations or memories. In *Proceedings of the Eleventh International Workshop on Qualitative Reasoning*, pages 3–6, 1997. 89

[FGMF98] Kenneth D. Forbus, Dedre Gentner, Arthur B. Markman, and Ronald W. Ferguson. Analogy just looks like high level perception: Why a domain-general approach to analogical mapping is right. *JETAI*, 10:231–257, 1998. 103

[FHR01] C. Ferri, J. Hernández-Orallo, and M. J. Ramírez-Quintana. Incremental learning of functional logic programs. In Herbert Kuchen and Kazunori Ueda, editors, *Functional and Logic Programming*, volume 2024 of *Lecture Notes in Computer Science*, pages 233–247. Springer Berlin Heidelberg, 2001. 24, 33, 37

[Fla00] Peter A. Flach. The use of functional and logic languages in machine learning. In *9th International Workshop on Functional and Logic Programming, WFLP'2000, Benicassim, Spain, September 28-30, 2000*, pages 225–237, 2000. 13, 15, 24

[FLB$^+$13] D. Ferrucci, A. Levas, S. Bagchi, D. Gondek, and E. T. Mueller. Watson: Beyond jeopardy! *Artificial Intelligence*, 199:93–105, 2013. 1, 76

[Fle97] Pierre Flener. Inductive logic program synthesis with dialogs. In Stephen Muggleton, editor, *Inductive Logic Programming*, volume 1314 of *Lecture Notes in Computer Science*, pages 175–198. Springer Berlin Heidelberg, 1997. 32

[Fou06] H. E. Foundalis. Phaeaco: A cognitive architecture inspired by Bongard's problems. *Doctoral thesis, Indiana University, Bloomington*, 2006. 87, 91, 99, 102

[Fre97] Robert M. French. Pseudo-recurrent connectionist networks: An approach to the "sensitivity-stability" dilemma. *Connection Science*, 9:353–379, 1997. 154

[FS08] P Flener and U. Schmid. An introduction to inductive programming. *Artificial Intelligence Review*, 29(1):45–62, 2008. 19, 94, 104, 128

[FU02] K. D. Forbus and J. Usher. Sketching for knowledge capture: A progress report. In *Proceedings of the 7th International Conference on Intelligent User Interfaces*, IUI '02, pages 71–77, New York, NY, USA, 2002. ACM. 87, 98

[FUL$^+$08] Kenneth Forbus, Jeffrey Usher, Andrew Lovett, Kate Lockwood, and Jon Wetzel. Cogsketch: Open-domain sketch understanding for cognitive science research and for education. In *Proceedings of the Eurographics Workshop on Sketch-Based Interfaces and Modeling*, 2008. 88, 98

[FY99] Pierre Flener and Serap Yiılmaz. Inductive synthesis of recursive logic programs: Achievements and prospects. *The Journal of Logic Programming*, 41(2):141–195, 1999. 19

[G$^+$72] Eugene Garfield et al. Citation analysis as a tool in journal evaluation. American Association for the Advancement of Science, 1972. 47

[Gär03] Thomas Gärtner. A survey of kernels for structured data. *ACM SIGKDD Explorations Newsletter*, 5(1):49–58, 2003. 33

[Gär05] T. Gärtner. *Kernels for structured data.* PhD thesis, Universität Bonn, 2005. 14, 34

[GB13] Michael Genesereth and Yngvi BjÃűrnsson. The international general game playing competition. *AI Magazine*, 34:107–111, 2013. 100

[Gen83] Dedre Gentner. Structure-mapping: A theoretical framework for analogy. *Cognitive Science*, 7(2):155–170, 1983. 103

[Geo08] Nikolay Georgiev. Item analysis of c, d and e series from Raven's standard progressive matrices with item response theory two-parameter logistic model. *Europe's Journal of Psychology*, 4(3), 2008. 139

[Get07] Lise Getoor. *Introduction to statistical relational learning*. MIT press, 2007. 33

[GGL+13] Pankaj Gupta, Ashish Goel, Jimmy Lin, Aneesh Sharma, Dong Wang, and Reza Zadeh. Wtf: The who to follow service at twitter. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, pages 505–514, Republic and Canton of Geneva, Switzerland, 2013. International World Wide Web Conferences Steering Committee. 51

[GHOK+15a] S. Gulwani, J. Hernández-Orallo, E. Kitzelmann, S. H. Muggleton, U. Schmid, and B. Zorn. Inductive programming meets the real world, 2015. Communications of the ACM, to appear. 4, 30, 31

[GHOK+15b] S. Gulwani, J. Hernández-Orallo, E. Kitzelmann, S. H. Muggleton, U. Schmid, and B. Zorn. Inductive programming meets the real world, 2015. Communications of the ACM, to appear. 94, 104

[GL02] Dedre Gentner and Jeffrey Loewenstein. Relational language and relational thought. *Language, literacy, and cognitive development: The development and consequences of symbolic communication*, pages 87–120, 2002. 103

[GLF+09] Alex Graves, Marcus Liwicki, Santiago Fernandez, Roman Bertolami, Horst Bunke, and Jurgen Schmidhuber. A novel connectionist system for unconstrained handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(5):855–868, 2009. 34

[GLP05] Michael Genesereth, Nathaniel Love, and Barney Pell. General game playing: Overview of the aaai competition. *AI Magazine*, 26(2):62–72, 2005. 77, 115

[goo] Google's Google Now. https://www.google.com/landing/now/. 1

[Gri54] Ruth Griffiths. *The abilities of babies: a study in mental measurement.* McGraw-Hill, 1954. 118, 123

[Gro87] Stephen Grossberg. Competitive learning: From interactive activation to adaptive resonance. *Cognitive Science*, 11(1):23–63, 1987. 36

[Gro13] Stephen Grossberg. Adaptive resonance theory: How a brain learns to consciously attend, learn, and recognize a changing world. *Neural Netw.*, 37:1–47, January 2013. 154

[GSSS12] Shane Griffith, Jivko Sinapov, Vladimir Sukhoy, and Alexander Stoytchev. A behavior-grounded approach to forming object categories: Separating containers from noncontainers. *Autonomous Mental Development, IEEE Transactions on*, 4(1):54–69, 2012. 111, 124

[Gul11] Sumit Gulwani. Automating string processing in spreadsheets using input-output examples. In *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '11, pages 317–330, New York, NY, USA, 2011. ACM. 24

[HB00] J. Holland and Booker. What is a learning classifier system? In *Learning Classifier Systems*, volume 1813 of *LNCS*, pages 3–32. Springer, 2000. 56

[Heb49] D.O. Hebb. *The organization of behavior.* Wiley, New York, 1949. 156

[Hen10] Robert Henderson. Incremental learning in inductive programming. In *Approaches and Applications of Inductive Programming*, pages 74–92. Springer, 2010. 37

[HHEK14] Mohammad Javad Hosseini, Hannaneh Hajishirzi, Oren Etzioni, and Nate Kushman. Learning to solve arithmetic word problems with verb categorization. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014. 81

[HKS09] Martin Hofmann, Emanuel Kitzelmann, and Ute Schmid. A unifying framework for analysis and evaluation of inductive programming systems. In B. Goertzel, P. Hitzler, and M. Hutter, editors, *Second Conference on Artificial General Intelligence*, pages 55–60. Atlantis Press, 2009. 32

[HKS14] Jacqueline Hofmann, Emanuel Kitzelmann, and Ute Schmid. Applying inductive program synthesis to induction of number series a case study with IGOR2. In *KI 2014: Advances in Artificial Intelligence*, pages 25–36. Springer, 2014. 91, 94, 99, 104, 105

[HM84] D. R. Hofstadter and M Mitchell. The copycat project, 1984. 85, 91, 101, 253

[HM12] Robert J Henderson and Stephen H Muggleton. Automatic invention of functional abstractions, 2012. 37

[HMS$^+$16] J. Hernández-Orallo, F. Martínez-Plumed, Ute Schmid, Michael Siebers, and David L. Dowe. Computer models solving intelligence test problems: Progress and implications. *Artificial Intelligence*, 230:74 – 107, 2016. 8, 76

[HO00a] J. Hernández-Orallo. Beyond the Turing Test. *J. Logic, Language & Information*, 9(4):447–466, 2000. 118, 151

[HO00b] J. Hernández-Orallo. Constructive reinforcement learning. *International Journal of Intelligent Systems*, 15(3):241–264, 2000. 23, 37, 155

[HO14a] José Hernández-Orallo. Ai evaluation: past, present and future. *arXiv preprint arXiv:1408.6908*, 2014. 2

[HO14b] José Hernández-Orallo. Deep knowledge: Inductive programming as an answer. In *Approaches and Applications of Inductive Programming (Dagstuhl Seminar 13502)*, 2014. 13, 30, 38, 39

[HOD10] J. Hernández-Orallo and D. L. Dowe. Measuring universal intelligence: Towards an anytime intelligence test. *Artificial Intelligence*, 174(18):1508–1539, 2010. 111, 118, 151

[HOD13] J. Hernández-Orallo and D. L. Dowe. On potential cognitive abilities in the machine kingdom. *Minds and Machines*, 23(2):179–210, 2013. 119, 151

[HODHL14] J. Hernández-Orallo, D. L. Dowe, and M. V. Hernández-Lloreda. Universal psychometrics: Measuring cognitive abilities in the machine kingdom. *Cognitive Systems Research*, 27:50âĂŞ74, 2014. 151

[Hof79] D. R. Hofstadter. *Gödel, g, Bach: An Eternal Golden Braid.* Basic Books, Inc., New York, NY, USA, 1979. 250

[Hof83] D. R. Hofstadter. The architecture of Jumbo. In *Proceedings of the International Machine Learning Workshop*, pages 161–170. University of Illinois Press, 1983. 85, 91, 101, 250, 253

[Hof08] D. R. Hofstadter. *Fluid concepts and creative analogies: Computer models of the fundamental mechanisms of thought.* Basic Books, 2008. 141, 246, 251

[HOGV00] José Hernández-Orallo and Ismael García-Varea. Explanatory and creative alternatives to the MDL priciple. *Foundations of Science*, 5(2):185–207, 2000. 155

[HORQ98] J. Hernández-Orallo and M. J. Ramírez-Quintana. Inverse narrowing for the induction of functional logic programs. In *APPIA-GULP-PRODE*, pages 379–392. Citeseer, 1998. 24, 32

[HORQ99] J. Hernández-Orallo and M. J. Ramírez-Quintana. A strong complete schema for inductive functional logic programming. In *Inductive Logic Programming*, pages 116–127. Springer, 1999. 33

[HPG82] Thomas G. Holzman, James W. Pellegrino, and Robert Glaser. Cognitive dimensions of numerical rule induction. *Journal of Educational Psychology*, 74(3):360–373, 1982. 95

[HT12] Emmanouil Hourdakis and Panos Trahanias. Computational modeling of observational learning inspired by the cortical underpinnings of human primates. *Adaptive Behavior*, 20(4):237–256, 2012. 155

[IGIS13] Shachar Itzhaky, Sumit Gulwani, Neil Immerman, and Mooly Sagiv. Solving geometry problems using a combination of symbolic and numerical reasoning. In *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 457–472. Springer, 2013. 81

[JC87] M. A. Just and P. A. Carpenter. *The psychology of reading and language comprehension.* Allyn & Bacon, 1987. 100

[JL87] J. Jaffar and J.-L. Lassez. Constraint logic programming. In *Proceedings of the 14th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '87, pages 111–119, New York, NY, USA, 1987. ACM. 20

[Kat05] Susumu Katayama. Systematic search for lambda expressions. In *Revised Selected Papers from the Sixth Symposium on Trends in Functional Programming, TFP 2005, Tallinn, Estonia, 23-24 September 2005.*, pages 111–126, 2005. 23, 32

[KAZB14] Nate Kushman, Yoav Artzi, Luke Zettlemoyer, and Regina Barzilay. Learning to automatically solve algebra word problems. *ACL (1)*, pages 271–281, 2014. 81

[Kee10] Ed Keedwell. Towards a staged developmental intelligence test for machines. In *Towards a Comprehensive Intelligence Test (TCIT): Reconsidering the Turing Test for the 21st Century Symposium*, pages 28–32, 2010. 118, 123

[Ken98] Claire J Kennedy. Evolutionary higher-order concept learning. In *Late Breaking Papers at the Genetic Programming 1998 Conference John R. Koza, Ed.. Stanford University Bookstore. University of Wisconsin, Madison, Wisconsin, USA*. Citeseer, 1998. 32

[Kes63] Maxwell Mirton Kessler. Bibliographic coupling between scientific papers. *American documentation*, 14(1):10–25, 1963. 46

[KFGQ00] Sven Kuehne, Kenneth Forbus, Dedre Gentner, and Bryan Quinn. Seql: Category learning as progressive abstraction using structure mapping. In *Proceedings of the 22nd Annual Meeting of the Cognitive Science Society*, pages 770–775, 2000. 103

[KFTK11] M. Klenk, K. Forbus, E. Tomai, and H. Kim. Using analogical model formulation with sketches to solve Bennett mechanical comprehension test problems. *Journal of Experimental & Theoretical Artificial Intelligence*, 23(3):299–327, 2011. 89, 91, 103, 254

[KGC99] Claire J Kennedy and Christophe Giraud-Carrier. An evolutionary approach to concept learning with structured data. In *Artificial Neural Nets and Genetic Algorithms*, pages 331–336. Springer, 1999. 32

[Kid87] A. L. Kidd, editor. *Knowledge Acquisition for Expert Systems: A Practical Handbook*. Plenum Press, New York, NY, USA, 1987. 31

[Kit07] Emanuel Kitzelmann. Data-driven induction of recursive functions from input/output-examples. In *Proceedings of the ECML/PKDD 2007 Workshop on Approaches and Applications of Inductive Programming (AAIPâĂŹ07)*, pages 15–26. Citeseer, 2007. 23, 32, 33

[Kit10] E. Kitzelmann. Inductive programming: A survey of program synthesis techniques. In *3rd Workshop AAIP*, volume 5812 of *LNCS*, 2010. 31, 128

[KK83] Robert Kowalski and Donald Kuehner. *Linear resolution with selection function*. Springer, 1983. 17

[KL06] Alan S Kaufman and Elizabeth O Lichtenberger. *Assessing adolescent and adult intelligence*. Wiley, 2006. 121

[Kle99] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, 46(5):604–632, September 1999. 47

[KMG10] M. Kunda, K. McGreggor, and A. Goel. Taking a look (literally!) at the Raven's intelligence test: Two visual solution strategies. In *Proc. 32nd Annual Meeting of the Cognitive Science Society, Portland*, pages 1691–1696, 2010. 89, 91, 99, 106, 126

[KMG12] M. Kunda, K. McGreggor, and A. Goel. Reasoning on the RavenâĂŹs Advanced Progressive Matrices test with iconic visual representations. In *34th Annual Conference of the Cognitive Science Society, Portland, OR*, pages 1828–1833, 2012. 89, 91, 99, 106, 126

[KMG13] M. Kunda, K. McGreggor, and A. Goel. A computational model for solving problems from the RavenâĂŹs Progressive Matrices intelligence test using iconic visual representations. *Cognitive Systems Research*, pages 22–23, 47–66, 2013. 89, 91, 99, 106

[KMLS92] Ross D King, Stephen Muggleton, Richard A Lewis, and MJ Sternberg. Drug design by machine learning: The use of inductive logic programming to model the structure-activity relationships of trimethoprim analogues binding to dihydrofolate reductase. *Proceedings of the national academy of sciences*, 89(23):11322–11326, 1992. 20, 32

[KMP98] Khalid Khan, Stephen Muggleton, and Rupert Parson. Repeat learning using predicate invention. In *Inductive Logic Programming*, pages 165–174. Springer, 1998. 37

[Kol68] Andrei Kolmogorov. Logical basis for information theory and probability theory. *Information Theory, IEEE Transactions on*, 14(5):662–664, 1968. 43

[KS73] Kenneth Kotovsky and Herbert A. Simon. Empirical tests of a theory of human acquisition of concepts for sequential patterns. *Cognitive Psychology*, 4(3):399 – 424, 1973. 142, 143

[KS97] D. Koller and M. Sahami. Hierarchically classifying documents using very few words. In *Proceedings of the Fourteenth International Conference on Machine Learning*, ICML '97, pages 170–178, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc. 34

