Imperial College London
Department of Computing

Logic Programs as Declarative and Procedural Bias in Inductive Logic Programming

Dianhuan Lin

November 2013

Submitted in part fulfilment of the requirements for the degree of
Doctor of Philosophy in Computing of Imperial College London
and the Diploma of Imperial College London

Statement of originality

I declare that all work presented in this dissertation is my own work, otherwise properly acknowledged.

Copyright

Abstract

Machine Learning is necessary for the development of Artificial Intelligence, as pointed out by Turing in his 1950 article "Computing Machinery and Intelligence". It is in the same article that Turing suggested the use of computational logic and background knowledge for learning. This thesis follows a logic-based machine learning approach called Inductive Logic Programming (ILP), which is advantageous over other machine learning approaches in terms of relational learning and utilising background knowledge. ILP uses logic programs as a uniform representation for hypothesis, background knowledge and examples, but its declarative bias is usually encoded using metalogical statements. This thesis advocates the use of logic programs to represent declarative and procedural bias, which results in a framework of single-language representation. We show in this thesis that using a logic program called the top theory as declarative bias leads to a sound and complete multi-clause learning system MC-TopLog. It overcomes the entailment-incompleteness of Progol, thus outperforms Progol in terms of predictive accuracies on learning grammars and strategies for playing Nim game. MC-TopLog has been applied to two real-world applications funded by Syngenta, which is an agriculture company. A higher-order extension on top theories results in meta-interpreters, which allow the introduction of new predicate symbols. Thus the resulting ILP system Metagol can do predicate invention, which is an intrinsically higher-order logic operation. Metagol also leverages the procedural semantic of Prolog to encode procedural bias, so that it can outperform both its ASP version and ILP systems without an equivalent procedural bias in terms of efficiency and accuracy. This is demonstrated by the experiments on learning Regular, Context-free and Natural grammars. Metagol is also applied to non-grammar learning tasks involving recursion and predicate invention, such as learning a definition of staircases and robot strategy learning. Both MC-TopLog and Metagol are based on a $\top$-directed framework, which is different from other multi-clause learning

systems based on Inverse Entailment, such as CF-Induction, XHAIL and IMPARO. Compared to another $\top$-directed multi-clause learning system TAL, Metagol allows the explicit form of higher-order assumption to be encoded in the form of meta-rules.

Acknowledgements

The first person I would like to thank is my supervisor Stephen Muggleton. This thesis would be impossible without his guidance and support. I really appreciate the numerous long discussions I had with Stephen. I benefit a lot from those discussions, but I know they took a lot of Stephen's time, for which I'm very grateful. I also want to thank Stephen for sharing his vision and ideas. I feel really lucky to have such an amazing supervisor in the journey of tackling challenging problems.

I also want to thank my industrial supervisor Stuart John Dunbar, who is always very positive and encouraging to the progress I make. I also would like to thank all members of the Syngenta University Innovation Centre at Imperial College London, in particular, Pooja Jain, Jianzhong Chen, Hiroaki Watanabe, Michael Sternberg, Charles Baxter, Richard Currie, Domingo Salazar. I also want to thank Syngenta for generously providing full funding for my PhD.

I also want to thank other colleagues, with whom I had interesting discussions, seminars and reading groups. They are: Robert Henderson, Jose Santos, Alireza Tamaddoni-Nezhad, Niels Pahlavi and Graham Deane. In particular, I want to thank Graham Deane for the discussion about using the ASP solver CLASP. I also would like to thank Katsumi Inoue for the fantastic opportunity of a research visit to NII in Tokyo. I also want to thank Krysia Broda, who supervised my MSc group project. It is from this project that I gained the confidence and skill in writing complex theorem provers in Prolog, which is tremendously useful in my PhD for developing ILP systems.

I also want to thank my external and internal examiners: Ivan Bratko and Alessandra Russo for reading through my thesis and providing very helpful feedback. I also would like to thank Stuart Russell for pointing out some of the related work.

Finally, I would like to thank my family and friends. In particular, I want

to thank my parents for their unfailing support. This thesis is dedicated to them. I am also very lucky to have a twin sister Dianmin. Together we enjoy and explore different parts of the world. Last but not least, special thanks for Ke Liu who has been accompanying me through the memorable journey of my PhD.

Contents

List of Tables

List of Figures

1. Introduction

1.1. Overview

The ability to learn makes it possible for human beings to acquire declarative and procedural knowledge, which are vital for the intelligent behaviour required for survival. Similarly, machine learning is indispensable for machine intelligence, otherwise acquiring this knowledge by way of programming is unacceptably costly [Tur50], and according to John McCarthy in the Lighthill Controversy Debate [LMMG73] "as though we did education by brain surgery, which is inconvenient with children". Moreover, machine learning techniques allow us to overcome human limitations [Mug06] when dealing with large-scale data. Thus Machine Learning [Mit97] is a critically important subarea of Artificial Intelligence [RN10] and Computer Science more generally.

This thesis focuses on learning structures from examples. The approach being followed is called Inductive Logic Programming (ILP) [Mug91], which has particular advantages in structure learning due to its logical representation. Logic programs are used as a uniform representation in ILP for its hypothesis, training examples and background knowledge. The expressiveness of logic programs makes it possible for ILP to learn Turing complete languages [Tar77], including relational languages. The basis in first-order logic also makes ILP's hypotheses readable, which is important for hypothesis validation.

On the other hand, the expressiveness of logic programs comes at a cost of efficiency due to an extremely large hypothesis space. Therefore ILP systems like Progol [Mug95] and FOIL [Qui90] have restricted their hypothesis spaces to those that subsume examples relative to background knowledge in Plotkin's sense [Yam97]. This means they can only generalise a positive example to a single clause, rather than a theory with multiple clauses. Additionally, it is unclear how to do predicate invention with this single-

clause restriction, because a theory with predicate invention has at least two clauses: one clause with the invented predicate in its body, while another clause with the invented predicate in its head, that is, defining the invented predicate. However, target concepts with a multi-clause theory naturally occur in real-world applications [LCW+11]. For example, a theory for parsing natural language is composed of multiple dependent grammar rules. Similarly, predicate invention is also important in real-world applications, because "it is a most natural form of automated discovery" [MRP+11]. As commented by Russell and Norvig [RN10]: "Some of the deepest revolutions in science come from the invention of new predicates and functionsfor example, Galileos invention of acceleration, or Joules invention of thermal energy. Once these terms are available, the discovery of new laws becomes (relatively) easy." On the other hand, when learning a multi-clause theory, the challenge lies in dealing with an exponentially increased hypothesis space. Predicate invention leads to an even larger hypothesis space due to introducing new predicate symbols.

To address Progol's entailment-incompleteness, ILP systems like CF-Induction [Ino04a], HAIL [RBR03], IMPARO [KBR09], TAL [CRL10] and MC-TopLog ($\top$DTD) [Lin09] were developed. They are all based on Inverse Entailment (IE) [Mug95] except TAL and MC-TopLog ($\top$DTD). The multi-clause IE-based systems require procedural implementations for the search control. They also perform greedy search, which is sub-optimal. In addition, they do not support predicate invention. In contrast to the IE-based systems, $\top$-directed ILP systems allow declarative implementations and optimal search. The $\top$-directed framework was first introduced in the system TopLog [MSTN08], but TopLog is restricted to single-clause learning like Progol. Both TAL and MC-TopLog extended TopLog to support multi-clause learning, while TAL is based on abductive proof procedural, in particular SLDNFA. Therefore TAL supports non-monotonic reasoning, which is not considered in this thesis due to the restriction to definite clause programs.

Based on TopLog and MC-TopLog ($\top$DTD), this thesis further explores the advantages of the $\top$-directed framework, in particular, leveraging the expressive power of a logic program to explicitly encode various declarative and procedural bias. For example, a logic program called meta-interpreter, which is an high-order extension of a top theory $\top$, can encode higher-

order logic in the form of metarules. Therefore the system Metagol based on a meta-interpreter supports higher-order logic learning, which includes predicate invention considering predicate invention is an intrinsically higher-order logic operation. The rest of this session explains the advantages of the $\top$-directed framework in more details, especially those explored in this thesis. In order to tackle the problem of an extremely large hypothesis space, one solution is to make use of the declarative bias available to constrain the hypothesis space. Apart from declarative bias, we also consider how to include procedural bias as a mean to reduce complex searches. The question is how to provide a mean which can explicitly represent various declarative and procedural bias.

Declarative bias says what hypothesis language is allowed, thus determining the size of an hypothesis space. Now the problem is: not all the declarative bias available can be encoded by the chosen representation. For example, one commonly used representation of declarative bias in ILP is mode declarations. Mode declarations are metalogical statements, which can only say what predicates or what types of arguments are allowed in an hypothesis language. Thus they are not able to capture strong declarative bias like partial information on what predicates should be connected or can not be connected together. For example, a noun phrase must consist of a noun, although the whole definition of a none phrase is unknown. Therefore, we advocate in this thesis the use of a logic program to represent declarative bias. The expressiveness of logic programs makes it flexible to encode strong declarative bias. We explore two forms of declarative bias in this thesis: top theories [MSTN08, MLTN11] and meta-interpreters [MLPTN13, ML13]. Meta-interpreters extends the first-order top theories to the higher-order in the form of metarules. Therefore the resulting ILP system is capable of introducing new predicate symbols for predicate invention.

Apart from the advantage of being able to encode strong declarative bias, using logic programs as declarative bias also leads to a single-language representation for an ILP system. Logic programs are already used in ILP as a uniform representation for its hypothesis H, examples E and background knowledge B. This is known as the "single-language trick" [LF01]. Firstly, when H and E are each logic programs, examples can be treated as most-specific forms of hypothesis, allowing reasoning to be carried out within a single framework. This contrasts with other representations used in

machine learning, such as decision trees, neural networks and support vector machines, in which examples are represented as vectors and hypotheses take the form of trees, weights and hyperplanes respectively, and separate forms of reasoning must be employed with the examples and hypotheses respectively. Secondly, in the case B and H are each logic programs, every hypothesis H can be used directly to augment the background knowledge once it has been accepted. Once more, this contrasts with decision trees, neural networks and support vector machines, in which the lack of background knowledge impedes the development of an incremental framework for learning. This thesis advocates declarative bias D also having the form of a logic program. It has advantages which distinguish top theories and meta-interpreters from other representations for declarative bias like antecedent description language (ADL) [Coh94], which is also expressive enough to capture strong declarative bias. The rest of this section explains these advantages in more details.

Firstly, it leads to two new approaches: $\top$-Directed Theory Derivation ($\top$DTD) and $\top$-Directed Theory co-Derivation ($\top$DTcD). These two approaches are alternatives to Inverse Entailment for hypothesis derivation. Since within these approaches, a logic program called top theories representing declarative bias can be reasoned directly with examples and background knowledge, so that the derivable hypotheses are bound to those that cover at least one positive example, that is, hold for $B \wedge H \models e^+$. A top theory can also directly constrain the hypothesis space exclusively to common generalisations of at least two examples. Similarly, the single-language representation also leads to a framework of meta-interpretive learning, where meta-interpreters can also be reasoned directly with training examples and background knowledge.

Secondly, this makes it possible for declarative bias to be learned directly using the same learning system to which they are input. For example, it has been demonstrated in [Coh92] that explicit declarative bias can be compiled from background knowledge. [BT07] also showed that declarative bias can be learned using ILP, while the learned bias "reduces the size of the search space without removing the most accurate structures" [BT07]. But this is not the focus of this thesis and is to be explored in future work.

Thirdly, it also results in a declarative machine learning framework called Meta-Interpretive Learning. Declarative Machine Learning [Rae12] has the

advantage of separating problem specifications from problem solving. In this way, the learning task of searching for an hypothesis can be delegated to the search engine in Prolog [Bra86, SS86] or Answer Set Programming (ASP) [Gel08]. ASP solvers such as Clasp [GKNS07] use a logic programming framework with efficient constraint handling techniques so that it competes favourably in international competitions with SAT-solvers. Clasp also features effective optimisation techniques based on branch-and-bound algorithms. It has recently been extended to UnClasp [AKMS12] with optimisation components based upon the computation of minimal unsatisfiable cores. This development of ASP solver from Clasp to UnClasp is made use of in this thesis.

Although declarative modelling makes it possible for users to focus on problem specification and not to worry about how to solve the problem step by step, it does not mean that users should forget about the control over search. As formulated in [Kow79], Algorithm = Logic + Control. During search, if it is known that the target hypothesis or its approximations are more likely to be in certain area of the hypothesis space, then that area should be searched first. Here is an analogy to finding a needle in a very large room: imagine you drop a needle in the middle of a very large room, you will start to search from your surrounding area, rather than searching systematically in an orderly fashion from one end of the room to the other end. In this thesis, we refer to the order in which search is conducted as procedural bias. To the author's knowledge, the term 'procedural bias' does not appear in existing literature, although it corresponds to the preference bias described in [Mit97]. Here we choose to use the term 'procedural bias' in order to contrast with 'declarative bias'. Existing ILP systems have procedural bias like top-down or bottom-up search implicitly built-in. So far there is no explicit mechanism for an ILP system to incorporate domain-specific procedural bias. To change the procedural bias in an existing ILP system would require changing the internal code of the system, which is inconvenient from both users' and developers' point of view.

In this thesis, we explore the possibility of encoding procedural bias using logic programs. Prolog is a fully-fledged programming language and it has both declarative and procedural semantics. Its procedural semantic is affected by the order of clauses and their body literals. Therefore the procedural bias of a learning system can be encoded in a Prolog program

via the order of clauses and their body literals. In contrast to Prolog, ASP has a totally different computational mechanism. Its procedure semantic is totally separated from its declarative semantics and not assigned to a logic program. Therefore, solutions returned by ASP are invariant to the order of clauses in a logic program and their body literals. Metagol, a Prolog implementation of Meta-Interpretive learning, is compared to a corresponding ASP version, as well as an ILP system without an equivalent procedural bias. The experimental results show that the Prolog version can outperform the others in terms of both efficiency and predictive accuracy.

1.2. Contributions

This thesis shows: using a logic program called the top theory as declarative bias leads to a new ILP system MC-TopLog, which can do multi-clause learning, outperforming state-of-art ILP systems like Progol in terms of predictive accuracies on several challenging datasets. A higher-order extension on top theories results in meta-interpreters, which allows the introduction of new predicate symbols. Thus the resulting ILP system Metagol can do predicate invention. Metagol also utilises the procedural semantic of logic programs to encode procedural bias, so that it can outperform both its ASP version and ILP systems without an equivalent procedural bias in terms of both efficiency and accuracy.

More details of the contributions are as follows:

- Development of MC-TopLog [MLTN11], which is a complete, declarative and efficient ILP system. MC-TopLog features multi-clause learning.
  - Proving the correctness of the two algorithms in MC-TopLog: $\top$DTD and $\top$DTcD
  - Implementing MC-TopLog and conducting all experiments
- Application of MC-TopLog to identifying metabolic control reactions in the two Syngenta projects: tomato project and predictive toxicology project. These two real-world applications demonstrate the advantages of multi-clause learning.

- Formulation of the framework of Meta-Interpretive Learning (MIL) in Answer Set Programming. This further supports MIL's characteristic as declarative modelling.
  - Implementation of ASP_{M}
  - All experiments involving Metagol and ASP_{M}, as well as their comparisons to other ILP systems without an equivalent procedural bias. Given exactly the same declarative bias while different procedural biases, Metagol has significantly shorter running time than the others, which shows the efficiency advantage of Metagol's procedural bias. Therefore this demonstrates the advantage of explicitly incorporating procedural bias.

1.3. Publications

The work in this thesis has been reviewed and published in the following venues.

- [MLTN11] Stephen Muggleton, Dianhuan Lin and Alireza Tamaddoni-Nezhad. MC-Toplog: Complete multi-clause learning guided by a top theory. In *Proceedings of the 21st International Conference on Inductive Logic Programming (ILP2011)*, LNAI 7207, pages 238-254, 2012. This paper is the basis for Chapter 3. In this paper, the initial idea and the theoretical framework are due to Stephen Muggleton. The author of this thesis contributed to (1) the theoretical framework, in particular, proving the soundness and completeness of the two algorithms in MC-TopLog: $\top$DTD and $\top$DTcD; (2) the implementation of MC-TopLog and its experiments; (3) writing the paper. The third author of this paper Alireza Tamaddoni-Nezhad contributed to conducting the Progol experiments of grammar learning, to which MC-TopLog is compared.

- [LCW+11] Dianhuan Lin, Jianzhong Chen, Hiroaki Watanabe, Stephen H. Muggleton, Pooja Jain, Michael J.E. Sternberg, Charles Baxter, Richard A. Currie, Stuart J. Dunbar, Mark Earll, Jose Domingo Salazar. Does multi-clause learning help in real-world applications?. In *Proceedings of the 21st International Conference on Inductive Logic*

Programming (ILP2011), LNAI 7207, pages 221-237, 2012. This paper is the basis for Chapter 4, where it demonstrates how multi-clause learning can help in real-world applications like the two biological applications considered in the paper. The author of this thesis contributed to applying MC-TopLog to the two biological domains, as well as writing the paper. Jianzhong Chen and Hiroaki Watanabe contributed to the Progol experiments. Stephen Muggleton conceived the study in this paper. Pooja Jain provided the biological interpretation of hypotheses suggested by MC-TopLog. All authors participated in the discussion of biological modelling using ILP.

- [MLPTN13] Stephen H. Muggleton, Dianhuan Lin, Niels Pahlavi and Alireza Tamaddoni-Nezhad, Meta-Interpretive Learning: application to Grammatical inference, *Machine Learning*, 2013. Published online. In this paper, the initial idea and the theoretical framework, as well as the initial implementation of MIL in Prolog are all due to Stephen Muggleton. The author of this thesis contributed to (1) formulating MIL in ASP; (2) conducting all experiments; (3) writing the sessions of Implementations and Experiments. The third author Niels Pahlavi contributed to the previous work on efficient Higher-order Logic learning in a First-order Datalog framework. The fourth author Alireza Tamaddoni-Nezhad contributed to providing relevant text for related work and the proof-reading of the paper. In terms of the writing of the paper, the author of this thesis wrote the sessions of 'Implementations' and 'Experiments', while the rest are written by Stephen Muggleton.

- [ML13] S.H. Muggleton and D. Lin. Meta-Interpretive learning of higher-order dyadic Datalog: Predicate invention revisited. In *Proceedings of the 23rd International Joint Conference Artificial Intelligence (IJCAI 2013)*, 2013. In Press. In this paper, Stephen Muggleton contributed to the idea and the theoretical framework. Stephen Muggleton also implemented Metagol$_D$ and wrote the paper. The author of this thesis conducted the experiment of robot strategy learning and provided relevant text.

1.4. Thesis Outline

Chapter 2 introduces the relevant background.

Chapter 3 introduces MC-TopLog, which is a new ILP system that uses a logic program called a top theory as its representation for declarative bias. MC-TopLog also features multi-clause learning and doing common generalisations.

Chapter 4 is about the real-world applications of MC-TopLog, which shows the advantage of multi-clause learning and common generalisation.

Chapter 5 introduces a framework for meta-interpretative learning (MIL), which supports predicate invention. The meta-interpreter, as a main component in the MIL framework, is itself a logic program, which can incorporate both declarative and procedural bias.

Chapter 6 discusses related work.

2. Background

2.1. Machine Learning

2.1.1. Overview

We start by providing a standard definition of Machine Learning.

Definition 1 *A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E. [Mit97]*

According to this definition, machine learning involves automatic improvement of a computer program through experience. This is similar to that of human learning, since children gain knowledge through experience when growing up. The analogy that machine must learn in the same way as a human child is first made by Turing in his 1950 article 'Computing Machinery and Intelligence' [Tur50]. Turing points out: developing a child machine is a much more effective way to achieve human-level AI than by manually programming a digital computer [Mug13].

Apart from pointing out the necessity of developing machine learning techniques for achieving human-level AI, Turing [Tur50] also suggests the use of computational logic as representation language. Inductive Logic programming (ILP) [Mug91] is such a logic-based Machine Learning technique. On the other hand, ILP is unable to handle uncertainty due to its logical representation. Therefore a new research area called Statistical Relational Learning [GT07] emerges. For example, methods like Stochastic Logic Programming [Mug96] and Markov Logic Network [RD06] can be viewed as extensions of ILP which employ probabilistic semantics.

Turing [Tur50] also suggests the use of background knowledge in learning. Based on Information Theory, he foresees the difficulty with *ab initio* machine learning, that is, learning without background knowledge [Mug13].

It was later confirmed by Valiant's theory of learnable [Val84] that "effective ab initio machine learning is necessarily confined to the construction of relatively small chunks of knowledge" [Mug13]. The ability to utilise background knowledge is exactly one of the advantages of ILP over other machine learning techniques.

According to the types of feedback available for learning, existing machine learning techniques can be divided into: supervised learning, semi-supervised learning and unsupervised learning. In supervised learning, training examples are labelled and the learning target is a function that maps inputs to outputs. While in the case of semi-supervised learning, the input contains both labelled and unlabelled data. Typically there are much more unlabelled examples than labelled ones due to the high cost of assigning labels. In contrast, labels of training examples are not available in unsupervised learning. ILP, the approach being followed in this thesis, belongs to supervised learning. Supervise learning is essentially a process of generalising an hypothesis from a set of observations. The generalised hypothesis can then be used for making predictions on unseen data.

2.1.2. Computational Learning Theory

How do we know whether good hypotheses are learnable from given examples and background knowledge? How do we know whether the provided examples are sufficient to learn a good hypothesis? Answers to these kind of questions vary with different settings. For example, what defines a good hypothesis and does it have to be the target hypothesis? In this thesis, we consider a setting called probably approximately correct (PAC) learning. PAC-learnability [Val84] is first proposed by Valiant [Val84]. It extends from Gold's theory of "language identification in the limit" [Gol67], which is an early attempt to formally study learnability. Gold's "language identification in the limit" is restricted to grammar induction, while PAC model extends it to general learning problems.

A definition of PAC-learnability is given in Definition 2, where ϵ is the bound on the error and (1-δ) corresponds to the probably of outputting an hypothesis with low error ϵ.

Definition 2 *Consider a concept class C defined over a set of instances X of length n and a learner L using hypothesis space H. C is* ***PAC-learnable***

by L using H if for all $c \in C$, distributions $\mathcal{D}$ over X, ϵ such that $0 < \epsilon < 1/2$, and δ such that $0 < \delta < 1/2$, learner L will with probability at least $(1-\delta)$ output an hypothesis $h \in H$ such that $error_{\mathcal{D}}(h) \leq \epsilon$, in time that is polynomial in $1/\epsilon$, $1/\delta$, n and size(c). [Mit97].

According to this definition, the PAC-learning model allows the learner to output an hypothesis that is an approximation to a target hypothesis with low error ϵ, rather than exactly the same as the target hypothesis. It does not require the learner to succeed all the time, but with high probability $(1-\delta)$. Additionally, it has restrictions on efficiency, as it requires the time complexity to be polynomial in $1/\epsilon$, $1/\delta$, n and size(c).

In order to show that a class of concepts C is PAC-learnable, one way is to first show that a polynomial number of training examples are sufficient for learning concepts in C, and then show the processing time of each example is also polynomial [Mit97]. It has been shown that propositional programs like conjunctions of boolean literals are PAC-learnable [Val84]. The PAC-learnability of first-order logic programs has been studied in [CP95, DMR93, Coh93]. Although arbitrary logic programs are not PAC-learnable, there are positive results on the learnability of a restricted class of logic programs. Specifically, "k-literal predicate definitions consisting of constrained, function-free, non-recursive program clauses are PAC-learnable under arbitrary distributions", where "a clause is constrained if all variables in its body also appear in the head" [DMR93]. There are also positive results on learning recursive programs, but it is highly restricted. Specifically, [Coh93] has shown that recursive programs restricted to $H_{2,1}$ are PAC-learnable, where $H_{2,1}$ means (1) each clause contains only two literals including head and body literals; (2) all predicates and functions are unary. For example, a clause like $p(f(f(f(Y)))) \leftarrow p(f(Y))$ belongs to the class of $H_{2,1}$.

Based on the PAC-learning model, we can study sample complexity, that is, how many examples are needed for successful learning. According to the fact that an hypothesis with high error will be ruled out with high probability after seeing sufficient number of examples, the following bound on the number of examples can be derived. It is usually known as the 'Blumer Bound' [BEHW89].

Blumer bound $m \geq \frac{1}{\epsilon}(ln|H| + ln\frac{1}{\delta})$

In the above m stands for the number of training examples, ϵ is the bound on the error, $|H|$ is the cardinality of the hypothesis space and $(1 - \delta)$ is the bound on the probability with which the inequality holds for a randomly chosen set of training examples. Note that when increasing $|H|$ you also increase the bound on the size of required training set. Given a fixed training set for which the bound holds as an equality, the increase in $|H|$ would need to be balanced by an increase in ϵ, i.e. a larger bound on predictive error. Therefore, the Blumer bound indicates that in the case that the target theory or its approximations are within both hypothesis spaces, a learning algorithm with larger search space is worse than the one with smaller search space in terms of both running time and predictive error bounds for a randomly chosen training set. Note that this Blumer bound argument only holds when the target hypothesis or its approximation is within the hypothesis spaces of both learners.