[KS06] Emanuel Kitzelmann and Ute Schmid. Inductive synthesis of functional programs: An explanation based generalization approach. *Journal of Machine Learning Research*, 7:429–454, 2006. 23, 32

[Kuh39] Fred Kuhlmann. *Tests of Mental Development: a Complete Scale for Individual Examination.* Educational Test Bureau, Educational Publishers, Incorporated, 1939. 118, 123

[KW92] Jörg-Uwe Kietz and Stefan Wrobel. Controlling the complexity of learning in logic through syntactic and task-oriented models. In *Inductive logic programming*. Citeseer, 1992. 37

[Lan11] P. Langley. Artificial intelligence and cognitive systems. *AISB Quarterly*, 2011. 118, 123

[LCA14] George Leu, Neville J. Curtis, and Hussein A. Abbass. Society of mind cognitive agent architecture applied to drivers adapting in a traffic context. *Adaptive Behaviour*, pages 123–145, 2014. 155

[LD93] Nada Lavrac and Saso Dzeroski. *Inductive Logic Programming: Techniques and Applications*. Routledge, New York, NY, 10001, 1993. 19, 20, 33

[LDG91] Nada LavraÄŊ, SaÅąo DÅ¿eroski, and Marko Grobelnik. Learning nonrecursive definitions of relations with linus. In Yves Kodratoff, editor, *Machine Learning âĂŤ EWSL-91*, volume 482 of *Lecture Notes in Computer Science*, pages 265–281. Springer Berlin Heidelberg, 1991. 19, 33

[Lev73] Leonid A Levin. Universal sequential search problems. *Problemy Peredachi Informatsii*, 9(3):115–116, 1973. 145

[Lev84] Leonid A Levin. Randomness conservation inequalities; information and independence in mathematical theories. *Information and Control*, 61(1):15–37, 1984. 145

[LF11] Andrew Lovett and Kenneth Forbus. Cultural commonalities and differences in spatial problem-solving: A computational analysis. *Cognition*, 121(2):281–287, 2011. 88, 91, 98, 103

[LF12] A. Lovett and K. Forbus. Modeling multiple strategies for solving geometric analogy problems. In *Proceedings of the 34th Annual Conference of the Cognitive Science Society*, pages 701–706, 2012. 88, 91, 98, 103

[LFU07] A. Lovett, K. Forbus, and J. Usher. Analogy with qualitative spatial representations can simulate solving RavenâĂŹs Progressive Matrices. In *Proceedings of the 29th Annual Conference of the Cognitive Society*, volume 30, page 34, 2007. 88, 89, 91, 92, 103, 126

[LFU10] A. Lovett, K. Forbus, and J. Usher. A structure-mapping model of RavenâĂŹs Progressive Matrices. In *Proceedings of CogSci*, volume 10, pages 2761–2766, 2010. 88, 91, 103, 126

[LG14] Vu Le and Sumit Gulwani. Flashextract: A framework for data extraction by examples. In *ACM SIGPLAN Notices*, volume 49, pages 542–553. ACM, 2014. 24

[LH07] S. Legg and M. Hutter. Universal intelligence: A definition of machine intelligence. *Minds and Machines*, 17(4):391–444, 2007. 151

[LL03] Jerome Lang and Paolo Liberatore. Propositional independence: Formula-variable independence and forgetting. *Journal of Artificial Intelligence Research*, 18:2003, 2003. 156

[LLF08] A. Lovett, K. Lockwood, and K. Forbus. A computational model of the visual oddity task. In *Proceedings of the 30th Annual Conference of the Cognitive Science Society. Washington, DC*, volume 25, page 29, 2008. 88, 91, 98, 103, 127

[Llo95] John W Lloyd. Declarative programming in escher. Technical report, Technical Report CSTR-95-013, Department of Computer Science, University of Bristol, 1995. 32

[Llo99] John W Lloyd. Programming in an integrated functional and logic language. *Journal of Functional and Logic Programming*, 3(1-49):68–69, 1999. 32

[Llo01] John W Lloyd. Knowledge representation, computation, and learning in higher-order logic, 2001. 24, 32

[LM00] R. Lempel and S. Moran. The stochastic approach for link-structure analysis (salsa) and the tkc effect. *Comput. Netw.*, 33(1-6):387–401, June 2000. 50

[LM01] J. Lafferty and A McCallum. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 282–289, 2001. 33

[LM10] Jérôme Lang and Pierre Marquis. Reasoning under inconsistency: A forgetting-based approach. *Artif. Intell.*, 174(12-13):799–823, August 2010. 156

[LMF08] Amy N Langville, Carl D Meyer, and Pablo FernÁndez. GoogleâĂŹs pagerank and beyond: The science of search engine rankings. *The Mathematical Intelligencer*, 30(1):68–69, 2008. 49

[LNR87] J. E. Laird, A. Newell, and Paul S. Rosenbloom. SOAR: An architecture for general intelligence. *Artificial Intelligence*, 33(1):1–64, September 1987. 102

[Lor80] F. M. Lord. *Applications of item response theory to practical testing problems.* Mahwah, NJ: Erlbaum, 1980. 80

[LR94] Fangzhen Lin and Ray Reiter. Forget it! In *In Proceedings of the AAAI Fall Symposium on Relevance*, pages 154–159, 1994. 156

[LS06] Tobias Lindahl and Konstantinos Sagonas. Practical type inference based on success typings. In *Proceedings of the 8th ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming*, PPDP '06, pages 167–178, New York, NY, USA, 2006. ACM. 240

[LTFU09] A. Lovett, E. Tomai, K. Forbus, and J. Usher. Solving geometric analogy problems through two-stage analogical mapping. *Cognitive Science*, 33(7):1192–1231, 2009. 88, 91, 98, 103

[LV08a] M. Li and P. Vitányi. *An introduction to Kolmogorov complexity and its applications (3rd ed.)*. Springer-Verlag, 2008. 80

[LV08b] Ming Li and Paul M.B. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer Publishing Company, 3 edition, 2008. 44, 45, 145

[LW11] Yongmei Liu and Ximing Wen. On the progression of knowledge in the situation calculus. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, page 976, 2011. 156

[Mac12] C. K. Machens. Building the human brain. *Science*, 338(6111):1156–1157, 2012. 78, 93

[Mar13] Sandra Marcus. *Automating knowledge acquisition for expert systems*, volume 57. Springer Science & Business Media, 2013. 37

[MB92] Stephen Muggleton and Wray Buntine. Machine invention of first-order predicates by inverting resolution. *Proceedings of the fifth international conference on machine learning*, pages 339–352, 1992. 19, 33, 37

[MB00] Stephen H Muggleton and Christopher H Bryant. Theory completion using inverse entailment. In *Inductive Logic Programming*, pages 130–146. Springer, 2000. 37

[MBHMM89] S. H. Muggleton, M. Bain, J. Hayes-Michie, and D. Michie. An experimental comparison of human and machine learning formalisms. In *In Proc. of 6th International Workshop on Machine Learning*, pages 113–118. Morgan Kaufmann, 1989. 173

[MC89] Michael Mccloskey and Neil J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. *The Psychology of Learning and Motivation*, 24:104–169, 1989. 34

[MC95] Raymond J. Mooney and Mary Elaine Califf. Induction of first-order decision lists: Results on learning the past tense of english verbs. *Journal of Artificial Intelligence Research*, 3:1–24, 1995. 20, 32

[McC07] J. McCarthy. What is artificial intelligence. *URL: http://www-formal.stanford.edu/jmc/whatisai.html*, 2007. 2, 76

[MCSK06] Noboru Matsuda, William W Cohen, Jonathan Sewall, and Kenneth R Koedinger. Applying machine learning to cognitive modeling for cognitive tutors. 2006. 38

[MD94] S. H. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *The Journal of Logic Programming*, 19:629–679, 1994. 20, 158

[MDRP+12] Stephen Muggleton, Luc De Raedt, David Poole, Ivan Bratko, Peter Flach, Katsumi Inoue, and Ashwin Srinivasan. Ilp turns 20. *Machine Learning*, 86(1):3–23, 2012. 33

[MF90] Stephen Muggleton and Cao Feng. *Efficient induction of logic programs*. Turing Institute, 1990. 19, 33

[MFHR13a] F. Martínez-Plumed, C. Ferri, J. Hernández-Orallo, and M. J. Ramírez-Quintana. Accelerating learning by reusing abstract policies in gerl. In Concha Bielza, Antonio Salmerón, Amparo Alonso-Betanzos, José Ignacio Hidalgo, Luis Martínez, Alicia Troncoso Lora, Emilio Corchado, and Juan M. Corchado, editors, *Multiconferencia CAEPIA'13*, VII Simposio de Teoría y Aplicaciones de Minería de Datos, TAMIDA'13, pages 1383–1392, 2013. 8, 37, 54, 72

[MFHR13b] F. Martínez-Plumed, C. Ferri, J. Hernández-Orallo, and M. J. Ramírez-Quintana. Learning with configurable operators and rl-based heuristics. In A. Appice, editor, *New Frontiers in Mining Complex Patterns*, volume 7765 of *Lecture Notes in Computer Science*, pages 1–16. Springer Berlin Heidelberg, 2013. 8, 54

[MFHR13c] F. Martínez-Plumed, C. Ferri, J. Hernández-Orallo, and M. J. Ramírez-Quintana. On the definition of a general learning system with user-defined operators. *CoRR*, abs/1311.4235, 2013. 8, 37, 54, 72

[MFHR13d] F. Martínez-Plumed, C. Ferri, J. Hernández-Orallo, and M. J. Ramírez-Quintana. Policy reuse in a general learning framework. In Concha Bielza, Antonio Salmerón, Amparo Alonso-Betanzos, José Ignacio Hidalgo, Luis Martínez, Alicia Troncoso Lora, Emilio Corchado, and Juan M. Corchado, editors, *Multiconferencia CAEPIA'13*, 15th Conference of the Spanish Association for Artificial Intelligence, CAEPIA'13, pages 119–128, 2013. 8, 37, 54, 72

[MFHR14] F. Martínez-Plumed, C. Ferri, J. Hernández-Orallo, and M. J. Ramírez-Quintana. A knowledge growth and consolidation framework for lifelong machine learning systems. In *Machine Learning and Applications (ICMLA), 2014 13th International Conference on*, pages 111–116, Dec 2014. 9, 154

[MFHR15a] F. Martínez-Plumed, C. Ferri, J. Hernández-Orallo, and M. J. Ramírez-Quintana. Forgetting and consolidation for incremental and cumulative knowledge acquisition systems. *CoRR*, abs/1502.05615, 2015. 9, 154

[MFHR15b] F. Martínez-Plumed, C. Ferri, J. Hernández-Orallo, and M. J. Ramírez-Quintana. Knowledge acquisition with forgetting: an incremental and developmental setting. *Adaptive Behavior*, 23(5):283–299, 2015. 9, 154

[MFHR16] F. Martínez-Plumed, C. Ferri, J. Hernández-Orallo, and M. J. Ramírez-Quintana. A computational analysis of general intelligence tests for evaluating cognitive development. *submitted (second revision)*, 2016. 8, 118

[MG11a] K. McGreggor and A. Goel. Finding the odd one out: a fractal analogical approach. In *Proceedings of the 8th ACM conference on Creativity and cognition*, pages 289–298, New York, NY, USA, 2011. ACM. 89, 91, 99, 106, 127

[MG11b] K. McGreggor and A. K. Goel. Fractally finding the odd one out: An analogical strategy for noticing novelty. In *AAAI Fall Symposium: Advances in Cognitive Systems*, pages 224–231, 2011. 89, 99, 106

[Mic83] RyszardS. Michalski. A theory and methodology of inductive learning. In RyszardS. Michalski, JaimeG. Carbonell, and TomM. Mitchell, editors, *Machine Learning*, Symbolic Computation, pages 83–134. Springer Berlin Heidelberg, 1983. 13

[Mil56] George A Miller. The magical number seven, plus or minus two: some limits on our capacity for processing information. *Psychological review*, 63(2):81, 1956. 3, 74

[Min82] Marvin Minsky. Semantic information processing. 1982. 1

[Min88] Marvin Minsky. *Society of mind*. Simon and Schuster, 1988. 76

[Mit82] Tom M. Mitchell. Generalization as search. *Artif. Intell.*, 18(2):203–226, 1982. 12

[Mit97] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997. 42

[MJMH07] S.T. Mueller, M. Jones, B.S. Minnery, and Julia M.H. Hiland. The bica cognitive decathlon: A test suite for biologically-inspired cognitive agents. In *Proceedings of Behavior Representation in Modeling and Simulation Conference, Norfolk*, 2007. 118, 123

[MKG10] K. McGreggor, M. Kunda, and A. Goel. A fractal analogy approach to the RavenâĂŹs test of intelligence. In *AAAI workshops at the 24th AAAI conference on Artificial Intelligence*, pages 69–75, 2010. 89, 91, 99, 106, 126, 127

[MKS92] Stephen Muggleton, Ross D King, and Michael JE Stenberg. Protein secondary structure prediction using logic-based machine learning. *Protein Engineering*, 5(7):647–657, 1992. 20, 32

[MKS$^+$13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013. 1

[ML13] Stephen Muggleton and Dianhuan Lin. Meta-interpretive learning of higher-order dyadic datalog: Predicate invention revisited. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 1551–1557. AAAI Press, 2013. 54

[MLPTN14] Stephen H Muggleton, Dianhuan Lin, Niels Pahlavi, and Alireza Tamaddoni-Nezhad. Meta-interpretive learning: application to grammatical inference. *Machine Learning*, 94(1):25–49, 2014. 54, 73

[MMPS94] Donald Michie, Stephen Muggleton, David Page, and Ashwin Srinivasan. To the international computing community: A new east-west challenge. *Distributed email document available from http://www. doc. ic. ac. uk/˜ shm/Papers/ml-chall. pdf*, 1994. 31

[MODS97] Fumio Mizoguchi, Hayato Ohwada, Makiko Daidoji, and Shiroteru Shirato. Learning rules that classify ocular fundus images for glaucoma diagnosis. In Stephen Muggleton, editor, *Inductive Logic Programming*, volume 1314 of *Lecture Notes in Computer Science*, pages 146–159. Springer Berlin Heidelberg, 1997. 20, 32

[Mon12] M. Montessori. *The Montessori method.* Frederick A. Stokes Co., New York City, USA, 1912. 251

[Mon95] David J Montana. Strongly typed genetic programming. *Evolutionary computation*, 3(2):199–230, 1995. 32

[MS95] Eric McCreath and Arun Sharma. Extraction of meta-knowledge to restrict the hypothesis space for ilp systems. In *AI-CONFERENCE-*, pages 75–82. Citeseer, 1995. 37

[MT93] Tom M. Mitchell and Sebastian B. Thrun. Explanation-based neural network learning for robot control. In *Advances in Neural Information Processing Systems 5*, pages 287–294. Morgan Kaufmann, 1993. 35

[Mue08] S.T. Mueller. Is the turing test still relevant? a plan for developing the cognitive decathlon to test intelligent embodied behavior. In *19th Midwest Artificial Intelligence and Cognitive Science Conference, MAICS*, 2008. 118, 123

[Mug87] Stephen Muggleton. Duce, an oracle-based approach to constructive induction. In *IJCAI*, pages 287–292. Citeseer, 1987. 37

[Mug95] S. Muggleton. Inverse entailment and progol. *New Generation Computing*, 13(3-4):245–286, 1995. 19, 32, 33, 175

[Mug99] S. H. Muggleton. Inductive logic programming: Issues, results, and the challenge of learning language in logic. *Artificial Intelligence*, 114(1–2):283–296, 1999. 19, 32

[NBBJ$^+$96] Ulric Neisser, Gwyneth Boodoo, Thomas J Bouchard Jr, A Wade Boykin, Nathan Brody, Stephen J Ceci, Diane F Halpern, John C Loehlin, Robert Perloff, Robert J Sternberg, et al. Intelligence: Knowns and unknowns. *American psychologist*, 51(2):77, 1996. 121

[New61] A. Newell. *Information processing language-V manual.* Prentice-Hall, 1961. 85, 104

[New73a] A. Newell. Productions systems : models of control structures. *Computer Science Department. Paper 2034, http://repository.cmu.edu/compsci/2034l*, 1973. 100