2.1.3. Inductive bias

The process of generalisation can be modelled as search [Mit82]. Thus when designing an inductive learning algorithm, the following issues have to be considered. They are all related to inductive bias.

1. what is the hypothesis space to be searched?
2. what is the order of search?
3. when to stop searching?

It appears as if an unbiased learner would be preferred over a biased one as an incorrect bias can be misleading. In fact, learning without any bias is futile, since an unbiased learner is unable to make a generalisation leap [Mit80]. Specifically, an unbiased hypothesis space contains the one that is simply the conjunction of all positive examples. Although this hypothesis is consistent with the training data, it is too specific to be generalised to any unseen data. Moreover, according to the Blumer bound explained earlier, it is preferable to have a stronger declarative bias[1] which defines a smaller hypothesis space, since a smaller hypothesis space would lead to a lower predictive error given a fixed amount of training examples.

[1] Assuming the stronger bias does not rule out a target hypothesis and its approximations.

Therefore, inductive bias is a critical part of a learning algorithm. The following discusses some of the commonly used biases in Machine Learning.

Declarative Bias

Definition 3 *Declarative bias specifies the hypothesis language that is allowed in the search space.*

Declarative bias is called language bias as well. It is also known as a restriction bias [Mit97], because it puts a hard restriction on the hypotheses that is considerable by a learner. For example, the CANDIDATE-ELIMINATION algorithm [Mit77] only considers conjunctions of attribute values as its candidate hypotheses. If the target hypothesis contains any disjunction, then the CANDIDATE-ELIMINATION algorithm will not be able to find it. Therefore the assumption that the target hypothesis or its approximations are within the declarative bias is not guaranteed to be correct, especially when the target hypothesis to be learned is unknown.

Procedural Bias

Traversing the entire hypothesis space to find a target hypothesis or its approximations is not just inefficient, but also infeasible in the case of an infinite hypothesis space. Therefore, an ordered search is required. One naturally occurred order in concept learning is the more-general-than partial order [Mit97]. H_1 is defined to be *more-general-than* H_2, if H_1 covers[2] more positive examples than H_2. Many concept learning algorithms leverage this more-general-than partial order for guiding the search.

Definition 4 *Procedural bias defines an order in which search is conducted*

In this thesis, we define procedural bias as in Definition 4. To the author's knowledge, the term 'procedural bias' does not appear in existing literature, although it corresponds to the preference bias described in [Mit97]. Here we choose to use the term 'procedural bias' in order to contrast with 'declarative bias'. Compared to declarative bias, which is a restriction bias, procedural bias is more desirable as an inductive bias [Mit97]. Since it only puts a

[2] An hypothesis covers an example means that the hypothesis explains the example [Mit97].

preference over the order of search, rather than a hard constraint that rules out some possible hypotheses. In addition, procedural bias can lead to effective pruning. For example, in a general-to-specific search, if an hypothesis H does not achieve any compression, then pruning can be applied to the whole subspace of hypotheses which are more specific than H. Therefore procedural bias allows learners to search within an infinite space without enumerating all hypotheses. It also provides a mean for improving efficiency by leveraging the information available.

Occam's Razor

The goal of search is to find an hypothesis that is consistent with examples. However, this criteria alone is not enough for a learner to choose one hypothesis over another as output. Since there could be multiple hypotheses which are all consistent with the examples. In that case it is unclear which one should be chosen. Occam's razor suggests choosing the shortest one from those consistent. However, two learners with different representations may suggest different hypotheses, even though they both follow the Occam's razor principle. This issue is addressed by the study of Kolmogorov complexity in [Sol64, Kol65], which formally define the notion of simplicity suggested in Occam's razor. Specifically, it is proposed in [Sol64, Kol65] to measure the simplicity of an hypothesis by a shortest equivalent Turing machine program. In this way, the definition of simplicity no longer depends on any particular representation used by learners. Minimum Description Length [Ris78] is an approximation to such representation-free measures. It is also a measure which is a trade-off between the complexity of an hypothesis and coverage of this hypothesis, thus it avoids overfitting.

2.1.4. Evaluating Hypotheses

An hypothesis H can be evaluated by its predictive accuracy, that is, the portions of unseen data that H makes correct prediction. In order to make this evaluation, we have to make the assumption that examples are independent and identically distributed (iid). This guarantees that test and training examples are drawn from the same distribution. The simplest approach to get predictive accuracy is holdout cross-validation. It randomly splits the available data into two sets: one for training and one for testing.

This approach is only applicable when the data is in abundance. Another approach is called k-fold cross validation. It randomly splits the available data into k folds with similar size. Then k rounds of learning are performed. On each round, one fold is hold out as test data, while the remaining are used for training. Leave-one-out cross validation (LOOCV) is a special case of k-fold cross validation, when $k = n$ (n is the number of examples). More details about cross-validation can be found in [Sto74].

2.2. Logic Programming

Logic programming is a declarative programming paradigm. The advantage of declarative programming is to separate the specification of problem from how to solve the problem. As formulated in [Kow79], Algorithm = Logic + Control. Therefore, despite being declarative, logic programming languages are accompanied with procedural semantics in its solvers. The procedural semantic lies in the logical inference, which is different from the declarative semantic directly conveyed by the the language itself, since it is based on first-order logic. This will be discussed in more details later. The following introduces the logical notations used in this thesis first, and then explains two different logic programming languages: Prolog and Answer Set Programming (ASP).

2.2.1. Logical Notation

A variable is represented by an upper case letter followed by a string of lower case letters and digits. A function symbol or predicate symbol is a lower case letter followed by a string of lower case letters and digits. The set of all predicate symbols is referred to as the predicate signature and denoted $\mathcal{P}$. The arity of a function or predicate symbol is the number of arguments it takes. A constant is a function or predicate symbol with arity zero. The set of all constants is referred to as the constant signature and denoted $\mathcal{C}$. Functions and predicate symbols are said to be monadic when they have arity one and dyadic when they have arity two. Variables and constants are terms, and a function symbol immediately followed by a bracketed n-tuple of terms is a term. A variable is first-order if it can be substituted for by a term. A variable is higher-order if it can be

substituted for by a predicate symbol. The process of replacing (existential) variables by constants is called Skolemisation. The unique constants are called Skolem constants. A predicate symbol or higher-order variable immediately followed by a bracketed n-tuple of terms is called an atomic formula or atom for short. The negation symbol is $\neg$. Both A and $\neg A$ are literals whenever A is an atom. In this case A is called a positive literal and $\neg A$ is called a negative literal. A finite set (possibly empty) of literals is called a clause. A clause represents the disjunction of its literals. Thus the clause $\{A_1, A_2, ..\neg A_i, \neg A_{i+1}, ...\}$ can be equivalently represented as $(A_1 \vee A_2 \vee ..\neg A_i \vee \neg A_{i+1} \vee ...)$ or $A_1, A_2, .. \leftarrow A_i, A_{i+1}, ...$ A Horn clause is a clause which contains at most one positive literal. A Horn clause is unit if and only if it contains exactly one literal. A denial or goal is a Horn clause which contains no positive literals. A definite clause is a Horn clause which contains exactly one positive literal. The positive literal in a definite clause is called the head of the clause while the negative literals are collectively called the body of the clause. A unit clause is positive if it contains a head and no body. A unit clause is negative if it contains one literal in the body. A set of clauses is called a clausal theory. A clausal theory represents the conjunction of its clauses. Thus the clausal theory $\{C_1, C_2, ...\}$ can be equivalently represented as $(C_1 \wedge C_2 \wedge ...)$. A clausal theory in which all predicates have arity at most one is called monadic. A clausal theory in which all predicates have arity at most two is called dyadic. A clausal theory in which each clause is Horn is called a Horn logic program. A logic program is said to be definite in the case it contains only definite clauses. Literals, clauses and clausal theories are all well-formed-formulae (wffs) in which the variables are assumed to be universally quantified. Let E be a wff or term and σ, τ be sets of variables. $\exists\sigma.E$ and $\forall\tau.E$ are wffs. E is said to be ground whenever it contains no variables. E is said to be higher-order whenever it contains at least one higher-order variable or a predicate symbol as an argument of a term. E is said to be Datalog if it contains no function symbols other than constants. A logic program which contains only Datalog Horn clauses is called a Datalog program. The set of all ground atoms constructed from $\mathcal{P}, \mathcal{C}$ is called the Datalog Herbrand Base. $\theta = \{v_1/t_1, .., v_v/t_n\}$ is a substitution in the case that each v_i is a variable and each t_i is a term. $E\theta$ is formed by replacing each variable v_i from θ found in E by t_i. μ is called a unifying substitution for atoms A, B

in the case $A\mu = B\mu$. We say clause C θ-subsumes clause D or $C \succeq_\theta D$ whenever there exists a substitution θ such that $C\theta \subseteq D$.

A logic program is said to be higher-order in the case that it contains at least one constant predicate symbol which is the argument of a term. A meta-rule is a higher-order wff

$$\exists\sigma\forall\tau P(s_1, .., s_m) \leftarrow .., Q_i(t_1, .., t_n), ..$$

where σ, τ are disjoint sets of variables, $P, Q_i \in \sigma \cup \tau \cup \mathcal{P}$ and $s_1, .., s_m, t_1, .., t_n \in \sigma \cup \tau \cup \mathcal{C}$. Meta-rules are denoted concisely without quantifiers as

$$P(s_1, .., s_m) \leftarrow .., Q_i(t_1, .., t_n), ..$$

2.2.2. Prolog

Prolog is restricted to Horn clauses. Its inference rule is SLD-resolution. The evaluation of a Prolog program is based on depth-first search and left-most selection rule. As a result, the procedural semantic of a Prolog program depends on the order of clauses, as well as the order of literals in a clause. For example, the Prolog program P_2 in Figure 2.1(b) is similar to P_1 (Figure 2.1(a)) except that the order of literals is different between C_{12} and C_{22}. However, P_2 is less efficient than P_1. P_2 is also in the dangerous of running into non-termination if no depth-bound is imposed. More details about Prolog can be found in [Bra11, SS86]. Among Prolog compilers, YAP [Cos] is one of the high-performance ones. Thus it is used in all of our experiments related to Prolog.

C_{11}	$path(X, Y) :- edge(X, Y)$
C_{12}	$path(X, Z) :- edge(X, Y), path(Y, Z)$

(a) P_1

C_{21}	$path(X, Y) :- edge(X, Y)$
C_{22}	$path(X, Z) :- path(X, Y), edge(Y, Z)$

(b) P_2

Figure 2.1.: Tail Recursion (P_1) vs. Non-Tail Recursion (P_2)

2.2.3. Answer Set Programming (ASP)

ASP has similar syntax to Prolog, except language extensions like cardinality constraints and optimisation statements. Details of the language extensions can be found in [GKKS12]. ASP is based on stable model semantics [GL88]. Its computation mechanism is different from that of Prolog. Specifically, it leverages the high-efficiency of constraint solvers. Therefore ASP can tackle computationally hard problems [GKKS12].

Different ASP solvers have different search algorithms, but they are all based on the Davis-Putnam-Logemann-Loveland (DPLL) algorithm [MLL62]. Clasp [GKNS07] is an ASP solver that competes favourably in international competitions with SAT-solvers. It uses a search algorithm called conflict-driven nogood learning. Its main difference from DPLL is: when certain assignment is unsatisfiable, it analyses the conflict and learns from the conflict by adding a conflict constraint instead of systematic backtracking. Although logic programs in ASP do not encode procedural semantics based on the order of clauses or literals like that in Prolog, ASP solvers do have their own built-in procedural bias.

2.3. Deduction, Abduction and Induction

Deduction is based on the sound inference rule. For example, in Figure 2.2, $E = \{bird(a)\}$ can be derived from B and H according to Modus Ponens or Resolution.

Unlike deduction that has sound inference rule, abduction and induction are empirical. Both abduction and induction can be viewed as the inverse of deduction, while an abductive hypothesis is about ground or existentially quantified facts which requires minimal answer, and an inductive hypothesis is about general rules. For example, $B = \{hasFeather(a)\}$ in Figure 2.2 can be abduced from the given example bird(a) based on the general rule H. In contrast, the general rule $H = \{bird(X) \leftarrow hasFeather(X)\}$ can be induced from the given example bird(a) based on the ground fact $B = \{hasFeather(a)\}$.

$B = \{hasFeather(a)\}$
$H = \{bird(X) \leftarrow hasFeather(X)\}$ E = {bird(a)}

Figure 2.2.: Bird example

2.4. Inductive Logic Programming

Inductive Logic Programming [MR94] lies in the intersection of Machine Learning and Logic Programming. It is a branch of Machine Learning that uses logic programs as a uniform representation for its hypothesis, background knowledge and examples. Compared to other machine learning approaches, ILP is not only based on computational logic, but also capable of using background knowledge. Both of these advantages are considered by Turing as critical for achieving human-level AI [Tur50], as mentioned in Section 2.1.1. The following summaries the advantages of ILP, where the last two advantages are due to its logical representation.

1. Capable of using background knowledge
2. Relational Learning. e.g. structure learning
3. Comprehensible, which is important for hypothesis validation.

2.4.1. Logical setting

There are three different learning settings in ILP: learning from interpretations, learning from entailment, learning from proofs [MRP$^+$11]. In this thesis, we consider learning from entailment, where entailment means true in all minimal Herbrand models while a training example is an observation of the truth or falsity of a logical formula [MR94].

A general setting of ILP allows background knowledge B, hypothesis H and examples E to be any wff, but many ILP systems restricted to definite clauses. This is called the definite setting [MRP$^+$11]. The advantage of restricting to being definite is "a definite clause theory has a unique minimal Herbrand model and any logical formulae is either true or false in the minimal model" [MR94]. In this thesis, we consider a special case of the definite setting, where examples are restricted to a set of true and false ground facts, rather than general clauses. This setting is called example setting [MR94].

E and B are inputs to an ILP system, while H is output. Their logical relations can be described by the following prior and posterior conditions [MR94]. The prior satisfiability ensures the consistency of the input data. The prior necessity requires that there are positive examples unexplainable by the background knowledge, so that learning is necessary. The aim of learning is to derive an hypothesis that covers all positive examples while none of the negative examples, as described by the two posterior conditions.

$$B \not\models E^- \ \ (prior\ satisfiability) \tag{2.1}$$

$$B \not\models E^+ \ \ (prior\ necessity) \tag{2.2}$$

$$B \wedge H \not\models E^- \ \ (posterior\ satisfiability) \tag{2.3}$$

$$B \wedge H \models E^+ \ \ (posterior\ sufficiency) \tag{2.4}$$

2.4.2. Inductive bias

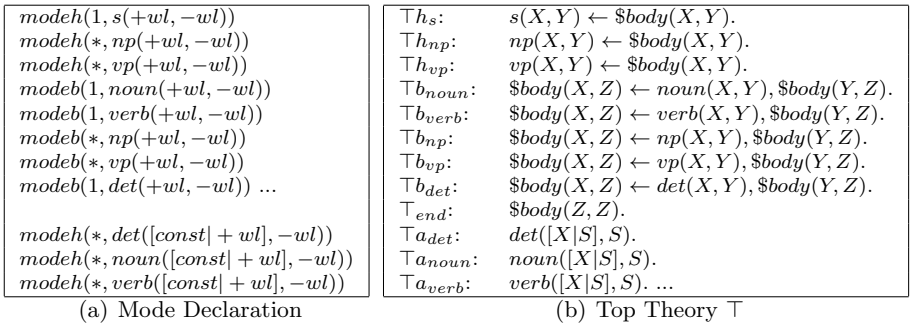
As a machine learning technique, ILP has the inductive bias mentioned in Section 2.1.3. This subsection explains how ILP represents and implements those inductive bias, especially declarative and procedural bias.

Declarative bias

Various representations have been employed by different ILP systems for representing declarative bias. Mode declaration [Mug95] is one of the widely used declarative bias. This thesis considers using top theory, which was first introduced in [MSTN08]. The following explains what is a top theory and how to use it for representing a declarative bias.

Top theories as declarative bias A top theory $\top$ is essentially a logic program. Similar to a context-free grammar, a top theory consists of terminals and non-terminals. The terminal literals are those in the hypothesis language, such as $s(X, Y)$ in Figure 2.3(b); while the non-terminal literals like $\$body(X, Y)$ in Figure 2.3(b) are not allowed to appear in neither the hypothesis language nor background knowledge. In order to distinguish the non-terminals, they are prefixed with the symbol '$'. Although the non-terminals do not appear in the hypothesis language, they play important

role in composing the hypothesis language. More examples of various non-terminals can be found in [Lin09]. Figure 2.3(b) shows a top theory for grammar learning, while its corresponding[3] version of a mode declaration is in Figure 2.3(a).



$modeh(1, s(+wl, -wl))$
$modeh(*, np(+wl, -wl))$
$modeh(*, vp(+wl, -wl))$
$modeb(1, noun(+wl, -wl))$
$modeb(1, verb(+wl, -wl))$
$modeb(*, np(+wl, -wl))$
$modeb(*, vp(+wl, -wl))$
$modeb(1, det(+wl, -wl))$ ...

$modeh(*, det([const| + wl], -wl))$
$modeh(*, noun([const| + wl], -wl))$
$modeh(*, verb([const| + wl], -wl))$

(a) Mode Declaration

$\top h_s$:	$s(X,Y) \leftarrow \$body(X,Y).$
$\top h_{np}$:	$np(X,Y) \leftarrow \$body(X,Y).$
$\top h_{vp}$:	$vp(X,Y) \leftarrow \$body(X,Y).$
$\top b_{noun}$:	$\$body(X,Z) \leftarrow noun(X,Y), \$body(Y,Z).$
$\top b_{verb}$:	$\$body(X,Z) \leftarrow verb(X,Y), \$body(Y,Z).$
$\top b_{np}$:	$\$body(X,Z) \leftarrow np(X,Y), \$body(Y,Z).$
$\top b_{vp}$:	$\$body(X,Z) \leftarrow vp(X,Y), \$body(Y,Z).$
$\top b_{det}$:	$\$body(X,Z) \leftarrow det(X,Y), \$body(Y,Z).$
$\top_{end}$:	$\$body(Z,Z).$
$\top a_{det}$:	$det([X\|S],S).$
$\top a_{noun}$:	$noun([X\|S],S).$
$\top a_{verb}$:	$verb([X\|S],S).$ ...

(b) Top Theory $\top$

Figure 2.3.: Declarative bias for grammar learning

Composing hypothesis language using a top theory Considering a top theory $\top$ defines a hypothesis space, a hypothesis H within the hypothesis space holds for $\top \models H$. According to the Subsumption theorem [NCdW97], the following two operators are sufficient to derive any H that meets $\top \models H$: SLD-resolution and substitution. By applying SLD-resolution to resolve all the non-terminals in an SLD-derivation sequence, an hypothesis clause with only terminals can be derived. For example, an hypothesis clause $s(S1, S2) \leftarrow np(S1, S3), vp(S3, S4), np(S4, S2)$ can be derived from an SLD-derivation sequence $[\top h_s, \top b_{np}, \top b_{vp}, \top b_{np}, \top_{end}]$, where clauses $\top_i$ are in Figure 2.3. Different from SLD-resolution, which is to do with connecting terminal literals, substitution is required to deal with ground values in the terminal literals. For example, abductive hypotheses are ground facts, while their corresponding top theories are universally quantified, e.g. $noun([X|S], S)$ in Figure 2.3(b). In this thesis, *translation* refers to the process of deriving H from $\top$, and $\top$ *version of* H refers to the set of clauses in $\top$ that derives H.

[3]The first argument in a mode declaration is about the times of recall. Such information is not included in the top theory in 2.3(b), but it is feasible to be included via additional arguments in a top theory

Procedural bias

The notion of 'generalisation as entailment' is first introduced by Plotkin [Plo69]. Later it is extended to relative generalisation, that is, generalisation with respect to background knowledge [Plo71b]. Generalisation order naturally provides a partial order for search. However, due to the undecidability of entailment, a more restricted form, that is, subsumption is considered. The concept of refinement operator based on subsumption was first introduced by Shapiro [Sha83]. More details about refinement operator can be found in [NCdW97]. There are also other attempts like generalised subsumption [Bun86] to overcome the limitation of subsumption in defining lattice properties of hypothesis space, while not sacrifice the decidability.

There are ILP systems like TopLog [MSTN08] with no procedural bias. It enumerates all hypotheses consistent with examples. Most ILP systems perform an ordered search based on subsumption lattice. This is built-in, thus not modifiable by users. No existing ILP system has made the procedure bias as an explicit component. For example, Progol performs a top-down search[4]. In the case that a target hypothesis is closer to a bottom clause, Progol would find the target hypothesis with a much shorter amount of time if it can switch from top-down to bottom-up. Additionally, in the case that there is any other domain-specific procedural bias, it is not encodable by existing ILP systems unless doing a 'surgery' to the system.

2.4.3. Theory derivation operators

Apart from an ordered search performed by refinement operators, bounding the search space to those that cover at least one positive example is another solution to a large hypothesis space. We use the term 'theory derivation operators' to refer to operators that only derive hypotheses that cover at least one positive example. The following discusses three types of such operators: (a) Inverse Entailment; (b) $\top$-directed operator; (c) Common Generalisation.

[4] Although Progol uses a bottom clause $\bot$ to bound its search space, its search procedural is general-to-specific.

Inverse Entailment

Inverse Entailment (IE) [Mug95] is an inverse operator that derives H from B and E^+. The following equation is the basis for Inverse Entailment, which shows $\neg H$ can be derived as the consequences of B and $\neg E^+$. In this way, all the derivable hypotheses cover at least one positive examples, while those do not cover any positive examples are pruned by this data-driven approach.

$$B, \neg E^+ \models \neg H \tag{2.5}$$

$\top$-directed operator

As an alternative to Inverse Entailment, a $\top$-directed operator does not involve an inverse process to deduction, but deductively find a hypothesis that explains at least one positive example. This is only feasible via a schema to take the place of the unknown hypothesis in a deductive proof. This schema is a top theory $\top$, which can be used to represent a declarative bias, as explained in Section 2.4.2. The equations below are basis for

$$B, \top \models E^+ \tag{2.6}$$

$$\top \models H \tag{2.7}$$

Common Generalisation

The previous two types of theory derivation operators generalise a single example to a hypothesis. In other words, they use a single example as a seed example. This is referred to as 'solo-generalisation' in this thesis. In contrast, common generalisation operators generalise multiple examples together. Thus they are referred as 'co-generalisation', which makes it possible to constrain a search space to common generalisations.

2.4.4. Leveraging ASP for ILP

Considering ILP uses logic programs as its representation language, an ILP system can be implemented using ASP, apart from Prolog. Apart from the potential advantage in efficiency, the use of ASP can improve the predictive accuracy, since an ASP solver's optimisation component is handy for finding

a globally optimal hypothesis, while a globally optimal hypothesis might have higher predictive accuracy than a locally optimal hypothesis.

2.5. Grammatical inference

Grammatical inference (or grammatical induction) is the process of learning a grammar from a set of examples. It is closely related to the fields of machine learning as well as the theory of formal languages. It has numerous real-world applications including speech recognition (e.g. [Sto95]), computational linguistics (e.g. [Flo02]) and computational biology (e.g. [SB02]).

2.5.1. Formal language notation

Let Σ be a finite alphabet. Σ^* is the infinite set of strings made up of zero or more letters from Σ. λ is the empty string. uv is the concatenation of strings u and v. $|u|$ is the length of string u. A language L is any subset of Σ^*. Let ν be a set of non-terminal symbols disjoint from Σ. A production rule $r = LHS \rightarrow RHS$ is well-formed in the case that $LHS \in (\nu \cup \Sigma)^*$, $RHS \in (\nu \cup \Sigma \cup \lambda)^*$ and when applied replaces LHS by RHS in a given string. A grammar G is a pair $\langle s, R \rangle$ consisting of a start symbol $s \in \nu$ and a finite set of production rules R. A grammar is Regular Chomsky-normal in the case that it contains only production rules of the form $S \rightarrow \lambda$ or $S \rightarrow aB$ where $S, B \in \nu$ and $a \in \Sigma$. A grammar is Linear Context-Free in the case that it contains only Regular Chomsky-normal production rules or rules of the form $S \rightarrow Ab$ where $S, A \in \nu$ and $b \in \Sigma$. A grammar is Context-Free in the case that it contains only Linear Context-Free Chomsky-normal production rules or rules of the form $S \rightarrow AB$ where $S, A, B \in \nu$.[5] A Context-Free grammar is said to be deterministic in the case that it does not contain (a) two Regular Chomsky-normal production rules $S \rightarrow aB$ and $S \rightarrow aC$ where $B \neq C$ (b) two Regular Chomsky-normal production rules $S \rightarrow Bb$ and $S \rightarrow Cb$ where $B \neq C$. A sentence $\sigma \in \Sigma^*$ is in the language defined by a grammar Giff given a start symbol $S \in \nu$ there exists a sequence of production rule applications $S \rightarrow_{R_1} \ldots \rightarrow_{R_n} \sigma$ where $R_i \in G$. A language L is Regular, Linear Context-free or Context-Free in the case there exists a grammar G for which $L = L(G)$ where G is Regular, Linear

[5] This is an adaptation of Chomsky-normal form Context-free, which only permits productions of the form $S \rightarrow \lambda$, $S \rightarrow a$ and $S \rightarrow AB$.