[New73b] A. Newell. You can't play 20 questions with nature and win: Projective comments on the papers of this symposium. In *Visual Information Processing, ed. W. Chase*, pages 283–308. New York: Academic Press, 1973. 77

[Nil05] Nils J Nilsson. Human-level artificial intelligence? be serious! *AI magazine*, 26(4):68, 2005. 2

[NS63] A. Newell and H.A. Simon. GPS, A program that simulates human thought. In E.A. Feigenbaum and J. Feldman, editors, *Computers and Thought*, pages 279–293. McGraw-Hill, New York, 1963. 80

[NTHW10] Andrew Nuxoll, Dan Tecuci, Wan Ching Ho, and Ningxuan Wang. Comparing forgetting algorithms for artificial episodic memory systems. In *Proc. of the Symposium on Human Memory for Artificial Agents. AISB*, pages 14–20, 2010. 155

[OK07] Pierre-Yves Oudeyer and Frédéric Kaplan. Dialog: How can we assess open-ended development? *The Newsletter of the Autonomous Mental Development Technical Committee*, 4(1):1–3, 2007. 119, 122

[Ols95] Roland Olsson. Inductive functional programming using incremental program transformation. *Artificial intelligence*, 74(1):55–81, 1995. 23, 31, 32, 37

[Ols99] J Roland Olsson. How to invent functions. In *Genetic Programming*, pages 232–243. Springer, 1999. 37

[OSTU13] S. Ohlsson, R. H. Sloan, G. Turán, and A. Urasky. Verbal IQ of a four-year old achieved by an AI system. In *COMMONSENSE 2013, 11th International Symposium on Logical Formalizations of Commonsense Reasoning*, page 6, 2013. 91, 94, 108, 253

[PG15] Oleksandr Polozov and Sumit Gulwani. Flashmeta: A framework for inductive program synthesis. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pages 107–126. ACM, 2015. 24

[Pia64] Jean Piaget. Part i: Cognitive development in children: Piaget development and learning. *Journal of research in science teaching*, 2(3):176–186, 1964. 123

[Plo70] G. Plotkin. A note on inductive generalization. *Machine Intelligence*, 5, 1970. 33

[Plo72] Gordon Plotkin. *Automatic methods of inductive inference.* PhD thesis, Edinburgh University, 1972. 33

[PR11] H. Prade and G. Richard. Analogy-making for solving IQ tests: A logical view. *Case-Based Reasoning Research and Development*, pages 241–257, 2011. 91, 92, 99, 107, 127

[PR13] Henri Prade and Gilles Richard. Picking the one that does not fit - A matter of logical proportions. In *Proceedings of the 8th conference of the European Society for Fuzzy Logic and Technology, EUSFLAT-13, September 11-13, 2013*, 2013. 91, 92, 107

[PR14] Henri Prade and Gilles Richard. From analogical proportion to logical proportions: A survey. In Henri Prade and Gilles Richard, editors, *Computational Approaches to Analogical Reasoning: Current Trends*, volume 548 of *Studies in Computational Intelligence*, pages 217–244. Springer Berlin Heidelberg, 2014. 91, 92, 99, 107

[PY10] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Trans. on Knowl. and Data Eng.*, 22(10):1345–1359, 2010. 35

[QCJ95] J. Ross Quinlan and R. Mike Cameron-Jones. Induction of logic programs: Foil and related systems. *New Generation Computing*, 13(3-4):287–312, 1995. 19, 32, 33

[Qui96] J. R. Quinlan. Learning first-order definitions of functions. *Journal of Artificial Intelligence Research*, 5:139–161, 1996. 32

[Qui12] R. Q. Quiroga. Concept cells: the building blocks of declarative memory functions. *Nature Reviews Neuroscience*, 13:587–597, 2012. 155

[Rav01] Raven's progressive matrices, 2001. 134

[RC96] J.C. Raven and J.H. Court. *Manual for Raven's Progressive Matrices and Vocabulary Scales: Standard progressive matrices*. Manual for Raven's Progressive Matrices and Vocabulary Scales. Oxford Psychologists Press, 1996. 134, 148, 247

[RCR92] J. C. Raven, J. H. Court, and J. Raven. *Manual for Raven's Progressive Matrices and Vocabulary Scale*. San Antonio, TX: Psychological Corporation, 1992. 85, 86, 124, 247

[Rea89] Chris Reade. *Elements of Functional Programming*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989. 21

[Rin93] Mark Ring. Learning sequential tasks by incrementally adding higher orders. In *Advances in Neural Information Processing Systems 5*, pages 115–122. Morgan Kaufmann, 1993. 36

[Rin97] Mark B. Ring. Child: A first step towards continual learning. In *Machine Learning*, pages 77–104, 1997. 36

[Ris78a] J. Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465 – 471, 1978. 44

[Ris78b] Jorma Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978. 45

[RK11] M. Ragni and A. Klein. Predicting numbers: an AI approach to solving number series. In *KI 2011: Advances in Artificial Intelligence*, pages 255–259. Springer, 2011. 90, 91, 93, 96, 99, 105

[RM91] Bradley L. Richards and Raymond J. Mooney. First-order theory revision. In *Proceedings of the Eighth International Machine Learning Workshop*, pages pp. 447–451, Evanston, IL, June 1991. 19, 37

[RMPRG86] David E. Rumelhart, James L. McClelland, and CORPORATE PDP Research Group, editors. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*. MIT Press, Cambridge, MA, USA, 1986. 14

[RN12] M. Ragni and S. Neubert. Solving RavenâĂŹs IQ-tests: An AI and cognitive modeling approach. In *ECAI*, pages 666–671. IOS Press, 2012. 90, 91, 98, 101, 126, 135, 138

[RN14] Marco Ragni and Stefanie Neubert. Analyzing ravenâĂŹs intelligence test: Cognitive model, demand, and complexity. In Henri Prade and Gilles Richard, editors, *Computational Approaches to Analogical Reasoning: Current Trends*, volume 548 of *Studies in Computational Intelligence*, pages 351–370. Springer, 2014. 90, 91, 98, 101

[Rui11] P. E. Ruiz. Building and solving odd-one-out classification problems: A systematic approach. *Intelligence*, 39(5):342–350, 2011. vii, 90, 91, 106, 127, 129, 130, 131, 133

[RV08] B. Raducanu and J. VitriÃă. Learning to learn: From smart machines to intelligent machines. *Pattern Recognition Letters*, 29(8):1024 – 1032, 2008. 155

[Sam59] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959. 76

[SAU13] C. Strannegård, M. Amirghasemi, and S. Ulfsbücker. An anthropomorphic method for number sequence problems. *Cognitive Systems Research*, 22âĂŞ23:27–34, 2013. 91, 94, 99, 104, 105, 114

[SB06] D Sculley and Carla E Brodley. Compression and machine learning: A new perspective on feature space vectors. In *Data Compression Conference, 2006. DCC 2006. Proceedings*, pages 332–341. IEEE, 2006. 46

[Sch04] Jürgen Schmidhuber. Optimal ordered problem solver. *Machine Learning*, 54(3):211–254, 2004. 37

[Sch10] K. Warner Schaie. *Primary Mental Abilities*. John Wiley & Sons, Inc., 2010. 82

[Sch13] C. Schenck. Intelligence tests for robots: Solving perceptual reasoning tasks with a humanoid robot. Master's thesis, Iowa State University, 2013. 6, 91, 93, 99, 107, 108, 111, 118, 123

[SCS13] C. Strannegård, S. Cirillo, and V. Ström. An anthropomorphic method for progressive matrix problems. *Cognitive Systems Research*, 22âĂŞ23(0):35–46, 2013. 91, 94, 98, 101, 127

[SD03] P. Sanghi and D. L. Dowe. A computer program capable of passing IQ tests. In *4th Intl. Conf. on Cognitive Science (ICCS'03), Sydney*, pages 570–575, 2003. 78, 87, 91, 96, 99, 100, 109, 120, 123, 127, 128, 148

[Sel] Google Self-Driving Car Project. `http://www.popsci.com/cars/article/2013-09/google-self-driving-car`. 1

[Sha48] C. E. Shannon. A mathematical theory of communication. *Bell system technical journal*, 27, 1948. 44

[Sha83] Ehud Y. Shapiro. *Algorithmic Program DeBugging*. MIT Press, Cambridge, MA, USA, 1983. 19

[Sha01] Lokendra Shastri. Biological grounding of recruitment learning and vicinal algorithms in long-term potentiation. In Stefan Wermter, Jim Austin, and David Willshaw, editors, *Emergent Neural Computational Architectures Based on Neuroscience*, volume 2036 of *Lecture Notes in Computer Science*, pages 348–367. Springer Berlin Heidelberg, 2001. 156

[SHFE14] Min Joon Seo, Hannaneh Hajishirzi, Ali Farhadi, and Oren Etzioni. Diagram understanding in geometry questions. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014. 81

[SHK09] Ute Schmid, Martin Hofmann, and Emanuel Kitzelmann. Analytical inductive programming as a cognitive rule acquisition devise. In B. Goertzel, P. Hitzler, and M. Hutter, editors, *Second Conference on Artificial General Intelligence*, pages 162–167. Atlantis Press, 2009. 23, 31, 38

[SHM$^+$16] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016. 1

[Shu12] Thomas R Shultz. A constructive neural-network approach to modeling psychological development. *Cognitive Development*, 2012. 127

[Sin01] P. Singh. The public acquisition of commonsense knowledge. Proceedings of AAAI Spring Symposium: Acquiring (and Using) Linguistic (and World) Knowledge for Information Access, 2001. 94, 108

[Sir] Apple's Siri. https://support.apple.com/en-us/HT204389. 1

[SJT08] R.L. Simpson Jr and C.R. Twardy. Refining the cognitive decathlon. In *Proceedings of the 8th Workshop on Performance Metrics for Intelligent Systems*, pages 124–131. ACM, 2008. 118, 123

[SK63] H. A. Simon and K. Kotovsky. Human acquisition of concepts for sequential patterns. *Psychological Review*, 70(6):534, 1963. 77, 84, 85, 86, 91, 99, 104, 114, 126, 141, 142, 144, 146, 246

[SK11] Ute Schmid and Emanuel Kitzelmann. Inductive rule learning on the knowledge level. *Cognitive Systems Research*, 12(3):237–248, 2011. 23, 95, 128

[SKN91] T. Simon, D. Klahr, and A. Newell. SCSoar: Pattern induction in series completion problem solving. In *European Soar Workshop*, page 17, Cambridge, 1991. 86, 91, 99, 102

[Slo03] N. J. A. Sloane. The on-line encyclopedia of integer sequences. *Notices of the AMS*, 50(8):912–915, 2003. 246

[SM02] DanielL. Silver and RobertE. Mercer. The task rehearsal method of life-long learning: Overcoming impoverished data. In Robin Cohen and Bruce Spencer, editors, *Lecture Notes in Computer Science*, volume 2338, pages 90–101. Springer, 2002. 35

[Sma73] Henry Small. Co-citation in the scientific literature: A new measure of the relationship between two documents. *Journal of the American Society for information Science*, 24(4):265–269, 1973. 46

[Smi84] Douglas R Smith. The synthesis of lisp programs from examples: A survey. *Automatic program construction techniques*, pages 307–324, 1984. 22

[SMKS94a] A. Srinivasan, S. H. Muggleton, R.D. King, and M.J.E. Sternberg. Mutagenesis: Ilp experiments in a non-determinate biological domain. In *Proceedings of the 4th International Workshop on Inductive Logic Programming, volume 237 of GMD-Studien*, pages 217–232, 1994. 20, 32

[SMKS94b] Ashwin Srinivasan, Stephen Muggleton, Ross D King, and Micheal JE Sternberg. Mutagenesis: Ilp experiments in a non-determinate biological domain. In *Proceedings of the 4th international workshop on inductive logic programming*, volume 237, pages 217–232. Citeseer, 1994. 31

[SNSE13] C. Strannegård, A. Nizamani, A. Sjöberg, and F Engström. Bounded Kolmogorov complexity based on cognitive models. In K. U. Kühnberger, S. Rudolph, and P. Wang, editors, *Artificial General Intelligence*, volume 7999 of *Lecture Notes in Computer Science*, pages 130–139. Springer Berlin Heidelberg, 2013. 91, 94, 99, 104, 114

[Sol64a] R. J. Solomonoff. A formal theory of inductive inference. Part I. *Information and control*, 7(1):1–22, 1964. 80

[Sol64b] Ray J Solomonoff. A formal theory of inductive inference. part i. *Information and control*, 7(1):1–22, 1964. 43

[Sol64c] Ray J Solomonoff. A formal theory of inductive inference. part ii. *Information and control*, 7(2):224–254, 1964. 43

[Sol66] R. J. Solomonoff. Some recent work in artificial intelligence. *Proceedings of the IEEE*, 54(12):1687–1697, 1966. 84

[Sol02] Ray J Solomonoff. Progress in incremental machine learning. In *NIPS Workshop on Universal Learning Algorithms and Optimal Search, Whistler, BC*. Citeseer, 2002. 37

[SP04] DanielL. Silver and Ryan Poirier. Sequential consolidation of learned task knowledge. In *Advances in Artificial Intelligence*, volume 3060 of *Lecture Notes in Computer Science*, pages 217–232. Springer, 2004. 35

[SP06] Daniel L. Silver and Ryan Poirier. Machine life-long learning with csmtl networks. In *AAAI*. AAAI Press, 2006. 35

[Spe04] C. Spearman. General Intelligence, Objectively Determined and Measured. *The American Journal of Psychology*, 15(2):201–92, 1904. 86

[Sri04] A. Srinivasan. *The Aleph Manual (Technical Report)*, 2004. 33

[SS06] Sylvain Sirois and Thomas R Shultz. Preschoolers out of adults: Discriminative learning with a cognitive load. *The quarterly Journal of experimental psychology*, 59(8):1357–1377, 2006. 149

[SS10] J. Sinapov and A. Stoytchev. The odd one out task: Toward an intelligence test for robots. In *Development and Learning (ICDL), 2010 IEEE 9th International Conference on*, pages

126–131. IEEE, 2010. 6, 89, 91, 93, 99, 107, 108, 111, 118, 123, 124

[SS12a] C. Schenck and A. Stoytchev. The object pairing and matching task: Toward montessori tests for robots. In *Ugur, E., Nagai, Y., Oztop, E., and Asada, M. (Eds) Proceedings of Humanoids 2012 Workshop on Developmental Robotics: Can developmental robotics yield human-like cognitive abilities?*, pages 7–13, 2012. 93, 99, 107, 108, 111, 252

[SS12b] M. Siebers and U. Schmid. Semi-analytic natural number series induction. In *KI 2012: Advances in Artificial Intelligence*, pages 249–252. Springer, 2012. 38, 91, 93, 96, 99, 104, 105

[SSF06] A. SjÃűberg, S. SjÃűberg, and K. ForssÃl'n. *Predicting Job Performance.* Assession Intenational, Stockholm, 2006. 94

[SSLZ09] Kaile Su, Abdul Sattar, Guanfeng Lv, and Yan Zhang. Variable forgetting in reasoning about knowledge. *Journal of Artificial Intelligence Research*, 35(2):677, 2009. 156

[SSS12] C. Schenck, J. Sinapov, and A. Stoytchev. Which object comes next? Grounded order completion by a humanoid robot. *Cybernetics and Information Technologies*, 12(3):5–16, 2012. 93, 99, 108, 111

[Ste00] R. J. Sternberg (ed.). *Handbook of intelligence.* Cambridge University Press, 2000. 77, 79

[Ste08] Robert J Sternberg. Increasing fluid intelligence is possible after all. *Proceedings of the National Academy of Sciences*, 105(19):6791–6792, 2008. 121

[Sum77] Phillip D. Summers. A methodology for lisp program construction from examples. *J. ACM*, 24(1):161–175, January 1977. 22, 23

[Sut98] R. Sutton. *Reinforcement Learning: An Introduction.* MIT Press, 1998. 55, 63, 68, 70

[SZW97] Jürgen Schmidhuber, Jieyu Zhao, and Marco Wiering. Shifting inductive bias with success-story algorithm, adaptive levin search, and incremental self-improvement. *Machine Learning*, 28(1):105–130, 1997. 37