Context-Free or Context-Free respectively. According to the Context-Free Pumping Lemma [HU79], if a language L is Context-Free, then there exists some integer $p \geq 1$ such that any string s in L with $|s| \geq p$ (where p is a constant) can be written as $s = uvxyz$ with substrings u, v, x, y and z, such that $|vxy| \leq p$, $|vy| \geq 1$ and $uv^n xy^n z$ is in L for every integer $n \geq 0$.

3. MC-TopLog: Complete Multi-clause Learning Guided by A Top Theory

This chapter introduces a new ILP system MC-TopLog, which has the following features:

1. entailment-complete multi-clause learning;
2. using a logic program called the top theory as declarative bias;
3. co-generalisation (common generalisation).

This chapter is based on the paper published in [MLTN11]. Although both ⊤DTD and ⊤DTcD are first published in [MLTN11], ⊤DTcD is the main contribution of this chapter since ⊤DTD was already described in the author's Master thesis [Lin09].

3.1. Introduction

Many ILP systems like Progol [Mug95] are restricted to deriving hypotheses that subsume E relative to B in Plotkin's sense [Plo71a], as first pointed out by Yamamoto [Yam97]. This type of incompleteness means that a positive example can only be generalised to a single clause, but not a theory with multiple clauses. In this chapter, we compare entailment-incomplete single-clause learning systems to entailment-complete multi-clause learning systems.

Yamamoto uses the learning of odd-numbers to demonstrate Progol's incompleteness. His example involves recursion and mutually dependent predicates (odd and even), making it unclear whether only applications with these properties might be affected by this type of incompleteness. Based

on the assumption that a theory can be built by sequentially adding single clauses, single-clause learners like Progol has been applied for automated discovery of food web [BCLM+11], where multiple clauses are required for encoding the eating relations among various animals.

We use a simplified version of grammar learning in this chapter to show that a multi-clause learner can improve upon the learning results of a single-clause learner. This is further demonstrated in the experiments of this chapter. Two data sets are used as experimental material: one does not involve recursion or mutually dependent predicates, while the other involve recursion. More experiments with real-world applications can be found in the next chapter, where target hypotheses are unknown in knowledge discovery tasks.

The challenge of doing multi-clause learning is a much larger search space, compared to doing single-clause learning. Therefore it is important to use all the information available to constrain the search space. This chapter shows that a logic program called the top theory can encode a strong declarative bias. In fact, the use of a top theory $\top$ not only provides a mechanism for naturally encoding a strong declarative bias, but also facilitates the bounding of a search space to those covering at least one positive example. By program transformation, a $\top$-directed method can further bound its search space to common generalisations of multiple examples. This is the main extension of $\top$-Directed Theory Derivation ($\top$DTD) to $\top$-Directed Theory co-Derivation ($\top$DTcD).

The structure of this chapter is as follows. It first explains the problem to be solved, where multi-clause learning is formally defined to avoid confusion with other learning problems. Then MC-TopLog is introduced with details about representing strong declarative bias using top theories, as well as its two key algorithms: $\top$DTD and $\top$DTcD. The experimental results of MC-TopLog and its comparisons to Progol are presented at the end.

3.2. Multi-clause Learning

Progol's entailment-incompleteness comes from the restriction in Plotkin's C-derivation [Plo71a], which requires a hypothesised clause C to be used only once in the refutation of e. In order to overcome Progol's entailment-incompleteness, C-derivation is generalised to K-derivation in [RBR04]. Ac-

cording to Definition 5 and Definition 6, the main difference between C-derivation and K-derivation is: C is a single clause that can be used at most once while K is a set of clauses that can be used at most once. K*-derivation [RBR04] is a refinement of K-derivation, which impose an additional restriction on the way clauses in K are used. Such restriction prevents HAIL from deriving some correct explanations, as pointed in [KBR09]. The restriction of clauses in K being used at most once also excludes the learning of recursive clauses, since a recursive clause can be called more than once in a derivation. For example, in Yamamoto's example of learning odd-numbers, the hypothesised clause $odd(s(X)) \leftarrow even(X)$ is used twice when proving the positive example $odd(s(s(s(0))))$.

In MC-TopLog, we consider M-derivation as defined in Definition 7, which generalises K-derivation. Specifically, there is no restriction on the times that a hypothesis clause is used and no restriction on the way a clause is used. Due to the removal of restrictions, M-derivation is essentially the same as SLD-derivation except depth bound. An example D is said to be SLD-derivable from T with respect to M, denoted $T \wedge M \vdash_m D$, iff there exists a SLD-derivation of D from T with respect to M within certain depth bound. Therefore, the hypotheses derivable by MC-TopLog can be characterised by those deriving examples together with T using SLD-derivation with certain depth bound.

Definition 5 *Let T be a theory, and C and D be clauses. Then a C-derivation of D from T with respect to C is a tree derivation of D from $T \cup C$ such that C is the generator of at most one input clause. A clause D is said to be C-derivable from T with respect to C, denoted $T \wedge C \vdash_c D$, iff there exists a C-derivation of D from T with respect to C. [Plo71a, RBR04]*

Definition 6 *Let T and K be theories, and let D be a clause. Then a K-derivation of D from T with respect to K is a tree derivation of D from $T \cup K$ such that each clause $k \in K$ (but not in T) is the generator of at most one input clause, which is called a k-input clause. Clause D is said to be K-derivable from T with respect to K, denoted $T \wedge K \vdash_k D$, iff there exists a K-derivation of D from T with respect to K. [RBR04]*

Definition 7 *Let T and M be theories, and let D be a clause. Then a M-derivation of D from T with respect to M is a tree derivation of D from*

$T \cup M$ such that each clause $m \in M$ (but not in T) is the generator of input clauses. Clause D is said to be M-derivable from T with respect to M, denoted $T \wedge M \vdash_m D$, iff there exists a SLD-derivation of D from T with respect to M. Let N be the cardinality of M. If $N = 1$, , then it is single-clause learning (SCL); otherwise if $N \geq 1$, it is multi-clause learning (MCL).

3.2.1. Example: grammar learning

Figure 3.1 shows a simplified version of grammar learning. It is used here to exemplify Definition 7. In this grammar learning task, single-clause and multi-clause learning methods will derive $H_{sc} = \{h_1, h_2, h_3\}$ and $H_{mc} = \{h_4, h_5, h_6, h_7\}$, respectively. Although there are multiple clauses in H_{sc}, each of them is derived independently from different examples by a single-clause learner. Specifically, h_1, h_2 and h_3 are independently generalised by a single-clause learner from e_1, e_2 and e_3, respectively. In contrast, clauses in H_{mc} are dependent, and they have to be generalised together in order to explain an example. For instance, hypothesising h_4 alone can not complete the refutation proof of the example e_1, since the definition about *np* is incomplete in B and the type of the word '*unknown*' is also missing from B. Thus another two input clauses, either $\{h_5, h_7\}$ or $\{h_8, h_9\}$, have to be derived together with h_4 in order to explain e_1.

In this example, H_{mc} is the target hypothesis which is not derivable by a single-clause learner. H_{mc} is also more compressive than H_{sc}, because H_{mc} has a shorter description length[1] than H_{sc} while covering the same number of examples. The shorter description length of H_{mc} results from the learning of multiple dependent clauses, which makes it possible to derive more general and more compact hypotheses.

3.2.2. MCL vs. MPL

As discussed earlier, clauses within a multi-clause hypothesis H_{mc} are dependent. This is similar to that in multiple predicate learning (MPL), where clauses about different predicates depend on each other. However, the MPL

[1] In this thesis, the description length (DL) of a clause is defined by the number of literals in the clause; while the compression is defined as $p - n - DL$ where p and n are the number of positive and negative examples covered by the clause.

Positive and Negative Examples E:
e_1:$s([an, unknown, alien, hits, the, house], [])$.
e_2:$s([a, small, boy, walks, a, dog], [])$.
e_3:$s([a, dog, walks, into, the, house], [])$.
e_4:$\neg s([dog, hits, a, boy], [])$.

Hypothesis language $\mathcal{L}$:
Predicates $=\{s, np, vp, det, noun, verb...\}$
Variables $=\{S_1, S_2, S_3, ...\}$
Constants $=\{a, the, ...\}$

Background Knowledge B:
b_1:$np(S1, S2) \leftarrow det(S1, S3), noun(S3, S2)$.
b_2:$vp(S1, S2) \leftarrow verb(S1, S2)$.
b_3:$vp(S1, S2) \leftarrow verb(S1, S3), prep(S3, S2)$.
b_4:$det([a|S], S)$. b_5:$det([an|S], S)$.
b_6:$noun([dog|S], S)$. b_7:$noun([boy|S], S)$.
b_8:$noun([house|S], S)$. b_9:$noun([alien|S], S)$.
b_{10}:$verb([hits|S], S)$. b_{11}:$adj([small|S], S)$.
b_{12}:$prep([into|S], S)$. b_{13}:$det([the|S], S)$.

Part of Hypothesis Space $\mathcal{H}$:
h_1:$s(S1, S2) \leftarrow det(S1, S3), S3 = [Word|S4], noun(S4, S5), vp(S5, S6), np(S6, S2)$.
h_2:$s(S1, S2) \leftarrow det(S1, S3), adj(S3, S4), noun(S4, S5), S5 = [Word|S6], np(S6, S2)$.
h_3:$s(S1, S2) \leftarrow np(S1, S3), S3 = [Word|S4], prep(S4, S5), np(S5, S2)$.
h_4:$s(S1, S2) \leftarrow np(S1, S3), vp(S3, S4), np(S4, S2)$.
h_5:$np(S1, S2) \leftarrow det(S1, S3), adj(S3, S4), noun(S4, S2)$
h_9:$np(S1, S2) \leftarrow det(S1, S3), prep(S3, S4), noun(S4, S2)$
h_6:$verb([walks|S], S)$. h_7:$adj([unknown|S], S)$. h_8:$prep([unknown|S], S)$.

Figure 3.1.: Grammar Learning

discussed in [RLD93] is essentially single-clause learning. Since each predicate to be learned are observable and provided as examples. Therefore there is only one clause to be hypothesised for each example. Applying an MPL method to the learning problem in Figure 3.1 would require the predicates *np* and *vp* to be observable and provided as training examples.

3.2.3. Increase in Hypothesis Space

Although the entailment-completeness of MCL makes it possible to find hypotheses with higher compression than SCL, this comes at the cost of a much larger hypothesis space. Specifically, a single-clause learner's upper bound on the size of its hypothesis space is $O(2^N)$, where N is the number of distinct atoms derivable from an hypothesis language. In contrast, it is $O(2^{2^N})$ for a multi-clause learner, because it does not ignore the hypotheses with dependent clauses. The increase of hypothesis space posts challenges to not just the complexity of learning algorithm, but also the sample complexity. According to the Blumer bound explained in Section 2.1.2, with the increase in the size of a hypothesis space, more observed examples are required. In the case when the training examples are fixed, SCL's error bound is linear with respect to N according to Lemma 1; while MCL's error bound grows exponentially with the increase of N according to Lemma 2. Figure 3.2 depicts this difference. That is why it is particularly important for MCL to leverage the information available to bound its hypothesis space.

Lemma 1 *SCL's error bound is linear with respect to N:* $\epsilon \geq \frac{1}{m}(Nln2 + ln\frac{1}{\delta})$*, where N is the number of distinct atoms derivable from an hypothesis language.*

Proof. *Considering SCL has* $|H| = 2^N$*, then by replacing* $|H|$ *in Equation 3.1 about the Blumer bound, we can derive Equation 3.2. Then considering both* m *and* ϵ *is both larger than 0, we can derive Equation 3.3.*

$$m \geq \frac{1}{\epsilon}(ln|H| + ln\frac{1}{\delta}) \tag{3.1}$$

$$m \geq \frac{1}{\epsilon}(Nln2 + ln\frac{1}{\delta}) \tag{3.2}$$

$$\epsilon \geq \frac{1}{m}(Nln2 + ln\frac{1}{\delta}) \tag{3.3}$$

Lemma 2 *MCL's error bound grows exponentially with the increase of N:* $\epsilon \geq \frac{1}{m}(2^N ln2 + ln\frac{1}{\delta})$*, where N is the number of distinct atoms derivable from an hypothesis language.*

Proof. *Considering MCL has* $|H| = 2^{2^N}$*, then by replacing* $|H|$ *in Equation 3.1 about the Blumer bound, we can derive Equation 3.4. Then considering both* m *and* ϵ *is both larger than 0, we can derive Equation 3.5.*

$$m \geq \frac{1}{\epsilon}(2^N ln2 + ln\frac{1}{\delta}) \tag{3.4}$$

$$\epsilon \geq \frac{1}{m}(2^N ln2 + ln\frac{1}{\delta}) \tag{3.5}$$

3.3. MC-TopLog

This section first explains how to use a top theory to encode strong declarative bias, and then explains how to derive an hypothesis using a top theory. Finally, we explain how to constrain the search space to common generalisations using the ⊤DTcD algorithm.

3.3.1. Top theories as strong declarative bias

As explained in Section 2.4.2, declarative bias can be represented by a top theory. For example, the one shown in Figure 6.1(b). Its corresponding

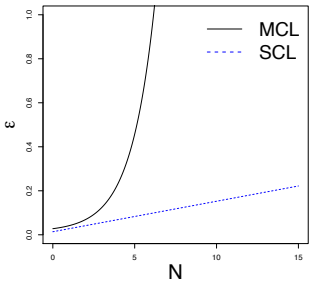
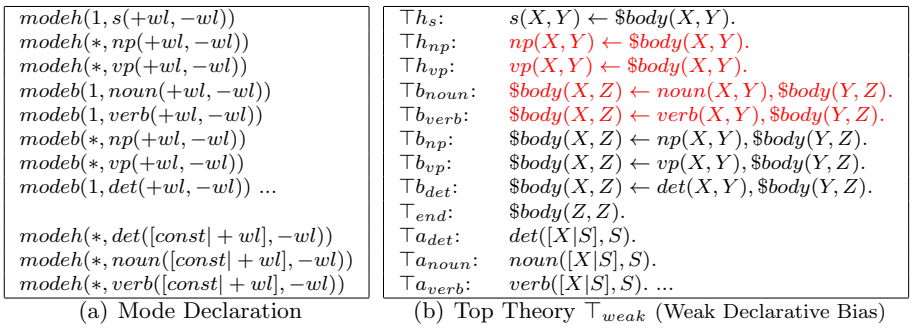


Figure 3.2.: Blumer bounds for MCL and SCL

version of a mode declaration is given in Figure 6.1(a). This kind of declarative bias only tells what predicates are allowed in the head or body of an hypothesis clause. However, in the case that a stronger declarative bias does exist, it is definitely worth to use that information to further constrain the hypothesis space. For example, in the grammar learning task, we know a noun phrase always consists of a noun and a verb phrase always has a verb. This provides information about how predicates should be connected. However, there is no way for a mode declaration to capture this information, while a top theory can encode it as that in Figure 3.3(c). Such a top theory will avoid deriving clauses like $np(S1, S3) \leftarrow det(S1, S2), adj(S2, S3)$, which defines a noun phrase without a noun. Another example of strong bias exists for learning tasks whose target hypothesis is known to be recursive. In that case, it would be more efficient if non-recursive clauses are excluded from the hypothesis space. An example of such declarative bias encoded by a top theory can be found in 3.13, where the clause highlighted in red has the predicate 'compute/3' in both its head and body, thus forces the recursion and excludes those non-recursive clauses. Apart from the strong bias about the connection of predicates, there are other types of strong bias, such as the restriction on function terms. For example, in Yamamoto's example of learning odd-numbers, it would be undesirable to have a clause like $odd(s(X)) \leftarrow even(s(s(X)))$ in the hypothesis space, since it will lead to the expansion of function terms during reasoning. A top theory encoding such declarative bias can be found in Figure 3.8, where the clause T_{nt3} restricts the expansion of function terms.



(a) Mode Declaration

(b) Top Theory $\top_{weak}$ (Weak Declarative Bias)

$\top h_s$:	$s(X,Y) \leftarrow \$body(X,Y)$.
$\top h_{np\_noun}$:	$np(X,Y) \leftarrow \$body(X,M1), noun(M1,M2), \$body(M2,Y)$.
$\top h_{vp\_verb}$:	$vp(X,Y) \leftarrow \$body(X,M1), verb(M1,M2), \$body(M2,Y)$.
	... (The rest are the same as that in Figure 6.1(b))

(c) Top Theory $\top_{strong}$(Strong Declarative Bias)

Figure 3.3.: Declarative bias for grammar learning. (a) and (b) are the same as those in Figure 2.3. They are repeated here for the convenience of reference and comparison

3.3.2. ⊤-directed Theory Derivation (⊤DTD)

⊤DTD is to derive all the candidate hypotheses that satisfy (3.6), where $\vdash_h$ denotes a derivation in at most h resolutions. ⊤DTD uses the top theory to direct the search for such hypotheses. A full description of ⊤DTD is given in Algorithm 1, where step 2, 4 and 5 are the key steps. Step 2 finds all the refutations of e that satisfy (3.7), where $\vdash_{h'}$ has the same semantic as $\vdash_h$ except[2] $h' \geq h$. It is the use of ⊤ that makes refutations of e derivable, otherwise e cannot be proved by B alone, because of the missing clauses to be hypothesised.

After deriving all the refutations of e, each refutation sequence R_i is processed to derive the corresponding H_i. This involves step 4 and 5 in Algorithm 1. Step 4 is about extracting derivation sequences D_i from each refutation sequence R_i. Each extracted sequence in D_i preserves the same order as that in R_i. This guarantees that the pair of literals resolved in D_i is the same as that in R_i. To facilitate the extraction, it requires R_i to be recorded as a list with nested sub-lists, instead of a linear sequence. More details about how to extract D_i from the R_i can be found in [Lin09]. Step 5

[2]Apart from the terminals in (1), (2) have non-terminals to be resolved, thus requires larger depth limit

is about translating D_i into H_i, which are explained in Section 2.4.2. In the case that ground values are required, the values to be substituted come from the unification that happens when refuting e using $\top$ and B. Therefore it requires R_i to record the ground values unified during the refutation.

The correctness (ie. soundness and completeness) of $\top$DTD is proved in Theorem 3.3.2. An example of how $\top$DTD works is given in Example 1. The cover set algorithm using $\top$DTD is given in Algorithm 2.

$$B \wedge H \vdash_h e \ (e \in E^+) \tag{3.6}$$

$$B \wedge \top \vdash_{h'} e \ (e \in E^+, h' \geq h) \tag{3.7}$$

$$\top \models H \tag{3.8}$$

Algorithm 1 $\top$-directed Theory Derivation ($\top$DTD)

Input: a positive example e, background knowledge B, top theory $\top$ and h'
Output: $\mathcal{H} = \{H_i : B \wedge H_i \vdash_h e\}$, where $h \leq h'$
1: Let $\mathcal{H} = \emptyset$
2: $\mathcal{R} = \{R_i : R_i = Refs(e, B, \top, h)\}$ %Find all the refutations of e that satisfy the formula 3.7
3: **for all** R_i in $\mathcal{R}$ **do**
4: $D_i = DSeqs(R_i)$ %Obtain derivation sequences D_i by extracting $\top$ clauses from R_i.
5: $H_i = Trans(D_i)$ %Translate D_i into an hypothesis theory H_i
6: $\mathcal{H} = \mathcal{H} \cup H_i$
7: **end for**
8: **return** $\mathcal{H}$

Algorithm 2 Cover set algorithm of $\top$DTD

Input: examples E, background knowledge B, top theory $\top$ and h'
Output: an hypothesis H
1: Let $H = \emptyset$ and $E^+ =$ all positive examples in E
2: **for all** $e_i \in E^+$ **do**
3: $\mathcal{H}_i = TDTD(e_i, B, \top, h')$
4: $\mathcal{H} = \mathcal{H} \cup \mathcal{H}_i$
5: **end for**
6: **while** $E^+ \neq \emptyset$ **do**
7: Let H' be the one in $\mathcal{H}$ with highest compression and $H = H \cup H'$
8: Let E' be the positive examples covered by H' and $E^+ = E^+ - E'$
9: Let $\mathcal{H}'$ be the set of candidate hypotheses that are redundant with respect to H
10: $\mathcal{H} = \mathcal{H} - \mathcal{H}'$ %Remove redundant candidates hypotheses, which cover a subset of the examples that already covered by H
11: **end while**
12: **return** H

Correctness of $\top$DTD Given e, B, $\top$ and h', Algorithm 1 returns all candidate hypotheses that satisfy (3.6), where H is within the hypothesis space defined by $\top$.

Proof. *Assume the theorem is false. Then either (a) the algorithm does*

not terminate or (b) a theory H derived by the algorithm does not satisfy (3.6) or (c) the algorithm cannot derive a theory H that is within the hypothesis space defined by $\top$, as well as satisfies (3.6).

First consider (a). Due to the restriction of at most h' resolutions in formula(3.7), $\mathcal{R}$ derived at step 2 is a finite set. Therefore there are only finite number of loops between step 3 and 7. Also each operation within the loop terminates in finite time. This refutes (a).

Secondly suppose (b) is true, which means $B \wedge H \wedge \neg e \nvDash \square$. However, step 2 can find at least one refutation R_i that satisfies (3.7), that is, $B \wedge \top \wedge \neg e \nvDash \square$. This means clauses appearing in R_i form pairs of complementary literals. Following step 4, derivation sequences D_i can be extracted from the refutation sequence R_i. Then at step 5, there are three possible ways to translate D_i into H: (1) only SLD-resolution (2) only substitution; (3) both SLD-resolution and substitution. In case (1), all the non-terminals are resolved using SLD-resolution in order to compose hypothesis clauses with only terminals. The resolved literals must be in pairs, otherwise there will be at least one literal left unresolved, which means there will be non-terminals remaining in the derived H. If replacing the $\top$ clauses in R_i with their corresponding H, whose only difference from the replaced $\top$ clauses are pairs of non-terminals, then the clauses in this new sequence still form pairs of complementary literals. Therefore it contradicts the assumption that $B \wedge H \wedge \neg e \nvDash \square$. In case (2), if replacing the $\top$ clauses with H, which is derived by substituting the variables in $\top$ with the ground values unified during the refutation, then the clauses in this new sequence still form pairs of complementary literals. Thus it also contradicts the assumption. In case (3), the assumption is also contradicted considering both case (1) and (2).

Lastly consider (c), which implies that the corresponding $\top$ version of H from which it is translated cannot be used to prove e with B, that is, the step 2 cannot be executed. However, considering that H is translatable from $\top$, that is, within the hypothesis space defined by $\top$, the formula (3.8) holds. Then (3.9) holds and (3.7) can be derived accordingly. This means a refutation using B and the $\top$ version of H does exist for e. This contradicts the assumption and completes the proof.

$$B \wedge \top \models B \wedge H \tag{3.9}$$

Example 1 *For the learning task in Figure 3.1, one of the refutations for* e_1 *is as shown in Figure 3.4. Its corresponding SLD-refutation sequence is recorded as* $R_1 = [\neg e_1, [Th_s, Tb_{np}, [Th_{np\_noun}, Tb_{det}, b_5, Tb_{prep}, [Ta_{prep}(unknown)], T_{end}, b_9, T_{end}], Tb_{vp}, b_2, b_{10}, Tb_{np}, b_1, b_{13}, b_8, T_{end}]]$. *Using the extraction algorithm explained in [Lin09],* D_1 *consisting of three derivation sequences can be extracted from* R_1. *They are:* $d_1 = [Th_s, Tb_{np}, Tb_{vp}, Tb_{np}, T_{end}]$, $d_2 = [Th_{np\_noun}, Tb_{det}, Tb_{prep}, T_{end}, T_{end}]$ *and* $d_3 = [Ta_{prep}(unknown)]$, *which are highlighted by the three square boxes in Figure 3.4. Then by applying SLD-derivation and substitution to* D_1, $T_1 = \{h_4, h_8, h_9\}$ *can be derived, where* h_i *is in Figure 3.1.*

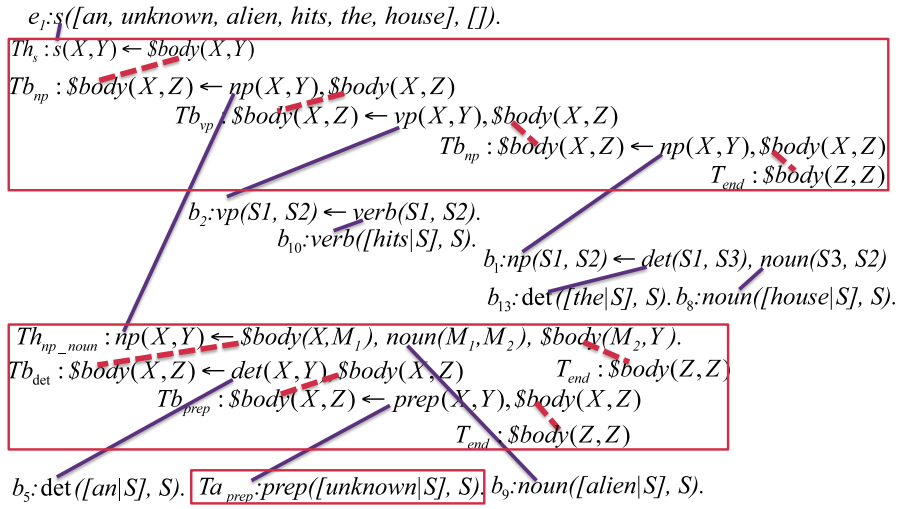


Figure 3.4.: Refutation of e_1 using clauses in B and $\top_{strong}$(Figure 3.3(c)). The dash lines represent resolving a pair of non-terminal literals, while the solid lines correspond to the terminals.

3.3.3. ⊤-directed Theory Co-Derivation (⊤DTcD)

In order to constrain the derivable hypotheses to common generalisations, ⊤DTcD extends ⊤DTD based on co-refutation. Co-refutation combines the refutations that are the same except the instantiation of variables. Co-refutation can be done via program transformation. Specifically, literals

of the same predicate can be combined into one literal by combining their corresponding arguments into a compound. For example, the refutation proof in Fig 3.5(c) is the result of combining the two refutation proofs in Fig 3.5(a) and Fig 3.5(b). Co-refutation has the advantage of proving several examples together in a compound proof. More importantly, it proves them using the same non-ground clauses.

The design of ⊤DTcD is based on the fact that if a theory is common to multiple examples E, then the refutation proofs of each example in E using that common theory will have the same structure, that is, the proofs are the same except the instantiation of variables. Those same-structure refutation proofs can be combined into co-refutation by combining corresponding arguments. It is the combined proof that forces the co-generalised examples to be proved using the same non-ground rules.

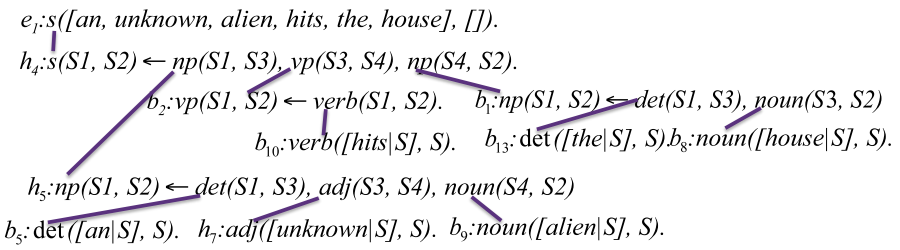
In terms of how to choose the examples to be generalised together, rather than randomly sample a pair of examples as that in ProGolem, ⊤DTcD takes all positive examples as input. The examples that do not fit the compound proof are filtered out during the derivation of a compound refutation proof. At the end of a refutation, not only an hypothesis is derived, but also the maximum set of examples that can be explained by that hypothesis.

The algorithm of ⊤DTcD is given in Algorithm 3. It is the same as Algorithm 1 except (1) its input and output; (2) its step 2 and 3, where it combines examples to be generalised together into a compound and queries the compound instead of a single example. The cover set algorithm of ⊤DTcD is also slightly different from that of ⊤DTD, since the output of ⊤DTcD contains the candidate hypotheses for all the positive examples, rather than just one example. Specifically, the steps 2-5 in Algorithm 2 need to be replaced by a single step: $\mathcal{H} = TDTcD(E^+, B, \top, h')$. The correctness of ⊤DTcD is given in Theorem 3.3.3. We also give an example of how ⊤DTcD works in Example 2.

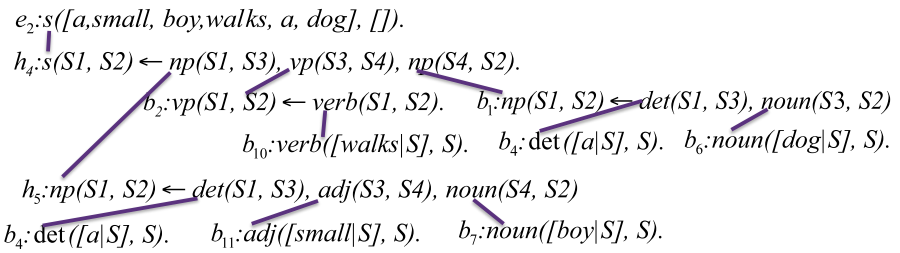
$$B \wedge H \vdash_h E_i \tag{3.10}$$

$$B \wedge \top \vdash_{h'} E_i \tag{3.11}$$

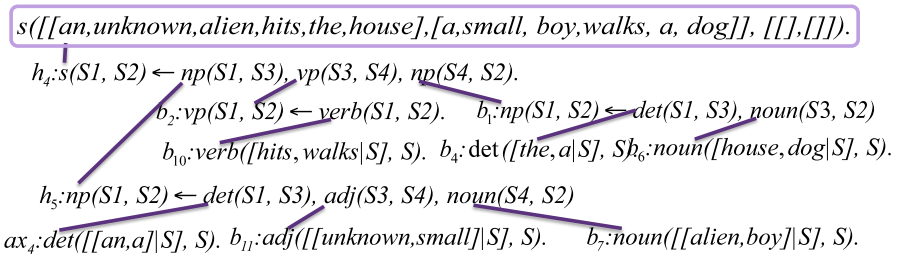
$where\ h' \geq h \wedge (E_i \subset E^+ \wedge |E_i| > 1) \wedge (\forall e_j \in E_i, sameRefStru(e_j))$



(a) Refutation-proof of e_1



(b) Refutation-proof of e_2



(c) Co-refutation of e_1 and e_2

Figure 3.5.: Combine same structure refutation-proofs

Correctness of ⊤DTcD Given E^+, B, $\top$ and h', Algorithm 3 returns all candidate hypotheses that hold for (3.10), where (1) H is within the hypothesis space defined by $\top$; (2) $E_i \subset E^+, |E_i| > 1$ and each $e_j \in E_i$ shares the same structure of refutation proofs.

Proof. *Assume the theorem is false. Then either (a) the algorithm does not terminate or (b) a theory H is derived by the algorithm as a co-generalisation of E_i, while $\exists e_j \in E_i, B \wedge H \not\models e_j$. or (c) the algorithm cannot derive a theory H that is within the hypothesis space defined by $\top$, as well as satisfies (3.10).*

First consider (a). Similar to that in the proof of Theorem 3.3.2, case (a)

Algorithm 3 $\top$-directed Theory co-Derivation ($\top$DTcD)

Input: All positive examples E^+ , background knowledge B, top theory $\top$ and h'
Output: $\mathcal{H} = \{H_i : B \wedge H_i \vdash_h E_i\}$, where $E_i \subset E^+$, $|E_i| > 1$ and $h \leq h'$

```
Let H = ∅
e_comp = Aggr(E+) %Aggregate all positive examples E+ into a compound literal e_comp
R = {R_i : R_i = Refs(e_comp, B, ⊤, h)} %Find all the refutations that satisfy the formula 3.11

for all R_i in R do
    D_i = DSeqs(R_i) %Obtain derivation sequences D_i by extracting ⊤ clauses from R_i.
    H_i = Trans(D_i) %Translate D_i into an hypothesis theory H_i
    H = H ∪ H_i
end for
return H
```

is refuted because: (1) the bound h' on the resolution steps guarantees that $\mathcal{R}$ is a finite set; (2) each operation within the for-loop terminates in finite time.

Secondly suppose (b) is true, but at step 3, a co-refutation of E_i using B and $\top$ can be found, which means $\forall e_j \in E_i, B \wedge \top \vdash_{h'} e_j$. Considering that the rest of the algorithm is the same as that in Algorithm1 and the correctness of Algorithm1 which is proved in Theorem 3.3.2, the hypothesis H derived will satisfy $\forall e_j \in E_i, B \wedge H \vdash_h e_j$, which contradicts the assumption and refutes (b).

Lastly consider (c), which implies that the step 3 cannot be executed either because (1) the corresponding $\top$ version of H from which it is translated cannot be used to prove E_i with B; or (2) the refutation of each e_j in E_i cannot be combined into a co-refutation. For case (1), similar to that in the proof of Theorem 3.3.2, (3.11) can be derived from the formulae (3.8) and (3.10). This means refutation using B and the $\top$ version of H does exist for the set of examples E_i that share the same structure of refutation proofs. The case (2) contradicts the fact that each $e_j \in E_i$ shares the same structure of refutation proofs so that their refutations can be combined, therefore completes the proof.

Example 2 *For all the positive examples in Figure 3.1, the $\top$DTcD method first combines them into a compound example $s([[an, unknown, alien, hits, the, house], [a, small, boy, walks, a, dog], [a, dog, walks, into, the, house]], [[], [], []])$. Then proves it using clauses in B and $\top$. In this way, we can derive the hypothesis $H_2 = \{h_4, h_5, h_7\}$ that co-generalises examples e_1 and e_2. Please note that H_2 does not cover e_3, since e_3 is filtered out in the refutation using the $\top$ version of H_2. As visualised in Figure 3.6, e_3 would be filtered out at the*

goal marked with a cross symbol, because the word 'dog' in e_3 is known to be a noun, rather than an adjective, thus it has to be filtered out in order to succeed the other part of the compound goal. Here we also give an example of the hypotheses that are pruned due to non-common generalisations: the hypothesis $H_1 = \{h_4, h_8, h_9\}$ derived when generalising e_1 alone is no longer derivable because apart from e_1 it cannot generalise either e_2 or e_3. Specifically, both e_2 and e_3 have their second words known as non-prepositions according to the given background knowledge, therefore they do not fit into the co-refutation using the $\top$ version of H_1.

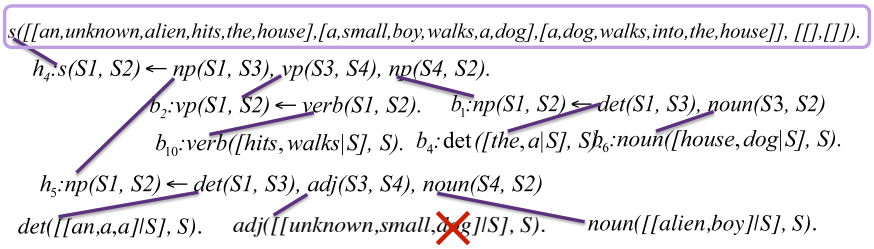


Figure 3.6.: Filter

3.3.4. Learning recursive concepts

Although $\top$DTcD requires its co-generalised examples to have the same structure of refutation proofs, it is still applicable to learning recursive theories. Since the refutations using the recursive theory have at least one recursive step in common, even though the lengths of refutation vary because of applying the recursive theory different times. Therefore, $\top$DTcD's solution for learning a recursive theory is: stop after finishing one recursive step and collect the unfinished subgoals as secondary examples, rather than proving a compound goal to the end where empty is derived.

The concept of secondary examples introduced here is similar to that in IMPARO [KBR09], because they both refer to the unexplained goals. However, the search of refutations continues for secondary examples in IMPARO, while here the search stops at the secondary examples, which cuts down the search space, thus improves the efficiency. After the search terminates at secondary examples, those secondary examples are collected as uncover examples and they are used together with the original training examples to

compute the compression of a corresponding hypothesis. Secondary examples correspond to only recursive subgoals in MC-TopLog, while they can be any subgoals in IMPARO. It is the assumption of learning a recursive theory[3] that makes it possible to stop searching at the secondary examples without scarifying the completeness of MC-TopLog.

Example

Here we use the odd-even example given in [Yam97] to exemplify how to apply ⊤DTcD to learn a recursive theory. The background knowledge B and the positive examples E^+ are in Fig 3.7, while the top theory $\top$ is in Fig 3.8. Note that the integers are represented by Peano numbers.

$$B = \begin{cases} b1 : even(0). \\ b2 : even(s(X)) \leftarrow odd(X). \end{cases} \quad E^+ = \begin{cases} e_1 = odd(s(s(s(0)))). \\ e_2 = odd(s(s(s(s(s(0))))))). \end{cases}$$

Figure 3.7.: Yamamoto's odd-even example

$T_1 : odd(Xs) \leftarrow \$bindF(Xs, Ys), \$body(Ys)$
$T_2 : \$body(Xs) \leftarrow even(Xs), \$bindF(Xs, Ys), \$body(Ys)$
$T_3 : even(Xs) \leftarrow \$bindF(Xs, Ys), \$body(Ys)$
$T_4 : \$body(Xs) \leftarrow odd(Xs), \$bindF(Xs, Ys), \$body(Ys)$
$T_{nt0} : \$body(Xs).$

$T_{nt1} : \$bindF([0,0],[0,0]).$
$T_{nt2} : \$bindF(Xs, Xs).$
$T_{nt3} : \$bindF([s(X_1), s(X_2)], Ys) \leftarrow \$bindF([X_1, X_2], Ys)$

Figure 3.8.: A top theory for odd-even example[Yam97]

At the first step of ⊤DTcD, the compound goal is proved using clauses in $\top$ and B. The refutation in Fig 3.9 is one of those found proofs. Note that this refutation stops when it encounters the subgoal with predicate *odd*, which means one recursive step has finished. In Fig 3.9, $odd(s(0))$ is collected as secondary example, while $odd(s(s(s(0))))$ is the original example e_1. The hypothesis derived from this co-refutation is $H_1 = \{odd(s(X)) \leftarrow even(X)\}$, which corresponds to the SLD-derivation sequence $[T_1, T_{nt3}, T_{nt2}, T_2, T_{nt2}, T_{nt0}]$.

[3]Knowing a target hypothesis is a recursive theory, we can assume the secondary examples collected after one recursive step can be proved by the hypothesised theory learned in the previous recursive step.

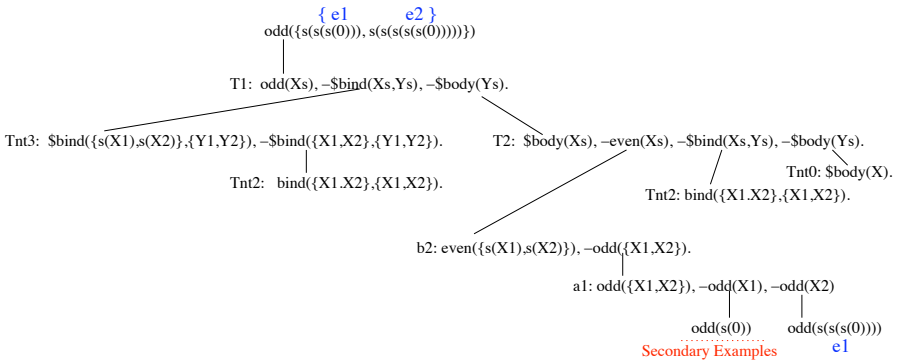


Figure 3.9.: A Co-Refutation of One Recursive Step

The co-refutation stops after one recursive step. Considering $odd(s(0))$ is not in the training examples, it is collected as part of the secondary examples, which will be used to compute the compression of H_1.

Without co-generalisation, there are 2412 candidate hypotheses that can be solo-generalised from e_2. In contrast, the size of search space decreased to 35 candidate hypotheses when e_2 are co-generalised with e_1. This shows the search space is dramatically decreased when restricting to common generalisations. The pruned hypotheses are those explain only one of e_1 and e_2. For example, the hypothesis $H_2 = \{odd(s(s(s(s(s(X)))))) \leftarrow even(X)\}$, which explains only e_2 but not e_1, is no longer derivable.

3.4. Experiments

The null hypotheses to be empirically investigated in the study are as follows. MC-TopLog and Progol5 [MB00] are the two ILP systems used in this experiment. All materials can be found at http://ilp.doc.ic.ac.uk/mcTopLog.

Null Hypothesis 1 A multi-clause learning method does not have higher predictive accuracies than a single-clause learning method.

Null Hypothesis 2 The search space of ⊤DTcD is not smaller than that of ⊤DTD.

```
s(S1,S2) :- np(S1,S3), vp(S3,S4), np(S4,S2).
s(S1,S2) :- np(S1,S3), vp(S3,S4), np(S4,S5), prep(S5,S6), np(S6,S2).
np(S1,S2) :- det(S1,S3), noun(S3,S2).
np(S1,S2) :- det(S1,S3), adj(S3,S4), noun(S4,S2).
vp(S1,S2) :- verb(S1,S2).
vp(S1,S2) :- verb(S1,S3), prep(S3,S2).
det([a|S],S).      det([the|S],S).
adj([big|S],S).    adj([small|S],S).    adj([nasty|S],S).
noun([man|S],S).   noun([dog|S],S).     noun([house|S],S).    noun([ball|S],S).
verb([takes|S],S). verb([walks|S],S).   verb([hits|S],S).
prep([at|S],S).    prep([to|S],S).      prep([on|S],S).       prep([in|S],S).     prep([into|S],S).
```

Figure 3.10.: A complete grammar

3.4.1. Experiment 1 - Grammar learning

Materials A complete grammar is given in Figure 3.10. The background knowledge B for each learning task is generated by randomly removing certain number of clauses from the complete theory, and those left-out clauses form the corresponding target hypothesis. The top theory is as that in Figure 3.3(c). Part of the examples are in Figure 3.11. There are 50 in total and half of them are negative. Therefore the default accuracy is 50%.

```
s([the,dog,takes,the,ball,to,the,house],[]).      ¬s([the, dog],[]).
s([the,small,dog,walks,on,the,house],[]).          ¬s([dog,the,man,the,walks],[]).
s([a,ball,hits,the,dog],[]).                      ¬s([ball,a,dog,a,hits],[]).
```

Figure 3.11.: Part of the Training Examples for Grammar Learning

Methods The null hypothesis 1 was investigated by comparing the learning results of MC-TopLog and Progol5 [MB00] for randomly chosen samples. For each size of leave-out on the complete theory, we sampled ten times and the predictive accuracies results of ten samples were averaged. The predictive accuracies were measured by leave-one-out cross validation. The null hypothesis 2 was examined by comparing the sizes of search space and running time of $\top$DTD and $\top$DTcD. The search space is measured by the number of candidate hypotheses generated during learning.

Results The x-axis in Figure 3.12(a) corresponds to the percentage of clauses remaining in the background knowledge. The smaller the percentage, the more clauses are left-out and to be learned. The line marked with 'before' represents the predictive accuracies before learning, which shows the degree of incompleteness in the background knowledge. Progol's predictive accuracy line is above the 'before learning' line, which shows the effectiveness in learning. However, when the percentage of remaining clauses decreases to half, Progol fails to reconstruct the multiple missing clauses due to its

single-clause limitation, therefore its accuracy drops to default. In contrast, MC-TopLog's ability of deriving multi-clause hypotheses makes it possible to hypothesise the missing clauses or their approximations even when half of the background knowledge is left-out. Therefore in Figure 3.12(a) MC-TopLog's predictive accuracies are always higher than that of Progol, and their difference increases as the background knowledge becomes more incomplete. Thus the null hypothesis (a) is refuted. Such results show that there are multi-clause learning tasks where Progol no longer performs as well as that in learning food webs [BCLM$^+$11], where multiple clauses are required for encoding the eating relations among various animals.

There is no significant difference between ⊤DTD and ⊤DTcD in terms of predictive accuracies. Therefore the accuracy lines of ⊤DTD and ⊤DTcD overlap in Figure 3.12(a). Figure 3.12(b) shows that the search space is reduced dramatically when the learning method switches from ⊤DTD to ⊤DTcD, thus the null hypothesis (b) is refuted. Additionally, the search space of ⊤DTD grow exponentially with more missing clauses; while ⊤DTcD's search space grows logarithmically because it is bound to common generalisations. The averaged running time plotted in Figure 3.12(c) shows similar pattern to that of Figure 3.12(b), which further confirms the improvement of ⊤DTcD over ⊤DTD in terms of efficiency.

3.4.2. Experiment 2 - Learning game strategies

Materials For this experiment, we choose the learning of game strategies for Nim [MX11], because the target hypothesis not only contains recursion, but also involves non-observable predicate learning. The rules of Nim game are: two players take turns to remove objects from distinct heaps; on each turn, a player can take any number of objects, provided at least one object is taken and they all come from the same heap. The learning task is to generalise a theory for identifying a P-position, which is a position that players are guaranteed to win if continue to play optimally, that is, identifying the precondition for grasping the winning strategy. Although Progol can suggest a single-clause hypothesis like the first clause in Figure 3.15(a) [MX11], this is not the target hypothesis unless the number of heaps is fixed to be three. To handle a more general case where the number of heaps is not fixed, that hypothesis is too specific and needs to be

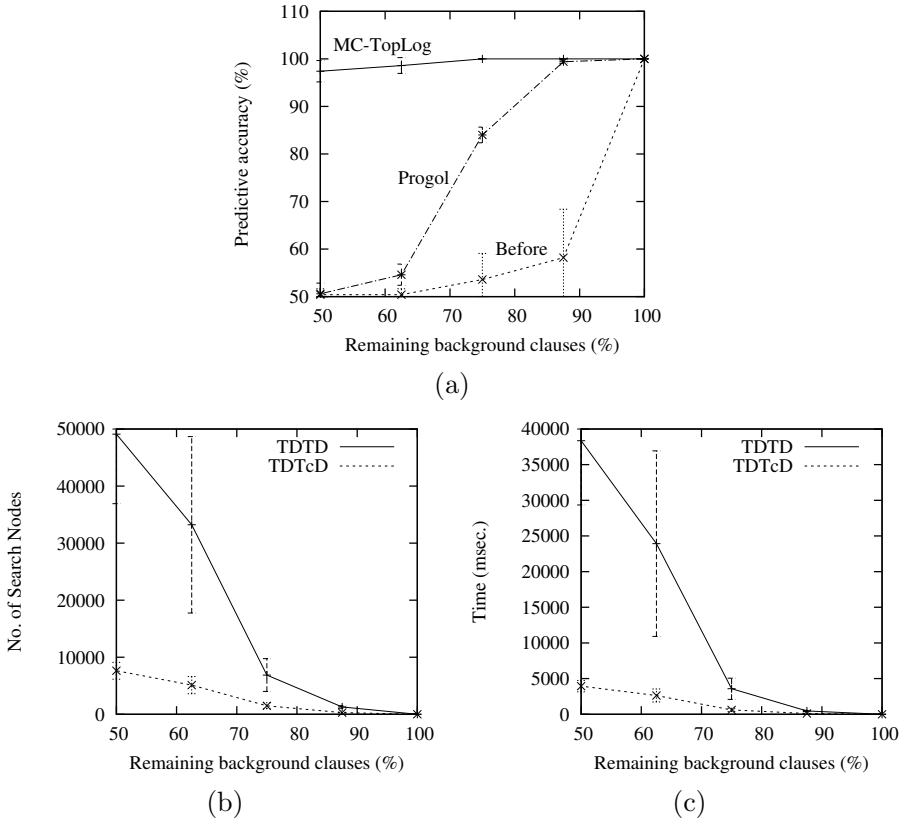


Figure 3.12.: Average (a) predictive accuracies, (b) sizes of search spaces and (c) running time for learning grammars

further generalised. The background knowledge available for this learning task includes the definition of mathematical functions like *and*, *or* and *xor*. The training examples are in the form of $play([3,4,5])$, in which the number sequence records the number of sticks in each heap. The declarative bias given to Progol and MC-TopLog are as in Figure 3.13. The top theory in Figure 3.13(b) encodes a strong declarative bias, which restricts the hypothesis clause about $compute/3$ to be recursive, while the same information is not encodable by a mode declaration.