[tag08] Handbook of behavioral neuroscience. In Lynn Nadel Ekrem Dere, Alexander Easton and Joseph P. Huston, editors, *Handbook of Episodic Memory*, volume 18 of *Handbook of Behavioral Neuroscience*, pages iii –. Elsevier, 2008. 156

[TD09] Elliot M Tucker-Drob. Differentiation of cognitive abilities across the life span. *Developmental psychology*, 45(4):1097–1118, 2009. 124

[THJA04] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. Support vector machine learning for interdependent and structured output spaces. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, pages 104–, New York, NY, USA, 2004. ACM. 34

[Thr96a] S. Thrun. *Explanation-Based Neural Network Learning: A Lifelong Learning Approach*. Kluwer Academic Publishers, Boston, MA, 1996. 35

[Thr96b] Sebastian Thrun. Is learning the n-th thing any easier than learning the first. In *Advances in Neural Information Processing Systems*, volume 8, pages 640–646, 1996. 35

[Thu38] Louis Leon Thurstone. *Primary mental abilities*. Chicago, Ill. : The University of Chicago press, 1938. 82, 87, 245

[TLFU05] E. Tomai, A. Lovett, K. Forbus, and J. Usher. A structure mapping model for solving geometric analogy problems. In *Proceedings of the 27th Annual Conference of the Cognitive Science Society*, pages 2190–2195, Stressa, Italy, 2005. Cognitive Science Society, Inc. 87, 88, 89, 91, 98, 103

[TO96] Sebastian Thrun and Joseph O'Sullivan. Discovering structure in multiple learning tasks: The tc algorithm. In *In International Conference on Machine Learning*, pages 489–497. Morgan Kaufmann, 1996. 35

[TT41] L.L. Thurstone and T.G. Thurstone. *Factorial studies of intelligence*. Psychometrika monograph suplements. The University of Chicago press, 1941. 77, 82, 124, 140, 245

[TT47] L. L. Thurstone and T. G. Thurstone. *American Council on Education for college freshmen: Manual of Instructions*. Cooperative Test Division, Educational Testing Service, 1947. 246

[Tur50a] A. M. Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950. 118

[Tur50b] Alan M Turing. Computing machinery and intelligence. *Mind*, pages 433–460, 1950. 187

[Tur11] P. D Turney. Analogy perception applied to seven tests of word comprehension. *Journal of Experimental & Theoretical Artificial Intelligence*, 23(3):343–362, 2011. 90, 91, 106, 108, 255

[TY97] Fumihide Tanaka and Masayuki Yamamura. An approach to lifelong reinforcement learning through multiple environments. In *Proceedings of the 6th European Workshop on Learning Robots (EWLR-6)*, pages 93–99, 1997. 36

[TY03a] F. Tanaka and M. Yamamura. Exploiting value statistics for similar continuing tasks. In *Robot and Human Interactive Communication*, pages 271–276, Oct 2003. 36

[TY03b] Fumihide Tanaka and Masayuki Yamamura. Multitask reinforcement learning on the distribution of mdps. In *Computational Intelligence in Robotics and Automation*, volume 3, pages 1108–1113 vol.3, July 2003. 36

[vABL04] L. von Ahn, M. Blum, and J. Langford. Telling humans and computers apart automatically. *Communications of the ACM*, 47(2):56–60, 2004. 119

[VAMM$^+$08] L. Von Ahn, B. Maurer, C. McMillen, D. Abraham, and M. Blum. RECAPTCHA: Human-based character recognition via web security measures. *Science*, 321(5895):1465, 2008. 119

[VL97] Paul Vitányi and Ming Li. On prediction by data compression. In *Machine Learning: ECML-97*, pages 14–30. Springer, 1997. 46

[VLDR01] Wim Van Laer and Luc De Raedt. How to upgrade propositional learners to first order logic: A case study. In *Machine Learning and Its Applications*, pages 102–126. Springer, 2001. 14

[VWW96] R. Virding, C. Wikström, and M. Williams. *Concurrent programming in ERLANG (2nd ed.)*. Prentice Hall International (UK) Ltd., Hertfordshire, UK,, 1996. 24, 55, 235

[Wal05a] Christopher S Wallace. *Statistical and inductive inference by minimum message length.* Springer Science & Business Media, 2005. 45

[Wal05b] C.S. Wallace. *Statistical and Inductive Inference by Minimum Message Length (Information Science and Statistics)*. Springer, 2005. 44

[Wat72] Satosi Watanabe. Pattern recognition as information compression. *Frontiers of Pattern Recognition. Academic Press, New York*, pages 561–567, 1972. 46

[WB68a] C. S. Wallace and D. M. Boulton. An information measure for classification. *The Computer Journal*, 11(2):185–194, 1968. 9, 44, 64, 159, 191

[WB68b] C. S. Wallace and D. M. Boulton. An information measure for classification. *Computer Journal*, 11(2):185–194, 1968. 43, 44, 80

[WBB11] Rachel Wood, Paul Baxter, and Tony Belpaeme. A review of long-term memory in natural and synthetic systems. *Adaptive Behavior*, page 1059712311421219, 2011. 154, 156

[WD92] C. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992. 36, 68

[WD99a] C. S. Wallace and D. L. Dowe. Minimum message length and Kolmogorov complexity. *Computer Journal*, 42(4):270–283, 1999. Special issue on Kolmogorov complexity. 80

[WD99b] C. S. Wallace and D. L. Dowe. Refinements of MDL and MML coding. *Comput. J.*, 42(4):330–337, 1999. 44

[Wec58] David Wechsler. *The Measurement and Appraisal of Adult Intelligence* . Williams & Wilkins Co, 1958. 121

[Wec81] D. Wechsler. *Wechsler Adult Intelligence Scale-Revised.* San Antonio, TX: Psychological Corporation, 1981. 87, 249

[Wen12] Juyang Weng. Symbolic models and emergent models: A review. *Autonomous Mental Development, IEEE Transactions on*, 4(1):29–53, 2012. 128

[Wik87] Ake Wikstrom. *Functional Programming Using Standard ML*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1st edition, 1987. 21

[Wix04] John T Wixted. The psychology and neuroscience of forgetting. *Annu. Rev. Psychol.*, 55:235–269, 2004. 155

[WWTP10] Zhe Wang, Kewen Wang, Rodney Topor, and JeffZ. Pan. Forgetting for knowledge bases in dl-lite. *Annals of Mathematics and Artificial Intelligence*, 58(1-2):117–151, 2010. 156

[WZZZ12] Yisong Wang, Yan Zhang, Yi Zhou, and Mingyi Zhang. Forgetting in logic programs under strong equivalence. In *KR*, pages 643–647, 2012. 156

[YLFA15] Yezhou Yang, Yi Li, Cornelia Fermüller, and Yiannis Aloimonos. Robot learning manipulation action plans by "watching" unconstrained videos from the world wide web. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, pages 3686–3693, 2015. 1

[Yon12] E. Yong. A large-scale model of the functioning brain. *Nature*, 29, November 2012. 78, 93, 122

[You15] Jia You. Beyond the turing test. *Science*, 347(6218):116–116, 2015. 115

[ZCC$^+$13] Liang-Jun Zang, Cong Cao, Ya-Nan Cao, Yu-Ming Wu, and Cun-Gen CAO. A survey of commonsense knowledge acquisition. *Journal of Computer Science and Technology*, 28(4):689–719, 2013. 31, 37

[ZF06] Yan Zhang and Norman Y. Foo. Solving logic program conflict through strong and weak forgettings. *Artificial Intelligence*, 170(8âĂŞ9):739 – 778, 2006. 156

[ZHE80] Thomas R. Zentall, David E. Hogan, and Charles A. Edwards. Oddity learning in the pigeon: Effect of negative instances, correction, and number of incorrect alternatives. *Animal Learning & Behavior*, 8(4):621–629, 1980. 129, 249

[ZHH74] Thomas Zentall, David Hogan, and Janice Holder. Comparison of two oddity tasks with pigeons. *Learning and Motivation*, 5(1):106–117, 1974. 129, 249

[ZZ09] Yan Zhang and Yi Zhou. Knowledge forgetting: Properties and applications. *Artificial Intelligence*, 173(16âĂŞ17):1525 – 1537, 2009. 156

Appendices

A

Erlang

Introduction

Erlang/OTP [VWW96] is a concurrent functional programming language (which includes environment and libraries) developed by Ericsson (and used in production systems for over 20 years) and “was designed from the ground up for writing scalable, fault-tolerant, distributed, non-stop and soft-realtime applications (like telecommunications systems)”. The core Erlang language consists of function definitions extended with message passing to handle concurrency and distribution, and *OTP* (which stands for *Open Telecom Platform*), a set of design principles and ready-to-use components that supports building fault-tolerant systems. The Erlang programming language is gaining momentum and its success is witnessed by the increasing number of its industrial applications, i.e.: *Facebook* and *Tuenti*'s chat back-end, *Whatsapp*, *SimpleDB* (a distributed database that is part of *Amazon Web Services*), *GitHub* (used for RPC proxies to ruby processes), *Motorola*, *Rakuten*, ...

The main reasons why we have chosen Erlang as the programming language of our system are: firstly, it is a free and open-source language with a large community of developers behind that implies that there is a large repository of libraries to deal, among other things, with the meta-level of the source code in an easy way; secondly, reflection and higher order, that allows us to interact easily with the meta-level representation of how rules and programs are transformed by operators; and finally, it is a unique representation language, which is appropriate for our requirements: operators, examples, models and background knowledge are represented in the same language. The advantages of using the same representation language has been previously shown by the fields of ILP, IFP and IFLP (except for operators).

Functional Erlang

A main difference between Erlang and more popular languages is that Erlang is primarily a functional programming language meaning that it emphasizes the evaluation of expressions rather than the execution of commands. Meanwhile the imperative programming paradigm rely on changing the state of the application during execution producing different results with the same input values, with functional programming, the functions and operations of the language are designed in a similar way to mathematical calculations where the expressions operate with functions to derive consistent basic values or results for the same inputs. This makes predicting the output of the function or program much easier and, therefore easier to debug and analyze.

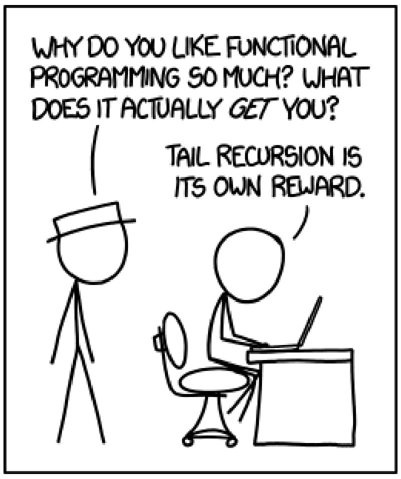

WHY DO YOU LIKE FUNCTIONAL PROGRAMMING SO MUCH? WHAT DOES IT ACTUALLY *GET* YOU?
TAIL RECURSION IS ITS OWN REWARD.

Figure A.1: Functional programming combines the flexibility and power of abstract mathematics with the intuitive clarity of abstract mathematics. Extracted from http://www.explainxkcd.com/1270

Particularly, Erlang is a relatively small (PROLOG-inspired[1]) and simple language which basics are: numbers (*integers* and *float*), *atoms* (for comparison and starting with a lowercase letter or you can delimit with single quotes), invariable *variables* (starting with a uppercase letter, once a variable is binding to a value, it cannot be changed) , *strings*, *tuples* (composite data type and are used to store collections of items delimited by curly brackets), *lists* (bracketed, comma-separated list of values), *bit strings*, *binaries* (enclosed in double angle brackets), `fun` expressions (to create anonymous functions) and

[1]The first implementation of Erlang was written in NU Prolog [AVW92]

process identifiers (*pids*, the “names” of processes) . Lists and tuples may look similar, but whereas a tuple can only be used in a comparison, lists allow a wider variety of manipulation operations to be performed (by using operators such as the *list comprehensions*, `head`, `tail`, |, ++ or -).

Erlang syntax includes a record construct providing syntactic sugar for accessing the elements of a tuple by name instead of by position. Patter matching-based functions are first class citizens in Erlang composed by their *name* (*atom* indicating its name), *arity* (number of patterns given as parameters), *variable identifiers* (any valid parameter), *guards* (additional clauses that can go in a function's head to make pattern matching more expressive) and the *body* which consists of an sequence of expressions, separated by commas; the value of the sequence, and therefore the value of the function, is the value of the last expression evaluated. For example, consider the declaration of the function factorial (Algorithm A.1 which calculates the factorial of an specified number.

```erlang
-module(fact).  % module declaration %
-export([factorial/1]).  % functions exported with their
    arity %
factorial(0) ->
  1;
factorial(N) when N > 0 ->
  N * factorial(N-1).
```

Code A.1: Factorial

Erlang, as a functional language, runs a program as a successive application of functions over an initial expression (free of variables). Branches of execution are selected based on pattern matching and loops are constructed using recursive functions. Erlang has a declarative syntax and is largely free from side-effects, i.e., most constructs (*pure functions*) return the same value given the same arguments regardless of the context of the call of the function. Exceptions are message passing and built-in functions (*BIFs*). One interesting feature of Erlang is its strong dynamic nature. Although variables are dynamically typed, there is no type checking at the compile-time, Erlang is *type safe*: all values are tagged with their type during run-time and an exception is thrown if a type clash occurs. Function identifiers are a special data type called *pids* and they can be generated at run-time and passed around in variables (higher order abilities). Execution threads are also created at run time, and they are identified by a dynamic system.

What makes Erlang a true functional programming language is the ability to use a functions just like any other sort of data. They can be passed as arguments to other functions and therefore, they can be stored in data structures like tuples and records or sent as messages to other processes. Most of all, they can be the result or output of the processing of other functions so that the functions passed around as data can be created dynamically within the program, and are not just pointers to statically defined functions. This powerful means of abstraction, named as higher order functions, is rooted in mathematics, mainly lambda calculus, where everything is a function, even numbers, and due to that, functions must accept other functions as parameters and can operate on them with even more functions. Erlang provides the classic set of higher-order functions that functional languages are famous for: anonymous functions (created on the fly maybe inside other piece of code), *map* (which applies a function to each element of a list, returning a new list with the results), *filter* (selects the element of a list that satisfy a predicate.), *fold* (also known as reduce, recursively apply a function to pair of elements from a list, starting with a seed (accumulator) that is passed to it and is usually the neutral element for the function), . . .

List comprehensions is another powerful construct whose roots lie in functional programming and, perhaps, it is a mild form of metaprogramming: they compress logic in new mathematical constructs. The logic of what you would write as `for()` cycles in an imperative language is exposed here in a generator, so that you can write the interesting part (what to extract) in the shortest possible form, associating a list to a variable representing its element. See for instance the code in Algorithm A.2 whose, in English, stands for: build a list, L with elements that have the value $X * X$, such that X is taken from the list $[1, 2, 3, 4]$, giving as output $[1, 4, 9, 16]$. Note that the previous list comprehension is similar to `lists:map(fun(X) -> X*X end, [1,2,3,4])` In fact, list comprehension provides a shorthand notation for most of the functions in the lists module.

```
L = [ X*X || X <- [1,2,3,4] ].
```

Code A.2: Easy list comprehension

`Quicksort` is a very famous example of the power of list comprehensions, Algorithm A.3 shows the code for the quicksort algorithm implemented in two lines by using lists comprehensions. In English, this would be read as break a list into head X and tail Xs and return a new list composed of three sublists: (1) ordered list (qsort) of all Y from the tail Xs such that Y is less than or equal to X, (2) list containing just X and (3) ordered list (`qsort`) of all Y from the tail Xs such that Y is greater than X.

```
qsort([X|Xs]) ->
  qsort([Y || Y <- Xs, Y =< X]) ++ [X] ++ qsort([Y || Y <-
    Xs, Y > X]).
```

Code A.3: Quicksort using list comprehensions

Dynamic typing in Erlang

That Erlang has a dynamic typing means that it is not necessary to write the type of a variable or the type of a function. This implies that when pattern matching, the code we write does not have to know what it would be matched against, namely, the tuple `{X,Y}` could be matched with `{atom, 123}` as well as `{"A string", «"binary stuff!"»}`, `{2.0, ["strings","and",atoms]}` or really anything at all. When an error occurs it is caught at runtime (unlike static type systems which enforce compilers to catch most errors before execute the code) and the compiler does not always indicates where things may result in failure. This can be seen as weakness in safety for many dynamic languages, however, Erlang begs to differ and certainly has a track record to prove it partially because Erlang is built on the notion that a failure in one of the components should not affect the whole system. Erlang uses a strategy for getting round errors coming from the programmer, hardware or network, where the language includes features which will allow you to distribute a program over to different nodes, handle unexpected errors, and never stop running.