Methods Similar to the experiment of grammar learning, the null hypothesis 1 was investigated by comparing the learning results of MC-TopLog and Progol5. However, different from the previous experiment, the background knowledge is fixed, since its size is too small to be randomly sampled. The accuracy curves in Figure 3.16(a) are drawn with the number of exam-

$modeh(*, play(+list)).$
$modeh(*, compute(+list)).$
$modeb(1, compute(+list)).$
$modeb(1, add(+int, +int, -int)).$
$modeb(1, minus(+int, +int, -int)).$
$modeb(1, multiply(+int, +int, int)).$
$modeb(1, xor(+int, +int, -int)).$
$modeb(1, and(+int, +int, -int)).$
$modeb(1, or(+int, +int, -int)).$
$modeb(1, equal(+int, int)).$
$modeb(1, mo(+int, int, int)).$

(a) Mode Declaration

$play([X|Xs]) \leftarrow compute(Xs, X, Result), \$body1(Result).$
$compute([X|Xs], R0, R) \leftarrow \$body([X, R0], [R1|\_]), compute(Xs, R1, R).$
$\$body(V0, V) \leftarrow bind(X, Y, V0), add(X, Y, Z), VUpdate(Z, V0, V1), \$body(V1, V).$
$\$body(V0, V) \leftarrow bind(X, Y, V0), minus(X, Y, Z), VUpdate(Z, V0, V1), \$body(V1, V).$
$\$body(V0, V) \leftarrow bind(X, Y, V0), multiply(X, Y, Z), VUpdate(Z, V0, V1), \$body(V1, V).$
$\$body(V0, V) \leftarrow bind(X, Y, V0), xor(X, Y, Z), VUpdate(Z, V0, V1), \$body(V1, V).$
$\$body(V0, V) \leftarrow bind(X, Y, V0), and(X, Y, Z), VUpdate(Z, V0, V1), \$body(V1, V).$
$\$body(V0, V) \leftarrow bind(X, Y, V0), or(X, Y, Z), VUpdate(Z, V0, V1), \$body(V1, V).$
$\$body1(Result) \leftarrow equal(Result, EqualResult).$
$\$body1(Result) \leftarrow mo(Result, Base, Result), equal(Result, ModResult).$
$\$body(Result, Result).$

(b) Top Theory

Figure 3.13.: Declarative bias for learning game strategies

$add(X, Y, Z) \leftarrow number(X), number(Y), Z = X + Y.$
$minus(X, Y, Z) \leftarrow number(X), number(Y), Z = X - Y.$
$multiply(X, Y, Z) \leftarrow number(X), number(Y), Z = X * Y.$
$and(X, Y, Z) \leftarrow number(X), number(Y), Z = X \wedge Y.$
$or(X, Y, Z) \leftarrow number(X), number(Y), Z = X \vee Y.$
$xor(X, Y, Z) \leftarrow number(X), number(Y), or(X, Y, Z1), and(X, Y, Z2), Z = Z1 - Z2.$
$mo(Num, Base, X) \leftarrow Num \neq 0, integer(Base), Base \neq 0, X = Num \bmod Base.$
$equal(X, X).$
$compute([], Result, Result).$

Figure 3.14.: Background knowledge for learning game strategies

ples on the x-axis. The null hypothesis 2 was examined by comparing the search spaces and running time of ⊤DTD and ⊤DTcD. Again, we varied the number of examples to see how the search space shrinks with more examples available to be co-generalised.

Results As shown in Figure 3.16(a), MC-TopLog only needs 6 examples to achieve accuracy of 100%, while Progol is not able to achieve accuracy of 100% even given 50 examples. Therefore the null hypothesis (a) is refuted. Progol's significantly lower accuracies results from its single-clause hypotheses which is too specific. For example, $\forall c_i \in H_s, H_m \models c_i$, where H_s and H_m are in Figure 3.15(a) and 3.15(b), respectively. H_m not only consists of a recursive clause, but also involves a non-observable predicate 'compute', therefore even methods that can learn recursive theories (e.g. [Mal03]) are

not able to derive H_m.

MC-TopLog's accuracy line in Figure 3.16(a) is derived under the learning mode of co-generalisation,while solo-generalisation is impractical for this learning task. Since there are so many mathematical functions which can be fit into a single example that the size of candidate hypotheses is much larger than what YAP (a Prolog interpreter) can handle. Therefore the null hypothesis (b) is refuted since Figure 3.16 shows that ⊤DTcD is applicable for this learning task where ⊤DTD fails. Figure 3.16(b) also shows that the power of co-generalisation is more effective with more examples. As can be seen from Figure 3.16(b), the number of‘search nodes decreases dramatically with increasing number of examples. This is consistent with the fact that the common part of different sets shrinks as the number of sets increases. In terms of running time, it decreases accordingly with the decreasing search space, as shown in Figure 3.16(c). However, the running time increases slightly after the number of examples increases to 20. This is due to the counteracting effect of binding more variables.

3.5. Discussions

⊤DHD in TopLog is entailment-incomplete, as pointed out in Chapter 2. To overcome this limitation, ⊤DHD is extended to ⊤DTD. Thus the system resulting from ⊤DTD is named MC-TopLog (Multi-clause TopLog). In order to improve the efficiency and scalability of ⊤DTD, the idea of common generalisation is introduced into ⊤DTD, which leads to ⊤DTcD. In comparison to TAL [CRL10], another ⊤-directed and entailment-complete ILP system mentioned in Chapter 2, MC-TopLog further explores the advan-

$play([N_1, N_2, N_3]) \leftarrow$
$\quad xor(N_1, N_2, N_3).$
$play([N_1, N_2, N_3, N_4]) \leftarrow$
$\quad xor(N_1, N_2, MidResult),$
$\quad xor(MidResult, N_3, N_4).$

(a) H_s by Progol

$play(Ns) \leftarrow compute(Ns, 0, 0).$

$compute([N|Ns], Resultsofar, Result) \leftarrow$
$\quad xor(N, Resultsofar, NewResultsofar),$
$\quad compute(Ns, NewResultsofar, Result).$

(b) H_m by MC-TopLog. The arguments of compute/3 mean: the list of numbers to be computed, the computed result sofar and the final computed result, respectively

Figure 3.15.: Hypotheses suggested by different ILP systems

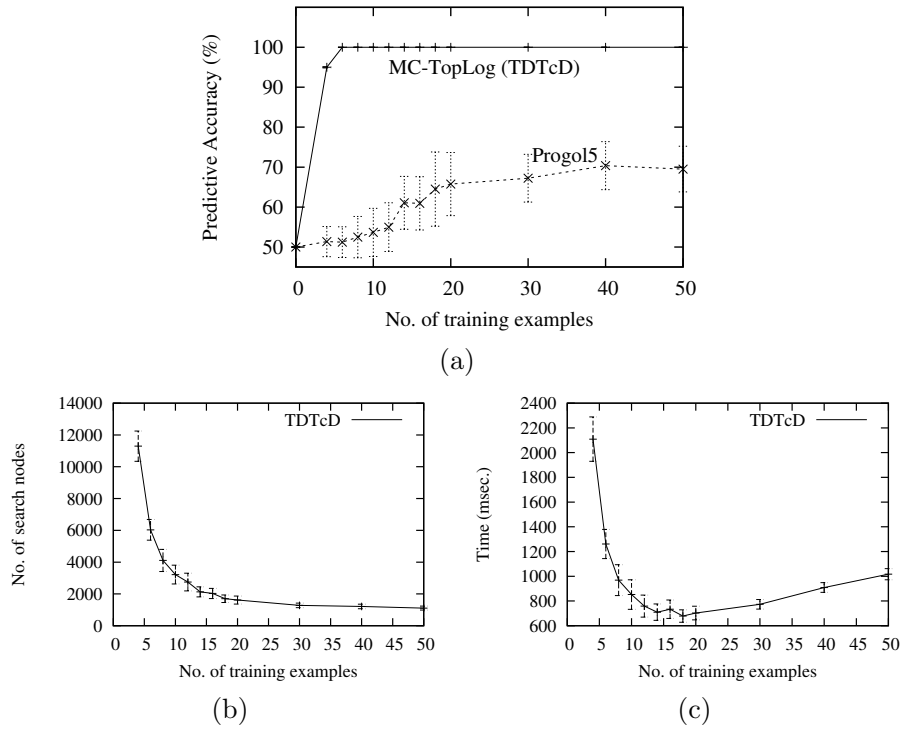


Figure 3.16.: Average (a) predictive accuracies, (b) sizes of search spaces and (c) running time for learning game strategies

tages of using logic programs as declarative bias. Specifically, this chapter explores the following two issues that were not considered in TAL: (1) the possibility of using a top theory to encode strong declarative bias; (2) the possibility of directly constraint a hypothesis space to common generalisations using a $\top$-directed method. In contrast, TAL uses a single example as a seed. It has not been extended to multiple examples so that the search space can be directly constrained to common generalisations.

3.6. Summary

The simplified version of grammar learning shows the importance of having an entailment-complete method, even for learning problems without recursion and mutually dependent predicates. Both $\top$DTD and $\top$DTcD are sound and complete for deriving hypotheses, but $\top$DTcD is more efficient

than ⊤DTD, while the improvement in efficiency does not come at the cost of lower predictive accuracy.

The experiments in this chapter demonstrate the advantage of multi-clause learning over single-clause learning in the case when a target hypothesis and its approximations are beyond the hypothesis space of a single-clause learner. In real-world applications where a target hypothesis is unknown, does multi-clause learning still help? We shall see the answer to this question in the next chapter.

4. Real-world Applications of MC-TopLog

This chapter is based on the paper [LCW+11], which aims to study whether multi-clause learning helps in real-world applications.

4.1. Introduction

4.1.1. Relationship between completeness and accuracy

It might be imagined that by achieving completeness of search, a learning algorithm necessarily increases the accuracy of prediction on unseen examples. However, the Blumer bound [BEHW89] indicates this is not necessarily the case, as explained in Section 2.1.2 of Chapter 2. Therefore on the face of it, the Blumer bound indicates that incomplete learning algorithms have lower error bounds than complete ones, as well as shorter running time. However, the Blumer bound only holds if the target theory or its approximation is within the hypothesis space for both algorithms. In the case that both the target theory and its approximation are within the hypothesis space of the complete learner but not within the hypothesis space of the incomplete learner, the complete learner will have a lower error bound. For an artificial dataset, it is possible to decide whether the target theory is within the hypothesis space before learning. However, this is not the case for a real-world dataset, so one of the motivations for this chapter is to see whether completeness in learning does lead to higher accuracy in at least one real-world dataset.

4.1.2. Experimental comparisons between SCL and MCL

Within ILP much effort has been put into designing methods that are complete for hypothesis finding. For example, ILP systems CF-Induction [Ino04b],

XHAIL [Ray09], TAL [CRL10] and MC-TopLog [MLTN11] were designed to overcome Progol's entailment-incompleteness. Indeed, different from SCL that restricts its hypothesis spaces to single-clause hypotheses, MCL is able to suggest multi-clause hypotheses, which are more compressive. The difference between single-clause and multi-clause hypotheses can be analogous to that between reductionist and systems hypotheses (see Section 4.3.1 for details). However, it is unclear whether systems hypotheses are definitely better than reductionist hypotheses, especially in real-world applications, while no direct comparison has been done before using real-world datasets[1]. At the same time, Progol's entailment-incompleteness does not stop it from being applied to real-world applications, because in certain cases, it is possible to construct a multi-clause hypothesis by sequentially adding single clauses. For example, a network of food webs, whose logical description consists of multiple clauses, can be constructed from scratch using Progol5 as shown in [TNBRM11] and [BCLM$^+$11]. Therefore, another motivation for this chapter is to use direct comparisons on the same datasets to demonstrate the necessity of having MCL, which is much more computationally expensive than SCL. We also analyse the cases when MCL does or does not improve the learning results of SCL.

4.1.3. Two biological applications

The two biological applications studied in this work are of commercial interest to Syngenta [syn], which is a leading agribusiness company providing crop protection and genetic solutions to growers. Developing tomato varieties optimised for shelf life, flavour and nutritional quality is a major part of Syngenta's breed selection and seed development programme. The aim of applying an ILP approach in this programme is to identify genetic control points regulating metabolic changes that occur during tomato fruit ripening. The other application about predictive toxicology is important to Syngentas crop protection initiatives. The objective is to identify control points for metabolic pathway perturbations caused by a liver tumour promoter (phenobarbital) in the rat. In both applications, the predictive

[1] Although [ISI$^+$09] has compared CF-Induction to Progol, no predictive accuracies are provided, but only learned hypotheses ranked by a probability measure. Although Progol's hypothesis is only ranked at 13th, it does not mean it has lower predictive accuracy than the one ranked at the top.

models derived would potentially influence the experimental design, thus saving time, experimental cost and labour involved with cycles of trial runs.

Why ILP?

For centuries scientists have used telescopes and microscopes to enhance their natural abilities to perceive the world. In an analogous way ILP can be used to enhance the abilities of scientists to reason about complex datasets. The biological applications to which ILP systems are applied in this work are typical of situations in which biologists have limited comprehension of the impact of perturbing a cellular pathway. The scale of the metabolic network and the interconnections among various pathways add another challenge to overcome. For example, during the tomato ripening, the genes that control the texture may also indirectly affect the flavour. It would be undesirable to sacrifice the taste of tomato to its firmness, although the firmness improves the shelf life. Therefore, all pathways related to flavour, texture and colour have to be considered together, which is difficult for biologists to conceptualise. Biologists therefore need a testable hypothesis suggested by an ILP system in order to carry out their studies. This is where ILP comes to their aid.

ILP has the advantage of suggesting readily comprehensible hypotheses, due to the use of logic programs as a uniform representation for B, E and H. Biologists can then examine the hypotheses using their existing knowledge. Those plausible hypotheses that are impossible to disprove can be considered for further experimental validation, while a biologically non-meaningful hypothesis may indicate that insufficient background knowledge has been provided. Being a knowledge discovery task it is often difficult to know a priori the depth of the knowledge required to circumvent such non-meaningful hypotheses. For example, in the predictive toxicology application, there are candidate hypotheses that explain the decrease of glucose and fructose from the reactions that produce them. However, in the given environment where glucose and fructose are provided externally, a decrease in glucose and fructose can only be explained by the reactions consuming them. Therefore, we updated the background knowledge with this knowledge as an integrity constraint to filter non-meaningful hypotheses. No matter whether a suggested hypothesis is disproved by biologists' existing knowledge or further tested

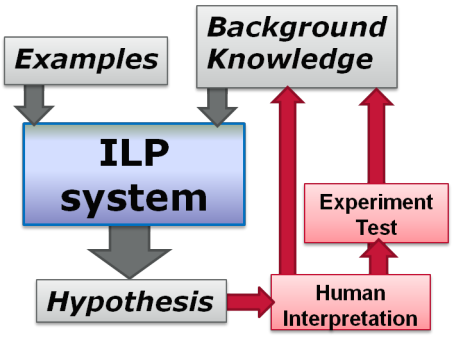


Figure 4.1.: Learning Cycle in ILP

by experiments, the background knowledge needs to be updated. ILP techniques make such a learning cycle feasible in a controlled manner. The diagram in Figure 4.1 not only shows such a learning cycle, but also highlights the fact that an ILP system does need scientists' help in providing/updating its input and interpreting its output. This supports our analogy of ILP technique as tools, which enhance scientists' capacity, rather than making scientists redundant.

4.1.4. Why these two applications?

The reason we chose these two applications to study the question that 'does multi-clause learning help in real-world applications?' is that they could potentially benefit significantly from multi-clause learning. Firstly, the background knowledge is highly incomplete, since none of the reaction states are known beforehand in the two applications. Secondly, the explanations for each example inevitably involve multiple reaction states, which will be explained later in Section 4.3. The same applications were also used in [MCW+10] to study how varying the background knowledge affects the accuracy, but the modelling has been extended by the more effective usage of transcriptomic data.

4.2. ILP models

4.2.1. Examples

The aim in both applications is to hypothesise the changes in reaction states, which reflect the genetic control of reactions. Although reaction states are not observable, they affect the flux through reactions, which leads to changes in metabolic abundance. Therefore, we can hypothesise the changes in reactions states through the changes in metabolic abundances that are observable. Accordingly, changes in metabolic abundance are used as examples E for learning. By comparing the treated group to the control group, three possible changes (i.e. *up*, *down* and *no-change*) in metabolic abundance can be observed. In the tomato application, the treated groups are obtained by knocking out specific genes related to the tomato ripening process, which results in ripening mutants, such as colourless non-ripening (CNR), ripening-inhibitor (RIN) and non-ripening (NOR); in the predictive toxicology application, the treated groups are Fischer F344 rats treated with different doses of phenobarbital.

4.2.2. Hypothesis space

The hypotheses are ground facts about reaction states. A reaction state can be *substrate limiting* or *enzyme limiting*. Substrate limiting means that the flux through a reaction is determined by the abundance of its substrates; while enzyme limiting implies that the flux through a reaction is controlled by the activity of its catalysing enzymes. Depending on the activity of catalysing enzymes, enzyme limiting can be further divided into three states: *catalytically increased*, *catalytically decreased* and *catalytically no-change*. These three states refer to the relative changes in the treated group against the control group, therefore they are not exactly the same as being activated or inhibited. For example, a relatively decreased reaction state does not necessarily mean inhibited.

An enzyme limiting reaction is assumed to be under genetic regulation, while a substrate limiting reaction is not, and its flux is affected by the nearby enzyme limiting reactions. Therefore, an hypothesis h_e about enzyme limiting contains more information than an hypothesis h_s about substrate limiting. Thus the description length for different hypotheses is dif-

ferent. Specifically, if h_s is encoded by L bit, then $k * L$ bits are required for h_e, where $k > 1$. Considering each metabolite's abundance is controlled by one regulatory reaction, each example is also encoded by L bits to make compression achievable. The difference in the description length can also be explained by Information Theory as follows. There are fewer reactions regulated by genes directly than indirectly, therefore the more frequent h_s is encoded using shorter description length than h_e to achieve minimum description length.

4.2.3. Background knowledge

Regulation rules

Figure 4.2 lists the seven regulation rules suggested by biologists. These rules tell how changes in reaction states affect metabolic abundances. For example, if a reaction is catalytically increased, which means the flux through that reaction increases, then the concentration of its product goes up, while its substrate's concentration goes down because of more rapid consumption. These are encoded as b_1 and b_2 in Figure 4.2. The rules b_1 to b_6 are all about enzyme limiting, and they are non-recursive, because the change in the substrate concentration will not affect the flux through the reaction but the enzyme activity itself. In contrast, the rule about substrate limiting (e.g. b_7) is recursive, because the substrate concentration would determine the flux through the reaction therefore affect the abundance of the product. These recursive rules essentially model the *indirect* effect of gene regulation.

These regulation rules seem to consider only one aspect, either enzyme limiting or substrate limiting, while in reality, both substrate abundances and enzyme activities may act together. However, it is unnecessary to consider the rules about the cumulative effect in our models, because the aim is to identify the dominating effect that is controlling the flux through a reaction, rather than knowing exactly what happens for each reaction. Similarly, as a node in a well-connected network, a metabolite's concentration is not just affected by one reaction's flux, but all reactions that consume or produce it. It seems the regulation rules should also capture this and consider how the fluxes from different reactions are balanced. However, no matter how fluxes from different branches are balanced, there is one branch whose effect dominates and leads to the final observed change. Therefore, the rules

in Figure 4.2 are sufficient for our simplified models.

b_1: $concentration(Metabolite, up, Time)$ ← produced_by(Metabolite,Reaction),
reaction_state(Reaction,enzymeLimiting,cataIncreased,Time).
b_2: $concentration(Metabolite, down, Time)$ ← consumed_by(Metabolite,Reaction),
reaction_state(Reaction,enzymeLimiting,cataIncreased,Time).
b_3: $concentration(Metabolite, down, Time)$ ← produced_by(Metabolite,Reaction),
reaction_state(Reaction,enzymeLimiting,cataDecreased,Time).
b_4: $concentration(Metabolite, up, Time)$ ← consumed_by(Metabolite,Reaction),
reaction_state(Reaction,enzymeLimiting,cataDecreased,Time).
b_5: $concentration(Metabolite, no\_change, Time)$ ← produced_by(Metabolite,Reaction),
reaction_state(Reaction,enzymeLimiting,cataNoChange,Time).
b_6: $concentration(Metabolite, no\_change, Time)$ ← consumed_by(Metabolite,Reaction),
reaction_state(Reaction,enzymeLimiting,cataNoChange,Time).
b_7: $concentration(Metabolite1, Change, Time)$ ←
produced_by(Metabolite1,Reaction), reaction_state(Reaction, substrateLimiting,_,Time),
consumed_by(Metabolite2,Reaction), concentration(Metabolite2,Change,Time).

Figure 4.2.: Regulation Rules

Metabolic networks

For the tomato application, the metabolic network is derived from the LycoCyc database [Lyc], which contains 1841 reactions, 1840 metabolites and 8726 enzymes. For the predictive toxicology application, it is obtained from the rat specific network in the KEGG database [OGS+99], which consists of 2334 reactions, 1366 metabolites and 1397 enzymes. In both applications, each reaction is considered as reversible. Therefore, the actual number of reactions N_r are doubled in the models. Since a subset of these reactions' states have to be hypothesised in order to explain the observed changes, the size of hypothesis spaces for the two applications are 2^{4N_r}, where the number 4 corresponds to the four possible reaction states (i.e. *substrate limiting*, *catalytically increased*, *catalytically decreased* and *catalytically no-change*).

Transcript profiles

Transcript profiles represent expression data for the genes encoding the enzymes. However, gene expression alone is not always indicative of the reaction states. This is due to the other cellular processes, such as post-translational modification that could change the activity of the enzyme. Therefore, instead of using transcription profiles as training examples, they were used as an integrity constraint in our model to filter hypotheses. Any hypotheses about enzyme limiting have to be consistent with their gene ex-

pression data. Specifically, if a reaction state is hypothesised to be catalytically increased, its expression data, if available, should be increased and vice versa. For example, without considering gene expression data, MC-TopLog would have to consider all the four candidate hypotheses shown in Figure 4.3. However, the hypotheses (b) and (c) have inconsistent reaction states (arrow color) with the change in the expression (colored squares), hence these two hypotheses will be filtered after applying the integrity constraint about gene expression.

Integrity constraint

Apart from the integrity constraint about gene expression, there is another constraint about reaction states: a reaction can not be in different states at the same time. Please note that, there is no constraint that a metabolite's concentration cannot be both up and down at the same time. As explained earlier, the model is about the dominated branch that leads to the final observation, while it is possible that different branches to the same metabolite have different contributions of fluxes.

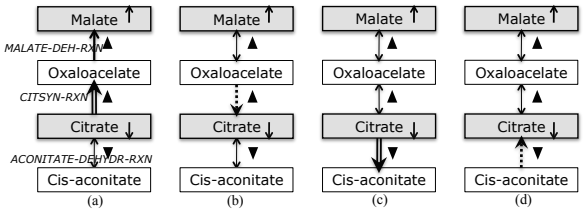


Figure 4.3.: Candidate Hypotheses for the decreased Citrate (Tomato Application). A reaction arrow is in double direction if its state is not hypothesised, otherwise it is not just in one direction, but also changed in the line style. The reaction states of substrate limiting, catalytically decreased and increased are respectively represented by thicker, dashed and double lines. Measured metabolites are highlighted in grey, and their corresponding values are annotated in their upper right corner. Gene expression levels are represented by the small triangles next to the reaction arrows. The upward and downward triangles mean increased and decreased.

4.3. Single-clause Learning *vs* Multi-clause Learning

Single-clause Learning (SCL) and Multi-clause Learning (MCL) has been defined in the previous chapter. This chapter compares SCL and MCL in the context of biological applications.

4.3.1. Reductionist vs. Systems hypothesis

SCL can only generalise an example to a single clause. Therefore its hypotheses are in the style of 'H_1 causes O_1, ... H_n causes O_n', where O_i represents an observation and each H_i is not necessarily related to the others. This kind of hypotheses can be referred to as reductionist hypotheses. In contrast, MCL is capable of generalising an example to a theory with multiple clauses so that its hypotheses are rich enough to be in the systems-level. MCL's hypotheses are in the style of 'H_1, H_2...H_j together cause O_1, O_2 ... O_i'. Table 4.1 summarises the differences between SCL and MCL.

Entailment-**Incomplete**	Entailment-**Complete**
a **single clause** per example	a **theory** per example
Constrained hypothesis space	**Less** constrained hypothesis space
Reductionist	**Systems**
H_1 causes O_1 ... H_n causes O_n	H_1, H_2...H_m together cause O_1, O_2 ... O_n

Table 4.1.: Single-clause Learning vs. Multi-clause Learning

4.3.2. SCL and MCL in the context of the two applications

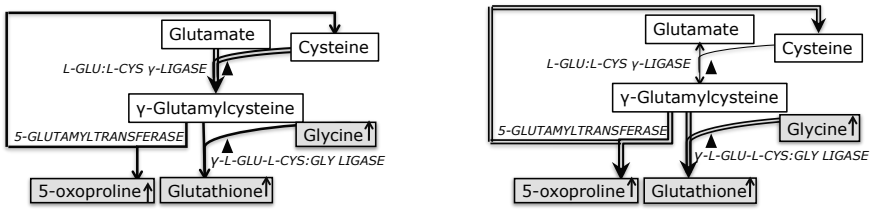
This subsection uses specific examples from the two applications to exemplify what has been discussed so far in this section. The two figures in Figure 4.4 are from the predictive toxicology application. They show two possible explanations for the increase in the abundances of glutathione and 5-oxoproline. Figure 4.4(a) says it is the reaction 'L-GLU:L-CYS γ-LIGASE' that is catalytically increased, which *indirectly* leads to the increase of glutathione and 5-oxoproline. In contrast, it is two different reactions whose activation that results in the increased glutathione and 5-oxoproline, as shown by the two double line arrows in Figure 4.4(b).

The explanation depicted in Figure 4.4(a) can be encoded by a logic program $H_{mc} = \{h_1, h_2, h_3\}$, where h_i is in Figure 4.5(a). Similarly, the

explanation in Figure 4.4(b) can be encoded as $H_{sc} = \{h_4, h_5\}$. Although both H_{mc} and H_{sc} consist of multiple clauses, H_{sc} is aggregated from two single-clause hypotheses: $H_{sc1} = \{h_5\}$ and $H_{sc2} = \{h_4\}$, which are respectively generalised from e_1 and e_2. In other words, each clause in H_{sc} is derived independently from different examples, and each alone is sufficient to explain an example. In contrast, H_{mc} comes from two multi-clause hypotheses: $H_{mc1} = \{h_1, h_3\}$ and $H_{mc2} = \{h_1, h_2\}$, which are also generalised from e_1 and e_2, respectively. However, none of the clauses in H_{mc} is able to explain any examples alone without the other clauses.

In the context of the two applications, single-clause learning means hypothesising a single reaction state for an example. This limitation restricts its derivable explanations to the reactions that directly connect to the observed metabolites. For example, the two double-line arrows in Figure 4.4(b) are connected directly to glutathione and 5-oxoproline, whose abundances are measurable. In contrast, a multi-clause learner is able to explore any possible regulatory reactions that are several reactions away from the observed metabolites. For example, the reaction arrow with double-line in Figure 4.4(a) is not directly connected to either glutathione or 5-oxoproline. However, the regulatory effect of this reaction is passed through the metabolite γ-glutamylcysteine, which is a common substrate of the two reactions 'γ-L-GLU-L-CYS:GLY LIGASE' and '5-GLUTAMYLTRANSFERASE'. The hypothesis H_{mc} in Figure 4.4(a) agrees with the one suggested by biologists [WCC+10], but it is not derivable by SCL.

In terms of compression, H_{mc} is more compressive than H_{sc}, according to the description length defined in the previous section. Intuitively, H_{mc} is more compact since it suggests a single control point for two observed



(a) Multi-clause hypotheses H_1

(b) Single-clause hypotheses H_2

Figure 4.4.: Explanations for the increase of Glutathione and 5-oxoproline

h_1: rs('γ-L-GLU-L-CYS:GLY LIGASE', substrateLimiting, _ , day14).
h_2: rs('5-GLUTAMYLTRANSFERASE', substrateLimiting, _ , day14).
h_3: rs('L-GLU:L-CYS γ-LIGASE', enzymeLimiting, cataIncreased, day14).
h_4: rs('5-GLUTAMYLTRANSFERASE', enzymeLimiting, cataIncreased, day14).
h_5: rs('L-GLU:L-CYS γ-LIGASE', enzymeLimiting, cataIncreased, dat14).

(a) Predictive Toxicology Application

h_6: rs('CITSYN-RXN', enzymeLimiting, cataIncreased, 'NOR_L').
h_7: rs('MALATE-DEH-RXN', substrateLimiting, _ , 'NOR_L').
h_8: rs('ACONITATE-DEHYDR-RXN', enzymeLimiting, cataDecreased, 'NOR_L').

(b) Tomato Application

Figure 4.5.: Candidate Hypothesis Clauses. The predicate 'rs' is short for 'reaction_state'.

metabolites, while H_{sc} involves two control points for the same number of observations. The higher compression of H_{mc} can also be explained by the fact that it is a systems-level description, thus more compact than the reductionist H_{sc}. Specifically, H_{mc} says it is the combination of h_1, h_2 and h_3 that leads to e_1 and e_2. In contrast, H_{sc} suggests that h_4 causes e_1 and h_5 causes e_2.

4.3.3. Reducing MCL to SCL

As mentioned earlier in the introduction, it is possible to construct a multi-clause hypothesis by sequentially adding single-clauses. The hypothesis H_{4a} drawn in Figure 4.3(a) gives such an example. H_{4a} consists of two clauses h_6 and h_7, which are in Figure 4.5(b). The single clause h_6 can be derived from the example of decreased Citrate. After h_6 is added to the background knowledge, another clause h_7 can be derived from the example of increased Malate. Despite the fact that H_{4a} can be sequentially constructed using Progol5, Progol5 does not necessarily suggest this hypothesis, but instead suggests H_{4d}=$\{h_8\}$ shown in Figure 4.3(d). Whether a MCL problem can be reduced to a SCL problem depends on the degree of incompleteness in the background knowledge and the distributions of given examples. For the two applications studied in this thesis, imagine an extreme case where all metabolite abundances are observable, then we can simply apply SCL to reconstruct each reaction state. However, not all metabolite abundances are measurable due to technological limitations.

4.4. Experiments

The two null hypotheses to be tested are: (1) MCL does not have higher predictive accuracies than SCL for any real-world datasets; (2) MCL always has higher predictive accuracies than SCL for all real-world datasets.

4.4.1. Materials

In the tomato application, transcript and metabolite profiles for three developmental stages (Early, Mid and Late) were obtained for wild type and three mutants (CNR, RIN, NOR) from Syngenta. This gave nine datasets in total (3 stages*3 mutants). In the cancer application, transcript and metabolite profiles were obtained for 1, 3, 7 and 14 days' post treatment, which were from a published study [WCC+10]. All the materials used in the experiments can be found at http://ilp.doc.ic.ac.uk/mcTopLog.

4.4.2. Methods

Progol5 [MB00] and MC-TopLog [MLTN11] were used to represent SCL and MCL respectively. Leave-one-out cross validation was used to compute the predictive accuracies. The closed world assumption applied during the testing phase was that "a reaction state is substrate limiting if it is not hypothesised". For the comparison of running time, we compared the number of search nodes instead, since Progol5 and MC-TopLog's running time are not comparable. Specifically, Progol5 was implemented in C, while MC-TopLog used Prolog and was executed using YAP. Since YAP is optimised towards efficiency, it is much faster, thus MC-TopLog's running time is even shorter than Progol5 despite of a much larger search space. For example, in the experiments, MC-TopLog takes maximum 10 mins for each run, while Progol 5 can take up to 3 hours.

4.4.3. Predictive accuracies

As shown in the tables below, there are two datasets (i.e. 'NOR_Mid' and 'NOR_Late') in the tomato application and one dataset (i.e. 'Day 3') in the predictive toxicology application, where MC-TopLog's accuracies are significantly higher than that of Progol5 at the 95% confidence level (i.e. p-value$\leq$ 0.05). While for the rest of the datasets, the two systems have

Timepoint	**default(no change)**,%	**Progol**,%	**MC-TopLog**,%	p-value
CNR_Early	63.64	86.36±7.32	81.82±8.22	0.576
CNR_Mid	36.36	86.36±7.32	86.36±7.32	1.000
CNR_Late	40.90	90.91±6.13	90.91±6.13	1.000
NOR_Early	86.36	86.36±7.32	86.36±7.32	1.000
NOR_Mid	50.00	**68.18**±9.93	**86.86**±7.32	**0.042**
NOR_Late	31.82	**68.18**±9.93	**86.36**±7.32	**0.042**
RIN_Early	100.00	100±0.00	100±0.00	1.000
RIN_Mid	90.91	90.91±6.13	90.91±6.13	1.000
RIN_Late	36.36	77.27±8.93	77.27±8.93	1.000

Table 4.2.: Predictive accuracies with standard errors in Tomato Application

Timepoint	**default(no change)**,%	**Progol**,%	**MC-TopLog**,%	p-value
Day 1	55.77	63.46±6.68	73.08±6.15	0.058
Day 3	30.77	**44.23**±6.89	**59.62**±6.80	**0.010**
Day 7	40.38	53.85±6.91	59.62±6.80	0.182
Day 14	48.08	61.54±6.75	63.46 ±6.67	0.569

Table 4.3.: Predictive accuracies with standard errors in Predictive Toxicology Application

the same or similar accuracies. Therefore both our null hypotheses are rejected by the accuracy results: (1) there is at least one dataset in both applications where MCL has significantly higher accuracy than SCL; (2) MCL does not outperform SCL all the time in terms of predictive accuracies. The explanation for such results will be given later after seeing a concrete example of the hypotheses derived by the two systems.

4.4.4. Hypothesis interpretation

This subsection exemplifies the different hypotheses suggested by Progol5 and MC-TopLog. The dataset used here is the abundances of six metabolites (Citrate, Malate, GABA, Alanine, Serine and Threonine) measured in the mutant 'CNR_Late' of the tomato application. MC-TopLog suggests a single control point to co-regulate the six metabolites. As shown in Figure 4.6(a), there is only one ground fact with enzyme limiting, while the rest are about substrate limiting, which are also indispensable in explaining the six observations together with the suggested control point. For the same set of observations, Progol suggests a reductionist hypothesis with six control points, since it hypotheses one control point for each metabolite. Thus all the ground facts in Figure 4.6(b) are about enzyme limiting.

rs(reversed-'GLYCINE-AMINOTRANSFERASE-RXN',enzymeLimiting,cataDecreased,'CNR_L').
rs('MALSYN-RXN',substrateLimiting,_,'CNR_L').
rs(reversed-'ALANINE–GLYOXYLATE-AMINOTRANSFERASE-RXN',substrateLimiting,_,'CNR_L').
rs(reversed-'GLYOHMETRANS-RXN',substrateLimiting,_,'CNR_L').
rs(reversed-'THREONINE-ALDOLASE-RXN',substrateLimiting,_,'CNR_L').
rs('GABATRANSAM-RXN',substrateLimiting,_,'CNR_L').
rs(reversed-'RXN-6902',substrateLimiting,_,'CNR_L').

(a) MC-TopLog's Hypothesis

rs('2.6.1.18-RXN',enzymeLimiting,cataIncreased,'CNR_L').
rs(reversed-'5.1.1.18-RXN',enzymeLimiting,cataDecreased,'CNR_L').
rs('THREDEHYD-RXN',enzymeLimiting,cataIncreased,'CNR_L').
rs(reversed-'ACONITATEDEHYDR-RXN',enzymeLimiting,cataDecreased,'CNR_L').
rs('GABATRANSAM-RXN',enzymeLimiting,cataIncreased,'CNR_L').
rs('1.1.1.39-RXN',enzymeLimiting,cataDecreased,'CNR_L').

(b) Progol's Hypothesis

Figure 4.6.: Hypothesis Comparison

.

Biological significance

Figure 4.7(a) visualises the hypothesis in Figure 4.6(a) suggested by MC-TopLog. It is the reaction 'GLYCINE-AMINOTRANS-RXN' that is suggested to be the control point for the six observations. This hypothesis is particularly interesting to biologists. Firstly, it is suggested in [FCS04] that the abundance of organic acids is controlled via TCA-Cycle, while this hypothesis indicates that the flux through the Malate can also be regulated by Glyoxylate shunt, independently of TCA cycle. Secondly, this hypothesis involves three intricately connected pathways (TCA-Cycle, Glyoxylate Shunt and GABA Shunt pathway), which is difficult for human beings to come up with. Different from the multi-clause hypothesis depicted in Figure 4.4(a) which has been confirmed by biologists [WCC$^+$10], no previous study is available to confirm the one in Figure 4.7(a), thus new biological experiments will be designed to test this hypothesis. Thirdly, this hypothesis could be of industrial interest since higher organic acid content in particular Malate is a commercially important quality trait for tomatoes [NNBC$^+$11].

4.4.5. Explanations for the accuracy results

The higher predictive accuracies by MC-TopLog in the three datasets can be explained by the fact that in those datasets neither target hypotheses nor their approximations are within the hypothesis space of Progol. Although the target hypotheses are unknown for the two real-world applications, the

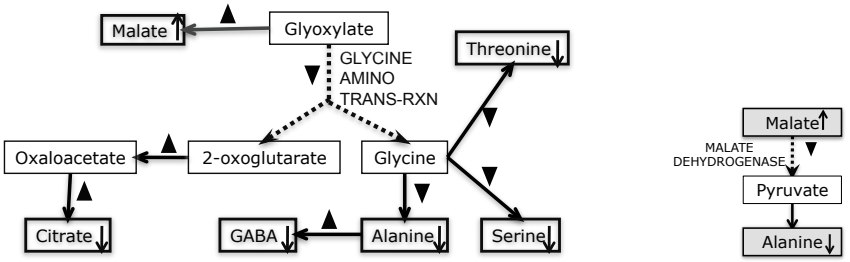


Figure 4.7.: MC-TopLog Hypotheses: (a) Three organic acids (Citrate, Malate, GABA) and three amino acids (Alanine, Serine and Threonine) are hypothesised to be controlled by the reaction 'GLYCINE-AMINOTRANS-RXN'. (b) Malate and Alanine are suggested to be controlled by the reaction catalysed by malate dehydrogenase.

hypotheses searched by Progol are less likely to be the targets. As mentioned before, Progol's hypotheses are not just reductionist, but also restricted to the reactions directly connected to the observed metabolites, so that they are usually specific to the example that they are generalised from. Such specific hypotheses may not be generalisable to the test data, thus they fail to predict the test data. In constrast, the multi-clause hypotheses suggested by MC-TopLog are not just in the systems-level, but also more compressive. For example, the multi-clause hypothesis in Figure 4.7(a) generalises six examples. When any of the six examples are left-out as test data, they can always be predicted by the hypothesis generalised from the remaining five examples. That is why MC-TopLog achieves higher accuracy for the three datasets.

On the other hand, it turns out that the systems hypotheses suggested by MC-TopLog does not always have higher predictive accuracies than the reductionist hypotheses suggested by Progol. That is because there do exist good approximations to the targets within the hypothesis space of Progol. Fig 4.7(b) shows such a good approximation, where a pair of metabolites are suggested to be co-regulated by Malate Dehydrogenase. This systems hypothesis is essentially derived by aggregating two reductionist hypotheses. Specifically, in Fig 4.7(b), the dash line denoting catalytically decrease is hypothesised from the increased Malate, while the solid line representing

substrate limiting is derived from the decreased Alanine. Although the number of co-regulated metabolites in Fig 4.7(b) is not as large as the one in Figure 4.7(a), it manages to predict one of the co-regulated metabolites when it is left-out as test data. There are other similar small co-regulated modules in Progol's hypothesis space, so that they together approximate the large module (Figure 4.7(a)) suggested by MC-TopLog. That is why in the dataset like 'CNR_Late' MC-TopLog does not outperform Progol5. In fact, the hypotheses with small co-regulated modules are not disprovable by existing knowledge. Additionally, there is no evidence that a control point regulating more metabolites is definitely better. Nevertheless, biologists tend to follow Occam's razor and prefer a more compressive hypothesis with fewer control points.

There is even one dataset 'CNR_Early' where Progol has a slightly higher accuracy than MC-TopLog. This is consistent with the Blumer bound argument, where it indicates that MC-TopLog is in the risk of overfitting when it searches within a much larger hypothesis space to find a high-compression hypothesis. In the context of the two applications, the high-compression hypotheses correspond to the control points that co-regulates as many metabolites as possible.

4.4.6. Search space and compression

Table 4.4 shows that MC-TopLog always has a larger search space than Progol5. This is consistent with the theoretical analysis discussed earlier. The larger search space makes it possible for MC-TopLog to find hypotheses with higher compression than Progol5. Indeed as shown in Table 4.4, hypotheses suggested by MC-TopLog always has higher compression than those suggested by Progol. In that table, the compression of an hypothesis H is defined as $N_p - N_n - DL$, where N_p and N_n are respectively the number of positive and negative examples covered by H, while DL is short for description length, which is defined in terms of the number of literals in H. As explained in Section 4.2.2, the DL of an hypothesis about substrate limiting and the one about enzyme limiting are respectively L and $k * L$. Here we choose $k = 10$ and $L = 1$, therefore a compression value of 10 in the Table 4.4 means only one example is compressed by H. Note that more compressive hypotheses does not necessarily correspond to higher accura-

cies, as you can see when lining up Table 4.4 with Table 4.2, This implies that a more complete search to find a more compressive hypothesis does not necessarily gain higher accuracies, which is consistent with the Blumer bound argument. However this does not mean that compression is not a good heuristic for search, but due to other problems like overfitting.

Timepoint	**Compression**		**Number of Search Nodes**	
	Progol	**MC-TopLog**	**Progol**	**MC-TopLog**
CNR_Early	0	49	352	1240
CNR_Mid	0	33	350	11890
CNR_Late	10	75	322	3654
NOR_Early	10	30	318	411
NOR_Mid	0	34	352	10851
NOR_Late	0	13	354	14032
RIN_Early	20	40	312	350
RIN_Mid	20	40	312	793
RIN_Late	0	14	354	14584

Table 4.4.: Comparing Compression and Search nodes (Tomato Application)

4.5. Discussions

There are other genome-wide metabolic modelling approaches, e.g. flux balance analysis [OTP10, FP08] and kinetic modelling [RPP09]. But ILP has several advantages, which can often be harder to obtain if using those approaches. Firstly, ILP does not require kinetic parameters for numerous metabolic reactions. Secondly, ILP can readily cope with incomplete data. ILP is still capable of hypothesising the reaction state if a gene annotation is not available or if a gene expression measurement for a catalysing enzyme is unavailable. Such hypotheses could not only locate the network gaps but potentially direct the network gap filling by suggesting candidate metabolic conversions. Thirdly, multi-clause ILP can generate an hypothesis with multiple clauses which describe the catalytic state of several neighbouring metabolic reactions. Hence, a multi-clause hypothesis carries substantial biological insight in terms of how the perturbations relating to neighbouring reactions affect a particular reaction and the concomitant change(s) in the participating metabolite(s).

Using abduction alone seems to be sufficient for the two applications considered in this chapter. Since the learning target is a reaction state that is not observable and could be simply a ground fact. However, an abductive system suggests all of the candidate hypotheses instead of the most promising ones. Although, an algorithm for ranking abductive hypotheses has already been proposed [ISI+09], it is not applicable to the current study due to the sheer number of candidate hypotheses generated[2]. Even if there are only 20 observations and for each observation only 10 candidate explanations are generated, the total number of candidate hypotheses[3] will be 10^{20}, which is far more than a billion. Hence, in this study we use covering algorithm to avoid the combinatorial explosion, as well as the concept of compression in ILP to select the most promising candidate hypotheses for further interpretation by biologists and experimental validation.

The use of transcriptomics together with metabolomics data in the modelling distinguishes the two applications from the previous biological application of ILP, e.g. MetaLog [TNCKM06]. This integrative omics approach is also different from the traditional approach used by biologists, where only transcriptomic data from treated groups and the control group is compared to find differentially expressed genes (control points). The integration of the metabolic data could take into account the effects due to the post-translational modification and protein-protein interactions that would otherwise not be captured by the differential gene expression alone.

4.6. Summary

This chapter demonstrates the necessity of multi-clause learning for real-world applications, despite the fact that multi-clause learning is computationally expensive. The next chapter will address the issue of predication invention, which has an even larger search space than multi-clause learning. The proposed solution comes from the extension of a top theory to a meta-interpreter, which is not just expressive enough to support predicate

[2]There are billions of candidate hypotheses, which exceeds the capacity of a Binary Decision Diagram (BDD), thus the algorithm in [ISI+09] is practically inapplicable here.

[3]The set of all candidate hypotheses is a cross product of the sets of candidate explanations that cover just one observation, since an hypothesis must cover all observations, instead of just one observation.

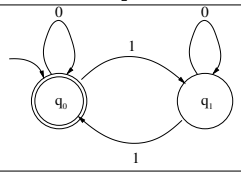
invention, but also capable of encoding procedural bias.

5. Meta-Interpretive Learning

This chapter introduces a new framework called Meta-Interpretive Learning (MIL), in which predicate invention is implemented using abduction with respect to a meta-interpreter. A meta-interpreter encodes higher-order meta-rules. Therefore it allows the introduction of new predicate symbols, thus supporting predicate invention. We consider applying such framework to grammar inference, as well as non-grammar learning tasks, such as learning a definition of staircases and robot strategies. In all these applications, learning recursion and predicate invention are required. This chapter is based on the papers published in [MLPTN13] and [ML13].

5.1. Introduction

a)

Finite acceptor	**Production rules**	**Definite Clause Grammar (DCG)**
	$q_0 \rightarrow$ $q_0 \rightarrow 0\ q_0$ $q_0 \rightarrow 1\ q_1$ $q_1 \rightarrow 0\ q_1$ $q_1 \rightarrow 1\ q_0$	$q_0([],[]) \leftarrow$ $q_0([0\|A],B) \leftarrow q_0(A,B)$ $q_0([1\|A],B) \leftarrow q_1(A,B)$ $q_1([0\|A],B) \leftarrow q_1(A,B)$ $q_1([1\|A],B) \leftarrow q_0(A,B)$

b)

E^+	E^-	**Meta-interpreter**	**Ground facts**
λ	1	$parse(S) \leftarrow parse(q0, S, []).$	
0	01		$acceptor(q0) \leftarrow$
00	10	$parse(Q, [], []) \leftarrow acceptor(Q).$	$delta1(q0, 0, q0) \leftarrow$
11	001	$parse(Q, [C\|X], Y) \leftarrow$	$delta1(q0, 1, q1) \leftarrow$
000	010	$delta1(Q, C, P),$	$delta1(q1, 0, q1) \leftarrow$
011	100	$parse(P, X, Y).$	$delta1(q1, 1, q0) \leftarrow$
101	111		

Figure 5.1.: a) Parity acceptor with associated production rules, DCG; b) positive examples (E^+) and negative examples (E^-), Meta-interpreter and ground facts representing the Parity grammar.

Consider the problem of using an ILP system to learn a Regular grammar which accepts all and only those binary sequences containing an even

number of 1s (see Figure 5.1). Since the 1950s automaton-based learning algorithms have existed [Moo56] which inductively infer Regular languages, such as *Parity*, from positive and negative examples. If we try to learn *Parity* using an ILP system the obvious representation of the target would be a Definite Clause Grammar (DCG) (see Figure 5.1(a)). However, if the ILP system were provided with examples for the predicate q_0 then the predicate q_1 would need to be invented since the only single state finite acceptor consistent with the examples would accept all finite strings consisting of 0s and 1s. It is widely accepted that Predicate Invention is a hard and under-explored topic within ILP [MRP$^+$11], and indeed state-of-the-art ILP systems, including MC-TopLog [MLTN11] and Progol [Mug95, MB00], are unable to learn grammars such as *Parity* in the form of a DCG using only first-order (non-metalogical) background knowledge since these systems do not support Predicate Invention. However, note that in Figure 5.1(a) each clause of the DCG has one of the following two forms.

$$\begin{aligned} Q([],[]) &\leftarrow \\ Q([C|x],y) &\leftarrow P(x,y) \end{aligned}$$

where Q, C, P are the only symbols which vary between the clauses. Figure 5.1(b) shows how these two forms of clauses above can be captured within the two clauses of a recursive meta-interpreter *parse/3* which uses the auxilliary predicates *acceptor/1* and *delta1/3*[1][2] to instantiate the predicate symbols and constants from the original DCG. The predicates *acceptor/1* and *delta1/3* can each be interpreted as Higher-Order Datalog [NP12] predicates since they take arguments which are predicate symbols q_0, q_1 from the DCG. By making *acceptor/1* and *delta1/3* abducible, *Parity*, and indeed any other Regular grammar, could in principle be learned from ground instances of *parse/1* using abduction. The chapter explores this form of learning with respect to a meta-interpreter.

We show that such abductively inferred grammars are a special case of Inverse Entailment. We also show that the hypothesis space forms a lattice

[1]Note that in the theory of automata [HU79] *delta1/3* corresponds to the transition function of the finite acceptor shown in Figure 5.1(a).

[2]Considering *delta1/3* as an arity 3 ground relation, if c, k are bounds on the number of terminals and non-terminals respectively then the number of possible definitions for *delta1/3* is 2^{ck^2}.

ordered by subsumption. The extensions of this use of abduction with respect to a meta-interpreter lead to a new class of inductive algorithm for learning grammars, as well as a broader class of Dyadic Datalog programs. The new approach blurs the normal distinctions between abductive and inductive techniques (see [FK00]). Usually abduction is thought of as providing an explanation in the form of a set of ground facts while induction provides an explanation in the form of a set of universally quantified rules. However, the meta-interpreter in Figure 5.1(b) can be viewed as projecting the universally quantified rules in Figure 5.1(a) onto the ground facts associated with *acceptor/1* and *delta1/3* in Figure 5.1(b). In this way abducing these ground facts with respect to a meta-interpreter is equivalent to induction, since it is trivial to map the ground *acceptor/1* and *delta1/3* facts back to the original universally quantified DCG rules.