Dynamic typing has also some advantages for the programmers due to it avoids the frustrations of having to convince the type system that one really knows what they are doing. Furthermore, since type declarations and annotations need not be typed (in), program development can progress more rapidly. Unfortunately, this freedom of expression comes with a price. Significantly less typos and other such mundane programming errors are caught by the compiler. More importantly, this freedom considerably obstructs program maintenance: it is extremely difficult to recall or decipher how a particular piece of code âĂŤ often written by some other programmer years ago âĂŤ can be used (comments can be unreliable, often cryptic and confusing).

Implementing such a type system would likely mean forcing Erlang to change its semantics: analyzers will want to actually prove that there will be no type errors at run time, as in mathematically prove. This means that in a few circumstances, the type checker will disallow certain practically valid operations for the sake of removing uncertainty that could lead to crashes. The other option is then to have a type system that will not prove the absence of errors, but will do a best effort at detecting whatever it can. You can make such detection really good, but it will never be perfect, and this is where *Dialyzer* [LS06] comes into play. Dialyzer is a type checker for Erlang that work without type declarations (although you can define types to help out Dialyzer in its task), is simple and readable, it adapts to the language (and not the other way around), and only complain on type errors that would guarantee a crash. Therefore, although defining types in Erlang have no effects or restrictions on actual compiling unless you enforce them yourself, it is possible to declare types and annotate functions in order to both document them and help formalize the implicit expectations about types we put in our code.

Metaprogramming in Erlang

The following Erlang expression:

```
Exprs = foo:bar(baz,17).
```

it is represented in abstract format by the following tuple:

```
Exprs= [{call,1,{remote,1,{atom,1,foo}, {atom,1,bar}},[{
    atom,1,baz},{integer,1,17}]}]
```

which, in fact, describes the tree representation of the abstract syntactic structure of source code written in Erlang. It is called as the *Abstract Syntax Tree* (AST) or just syntax tree. Figure A.2 represents the tree representation of the previous abstract expression. Each node of the tree denotes a construct occurring in the source code which does not include certain elements appearing in the real syntax (such as inessential punctuation and delimiters). Some advantages of the ASTs is that they can be edited and enhanced with properties and annotations for every element it contain and, furthermore, they contain extra information about the program (such as the position of an element in the source code).

In Erlang' ASTs, *modules* are represented with a list `[F1...Fn]`, where each `F` represents a *form*. A *form* is either an attribute or a function declaration. Concretely:

- For instance, the abstract format corresponding to the attribute `module (Mod)` is `{attribute,LINE,module,Mod}`.
- The abstract format corresponding to a *function declaration* is `{function, LINE,Name,Arity,[FC1...FCn]}` where each `FC` is the abstract format of a function clause, which in turn is represented by `{clause,LINE, [P1...Pn],[G1...Gn],[E1...En]}` where each `P`, `G` and `E` is the ab-

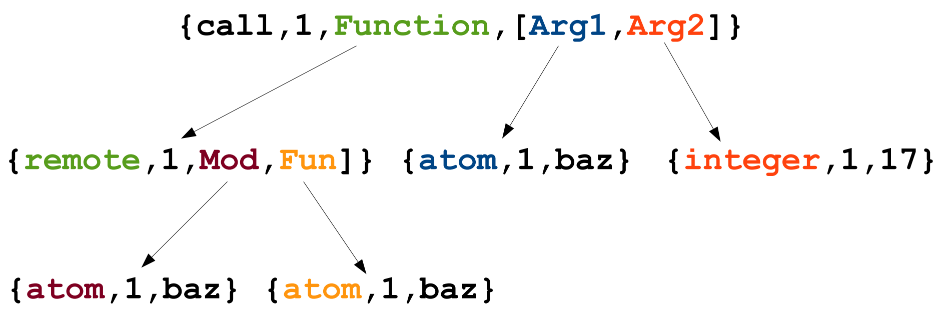


Figure A.2: Tree structure of the tuple: {call,1,{remote,1,{atom,1,foo}, {atom,1,bar}},[{atom,1,baz},{integer,1,17}]}

stract representation of one of its pattern, one of its guards and one of its bodyâĂŹs expressions respectively.

The easiest way to get the abstract form corresponding to a source code is using the web development framework for Erlang *ErlyWeb* and the `smerl.erl`[2] library (Simple Metaprogramming for Erlang) included. By using this module we can get the abstract form of all the function declarations of a module from its source file with a few lines of code:

```
{ok,Meta_model} = smerl:for_module(fact),
Form = smerl:get_forms(Meta_model),
```

The corresponding abstract form for its function declarations is:

```
[{function,4,factorial,1,
   [{clause,4,[{integer,4,0}],[],[{integer,5,1}]},
   {clause,6,
      [{var,6,'N'}],
         [[{op,6,'>',{var,6,'N'},{integer,6,0}}]],
         [{op,7,'*',
            {var,7,'N'},
            {call,7,
               {atom,7,factorial},
               [{op,7,'-',{var,7,'N'},{integer
   ,7,1}}]}}]}]}]
```

Additionally, with smerl it is possible to create and compile easily a new module allowing to call its functions by:

[2]`https://code.google.com/p/erlyweb/`

```
-module(try_smerl).
-export([test_smerl/0]).

test_smerl() ->
    M1 = smerl:new(foo),
    {ok, M2} = smerl:add_func(M1, "bar() -> 1 + 1."),
    smerl:compile(M2),
    foo:bar().
```

thus being possible to compile and launch this module (`try_smerl`) obtaining 2 as an output of the call (`foo:bar()`) made in the body of the function `test_smerl()`.

Concurrent and Distributed Erlang

One of the main reasons for using Erlang instead of other functional languages is Erlang's ability to handle concurrency (several threads of execution at the same time) and distributed programming. Erlang systems can be viewed as a collection of Erlang nodes where each one is a collection of processes, with a unique node name. The Erlang BIF `spawn` is used to create a new process (`Pid = spawn(Fun)`). The only method by which Erlang processes can exchange data is message passing.C ommunication is *asynchronous* and point-to-point, with one process sending a message (simply valid Erlang terms) to a second process identified by a process identifier, or *pid*, which uniquely identifies the process (`Pid ! Message`). The `receive` construct is used to allow processes to wait for messages from other processes. Each process has its own input queue for messages it receives, also referred to as mailbox. Informally, a mailbox is a sequence of values ordered by their arrival time which are matched (orderly) against the pattern in the `receive`.

As an alternative to addressing a process using its *pid*, Erlang provides a mechanism (the `register` *BIF*) for processes to be given names so that these names can be used as identities . The name, which must be an atom, is automatically unregistered when the associated process terminates. Message passing between processes in different nodes is transparent when pids are used, i.e., there is no syntactical difference between sending a message to a process in the same node, or to a remote node. However, the node must be specified when sending messages using registered names, as the *pid* registry is local to a node.

A unique feature of Erlang that greatly facilitates building fault-tolerant systems is that processes can be "linked together" in order to detect and recover from abnormal process termination. If a process P_1 is linked to a process P_2, and P_2 terminates with fault, process P_1 is automatically informed of the failure of P_2. It is possible to create links to processes at remote nodes.

As commented before, an integral part of Erlang, the OTP library provides a number of very frequently used design patterns (behaviours in Erlang terminology) for implementing robust distributed and concurrent systems (client/server architectures, supervisors, finite state machines, ...).

An echo process

As a simple example of a concurrent process we will create an echo process which echoes any message sent to it. Let us suppose that process A sends the message `{A, Msg}` to the echo process, so that the echo process sends a new message containing Msg back to process A. This is illustrated in Figure A.3

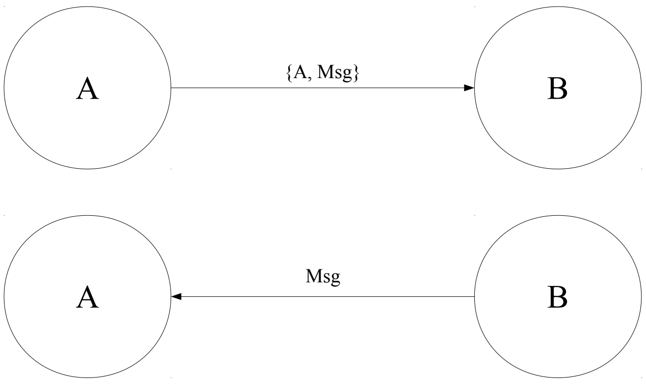


Figure A.3: An echo process in Erlang.

In Algorithm A.4 the function `echo:go()` creates a simple echo process which returns any message sent to it.

```
-module(echo).
-export([go/0, loop/0]).
go() ->
  % echo:loop() function (Pid2) is created and evaluated
    in parallel with the calling function %
  Pid2 = spawn(echo, loop, []),
  % Message "hello" sent to Pid2 %
  Pid2 ! {self(), hello},
  receive
    % A message from Pid2 is  %
    {Pid2, Msg} ->
      % Execution function if the message is received %
      io:format("P1 ~w~n",[Msg])
  end,
  Pid2 ! stop.

% function to be spawed by go() %
loop() ->
  receive
    % Message received for whatever process (which knows
    Pid2) %
    {From, Msg} ->
      % Execution function: Echo %
      From ! {self(), Msg},
      loop();
    after
      % if no message is received after 100ms , suspend
    the process %
      100 ->
        true
  end.
```

Code A.4: Echo program

B IQ tests

introduction

For the sake of exposition, we will arrange the test problems into three categories: general intelligence problems (usually related to the g-factor), other abstract pattern problems and knowledge-intensive problems. The two first categories do not require previous knowledge (or it is restricted to some basic arithmetic or geometrical operations). The last category requires the application of previous knowledge (and language).

General intelligence problems

In this first category we include several fluid intelligence problems that highly correlate with the g-factor. Typically, solving these problems involves inductive inference, abstraction, search and combination.

Thurstone letter series completion (Lett-S)

The letter series completion problems were introduced as part of some test batteries. These problems were developed to assess "reasoning ability". This is one of the seven factors (verbal comprehension, word fluency, number facility, spatial visualization, associative memory, perceptual speed and reasoning) identified by Thurstone in his Primary Mental Abilities (PMA) theory [Thu38, TT41]. The goal is to identify the following letter in a series (see Figure B.1) from five letter choices. To solve a letter series, an abstract pattern has to be identified which captures the regularity of the sequence. The correct answer can be generated by applying this pattern. Correctness in the context of induction problems typically presupposes that there is one continuation which is most plausible with respect to the given regularity. Typically, problems are carefully constructed in such a way that there is a unique solution. However, there is also research on ambiguous problems, for example in the domain of letter string analogies [Hof08].

ababababab___
aaabbbcccdd___
cadaeafa___
wxaxybyzczadab___
mnlnknjn___

Figure B.1: Examples of Thurstone letter series completion problems [SK63].

Number series (Numb-S)

Similar to the Thurstone letter series completion problems we find the continuation of number series. This kind of problem is also included in some intelligence tests (such as the German IST-2000 [ABLB99]). Number series problems can be constructed to be much more complex than letter series due to the greater number of combinations and operations, as, e.g., the Fibonacci sequence $1, 1, 2, 3, 5, 8$. In fact, the problem gets closer to a crystallised intelligence task (instead of a fluid intelligence task) when sequences are generated by the use of a large library of mathematical functions, so requiring a correct memory search and retrieval by the subject. For instance, some problems can be just obtained from the Online Encyclopedia of Integer Sequences (`http://oeis.org`)[Slo03].

Geometric analogy in American Council of Education tests (ACE-A)

The ACE tests [TT47] were designed to predict or measure the scholastic aptitude (or general intelligence) of the participants through the use of "differential prognosis", which involves a weighting of factors that are considered important in the curricula of American colleges. However, it has been subject to serious criticism due to the fact that intelligence is only one of factors that influence achievement. The examination consists of six tests arranged into two groups: *Quantitative Tests*, which includes arithmetical reasoning, number series, and figure analogies tasks; and *Linguistic Tests*, which includes same-opposite, completion, and verbal analogies tasks. In particular, we are interested in the the so-called "geometric-analogy" or "figure analogies" problems. The task to perform can be described as: "figure A is to figure B as figure C is to _” where “A”, “B”, and “C” are given and “D” should be derived (Figure B.2). Note that in order to solve these tasks, the process must involve some spatial reasoning and similarity recognition to identify the figures and occasionally perform spatial operations on them.

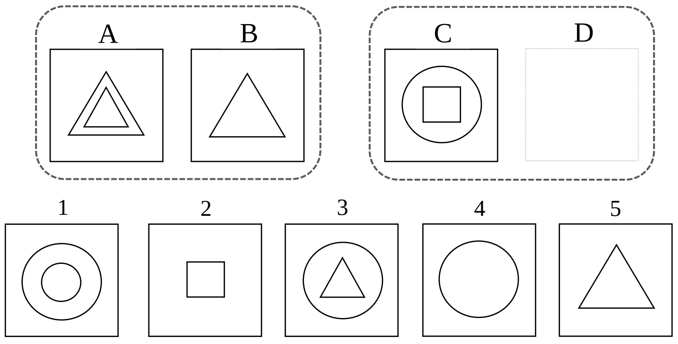


Figure B.2: ACE problem figure (redrawn following [Eva65], originally from the 1942 edition of the Psychological Test for College Freshmen of the American Council on Education, ACE).

Raven’s Progressive Matrices (RPM)

Originally from Raven [RCR92], Raven’s Progressive Matrices consist of a pattern or a set of items where a missing part or item has to be guessed. The most typical case is a 3×3 grid where a figure is placed at each of the nine positions except the bottom-right cell, which is empty (see Figure B.3)[1]. Eight possible choices (‘distractors’) to fill in the gap are displayed at the bottom. There is a logical relation between the figures, which can be seen either horizontally (rows) or vertically (columns). There are three different sets of Raven Progressive Matrices (RPM) for participants of different abilities [RCR92]: RPM for participants of different IQ ranges or different abilities [RC96]:

- Standard Progressive Matrices (SPM): the original set of matrices first published in 1938. It consist in five sets (from A to E) each one comprising 12 items —60 items in total. Each set starts with a problem which is, as far as possible, self-evident and becomes progressively more difficult.

[1]Please note that items from standardised IQ tests are not allowed to be published due to copyright issues. Therefore, we can only present some examples or problems which are similar but not identical to the ones used in IQ tests.

- Coloured Progressive Matrices (CPM): for children aged 5 through 11 years-of-age, the elderly, or people with learning difficulties, it consist in 3 sets of 12 items (A and B from SPM and a further set of 12 items inserted between the two, as set Ab). Most of them are presented on a coloured or b/n background to make the test visually stimulating for participants.

- Advanced Progressive Matrices (APM): developed to assess individuals of above-average intelligence, it contains 48 items (of increasingly difficulty) presented as one set of 12 (set I), and another of 36 (set II).

As in the previous case, these problems involve processes that require spatial reasoning abilities, similarity recognition, and analogies, but also sequences.

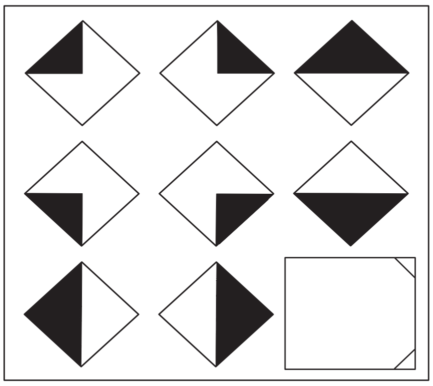


Figure B.3: Raven's Progressive Matrices example [own example].

Other abstract pattern problems

In this category, we include some problems that do not require the application of a significant amount of knowledge but cannot be considered general, as they measure very specific abilities.

Block design in WAIS (WAIS-B)

These are geometrical problems where blocks with only white, only red, and both white and red sides have to be arranged according to a presented pattern (see Figure B.4). This task aims at measuring spatial perception, visual abstract processing, and problem solving abilities. The difficulty does not lie in identifying the relation between the blocks but to figure out how to rotate and combine the blocks such that they correspond to a *given* pattern.