In this chapter, we show that the MIL framework can be directly implemented using declarative techniques such as Prolog and Answer Set Programming (ASP). In this way, the search for an hypothesis in a learning task is delegated to the search engine in Prolog or ASP. Although existing abductive systems can achieve predicate invention if loaded with the meta-interpreter introduced in this chapter, a direct implementation of MIL has the following advantages.

1. As a declarative machine learning [Rae12] approach, it can make use of the advances in solvers. For example, techniques ASP solvers such as Clasp [GKNS07] compete favourably in international competitions. Recently Clasp has been extended to UnClasp [AKMS12] by replacing its original branch-and-bound algorithm with the approach of finding minimum unsatisfiable cores, which has highly efficiency for optimisation tasks. This advance is exploited in the experiments of this chapter, as we use UnClasp for our experiments.

2. As demonstrated by the experiments in this chapter, direct implementation of the approach using a meta-interpter has increased efficiency due to an ordered search in the case of Prolog and effective pruning in the case of ASP. While existing abductive systems like SOLAR [NIIR10], A-System [KND01] and MC-TopLog do not have an ordered search, but instead enumerate all hypotheses that entail the data.

3. The resulting hypotheses achieve higher predictive due to global optimisation, as opposed to the greedy covering algorithm used in many systems including MC-TopLog.

The structure of this chapter is as follows. It first describes the theoretical framework for MIL, and then gives details about its various implementations supporting different language classes. Experiments on predicate invention for grammar inference, learning a definition of a staircase and structuring robot strategies are given in Section 5.4. In Section 5.5 we discuss the differences between MIL and its related work.

5.2. Meta-Interpretive Learning

5.2.1. Framework

The higher-order *meta-rules* incorporated within the Prolog meta-interpreter has been defined earlier in Chapter 2.

In general, unification is known to be semi-decidable for higher-order logic [Hue75]. We now contrast the case for higher-order Datalog programs.

Proposition 1 (Decidable unification) *Given higher-order Datalog atoms* $A = P(s_1, .., s_m)$, $B = Q(t_1, .., t_n)$ *the existence of a unifying substitution* μ *is decidable.*
Proof. A, B *has unifying substitution* μ *iff* $p(P, s_1, .., s_m)\mu = p(Q, t_1, .., t_n)\mu$.

This construction is used to incorporate meta-rules within clauses of the Prolog meta-interpreter shown in Figure 5.1(b).

Definition 8 (Meta-rule incorporation) *The meta-rule* $\exists\sigma\forall\tau P(s_1, .., s_m) \leftarrow .., Q_i(t_1, .., t_n), ..$ *is incorporated in the Prolog meta-interpreter* M *iff* M *contains a clause* $Head \leftarrow Body$ *where* $Head = prove(P, s_1, .., s_m)$ *and* $Body$ *contains atoms like* $prove(Q_i, t_1, .., t_n)$, $meta_r(\sigma)$ *and* $type_t(v)$ *for each* $v \in \tau$. *The atoms* $meta_r(\sigma)$ *and* $type_t(v)$ *provide groundings for all variables within the meta-rule.*

The Meta-Interpretive Learning (MIL) setting is a variant of the normal setting for ILP.

Definition 9 (Meta-Interpretive Learning setting) *A Meta-Interpretive Learning (MIL) problem consists of* $Input = \langle B, E\rangle$ *and* $Output = H$ *where*

the background knowledge $B = B_M \cup B_A$. B_M is a definite logic program[3] *representing a meta-interperter and B_A and H are ground definite Higher-Order Datalog programs consisting of positive unit clauses. The predicate symbol constants in B_A and H are represented by Skolem constants. The examples are $E = \langle E^+, E^- \rangle$ where E^+ is a ground logic program consisting of positive unit clauses and E^- is a ground logic program consisting of negative unit clauses. The Input and Output are such that $B, H \models E^+$ and for all e^- in E^-, $B, H \not\models e^-$.*

Inverse Entailment can be applied to allow H to be derived from B and E^+ as follows.

$$\begin{aligned} B, H &\models E^+ \\ B, \neg E^+ &\models \neg H \end{aligned} \tag{5.1}$$

Since both H and E^+ can each be treated as conjunctions of ground atoms containing Skolem constants in place of existential variables, it follows that $\neg H$ and $\neg E^+$ are universally quantified denials where the variables come from replacing Skolem constants by unique variables. We now define the concept of a Meta-Interpretive learner.

Definition 10 (Meta-Interpretive learner) *Let $\mathcal{H}_{B,E}$ represent the complete set of hypotheses H for the MIL setting of Definition 9. Algorithm A is said to be a Meta-Interpretive learner iff for all B, E such that H is the output of Algorithm A given B and E as inputs, it is the case that $H \in \mathcal{H}_{B,E}$.*

Example 3 (Parity example) *Let $B = \langle B_M, B_A \rangle$, $E = \langle E^+, E^- \rangle$ and $H \in \mathcal{H}_{B,E}$ represents the parity grammar. Figure 5.2 shows H as a possible output of a Meta-Interpretive learner, since H holds for $B, H \models E^+$.*

Note that this example of abduction produces Predicate Invention by introducing Skolem constants representing new predicate symbols. By contrast an ILP system, such as Progol, uses Inverse Entailment [Mug95] to construct

[3] Note that the meta-interpreter shown in Figure 5.1(b) is a definite logic program. Such a meta-interpreter is only a part of implementations such as those described later in Section 5.3. Within such an implementation negation-by-failure is used to implement operations such as abduction, so the implementation as a whole is not a definite logic program. However, this does not affect this definition or the later propositions which use it.

E^+	$\neg E^+$	E^-
$parse([]) \leftarrow$	$\leftarrow parse([])$,	$\leftarrow parse([1])$
$parse([1,1]) \leftarrow$	$parse([1,1])$,	$\leftarrow parse([0,1])$
$parse([0,1,1]) \leftarrow$	$parse([0,1,1])$,	$\leftarrow parse([1,0])$
$parse([1,0,1]) \leftarrow$	$parse([1,0,1])$,	$\leftarrow parse([0,0,1])$
$parse([1,1,0]) \leftarrow$	$parse([1,1,0])$.	$\leftarrow parse([1,1,1])$

H	$\neg H$
$acceptor(\$0) \leftarrow$	$\leftarrow acceptor(Q0)$,
$delta1(\$0,0,\$0) \leftarrow$	$delta1(Q0,0,Q0)$,
$delta1(\$0,1,\$1) \leftarrow$	$delta1(Q0,1,Q1)$,
$delta1(\$1,0,\$1) \leftarrow$	$delta1(Q1,0,Q1)$,
$delta1(\$1,1,\$0) \leftarrow$	$delta1(Q1,1,Q0)$.

Figure 5.2.: Parity example where B_M is the Meta-interpreter shown in Figure 5.1(b), $B_A = \emptyset$ and E^+, $\neg E^+$, E^-, H, $\neg H$, are as shown above. '$0' and '$1' in H are Skolem constants replacing existentially quantified variables.

a single clause from a single example, while a Meta-Interpretive learner uses Inverse Entailment to construct the set of all clauses H as the abductive solution to a single goal $\neg E^+$ using E^- as integrity constraints. In the example, the output hypothesis H is a set of ground facts in the meta-form. They can be translated by post-processing to the first-order DCG shown in Figure 5.1(a), which contains both invented predicates and mutual recursion. Neither predicate invention nor mutual recursion can be achieved with DCGs in this way using ILP systems such as Progol or MC-TopLog.

5.2.2. Lattice properties of hypothesis space

In this section we investigate orderings over MIL hypotheses.

Definition 11 ($\succeq_{B,E}$ relation in MIL) *Within the MIL setting we say that $H \succeq_{B,E} H'$ in the case that $H, H' \in \mathcal{H}_{B,E}$ and $\neg H' \succeq_\theta \neg H$.*

We now show that $\succeq_{B,E}$ forms a quasi-ordering and a lattice.

Proposition 2 (Quasi-ordering) *Within the MIL setting $\langle \mathcal{H}_{B,E}, \succeq_{B,E} \rangle$ forms a quasi-ordering.*

Proof. *Follows from the fact that $\langle \{\neg H : H \in \mathcal{H}_{B,E}\}, \succeq_\theta \rangle$ forms a quasi-ordering since each $\neg H$ is a clause [NCdW97].*

Proposition 3 (Lattice) *Within the MIL setting $\langle \mathcal{H}_{B,E}, \succeq_{B,E} \rangle$ forms a lattice.*

Proof. *Follows from the fact that $\langle \{\neg H : H \in \mathcal{H}_{B,E}\}, \succeq_\theta \rangle$ forms a lattice since each $\neg H$ is a clause [NCdW97].*

We now show that this ordering has a unique top element.

Proposition 4 (Unique $\top$ element) *Within the MIL setting there exists $\top \in \mathcal{H}_{B,E}$ such that for all $H \in \mathcal{H}_{B,E}$ we have $\top \succeq_{B,E} H$ and $\top$ is unique up to renaming of Skolem constants.*

Proof. *Let $\neg H' = \bigvee_{H \in \mathcal{H}_{B,E}} \neg H$ and $\neg \top = \neg H' \theta_v$ where v is a variable and $\theta_v = \{u/v : u \text{ variable in } \neg H'\}$. By construction for each $H \in \mathcal{H}_{B,E}$ it follows that $\neg \top \succeq_\theta \neg H$ with subsitution θ_v. Therefore for all $H \in \mathcal{H}_{B,E}$ we have $\top \succeq_{B,E} H$ and $\top$ is unique up to renaming of Skolem constants.*

This proposition can be illustrated with a grammar example.

Example 4 (Subsumption example) *In terms of the Meta-interpreter of Figure 5.1(a) the universal grammar $\{0,1\}^*$ can be expressed using $\top = \{(acceptor(\$0) \leftarrow), (delta1(\$0, 0, \$0) \leftarrow), (delta1(\$0, 1, \$0) \leftarrow)\}$. Letting H represent the Parity grammar from Example 3 it is clear that $\neg H \succeq_\theta \neg \top$ and so $\top \succeq_{B,E} H$. So unlike the subsumption relation between universally quantified clauses, binding all the (existentially quantified) variables in H to each other produces a maximally general grammar $\top$.*

We now show the circumstances under which a unique bottom element of the lattice can be constructed using Plotkin's lgg algorithm.

Proposition 5 (Unique $\bot$ element) *In the case that $\mathcal{H}_{B,E}$ is finite up to renaming of Skolem constants there exists $\bot \in \mathcal{H}_{B,E}$ such that for all $H \in \mathcal{H}_{B,E}$ we have $H \succeq_{B,E} \bot$ and $\bot$ is unique up to renaming of Skolem constants.*

Proof. *Since $\mathcal{H}_{B,E}$ is finite $\neg \bot = lgg(\{\neg H : H \in \mathcal{H}_{B,E}\})$ where lgg is Plotkin's algorithm for computing the least general generalisation of a set of clauses under subsumption [Plo69].*

For most purposes the construction of the unique bottom clause is intractable since the cardinality of the lgg clause increases exponentially in the cardinality of $\mathcal{H}_{B,E}$. We now show a method for reducing hypotheses.

In the rest of this subsection we show the existence of a compact bottom hypothesis in the case of MIL for Regular languages. Algorithms for learning the Regular languages have been widely studied since the 1970s within the topic of Grammatical inference [dlH05]. Many of these start with a prefix tree acceptor (PTA). PTA is a tree-like automation. Its states correspond to the prefixes of the positive examples, thus it only accept the examples

from which it is constructed. The states in the prefix tree acceptor are then progressively merged to construct a more general automata.

Proposition 6 (Unique $\bot$ for Regular languages) *Prefix trees act as a compact bottom theory in the MIL setting for Regular languages.*
Proof. *Follows from the fact that all deterministic Regular gramamrs which include the positive examples can be formed by merging the arcs of a prefix tree acceptor [Mug90]. Merging the arcs of the prefix tree is achieved by unifying the delta1 atoms in $\neg H$ within the MIL setting.*

Example 5 (Prefix tree) *Assume the MIL setting with B_M being the meta-interpreter for Regular languages. Let $E^+ = \{parse([1,1]), parse([1,1,0])\}$ then by always introducing a new state when parsing the examples, the following hypothesis corresponding to the prefix tree automaton can be derived: $\bot = \{delta1(\$0, 1, \$1),\ delta1(\$1, 1, \$2),\ acceptor(\$2),\ delta1(\$2, 0, \$3),\ acceptor(\$3)\}$.*

5.2.3. Reduction of hypotheses

Proposition 7 (Logical reduction of hypotheses.) *Suppose H' is an hypothesis in the MIL setting and $\neg H$ is the result of applying Plotkin's clause reduction algorithm[4] [Plo69] to $\neg H'$. Then H is a reduced hypothesis equivalent to H'.*
Proof. *Follows from the fact that $\neg H'$ is θ-subsumption equivalent to $\neg H$ by construction.*

Example 6 (Reduction example) *Let $H' = H \cup \{r\}$ where H is the Parity grammar from Figure 5.2 and $r = (delta1(\$0, 0, \$2) \leftarrow)$ represents an additional redundant grammar rule. Now Plotkin's reduction algorithm would reduce $\neg H'$ to the equivalent clause $\neg H$ and consequently grammar H is a reduced equivalent form of H.*

5.2.4. Language classes and expressivity

The meta-interpreter in Figure 5.3(a) can parse a ground definite Higher-order Datalog program consisting of unit clauses about $delta1/3$ and $acceptor/1$.

[4]This algorithm iteratively remove logically redundant literals from a clause until no redundant clauses remain.

This program can represent a Regular Grammar. Figure 5.3(b) shows how the meta-interpreter for Regular Grammars can be extended to Context-Free Grammars. The Chomsky language types form an inclusion hierarchy in which Regular $\subseteq$ Context-Free. Figure 5.3(c) shows how MIL can be further extended to support a broader class of Dyadic Datalog programs (H_2^2).

Language type	**Meta-interpreter**	**Example Grammar**	**L**
a) R	$parse(S) \leftarrow parse(Q, S, []).$ $parse(Q, X, X) \leftarrow acceptor(Q).$ $parse(Q, [C\|X], Y) \leftarrow delta1(Q, C, P),$ $\quad parse(P, X, Y).$	$S \rightarrow 0\ S$ $S \rightarrow 1\ T$ $T \rightarrow \lambda$ $T \rightarrow 1\ T$	0^+1^+
b) CF	$parse(S) \leftarrow parse(Q, S, []).$ $parse(Q, X, X) \leftarrow acceptor(Q).$ $parse(Q, [C\|X], Y) \leftarrow delta1(Q, C, P),$ $\quad parse(P, X, Y).$ $parse(Q, X, Y) \leftarrow delta2(Q, P, C),$ $\quad parse(P, X, [C\|Y]).$ $parse(Q, X, Y) \leftarrow delta3(Q, P, R),$ $\quad parse(P, X, Z),$ $\quad parse(R, Z, Y).$	$S \rightarrow \lambda$ $S \rightarrow T\ S$ $T \rightarrow 0\ U$ $U \rightarrow T\ 1$	$(0^n1^n)^*$
c) H_2^2	$prove(P, X, Y) \leftarrow meta1(P, X, Y).$ $prove(P, X, Y) \leftarrow meta2(P, Q),$ $\quad prove(Q, X, Y).$ $prove(P, X, Y) \leftarrow meta3(P, Q, R),$ $\quad prove(Q, X, Z),$ $\quad prove(R, Z, Y).$	$parent \rightarrow mother$ $parent \rightarrow father$ $ancestor \rightarrow parent$ $ancestor \rightarrow parent,$ $\quad ancestor$	Logic programs

Figure 5.3.: Meta-interpreters, Chomsky-normal form grammars and languages for a) Regular (R) b) Context-Free (CF) and c) H_2^2 languages.

Definition 12 (H_j^i program class) *Assuming i, j are natural, the class H_j^i contains all higher-order definite Datalog programs constructed from signatures $\mathcal{P}, \mathcal{C}$ with predicates of arity at most i and at most j atoms in the body of each clause.*

The class of dyadic logic programs with one function symbol has Universal Turing Machine (UTM) expressivity [Tar77]. In this thesis, we consider Dyadic Datalog programs (H_2^2), which have no function symbol. This fragment also has UTM expressivity, as demonstrated by the following H_2^2 encoding of a UTM in which $S, S1, T$ represent Turing machine tapes.

utm(S,S) $\leftarrow$ halt(S).
utm(S,T) $\leftarrow$ execute(S,S1), utm(S1,T).
execute(S,T) $\leftarrow$ instruction(S,F), F(S,T).

Below assume G is a Datalog goal and program $P \in H_2^2$.

Proposition 8 (Undecidable fragment of H_2^2) *The satisfiability of G, P is* undecidable *when $\mathcal{C}$ is infinite.*
Proof. Follows from undecidability of halting of UTM above.

The situation differs in the case $\mathcal{C}$ is finite.

[Decidable fragment of H_2^2] The satisfiability of G, P is *decidable* when $\mathcal{P}, \mathcal{C}$ is finite.
Proof. Suppose that the satisfiability of G, P is undecidable. However, given $\mathcal{P}, \mathcal{C}$ is finite, the set of Herbrand interpretations is finite, which means that the satisfiability of G, P is *decidable*, which contradicts the assumption and completes the proof.

5.3. Implementations

In this section, we describe the implementations of Meta-Interpretive Learning (MIL) using two different declarative languages: Prolog and Answer Set Programming (ASP). The resulting systems are called Metagol[5] and ASP_M[6], respectively.

5.3.1. Implementation in Prolog

The systems Metagol_R, Metagol_{CF}, Metagol_{RCF} and Metagol_D are four simple Prolog implementations of MIL supporting different language classes. These implementations leverage the procedure bias encoded in a Prolog meta-interpreter, which results in an ordered search through hypothesis spaces. In contrast, if directly using an abductive system, there is no ordered search and all consistent hypotheses would be enumerated.

$\textbf{Metagol}_R$

Before introducing Metagol_R, we first explain its simplified version noMetagol_R (non-optimising Metagol_R) as shown in Figure 5.4. The system noMetagol_R is based on the following abductive variant of the Regular Meta-interpreter from Figure 5.3 (the standard definition of member/2 is omitted for brevity).

[5] *Metagol* is MIL encoded within YAP Prolog. The name comes from the combination of *Meta-* and *gol*, where *Meta-* corresponds to the Meta-Interpreter, and *gol* is the reverse of *log* which is short for logic.
[6] The name ASP_M is MIL encoded within an ASP solver.

```
parse(S,G1,G2) :- parse(s(0),S,[],G1,G2). % S is the sequence to be parsed,
%G1 and G2 correspond to the grammars before and after parsing the sequence S

parse(Q,X,X,G1,G2) :- abduce(acceptor(Q),G1,G2). % Q is a state.
parse(Q,[C|X],Y,G1,G2) :- skolem(P), abduce(delta1(Q,C,P),G1,G3), parse(P,X,Y,G3,G2).
% C is a terminal to be parsed, while X and Y corresponds to the input and output strings.

abduce(X,G,G) :- member(X,G). % X is a clause which has been abduced.
abduce(X,G,[X|G]) :- not(member(X,G)).
% X is a newly abduced clause and added to the grammar so far G

skolem(s(0)). skolem(s(1)). ...
```

Figure 5.4.: noMetagol$_R$

The abduced atoms are simply accumulated in the extra variables $G1, G2, G3$. The term $s(0)$ represents the start symbol and a finite set of Skolem constants is provided by the monadic predicate *Skolem*. Hypotheses are now the answer substitutions of a goal such as the following.

```
:-  parse([],[],G1), parse([0],G1,G2), parse([0,0],G2,G3), parse([1,1],G3,G4),    % Pos
    parse([0,0,0],G4,G5), parse([0,1,1],G5,G6), parse([1,0,1],G6,G),
    not(parse([1],G,G)), not(parse([0,1],G,G)).                                  % Neg
```

Note that each of the positive examples are provided sequentially within the goal and the resulting grammar is then tested for consistency, which includes integrity constraints and non-coverage on any negative examples. The final grammar returned in the variable G is a solution which covers all positives and none of the negatives. In the case shown above the first hypothesis found by Prolog is as follows.

```
G = [delta1(s(1),0,s(1)),delta1(s(1),1,s(0)),delta1(s(0),1,s(1)),
     delta1(s(0),0,s(0)),acceptor(s(0))]
```

This hypothesis correctly represents the Parity acceptor of Figure 5.1. All other consistent hypotheses can be generated by making Prolog backtrack through the SLD proof space.

Metagol$_R$ We will now explain the following procedural biases, which extends noMetagol$_R$ to Metagol$_R$.

Minimal cardinality hypothesis Occam's razor suggests to select the shortest hypothesis that fits the data. Therefore we introduce the clause bound into Metagol$_R$ so that the search starts from shorter hypotheses. In Metagol$_R$ (Figure 5.5) the variables K,$K1$,$K2$ and $K3$ are related to the clause bound. They are instantiated with Peano numbers $(s(0), s(1), ..)$ representing a bound on the maximum number of abduced clauses. Thus

```
parse(S,G1,G2,S1,S2,K1,K2) :- parse(s(0),S,[],G1,G2,S1,S2,K1,K2).

parse(Q,X,X,G1,G2,S,S,K1,K2) :- abduce(acceptor(Q),G1,G2,K1,K2).
parse(Q,[C|X],Y,G1,G2,S1,S2,K1,K2) :- skolem(P,S1,S3),
        abduce(delta1(Q,C,P),G1,G3,K3,K2), parse(P,X,Y,G3,G2,S3,S2,K3,K2).

abduce(X,G,G,K,K) :- member(X,G).
abduce(X,G,[X|G],s(K),K) :- not(member(X,G)).

skolem(s(N),[s(Pre)|SkolemConsts],[s(N),s(Pre)|SkolemConsts]):- N is Pre+1.
skolem(S,SkolemConsts,SkolemConsts):-member(S,SkolemConsts).
```

Figure 5.5.: Metagol$_R$

the second clause of *abduce*/5 fails once $K1$ has a value of 0. $K1$ is iteratively increased until an hypothesis is found within that bound. The search thus guarantees finding an hypothesis with minimal description length. In Metagol, the preference of minimal cardinality hypothesis is given higher priority than other procedure bias, such as specific-to-general search explained in the next paragraph.

Specific-to-General Within the MIL setting an hypothesis H_s is said to be more specific than H_g in the case that $\neg H_s \succeq_\theta \neg H_g$, as explained in Section 5.2.2. Therefore H_s is a refinement of H_g by renaming with new Skolem constants. In Metagol$_R$ the Skolem constants are enumerated by the program of $Skolem/3$. The first clause of $Skolem/3$ introduces a new Skolem constant, while the second clause of $Skolem/3$ provides a Skolem constant that has already been used in the deriving hypothesis. Due to Prolog's procedural semantics, the first clause of $Skolem/3$ will be tried before the second one, thus H_s, that is, the one with more Skolem constants, will be considered before H_g[7]. Switching the order of the two clauses in $Skolem/3$ will result in a general-to-specific search. In that case, the universal grammar will be considered first, since it is maximally general and can be expressed with only one Skolem constant (see Example 4).

[7]provided H_g and H_s are within the same clause bound

Metagol$_{CF}$

The Metagol$_{CF}$ system is based on an abductive variant of the Context-Free Meta-interpreter from Figure 5.3(b). The full Prolog description is included in Appendix A.1. Once more, abduction is carried out with respect to a single goal as in Metagol$_R$.

Metagol$_{RCF}$

The Metagol$_{RCF}$ system simply combines Metagol$_R$ and Metagol$_{CF}$ sequentially, as shown below in Figure 5.6. Due to Prolog's procedural semantics, the hypothesis returned will be Regular in the case Metagol$_R$ finds a consistent grammar and otherwise will be the result of Metagol$_{CF}$.

```
metagolRCF(G):- metagolR(G).
metagolRCF(G):- metagolCF(G).
```

Figure 5.6.: Metagol$_{RCF}$

Metagol$_D$

Metagol$_D$ is a modification of the Meta-Interpreter shown in Figure 5.3(c). Compared to Metagol$_R$ and Metagol$_{CF}$ that support Type 2,3 in Chomsky hierarchy, Metagol$_D$ supports the broader class of Dyadic Datalog programs, which is Turing equivalent. Thus it has a much larger hypothesis space than the other Metagol systems. Therefore it requires additional modifications in order to handle the significantly increased search space. In particular, the modifications include methods for a) ordering the Herbrand Base, b) logarithmic-bounding the maximum depth of iterative deepening and c) using a series of episodes for ordering multi-predicate incremental learning. The full Prolog description of Metagol$_D$ with these modifications is included in Appendix A.2.