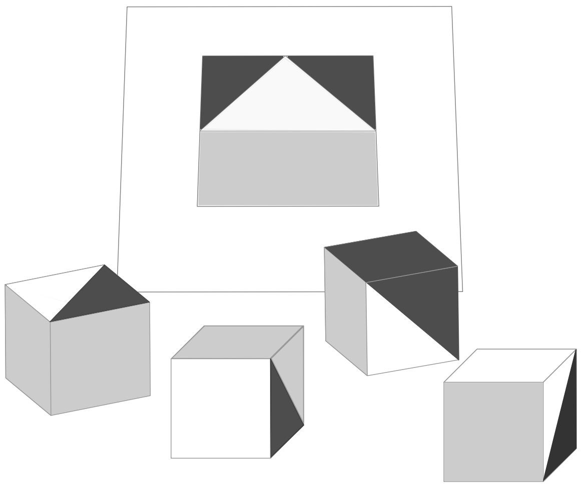


Figure B.4: A block problem such as those found in Wechsler Adult Intelligent Scale, WAIS [Wec81]. The picture shows one problem from the Kohs Block Design test developed by Samuel Calmin Kohs and subsequently adapted into WAIS by David Wechsler. (Taken from Wikimedia Commons: http://commons.wikimedia.org/wiki/File:Kohs_Block_Design_Test_-_Figure_1.jpg)

Odd-one-out problems (OOO)

The odd-one-out problems are focussed on geometry and spatial understanding, where the goal is to spot the most dissimilar object from the rest (see Figure B.5). They were first introduced in [ZHH74, ZHE80] as a non-verbal test in the study of comparative animal learning (Zentall et al. [ZHH74] assessed pigeon's intelligence), where some species appears to learn it readily, slowly or nothing at all. The oddity task has also been used for cross-cultural testing to probe the conceptual primitives of geometry in the Mundurukú, an isolated Amazonian indigenous group, for which Deheane et al. [DIPS06] designed a visual oddity task. This is a fundamental task used in a wide variety of intelligence tests to evaluate human or animal intelligence and some cognitive abilities are required, including visual encoding, pattern detection, similarity assessment, analogical transfer and creativity. Given a set of items, the participant is asked to decide which one among them is most dissimilar from the rest, namely addressing the identification of relations in geometrical patterns. Typically, the presented items can vary along several dimensions (e.g., shape, size, quantity). The problems can be automatically generated by using a complex set of algorithms (http://www.cambridgebrainsciences.com/). Due to this automatic generation and the possibility of generating countless novel problems, the participant cannot rote learn how to solve them in advance.Consequently, “the task is suitable for training reasoning abilities or taking many repeated measures”.

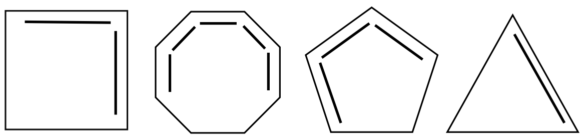


Figure B.5: An example of an odd-one-out problem [own example].

Bongard’s analogy problems (Bong-A)

Bongard’s analogy problems are a set of puzzles about visual categorisation that were originally introduced for *visual pattern recognition*. They were invented by the Russian computer scientist M. M. Bongard [Bon70], who provided a notable subset of a hundred problems in the appendix of the original publication. Each problem is formed by six boxes on the left side and another six on the right side containing relatively simple diagrams (see Figure B.6). The goal is to guess the pattern, or rule, that appears in the left set which is lacking in all the diagrams of the set on the right. For instance, the left set in Figure B.6 follows two patterns: (1) all figures have only 3 sides, and (2) 2 out of the 4 small white circles that appear in each diagram are closer together than the others. These previous patterns do not appear on the right side. Bongard was interested in the automation of visual perception, so the original aim to develop them was to provide a computer system with a black-and-white set of static (outlined or filled) figures in two dimensions, as challenges for pattern recognition algorithms. However, a few years later, Hofstadter [Hof79] was more interested in the problems themselves as abstract analogy problems, and referred to them as “Bongard Problems”, which is the term that has prevailed thereafter in the literature.

String analogies (Str-A)

Originally referred to as “word analogies” (we use the term ‘string analogies’, as they are not really words of a language), Hofstadter [Hof83] introduced short alphabetic sequence puzzles (see Figure B.7) as input problems. He

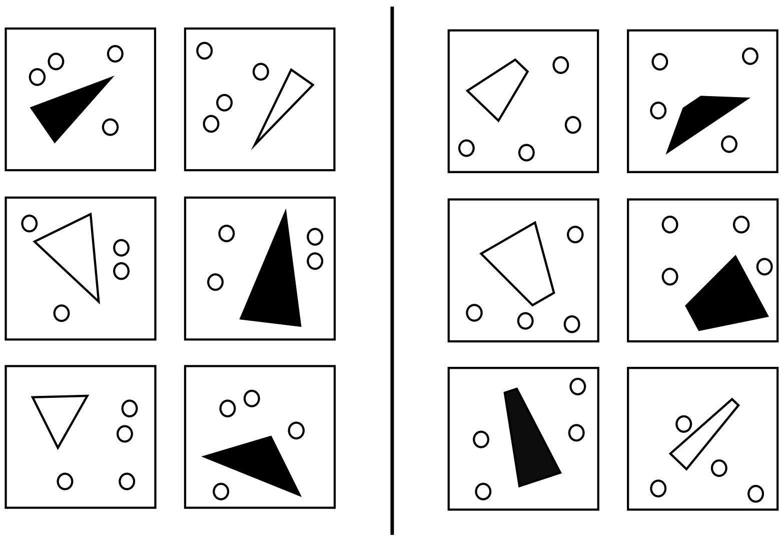


Figure B.6: An example of a Bongard problem [own example].

focussed on Bongard's problems, and simplified them in order to avoid the difficulties arising when constructing representations for visual problems. The processes involved in this problem are similar to the letter series problem but with the additional use of analogical transfer.

abc ⇒ abd	ijk ⇒ ?
aabc ⇒ aabd	ijkk ⇒ ?
aac ⇒ abd	kji ⇒ ?
abc ⇒ abd	mrrjjj ⇒ ?
abc ⇒ abd	xyz ⇒ ?

Figure B.7: String analogy problems (originally referred to as "word analogies") analysed by the Copycat project [Hof08].

Montessori's object matching (Mont-O)

The Montessori method [Mon12] is an educational method developed by Maria Montessori, an influential Italian physician and educator, in 1897 (but popularised in the 1910s). This method encouraged children to focus their attention on one particular 'quality', working at his/her own optimum level with a natural psychological, physical, and social development, so it makes emphasis on independence and freedom within limits. This method tries to stimulate, among other abilities, sensory, language and numeracy skills. Several tasks require the children to compare objects, moving, rotating, or touching them (see Figure B.8). One typical task inside the Montessori method is *Object-to-object matching*, where the child is asked to find the matches from one set to another, e.g., matching coloured tiles, 3-dimensional shapes, or pieces of textured cloth. This problem requires some cognitive abilities, including visual encoding, object grouping, similarity assessment, category recognition, and object ordering. Recently, a sensorimotor robot has been able to solve this kind of object pairing task [SS12a].

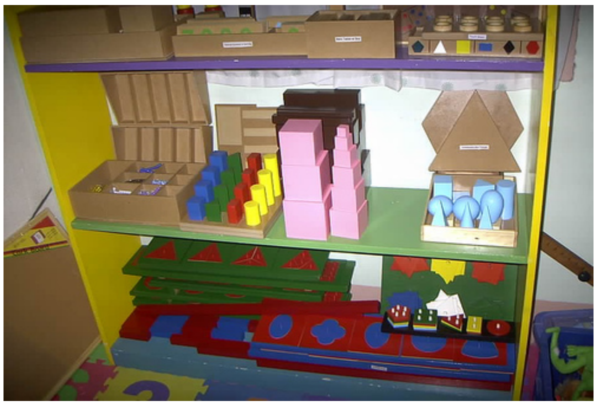


Figure B.8: Materials used in some exercises and tests of the Montessori method. ["Montessori Materials", Diamond Montessori, retrieved from https://www.flickr.com/photos/rossmenot/351505158/, CC BY 2.0]

Knowledge-intensive problems

Finally, we group another set of tasks that depend on the crystallised use of knowledge and experience.

Hofstadter's anagrams (Jumbles)

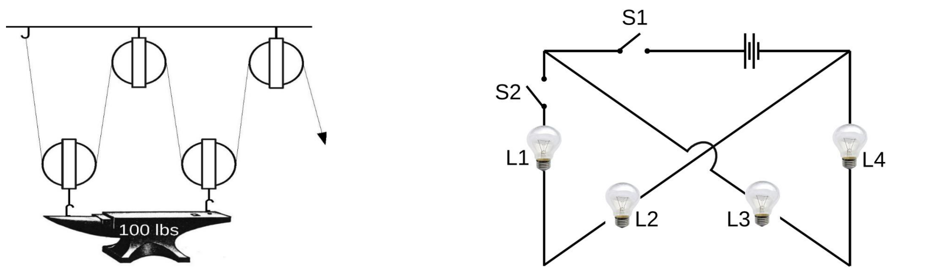
Frequently found in newspaper puzzle sections, jumbles are games where five or six letters are given with the aim of unscrambling them into an English word. In order to solve them, humans try to find common letter patterns in order to generate blocks. For instance, in the jumble "yurfip" we are able to find the common letter sequence "ify" because all the letters are present. Once we subtract these letters, the remaining letters are "urp", so there are two plausible possibilities (consonant + vowel + consonant): “rupify” and “purify”, the latter being the right one. Clearly, these tasks require the use of a large background knowledge (vocabulary) but also the ability of making combinations. Jumbles were used by Hofstadter as a domain of his prototype system Jumbo [Hof83], prelude to the Copycat project [HM84].

Verbal common-sense reasoning problems in WPPSI (WPPSI)

Wechsler developed several variants of WAIS that were aimed for individuals aged under 16: the Wechsler Intelligence Scale for Children (WISC) and the Wechsler Preschool and Primary Scale of Intelligence (WPPSI). Similar to WAIS, the WISC is used as an intelligence test for individuals between 6 and 16 years old, which includes the same kind of tests as WAIS-IV but features two supplemental tests: Cancellation and Word Reasoning. These latter tests are used to accommodate rare individuals. WAIS can also be used as a clinical tool to diagnose attention-deficit hyperactivity disorder (ADHD) and learning disabilities. WPPSI is a descendent of the WAIS and WISC tests, and is also used as an intelligence test but for individuals between 2 years 6 months and 7 years 3 months using the same kind of test as the previous ones. The verbal part of the WPPSI (WPPSI-III, third Edition) includes questions of the type: “What color is the sky?”, “What are pancakes made out of?”, “Finish what I say. Pen and pencil are both ...”, “You can see through it”, “It is square and you can open it”, etc. This task has been used to evaluate the common-sense reasoning system ConceptNet 4 [OSTU13]. The processes involved are language understanding and common-sense reasoning.

Bennett mechanical comprehension tests (BMCT)

The Bennett Mechanical Comprehension Test (BMCT) [Ben69] is an aptitude test for the evaluation of human common sense relying on everyday reasoning. Namely, it measures the ability to perceive, understand, and reason about the physical world (with an important content about physics and contextual information) and is commonly used to evaluate applicants for technical positions (typically, electrical and mechanical positions). The BMCT is composed of 68 diagrams depicting physical situations involving many different mechanisms, and is accompanied by multiple-choice questions about their qualitative properties. The test is time-limited (30 minutes) and the required reading and exercise level of concentration is below or at a sixth-grade reading level. To solve this test, some abilities are required: spatial reasoning, conceptual knowledge spanning a broad range of domains (dynamics, acoustics, statics, electricity, and heat), and experience with a wide variety of everyday situations (boats, trains, bicycles, cranes, hoists, . . .). In Figure B.9 we can see two examples that represent the two general types of problems that appear on the BMCT [KFTK11]: (a) *outcome problems*, where we need to predict a property of the system, and (b) *model formulation*, where we need to reasoning qualitatively using a broad set of everyday life concepts observing a causal model. In case of the weight (Figure 1 in B.9), the examinee must remember to divide the weight by the number of sections of rope supporting it to get the force needed to lift the weight, so it is required a force equal to 12 lbs (answer D), whereas, in the case of the tools for breaking concrete, it seem to be more suitable the answer D, based on some previous background knowledge and knowing that the rest of tools are for hammering nails (answer B), made of rubber and very weak (answer C) or are used in metalworking (answer E). We have included this test (even if it is not properly an IQ test) as a good example of a test that requires many abilities.



1. Approximately how much force is needed to lift the anvil?

a) 100 lbs b) 50 lbs c) 25 lbs d) 10 lbs e) 200 lbs

2. If the switch S1 is closed and switch S2 remains open, which lights will turn on?

a) L1&L2 b) All c) L3&L4 d) L1 &L4 e) None

Figure B.9: Figurative recreation of Bennett mechanical comprehension test problems [own example].

Word analogies in Scholastic Assessment Test (SAT-A)

The Scholastic Assessment Test (SAT reasoning test) is a standardised test (or entrance exam) developed by the non-profit organisation "College Board", which is used in the USA as an admission prerequisite. The test is intended to assess literacy and writing skills that are needed for university. SAT consists of three major sections: *Critical Reading*, including sentence completions and questions about short passages; *Mathematics*, including multiple choice and grid-in about topics such as algebra and geometry (calculator use is allowed); and *Writing*, including multiple choice questions and a brief essay. It includes some tasks that are very similar to the problems we may find in intelligence tests. For instance, the SAT analogy questions (word analogies) include exercises of the form *teacher : chalk :: soldier : ?* or *sun : planet :: earth : ?*. The procedure to solve word analogy problems corresponds to the procedure to solve geometrical analogies or letter string analogies. However, to detect the relation between the first pair of words and to create the solution, conceptual knowledge and verbal knowledge is necessary. This problem has been used by [Tur11] (374 multiple-choice questions).

C
Coverage Graphs: Experimental

Consolidation without forgetting

In a first experiment we try to show what would happen without applying the forgetting mechanism and check whether the MML-based measures work successfully for knowledge acquisition. Table C.1 shows all the rules in W at step 500 where we can see that the consolidated set almost represent all the legal chess moves. Figure C.1 for their coverage relations.

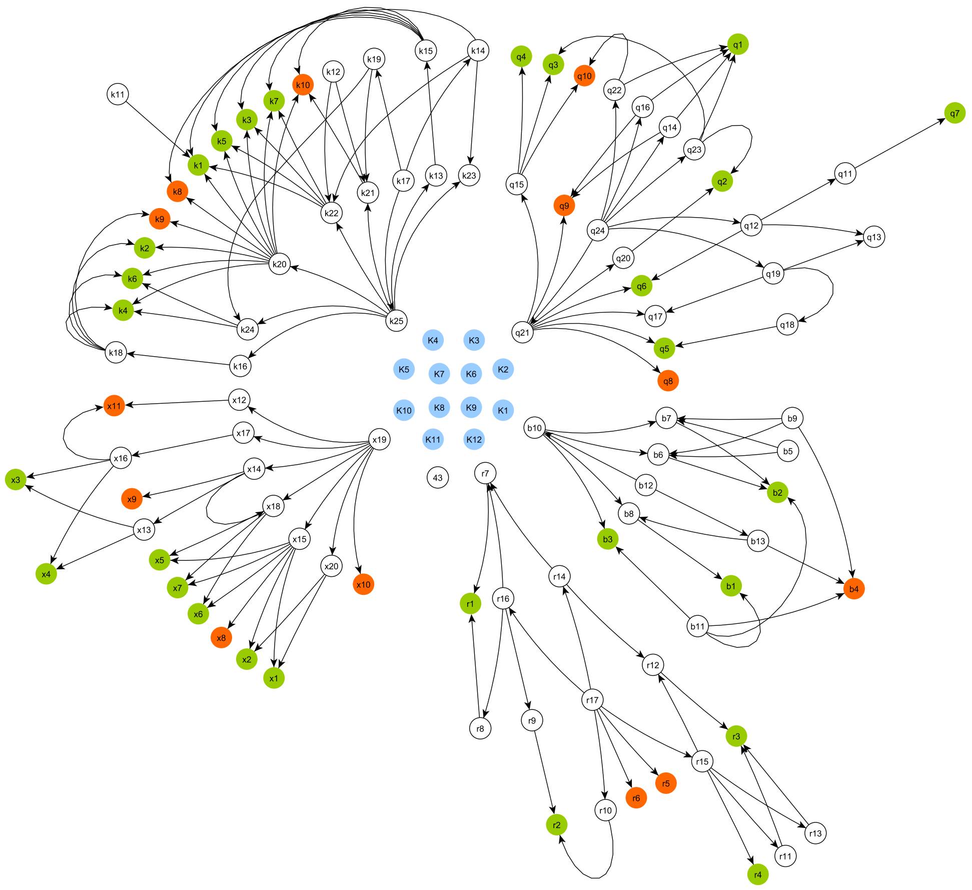


Figure C.1: Coverage graph *that represents the coverage relations between the individuals for the chess problem (without oblivion) at step* 400*. The metrics are shown in Table C.1. Green and red nodes refer to positive and negative examples respectively. Original background knowledge is represented as blue nodes.*

Table C.1: The complete set of rules and their metrics (same as in Table 8.7) for the chess problem without *the oblivion mechanism at step* 500. *Consolidated rules are placed in the table at the top while the rest of rules in W are placed in the table at the bottom.*

ID	ρ	$L(\rho)$	$\dot{S}_+$	$\dot{S}_-$	$-\dot{L}_+$	$-\dot{L}_-$	opt_+	opt_-	$Perm$
r15	move(rook,pos(A,B),pos(A,C)).	22.133	49.594	0	27.461	-22.133	2.746	-46.847	2.746
q19	move(queen,pos(A,B),pos(C,B))	13.214	27.052	0	13.838	-13.214	1.383	-25.668	1.383
q12	move(queen,pos(A,C),pos(A,D))	13.214	27.052	0	13.838	-13.214	1.383	-25.668	1.383
r16	move(rook,pos(A,B),pos(C,B)).	20.455	33.815	0	13.36	-20.455	1.336	-32.479	1.336
x18	move(king,pos(A,B),pos(C,D)) :- rdiff(B,D,1), fdiff(A,C,1).	34.275	40.578	0	6.303	-34.275	0.63	-39.947	0.63
x20	move(king,pos(A,B),pos(A,C)) :- rdiff(B,D,1).	22.918	27.052	0	4.134	-22.918	0.413	-26.638	0.413
x13	move(king,pos(A,B),pos(C,B)) :- fdiff(A,C,1).	22.918	27.052	0	4.134	-22.918	0.413	-26.638	0.413
q23	move(queen,pos(A,B),pos(C,D)) :- rdiff(B,D,E),fdiff(A,C,E).	28.993	32.462	0	3.469	-28.993	0.346	-32.115	0.346
b10	move(bishop,pos(A,B),pos(C,D)) :- rdiff(B,D,E), fdiff(A,C,E).	24.534	26.3	0	1.766	-24.534	0.176	-26.123	0.176
x6	move(king,pos(f,6),pos(g,7)).	27.052	27.052	0	0	-27.052	0	-27.052	-0.63
x4	move(king,pos(d,3),pos(e,3)).	27.052	27.052	0	0	-27.052	0	-27.052	-0.413
b3	move(bishop,pos(e,6),pos(a,2)).	27.052	27.052	0	0	-27.052	0	-27.052	-0.176
q6	move(queen,pos(d,2),pos(d,5)).	27.052	27.052	0	0	-27.052	0	-27.052	-1.383
k7	move(knight,pos(f,1),pos(g,3)).	27.052	27.052	0	0	-27.052	0	-27.052	0
r3	move(rook,pos(d,2),pos(d,5))	27.052	27.052	0	0	-27.052	0	-27.052	-2.746
k4	move(knight,pos(c,4),pos(a,5)).	27.052	27.052	0	0	-27.052	0	-27.052	0
x9	:- move(king,pos(b,2),pos(c,4)).	27.052	0	27.052	-27.052	0	-27.052	0	0
q4	move(queen,pos(b,2),pos(e,2)).	27.052	27.052	0	0	-27.052	0	-27.052	-1.383
b4	:- move(bishop,pos(f,1),pos(b,3)).	27.052	0	27.052	-27.052	0	-27.052	0	0
x2	move(king,pos(d,5),pos(d,6)).	27.052	27.052	0	0	-27.052	0	-27.052	-0.413
q2	move(queen,pos(c,4),pos(e,6)).	27.052	27.052	0	0	-27.052	0	-27.052	-0.346
k2	move(knight,pos(b,2),pos(d,3)).	27.052	27.052	0	0	-27.052	0	-27.052	0
x3	move(king,pos(b,5),pos(c,5)).	27.052	27.052	0	0	-27.052	0	-27.052	-0.413
k3	move(knight,pos(c,4),pos(b,6)).	27.052	27.052	0	0	-27.052	0	-27.052	0
r4	move(rook,pos(a,4),pos(a,5))	27.052	27.052	0	0	-27.052	0	-27.052	-2.746
k6	move(knight,pos(f,1),pos(h,2)).	27.052	27.052	0	0	-27.052	0	-27.052	0
x11	:- move(king,pos(b,2),pos(d,2)).	27.052	0	27.052	-27	0	-27.052	0	0
x10	:- move(king,pos(b,2),pos(b,4)).	27.052	0	27.052	-27.052	0	-27.052	0	0
q9	:- move(queen,pos(f,1),pos(b,4)).	27.052	0	27.052	-27.052	0	-27.052	0	0
x7	move(king,pos(b,2),pos(c,3)).	27.052	27.052	0	0	-27.052	0	-27.052	-0.63
k1	move(knight,pos(b,2),pos(c,4)).	27.052	27.052	0	0	-27	0	-27	0
q5	move(queen,pos(a,5),pos(h,5)).	27.052	27.052	0	0	-27.052	0	-27.052	-1.383
k9	:- move(knight,pos(f,1),pos(c,2)).	27.052	0	27.052	-27.052	0	-27.052	0	0
r6	:- move(rook,pos(h,4),pos(a,2))	27.052	0	27.052	-27.052	0	-27.052	0	0
q3	move(queen,pos(e,6),pos(a,2)).	27.052	27.052	0	0	-27.052	0	-27.052	-0.346
q8	:- move(queen,pos(f,1),pos(b,3)).	27.052	0	27	-27	0	-27.052	0	0
r1	move(rook,pos(b,2),pos(e,2)).	27.052	27.052	0	0	-27.052	0	-27.052	-1.336
x1	move(king,pos(a,4),pos(a,5)).	27.052	27.052	0	0	-27.052	0	-27.052	-0.413
k5	move(knight,pos(f,1),pos(g,3)).	27.052	27.052	0	0	-27.052	0	-27.052	0
r2	move(rook,pos(a,5),pos(h,5))	27.052	27.052	0	0	-27.052	0	-27.052	-1.336

Continued on next page

Table C.1 – *Continued from previous page*

ID	ρ	$L(\rho)$	$\dot{S}_+$	$\dot{S}_-$	$-\dot{L}_+$	$-\dot{L}_-$	opt_+	opt_-	$Perm$
q7	move(queen,pos(a,4),pos(a,5)).	27.052	27.052	0	0	-27.052	0	-27.052	-1.383
k10	:- move(knight,pos(f,1),pos(h,3)).	27.052	0	27.052	-27.052	0	-27.052	0	0
q10	:- move(queen,pos(e,1),pos(b,8)).	27.052	0	27.052	-27.052	0	-27.052	0	0
q1	move(queen,pos(f,1),pos(c,4)).	27.052	27.052	0	0	-27.052	0	-27.052	-0.346
k8	:- move(knight,pos(f,1),pos(g,4)).	27.052	0	27.052	-27.052	0	-27.052	0	0
x8	:- move(king,pos(b,2),pos(d,3)).	27.052	0	27.052	-27.052	0	-27.052	0	0
x5	move(king,pos(d,4),pos(e,3)).	27.052	27.052	0	0	-27.052	0	-27.052	-0.63
r5	:- move(rook,pos(f,1),pos(b,3))	27.052	0	27.052	-27.052	0	-27.052	0	0
b1	move(bishop,pos(f,1),pos(c,4)).	27.052	27.052	0	0	-27.052	0	-27.052	-0.176
b2	move(bishop,pos(c,4),pos(e,6)).	27.052	27.052	0	0	-27	0	-27.052	-0.176
k22	move(knight,pos(A,B),pos(C,D)) :- rdiff(B,D,2), fdiff(A,C,1).	34.275	33.815	0	-0.46	-34	-0.046	-33.861	-0.046
r8	move(rook,pos(b,B),pos(e,B)).	19.133	13.526	0	-5.607	-19.133	-0.56	-14.086	-1.896
r9	move(rook,pos(a,B),pos(h,B)).	19.133	13.526	0	-5.607	-19.133	-0.56	-14.086	-1.896
q11	move(queen,pos(A,4),pos(A,5))	19.133	13.526	0	-5.607	-19.133	-0.56	-14.086	-1.943
r7	move(rook,pos(A,2),pos(B,2)).	19.718	13.526	0	-6.192	-19.718	-0.619	-14.145	-1.955
k24	move(knight,pos(A,B),pos(C,D)) :- rdiff(B,D,1), fdiff(A,C,2).	34.275	27.052	0	-7.223	-34.275	-0.722	-27.774	-0.722
x12	move(king,pos(A,B),pos(C,B)) :- fdiff(A,C,1).	22.918	0	13.526	-22.918	-9.392	-14.465	-0.939	-0.939
r10	move(rook,pos(a,A),pos(h,5)).	23.092	13.526	0	-9.566	-23.092	-0.956	-14.482	-0.956
r13	move(rook,pos(d,B),pos(d,5)).	18.633	9.017	0	-9.616	-18.633	-0.961	-9.978	-3.707
r12	move(rook,pos(d,2),pos(d,C)).	18.633	9.017	0	-9.616	-18.633	-0.961	-9.978	-3.707
r14	move(rook,pos(A,2),pos(C,D)).	21.133	11.271	0	-9.862	-21.133	-0.986	-12.257	-0.986
q20	move(queen,pos(A,B),pos(C,D)) :- fdiff(A,C,2).	23.884	13.526	0	-10.358	-23.884	-1.035	-14.561	-1.035
r11	move(rook,pos(d,B),pos(d,C)).	19.718	9.017	0	-10.701	-19.718	-1.07	-10.087	-3.816
q13	move(queen,pos(A,B),pos(A,B))	12.384	0	0	-12.384	-12.384	-1.238	-1.238	-1.238
k11	move(knight,pos(A,2),pos(C,4)).	19.718	6.763	0	-12.955	-19.718	-1.295	-8.058	-1.295
b5	move(bishop,pos(A,B),pos(C,D)) :- fdiff(A,C,2).	23.884	6.011	0	-17.873	-23.884	-1.787	-7.798	-1.787
q18	move(queen,pos(A,B),pos(C,B)) :- rdiff(B,B,0), fdiff(A,C,5)	32.987	13.526	0	-19.461	-32.987	-1.946	-15.472	-3.328
b8	move(bishop,pos(A,B),pos(C,D)) :- rdiff(B,D,3), fdiff(A,C,3).	34.275	13.526	0	-20.749	-34.275	-2.074	-15.6	-2.25
k21	move(knight,pos(A,B),pos(C,D)) :- rdiff(B,D,2), fdiff(A,C,2).	29.816	0	9.017	-29.816	-20.799	-11.096	-2.079	-2.079
q17	move(queen,pos(A,B),pos(C,B)) :- fdiff(A,C,1)	22.918	0	0	-22.918	-22.918	-2.291	-2	-2.291
b7	move(bishop,pos(A,B),pos(C,D)) :- rdiff(B,D,2), fdiff(A,C,2).	34.275	9.017	0	-25.258	-34.275	-2.525	-11.542	-2.701
b6	move(bishop,pos(A,B),pos(C,D)) :- rdiff(B,D,2), fdiff(A,C,2).	34.275	9.017	0	-25.258	-34.275	-2.525	-11.542	-2.701
k23	move(knight,pos(A,B),pos(C,D)) :- rdiff(B,D,1), fdiff(A,C,1).	34.275	0	0	-34.275	-34.275	-3.427	-3.427	-3.427
k19	move(knight,pos(A,B),pos(C,D)) :- fdiff(A,C,2).	23.884	13.526	3.005	-10.358	-20.879	-3.74	-14.261	-3.74
q14	move(queen,pos(A,1),pos(C,4))	15.258	5.41	9.017	-9.848	-6.241	-9.1	-5.493	-5.493
q16	move(queen,pos(A,1),pos(C,4))	19.718	5.41	9.017	-14.308	-10.701	-9.546	-5.939	-5.939
k12	move(knight,pos(A,B),pos(C,D)) :- rdiff(B,D,2).	23.884	28.179	7.514	4.295	-16.37	-6.333	-26.998	-6.333
q22	move(queen,pos(A,B),pos(C,D)) :- rdiff(B,D,E),fdiff(A,C,3).	32.16	5.41	13.526	-26.75	-18.634	-14.848	-6.732	-6.732
b9	move(bishop,pos(A,B),pos(C,D)) :- rdiff(B,D,2)	23.884	6.011	9.017	-17.873	-14.867	-9.902	-6.896	-6.896
b12	move(bishop,pos(A,B),pos(C,D)) :- rdiff(B,D,E), fdiff(A,C,F).	30.105	33.063	9.017	2.958	-21	-7.819	-31.865	-7.819
k15	move(knight,pos(A,B),pos(C,D)) :- rdiff(B,D,2), fdiff(A,A,0), fdiff(C,C,0).	34.275	33.815	9.017	-0.46	-25.258	-8.161	-32.959	-8.161
k13	move(knight,pos(A,B),pos(C,D)) :- rdiff(B,D,2), fdiff(C,C,0).	34.275	33.815	9.017	-0.46	-25.258	-8.161	-33	-8.161
b13	move(bishop,pos(A,B),pos(C,D)) :- rdiff(B,D,E), fdiff(A,C,3).	32.16	6.763	9.017	-25.397	-23.143	-10.655	-8.401	-8.401
b11	move(bishop,pos(A,B),pos(C,D)) :- rdiff(B,B,0), rdiff(D,D,0), fdiff(A,A,0).	43.635	36.069	9.017	-7.566	-34.618	-8.871	-35.923	-8.871
k14	move(knight,pos(A,B),pos(C,D)) :- fdiff(A,C,1).	23.884	11.271	13.526	-12.613	-10.358	-13.434	-11.179	-11.179

Continued on next page

Table C.1 – *Continued from previous page*

ID	ρ	$L(\rho)$	$\dot{S}_+$	$\dot{S}_-$	$-\dot{L}_+$	$-\dot{L}_-$	opt_+	opt_-	$Perm$
q15	move(queen,pos(A,B),pos(C,D)) :- fdiff(A,C,3)	23.884	32.462	13.526	8.578	-10.358	-11.315	-30.251	-11.315
k16	move(knight,pos(A,B),pos(C,D)) :- rdiff(B,D,1), fdiff(A,A,0).]	29.816	27.052	13.526	-2.764	-16.29	-12.449	-25.975	-12.449
x16	move(king,pos(A,B),pos(C,D)) :- rdiff(B,D,0), fdiff(B,B,0).	34.275	27.052	13.526	-7.223	-20.749	-12.895	-26.421	-12.895
x17	move(king,pos(A,B),pos(C,D)) :- rdiff(B,D,0), fdiff(A,C,0).	34.275	27.052	13.526	-7.223	-20.749	-12.895	-26.421	-12.895
k18	move(knight,pos(A,B),pos(C,D)) :- rdiff(B,B,0), rdiff(B,D,1), fdiff(A,A,0).	43.378	27.052	13.526	-16.326	-29.852	-13.806	-27.332	-13.806
x14	move(king,pos(A,B),pos(C,D)) :- fdiff(A,C,1).	23.884	67.63	27.052	43.746	3	-19.972	-61	-19.972
x15	move(king,pos(A,B),pos(C,D)) :- rdiff(B,B,0), rdiff(B,D,1), rdiff(D,D,0).	43.378	67.63	27.052	24.252	-16.326	-21.921	-62.499	-21.921
k20	move(knight,pos(A,B),pos(C,D)) :- rdiff(B,B,0), fdiff(A,A,0).	23.884	60.867	36	36.983	12.185	-28.763	-53.561	-28.763
q21	move(queen,pos(A,B),pos(C,D)) :- rdiff(B,D,E).	21.506	86.566	49.595	65.06	28.089	-38.129	-75.1	-38.129
r17	move(rook,pos(A,B),pos(C,D)).	22.777	108.205	54.104	85.428	31.327	-40.15	-94.251	-40.15
k25	move(knight,pos(A,B),pos(C,D)) :- rdiff(B,D,E), fdiff(A,C,F).	30.105	129.624	57.109	99.519	27.004	-41.446	-113.961	-41.446
k17	move(knight,pos(A,B),pos(C,D)) :- fdiff(A,C,E).	17.047	154.421	73.64	137.374	56.593	-52.538	-133.319	-52.538
q24	move(queen,pos(A,B),pos(C,D))	13.858	189.362	81.154	175.504	67.296	-55.488	-163.696	-55.488
x19	move(king,pos(A,B),pos(C,D))	22.918	189.363	108.208	166.445	85.29	-80.742	-161.897	-80.742

Consolidation with forgetting

After that, we repeat the same experiment, but using the forgetting mechanism. Table C.2 shows all the rules in W at step 500 where we can see that the consolidated set perfectly represent all the legal chess moves

Table C.2: *Rules and metrics (same as in Table 8.7) for the chess problem* with *the oblivion mechanisms at step* 500. *Consolidated rules are placed in the table at the top while the rest of rules in W are placed in the table at the bottom.*

ID	ρ	$L(\rho)$	$\dot{S}_+$	$\dot{S}_-$	$-\dot{L}_+$	$-\dot{L}_-$	opt_+	opt_-	$Perm$
x18	move(king,pos(A,B),pos(C,D)) :- rdiff(B,D,1), fdiff(A,C,1).	34.275	662.774	0	628.499	-34.275	62.849	-599.924	-22.681
x13	move(king,pos(A,B),pos(C,B)) :- fdiff(A,C,1).	22.918	459.884	0	436.966	-22.918	43.696	-416.187	-41.834
k22	move(knight,pos(A,B),pos(C,D)) :- rdiff(B,D,2), fdiff(A,C,1).	34.275	446.358	0	412.083	-34.275	41.208	-405.149	29.394
q23	move(queen,pos(A,B),pos(C,D)) :- rdiff(B,D,E),fdiff(A,C,E).	28.993	437.34	0	408.347	-28.993	40.834	-396.505	-1.513
x20	move(king,pos(A,B),pos(A,C)) :- rdiff(B,D,1).	22.918	392.254	0	369.336	-22.918	36.933	-355.32	8.116
r15	move(rook,pos(A,B),pos(A,C)).	22.133	389.999	0	367.866	-22.133	36.786	-353.212	6.602
q19	move(queen,pos(A,B),pos(C,B)).	13.214	365.202	0	351.988	-13.214	35.198	-330.003	-7.149
k24	move(knight,pos(A,B),pos(C,D)) :- rdiff(B,D,1), fdiff(A,C,2).	34.275	369.71	0	335.435	-34.275	33.543	-336.166	25.561
q12	move(queen,pos(A,C),pos(A,D)).	13.214	311.098	0	297.884	-13.214	29.788	-281.309	-12.559
r16	move(rook,pos(A,B),pos(C,B)).	20.455	284.046	0	263.591	-20.455	26.359	-257.686	-3.824
b10	move(bishop,pos(A,B),pos(C,D)) :- rdiff(B,D,E), fdiff(A,C,E).	24.534	275.028	0	250.494	-24.534	25.049	-249.978	12.731
x14	move(king,pos(A,B),pos(C,D)) :- fdiff(A,C,1).	23.884	1122.658	27.052	1.099	3.168	85.53	-1010.075	56.713
q24	move(queen,pos(A,B),pos(C,D)).	13.858	1167.744	81.156	1.154	67.298	42.348	-1044.239	42.348
r17	move(rook,pos(A,B),pos(C,D)).	22.777	811.559	54.104	788.782	31.327	30.184	-727.27	30.184
x19	move(king,pos(A,B),pos(C,D)).	22.918	1528.437	135.26	1505.519	112.342	28.817	-1364.359	28.817
b12	move(bishop,pos(A,B),pos(C,D)) :- rdiff(B,D,E), fdiff(A,C,F).	30.105	275.028	13.526	244.923	-16.579	12.318	-249.183	12.318
k12	move(knight,pos(A,B),pos(C,D)) :- rdiff(B,D,2).	23.884	223.179	9.017	199.295	-14.867	11.814	-202.347	11.814
k19	move(knight,pos(A,B),pos(C,D)) :- fdiff(A,C,2).	23.884	184.855	9.017	160.971	-14.867	7.981	-167.856	7.981
b5	move(bishop,pos(A,B),pos(C,D)) :- fdiff(A,C,2).	23.884	78.901	0	55.017	-23.884	5.501	-73.399	5.501
x12	move(king,pos(A,B),pos(C,B)) :- fdiff(A,C,1).	22.918	0	54.104	-22.918	31.186	-50.985	3.118	3.118
r13	move(rook,pos(d,B),pos(d,5)).	18.633	27.052	0	8.419	-18.633	0.841	-26.21	-26.21
k1	move(knight,pos(b,2),pos(c,4)).	27.052	27.052	0	0	-27.052	0	-27.052	-27.052
k7	move(knight,pos(f,1),pos(g,3)).	27.052	27.052	0	0	-27.052	0	-27.052	-27.052
k5	move(knight,pos(f,1),pos(g,3)).	27.052	27.052	0	0	-27.052	0	-27.052	-27.052
x3	move(king,pos(b,5),pos(c,5)).	27.052	27.052	0	0	-27.052	0	-27.052	-27.052
q4	move(queen,pos(b,2),pos(e,2)).	27.052	27.052	0	0	-27.052	0	-27.052	-27.052
r3	move(rook,pos(d,2),pos(d,5)).	27.052	27.052	0	0	-27.052	0	-27.052	-27.052
x6	move(king,pos(f,6),pos(g,7)).	27.052	27.052	0	0	-27.052	0	-27.052	-27.052
r2	move(rook,pos(a,5),pos(h,5)).	27.052	27.052	0	0	-27.052	0	-27.052	-27.052
r4	move(rook,pos(a,4),pos(a,5)).	27.052	27.052	0	0	-27.052	0	-27.052	-27.052
b2	move(bishop,pos(c,4),pos(e,6)).	27.052	27.052	0	0	-27.052	0	-27.052	-25.049
x9	:- move(king,pos(b,2),pos(c,4)).	27.052	0	27.052	-27.052	0	-27.052	0	0
b4	:- move(bishop,pos(f,1),pos(b,3)).	27.052	0	27.052	-27	0	-27	0	0
x10	:- move(king,pos(b,2),pos(b,4)).	27.052	0	27.052	-27.052	0	-27	0	0
k9	:- move(knight,pos(f,1),pos(c,2)).	27.052	0	27.052	-27.052	0	-27.052	0	0
r6	:- move(rook,pos(h,4),pos(a,2).	27.052	0	27.052	-27	0	-27.052	0	0
q8	:- move(queen,pos(f,1),pos(b,3)).	27.052	0	27.052	-27	0	-27.052	0	0
k10	:- move(knight,pos(f,1),pos(h,3)).	27.052	0	27.052	-27.052	0	-27.052	0	0
q10	:- move(queen,pos(e,1),pos(b,8)).	27.052	0	27.052	-27	0	-27.052	0	0
k8	:- move(knight,pos(f,1),pos(g,4)).	27.052	0	27.052	-27.052	0	-27.052	0	0
x8	:- move(king,pos(b,2),pos(d,3)).	27.052	0	27.052	-27.052	0	-27.052	0	0
r5	:- move(rook,pos(f,1),pos(b,3).	27.052	0	27.052	-27.052	0	-27.052	0	0

Continued on next page

Table C.2 – *Continued from previous page*

ID	ρ	$L(\rho)$	$\dot{S}_+$	$\dot{S}_-$	$-\dot{L}_+$	$-\dot{L}_-$	opt_+	opt_-	$Perm$
q9	:- move(queen,pos(f,1),pos(b,4)).	27.052	0	27.052	-27.052	0	-27.052	0	0
q13	move(queen,pos(A,B),pos(A,B)).	12.384	0	0	-12.384	-12.384	-1.238	-1.238	-1.238
r8	move(rook,pos(b,B),pos(e,B)).	19.133	0	0	-19.133	-19.133	-1.913	-1.913	-1.913
q21	move(queen,pos(A,B),pos(C,D)) :- rdiff(B,D,E).	21.506	13.526	81.156	-7.98	59.65	-73.838	-6.208	-6.208
b11	move(bishop,pos(A,B),pos(C,D)) :- rdiff(B,B,0), rdiff(D,D,0), fdiff(A,A,0).	43.635	9.017	13.526	-34.618	-30.109	-15.635	-11.126	-11.126
x15	move(king,pos(A,B),pos(C,D)) :- rdiff(B,B,0), rdiff(B,D,1), rdiff(D,D,0).	43.378	13.526	27.052	-30	-16.326	-27.332	-13.806	-13.806
k25	move(knight,pos(A,B),pos(C,D)) :- rdiff(B,D,E), fdiff(A,C,F).	30.105	448.611	63.121	418.506	33.016	-14.958	-400.448	-14.958
k20	move(knight,pos(A,B),pos(C,D)) :- rdiff(B,B,0), fdiff(A,A,0).	23.884	40.578	63.121	16.694	39.237	-55.139	-32.596	-32.595

Integration with gErl

It is also of interest the integration of out assessment setting with other inductive and deductive mechanisms in order to show its generality. Particularly, the application of the principles used (MML evaluations and cognitive mechanisms) in our declarative learning paradigm gErl seems to be an appealing idea since from the very beginning of it development we were looking on a proper foundation for detailed knowledge assessment metrics and criteria for forgetting.

We will focus on the problem of solving a single letter series completion problems, specifically, the problem number 11 in Table 7.6 ($rscdstdetuef\_$) following the problem definition stated in Section 7.4.3. The challenge we would like to face is the same as in the experiments performed in Section 8.5: knowledge discovery and acquisition in a progressive way from examples provided incrementally. A random set of examples and rules is given considering gErl as the inductive engine (which is generating rules during the whole process) and arriving to the system in a random order as well. gErl is also in charge of deduction, namely, to provide the coverage mechanism with the goal of establishing relations between individuals. We use gErl in the same way we used Prolog in the previous experiments: we take those rules generated by the learning system offline (around 120 rules including examples), we do not use our assessment setting to guide the learning which remains as future work.

Like the previous experiments, we have set the consolidation criterion with a threshold of optimality greater than the average of the optimality value of the rules in W, as in equation 8.17, and the β parameter equal to 0.1 in equation 8.12 with the aim of penalising those rules that are not pure. Furthermore, since this experiment has an illustrative aim, we will just show the performance of one configuration of forgetting percentage (50%) and maximum number of rules (50). Launching this experiment with whatever other configuration will have the same results in terms of consolidated rules.

Figure C.4 shows the evolution of the system during 500 steps. The variations in the amount of consolidated rules (dotted black line) and rules in the working space (dashed brown line) allow us to observe how the forgetting mechanism works (every 30 steps approximately). Table 8.9 presents the consolidated rules at the final step (500), the rule $q33$ being the solution of the letter series problem. If we could have a look at the consolidated set of rules at each step, we will see that, from the very begging, it is only formed by those rules covering the positive evidence and, it remains almost constant from step 220 (dotted black line) showing that the system has reached a stable situation. Like in the previous experiments using the forgetting mechanism, the average optimality of both the consolidated rules (dashed green line) and all the rules (dashed blue line) have an increasing trend due to the distribution with replacement used to populate the working space, while the appearance of new rules in the system or the execution of the forgetting mechanism mainly affect the average optimality of W (dashed blue line).

ID	***Rule***	$L(\rho)$	$\dot{S}_+$	$\dot{S}_-$	$-\dot{L}_+$	$-\dot{L}_-$	Opt_+	Opt_-	***Perm***
q33	letter(List) -> next(last(init(init(init(List)))))	14.9	1186.335	0	1171.404	-14.931	117.14	-1069.194	117.14
q52	letter(List) -> previous(next(last(init(init(List)))))	16.5	706.988	0	690.479	-16.509	69.047	-637.94	69.047
q65	letter(List) -> next(next(previous(last(List))))	12.6	682.046	0	669.437	-12.609	66.943	-615.102	66.943
q17	letter(List) -> next(last(List))	7.0	501.953	0	494.953	-7	49.495	-452.457	49.495
q19	letter(List) -> last(init(init(List)))	9.0	388.68	0	379.68	-9	37.968	-350.712	37.968
q57	letter(List) -> next(previous(last(init(init(List)))))	16.5	388.68	0	372.171	-16.509	37.217	-351.462	37.217
q114	letter(List) -> next(last(init(init(init(init(init(init(List))))))))	21.9	263.407	0	241.51	-21.897	24.151	-239.255	24.151

Table C.3: Consolidated rules and metrics for the letter series problem 13 with *forgetting at step* 500.

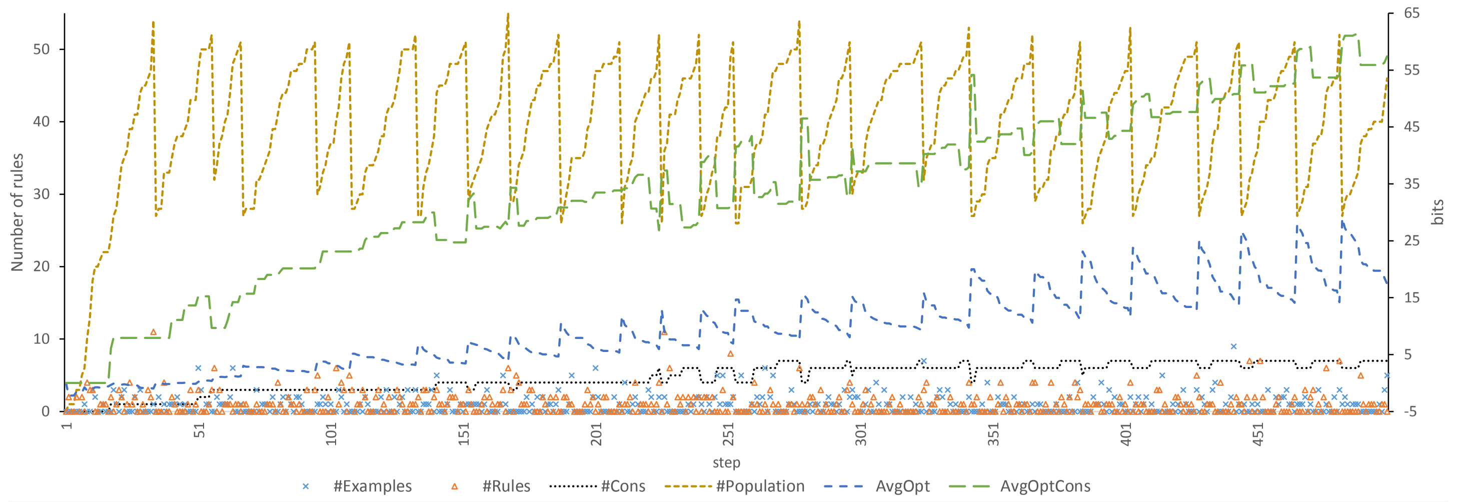


Table C.4: Evolution of some indicators for the letter series problem (13) with *forgetting: #Examples and #Rules show the examples that arrive and the rules that are generated by the inductive engine for each step, #Cons shows how many rules there are in the consolidated knowledge (initially the background knowledge) and #Population shows the total number of rules (magnitudes shown on the left y-axis). AvgOpt and AvgOptCons show, respectively, the average optimality for all rules and the average optimality for all consolidated rules (magnitudes shown on the right y-axis). The configuration establishes a maximum number of rules equal to* 50 *and up to 50% of rules forgotten for each forgetting step). We see a bumpier picture, where the forgetting mechanism takes place every 30 steps approximately.*