Ordering the Herbrand Base Within ILP, search efficiency depends on the partial order of θ-subsumption [NCdW97]. Similarly in $Metagol_D$ search efficiency is achieved using a total ordering to constrain deductive, abductive and inductive operations carried out by the Meta-Interpreter and to reduce redundancy in the search. In particular, we employ Knuth-Bendix [KB70, ZSM05] (lexicograhic) as well as interval inclusion total orderings

over the Herbrand Base to guarantee termination. To illustrate, consider the following ordered variant of the chain meta-rule from Figure 5.3(c)

$$P(x,y) \leftarrow Q(x,z), R(z,y), OrderTest(P,Q,R,x,y,z)$$

Figure 5.7 illustrates alternative OrderTests which each constrain the chain

Lexicographic	**Interval inclusion**
p1_parent(a_alice,b_ted) .. p2_parent(c_jake,d_john) .. p3_grandparent(a_alice,e_jane) p3_grandparent(c_jake,f_bob) ..	leq(0,0) leq(1,1) leq(2,2) .. leq(0,1) leq(1,2) leq(0,2)
Lex OrderTest	**Inclusion OrderTest**
$\langle P,x,y\rangle @ > \langle Q,x,z\rangle$ AND $\langle P,x,y\rangle @ > \langle R,z,y\rangle$	$x @ > z$ AND $z @ > y$

Figure 5.7.: Datalog Herbrand Base orderings with chain meta-rule OrderTests. $@>$ is "lexicographically greater"

meta-rule to descend through the Herbrand Base. In the *lexicographic* ordering predicates which are higher in the ordering, such as *grandparent*, are defined in terms of ones which are lower, such as *parent*. Meanwhile *interval inclusion* supports definitions of (mutually) recursive definitions such as *leq*, *ancestor* and *even/odd*. Interval inclusion assumes all atoms $p(u,v)$ precede the atom $q(x,y)$ in the ordering when the interval $[x,y]$ includes the interval $[u,v]$. Finite descent guarantees termination even when an ordering is infinitely ascending (e.g. over the natural numbers).

Logarithmic bounding and PAC model Apart from carrying out iterative deepening search like the other Metagol systems, $Metagol_D$ also set a maximum depth bound for iterative deepening as $d = log_2 m$, where m is the number of examples. Assuming c is the number of distinct clauses in H_2^2 for a given $\mathcal{P}, \mathcal{C}$ the number of hypotheses at depth d is

$$|H_d| \leq c^d = 2^{dlog_2 c} = 2^{log_2 m log_2 c} = m^{log_2 c}$$

Since the total number of hypotheses for an iterative deepening strategy is a constant, k, times those at the maximum depth considered, logarithmic bounding ensures the total number of hypotheses considered grows polynomially in m. Within the PAC model [Val84] the Blumer bound [BEHW89] can be used to show that

$$m \geq \frac{ln(k|H_d|) + ln\frac{1}{\delta}}{\epsilon} = O(ln(m))$$

for the boundary case of $\epsilon = \delta = \frac{1}{2}$. Therefore Metagol$_D$ with logarithmic bounding PAC-learns the class H_2^2.

Episodes for multi-predicate learning Learning definitions from a mixture of examples of two inter-dependent predicates such as *parent* and *grandparent* requires more examples and search than learning them sequentially in separate *episodes*. In the latter case the *grandparent* episode is learned once it can use the definition from the *parent* episode. This phenomenon can be explained by considering that time taken for searching the hypothesis space for the joint definition is a function of the product of the hypothesis spaces of the individual predicates. By contrast the total time taken for sequential learning of episodes is the sum of the times taken for the individual episodes[8] so long as each predicate is learned with low error.

5.3.2. Implementation in Answer Set Programming (ASP)

Compared to Prolog, ASP not only has advantages in handling non-monotonic reasoning, but also has higher efficiency in tackling search problems [GKKS12]. The systems ASP$_{\mathrm{M}R}$ and ASP$_{\mathrm{M}CF}$ are two simple ASP implementations of Meta-Interpretive learning. Each sequence is encoded as a set of facts. For example, the positive example $posEx(Seq2, [1, 1])$ is encoded in the second line in Figure 5.8, where seq2 is the ID of the sequence and the predicate $seqT(SeqID, P, T)$ means the sequence has a terminal T at position P. The meta-interpretive parser uses position to mark a substring, rather than storing the substring in a list. The goal of finding an hypothesis that covers all positive examples and none of the negatives is encoded as an integrity constraint.

[8] Misordering episodes leads to additional predicate invention.

List	Facts
posEx(e1,[0,0]).	posEx(e1). length(e1,2). seqT(e1,0,0). seqT(e1,1,0).
posEx(e2,[1,0,1]).	posEx(e2). length(e2,3). seqT(e2,0,1). seqT(e2,1,0). seqT(e2,2,1).
negEx(e3,[1]).	negEx(e3). length(e3,1). seqT(e3,0,1).
negEx(e4,[0,1]).	negEx(e4). length(e4,2). seqT(e4,0,0). seqT(e4,1,1).

Figure 5.8.: ASP representation of examples.

ASP$_{\mathbf{MR}}$ The program in Figure A.3 is an ASP implementation of the Regular Meta-interpreter in Figure 5.3. It is sectioned into parts describing generating, defining, testing, optimising, and displaying. The generating part specifies the hypothesis space as a set of facts about delta1/3 and acceptor/1. ASP choice rules are used to indicate that any subset of this set is allowed in the answer sets of this program. The defining part corresponds to the Regular Meta-interpreter. The testing part contains an integrity constraint saying that an answer set of this program should contain production rules which parse all positive examples and no negative examples. The display part restricts the output to containing only predicates $delta1/3$ and $acceptor/1$, which corresponds to the hypothesis.

In order to find a minimal hypothesis like that in Metagol$_R$, the optimisation component in ASP is used. Although the use of optimisation increases the computational complexity [GKKS12], it improves[9] the predictive accuracy. An optimisation statement like the one in Figure A.3 specifies the objective function to be optimised. The weight following each atom is part of the objective function. In our case, the objective function corresponds to the description length of an hypothesis. Therefore the weight is set to 1 for each atom, meaning the description length of a unit clause is 1.

Most ASP solvers do not support variables directly, therefore a grounder is needed for transforming the input program with first-order variables into an equivalent ground program. Then an ASP solver can be applied to find an answer set that satisfies all the constraints. The hypothesised grammar will be part of the returned answer set. In the case shown in Figure 5.8 the first hypothesis returned by ASP is the same as the one found by Metagol and correctly represents the Parity acceptor of Figure 5.1.

ASP solvers use efficient constraint handling techniques to efficiently find

[9]Occam's razor suggests that simpler hypotheses have higher predictive power. This is further supported by the experimental results described in Section 5.4, where the non-minimal hypotheses suggested by MC-TopLog have lower predictive accuracies than the minimal ones hypothesised by ASP$_{\mathrm{M}}$ and Metagol

```
% Instances
#const maxNumSkolemConstants=1.
skolem(0..maxNumSkolemConstants).
terminal(0;1).

% Generate: specify the hypothesis space
{acceptor(NT):skolem(NT)}.
{delta1(NT1,T,NT2):skolem(NT1):terminal(T):skolem(NT2)}.

% Defining Part
parse(ExID,MaxLengh,MaxLengh,NT):- length(ExID,MaxLengh),acceptor(NT).
parse(ExID,Position1,Position2,NT1):- seqT(ExID,Position1,T),
delta1(NT1,T,NT2), parse(ExID,Position1+1,Position2,NT2).

% Integrity constraint
:- negEx(ExID),length(ExID,MaxLengh),parse(ExID,0,MaxLengh,0)).
:- posEx(ExID),length(ExID,MaxLengh),not parse(ExID,0,MaxLengh,0).

% Optimisation
#minimize [delta1(NT1,T,NT2):skolem(NT1):terminal(T):skolem(NT2)=1,
acceptor(NT):skolem(NT)= 1].

% Displaying
#hide.
#show delta1/3.
#show acceptor/1.
```

Figure 5.9.: $\text{ASP}_{\text{M}R}$

stable models known as answer sets. This computational mechanism is very different from that of Prolog, leading to their different implementations, in particular, in the use of iterative deepening. In addition, the bound on clauses puts an implicit limit on Skolem constants, since the number of Skolem constants in a derived hypothesis is at most the number of clauses it contains. Therefore Metagol_R is immune to the number of Skolem constants pre-specified in the background knowledge. By contrast, $\text{ASP}_{\text{M}R}$ is largely affected by the number of Skolem constants due to its bottom-up search. Therefore $\text{ASP}_{\text{M}R}$ has to put an explicit bound on the number of Skolem constants. Specifically, the second line of 'Generate' $\{delta1(NT1, T, NT2) : skolem(NT1) : terminal(T) : skolem(NT2)\}$ has a default size of $T * NT^2$, where T corresponds to the number of terminals and NT denote the number of Skolem constants. While a cardinality constraint on this set does not always reduce the search space, because it can

lead to a quadratic blow-up in search space [GKKS12] when the cardinality constraint is translated into normal logic program during the grounding stage. Additionally, ASP solvers' built-in optimisation component is handy for finding a global minimal hypothesis. Thus $\text{ASP}_{\text{M}R}$ does not use iterative deepening on the clause bound like that in Metagol_R for finding a global minimal hypothesis.

ASP_{M}*CF* Similar to Metagol_{CF}, the $\text{ASP}_{\text{M}CF}$ system is based on a variant of the Context-Free Meta-interpreter from Figure 5.3(b). However, there is no equivalent ASP implementation to Metagol_{RCF}, since Metagol_{RCF} exploits the procedural semantics of Prolog programs, while there is no similar procedural semantics in ASP programs.

The implementation of $\text{ASP}_{\text{M}D}$ has not been explored in this thesis. It is one of the future work. In theory, it is similar to $\text{ASP}_{\text{M}R}$ and $\text{ASP}_{\text{M}CF}$ apart from the meta-rules. However, the extension from Types 2,3 in Chomsky hierarchy to the broader class of H_2^2 means a significant increase in hypothesis space. Therefore the lack of domain-specific procedural bias in ASP is more of a problem.

5.4. Experiments

In this section we describe experiments on learning tasks that requires learning recursion and predicate invention. The first four experiments are about grammar learning. It was shown in Section 5.1 that ILP systems cannot learn grammars in a DCG representation with predicate invention. However, an ILP system given a meta-interpreter as part of background knowledge becomes capable of doing predicate invention. In the experiments described below, the performance of the ILP system MC-TopLog, loaded with suitable meta-interpretive background, is compared against variants of Metagol and ASP_{M} as described in Section 5.3. MC-TopLog is chosen for this comparison since it can learn multiple dependent clauses from examples (unlike say Progol). This is a necessary ability for grammar learning tasks. In the last two experiments MIL is applied to non-grammar learning tasks: learning a definition of a staircase and robot strategy learning. All datasets and learning systems used in these experiments are available at http://ilp.doc.ic.ac.uk/MIL

5.4.1. Learning Regular Languages

We investigate the following Null hypotheses.

Null Hypothesis 1.1 Metagol$_R$, ASP$_{\mathrm{M}R}$ and a state-of-the-art ILP system cannot learn randomly chosen Regular languages.

Null Hypothesis 1.2 Metagol$_R$ and ASP$_{\mathrm{M}R}$ cannot outperform a state-of-the-art ILP system on learning randomly chosen Regular languages.

Null Hypothesis 1.3 Metagol$_R$ can not outperform ASP$_{\mathrm{M}R}$ on learning randomly chosen Regular languages.

Materials and Methods

Randomly chosen deterministic Regular grammars were generated by sampling from a Stochastic Logic Program (SLP) [Mug96] which defined the space of target grammars. Specifically, the SLP used for sampling consists of a meta-interpreter and all possible grammars. Then the following steps were conducted. Firstly, an integer i ($1 \leq i \leq 3$) was randomly sampled. This integer corresponds to the number of seed examples[10]. Secondly, the query "sample(parse(Seq,Grammar))' returned one sequence as well as the grammar that parse this sequence. Thirdly, the grammars were aggregated by issuing the query 'sample(parse(Seq,Grammar))" i times. Finally, each generated grammar was reduced using Plotkin's reduction algorithm (see Section 5.2.3) to remove redundancy and equivalent non-terminals. Non-deterministic and finite language grammars were discarded. Sampling of examples was also done using an SLP. Sampling was with replacement.

In this experiment, we used two different datasets sampled from different distributions. In dataset RG1, the examples were randomly chosen from Σ^* for $\Sigma = \{a, b\}$, while in RG2 $\Sigma = \{a, b, c\}$. RG2 has longer sequence lengths, as shown by Table 5.1. Both datasets contains 200 randomly chosen Regular grammars. We compared the performance of Metagol$_R$, ASP$_{\mathrm{M}R}$ and MC-TopLog on learning Regular grammars using RG1. Only Metagol$_R$ and ASP$_{\mathrm{M}R}$ were compared on RG2, since MC-TopLog failed to terminate due to the longer sequence examples. The performance was evaluated on

[10]The parsing of seed examples requires all rules in the grammar

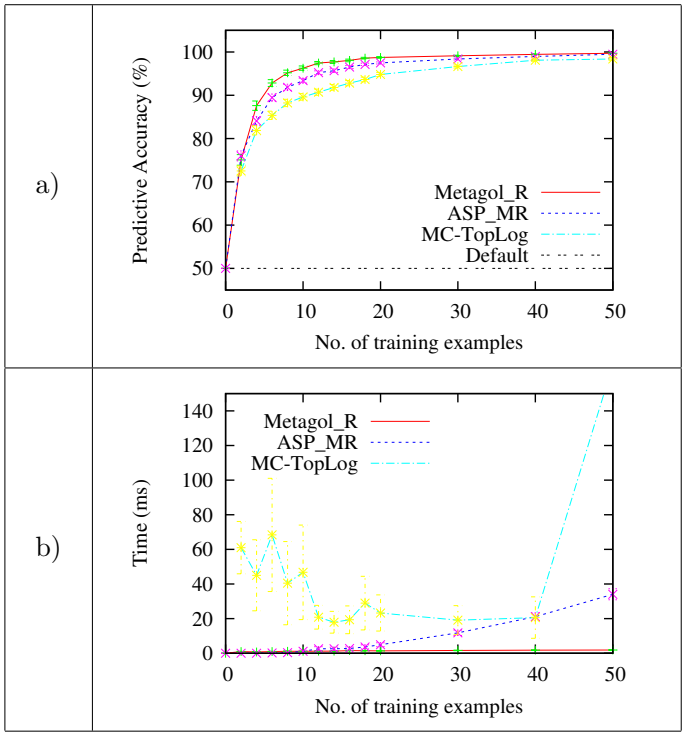


Figure 5.10.: Average (a) predictive accuracies and (b) running times for Null hypothesis 1 (Regular) on short sequence examples (RG1).

predictive accuracies and running time[11]. The results were averaged over 200 randomly sampled grammars. For each sample, we used a fixed test set of size 1000. The size of training set varied from 2 to 50 in RG1 and from 4 to 100 in RG2.

Results and Discussion

As shown by Figure 5.10(a), all three systems have predictive accuracies significantly higher than default. Therefore Null hypothesis 1.1 is refuted. MC-TopLog is not usually able to carry out predicate invention, but is enabled to do so by including a meta-interpreter as background knowledge. This is because predicate invention is an intrinsically higher-order logic operation,

[11]The running times of Metagol$_R$ and ASP$_{MR}$ are measured in terms of getting the first minimal hypothesis, rather than all minimal hypotheses.

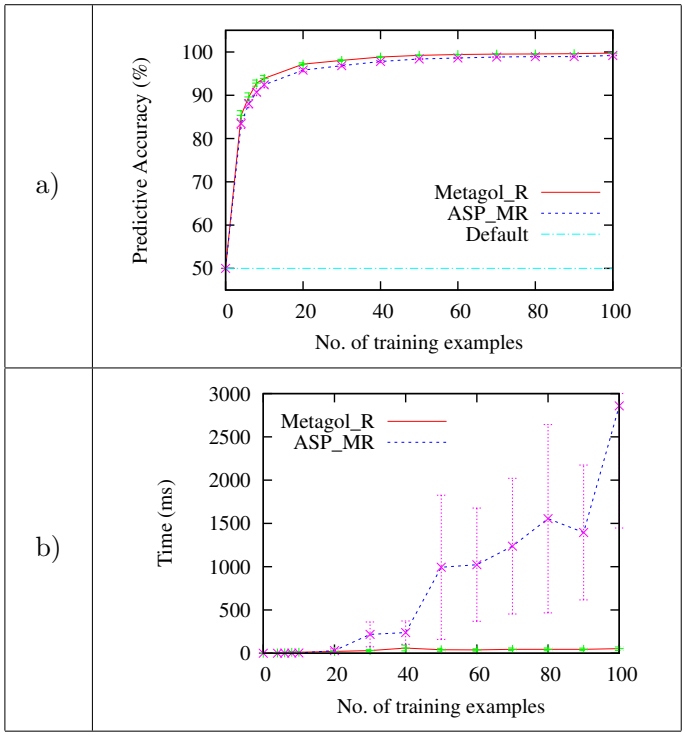


Figure 5.11.: Average (a) predictive accuracies and (b) running times for Null hypothesis 1 (Regular) on long sequence examples (RG2).

while a meta-interpreter embeds metarules which encode higher-order logic. Therefore MC-TopLog with a meta-interpreter in its background knowledge can output a hypothesis, which is in a similar form to that of Metagol$_R$. An example of the hypothesis induced by MC-TopLog is given in Figure 5.12.

As shown in Figure 5.10(b), MC-TopLog's running time is considerably longer than Metagol$_R$ and ASP$_{MR}$. MC-TopLog has slightly lower predictive accuracies than both Metagol$_R$ and ASP$_{MR}$. The difference is statistically significant according to a t-test ($p < 0.01$). Therefore, Null hypothesis 1.2 is refuted with respect to both predictive accuracy and running time. MC-TopLog's longer running time is due to the fact that it enumerates all candidate hypotheses within the version space. By contrast, both Metagol$_R$ and ASP$_{MR}$ do not traverse the entire space. In particular, ASP solver like Clasp incorporate effective optimisation techniques based on branch-and-bound algorithms [GKNS07]. The larger hypothesis space

leads to lower accuracy in MC-TopLog. This is consistent with the Blumer Bound [BEHW89], according to which the error bound decreases with the size of the hypothesis space. Moreover, MC-TopLog's accuracy is also affected by its covering algorithm which is greedy and does not guarantee finding a global optimum. By contrast, both Metagol$_R$ and ASP$_{\mathrm{M}R}$ find an hypothesis which is minimal in terms of description length. Figure 5.12 compares the different hypothesis suggested by the three systems. MC-TopLog's hypothesis $H_{mcTopLog}$ is longer than both of Metagol$_R$ and ASP$_{\mathrm{M}R}$. By contrast, both Metagol$_R$ and ASP$_{\mathrm{M}R}$ derive the one with minimal description length, although they are not exactly the same. $H_{metagolR}$ is more specific than H_{aspMR} due to the specific-to-general search in Metagol$_R$. In this example, $H_{metagolR}$ is the same as the target hypothesis.

E^+	E^-	$H_{metagolR}$	H_{aspMR}	$H_{mcTopLog}$
aa *aba* *abbba*	*abab* *aabaa* *baaababaa*	$s \rightarrow a\ s_1$ $s_1 \rightarrow b\ s_1$ $s_1 \rightarrow a\ s_2$ $s_2 \rightarrow$	$s \rightarrow a\ s_1$ $s_1 \rightarrow b\ s_1$ $s_1 \rightarrow a$ s s $\rightarrow$	$s \rightarrow a\ s$ $s \rightarrow$ $s \rightarrow b\ s_1$ $s_1 \rightarrow a\ s_2$ $s_2 \rightarrow$ $s_1 \rightarrow b\ s$

Figure 5.12.: Hypothesis Comparison

Figure 5.10(b) indicates that Metagol$_R$ has considerably lower running time than ASP$_{\mathrm{M}R}$, and the difference increases when examples are long, as shown in Figure 5.11(b). Metagol$_R$ also has slightly higher accuracy than ASP$_{\mathrm{M}R}$. A t-test suggests that their difference in accuracy is statistically significant ($p < 0.01$) as one is consistently higher than the other. Therefore, Null hypothesis 1.3 is refuted with respect to both predictive accuracy and running time. The reasons that Metagol$_R$ is faster than ASP$_{\mathrm{M}R}$ on learning regular languages are: (1) Metagol$_R$, as a Prolog implementation, can use forms of procedural bias which cannot be defined declaratively in ASP since the search in ASP is not affected by the order of clauses in the logic program; (2) there are few constraints in the learning task so that efficient constraint handling techniques in ASP do not increase efficiency.

Both Metagol$_R$ and ASP$_{\mathrm{M}R}$'s running times increase with the number of examples. By contrast, MC-TopLog's running time appears to be unaffected

	RG1	RG2	CFG3	CFG4
Average±STD	6.15±4.06	11.43±10.47	5.89±3.08	11.02±9.79
Maximum	15	78	15	68

Table 5.1.: Average and Maximum lengths of sampled examples for datasets R1, R2, CFG3 and CFG4.

by the number of examples. MC-TopLog's running time is determined by the size of the hypothesis space it enumerates, which depends on the lengths of examples. It therefore fails to learn from RG2 which has longer sequences (see Table 5.1).

5.4.2. Learning Context-Free Languages

We investigate the following Null hypotheses.

Null Hypothesis 2.1 Metagol$_{CF}$, ASP$_{\mathrm{M}CF}$ and a state-of-the-art ILP system cannot learn randomly chosen Context-Free languages.

Null Hypothesis 2.2 Metagol$_{CF}$ and ASP$_{\mathrm{M}CF}$ cannot outperform a state-of-the-art ILP system on learning randomly chosen Context-Free languages.

Null Hypothesis 2.3 Metagol$_{CF}$cannot outperform ASP$_{\mathrm{M}CF}$ on learning randomly chosen Context-Free languages.

Materials and Methods

Randomly chosen Context-Free grammars were generated using an SLP and reduced using Plotkin's reduction algorithm (see Section 5.2.3). Grammars were removed if they corresponded to finite languages or could be recognised using the pumping lemma for Context-Free grammars. However, not all Regular grammars can be filtered in this way, since it is undecidable whether a Context-Free grammar is Regular. Specifically, if a grammar is not pumpable, then it is definitely Regular, while a pumpable grammar is not necessarily non-Regular.

The examples were generated in the same way as that in the Regular-language experiment. There were two datasets, each containing 200 samples. Details are shown in Table 5.1. The comparisons of Metagol$_{CF}$, ASP$_{\mathrm{M}CF}$ and MC-TopLog on learning Context-Free grammars was done

using only dataset CFG3 since MC-TopLog failed to terminate on CFG4 with long-sequence examples. The evaluation method was the same as that for learning regular languages.

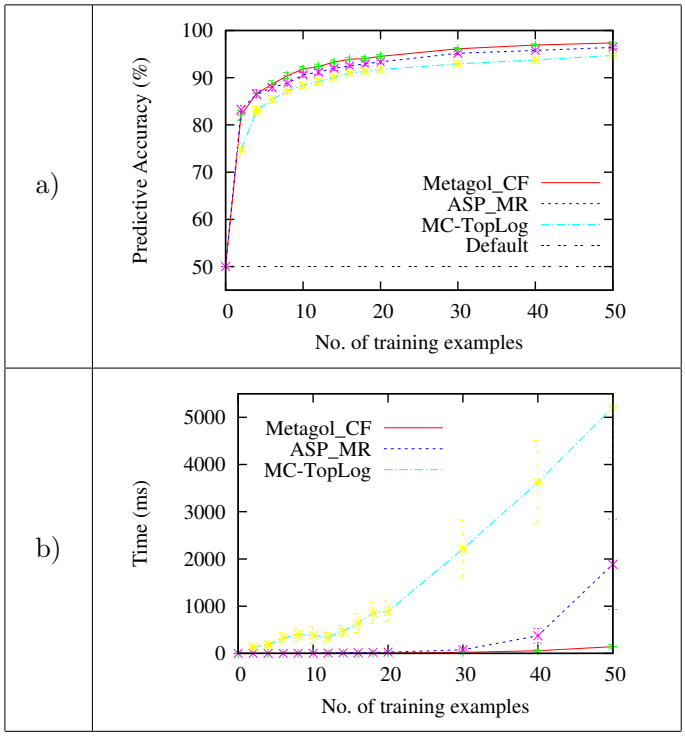


Figure 5.13.: Average (a) predictive accuracies and (b) running times for Null hypothesis 2 (Context-free) on short sequence examples (CFG3).

Results and Discussion

As shown in Figure 5.13(a), all three systems derive hypotheses with predictive accuracies considerably higher than default. Therefore Null hypotheses 2.1 is refuted. Compared to MC-TopLog, both Metagol$_{CF}$ and ASP$_{\mathrm{M}CF}$ have consistently higher averaged predictive accuracies. This is again explained by the Blumer Bound since MC-TopLog considers a larger hypothesis space. Metagol$_{CF}$ conducts a bounded search using a bottom clause so that it is feasible even though the version space is potentially infinite. ASP solvers can also deal with infinite spaces.

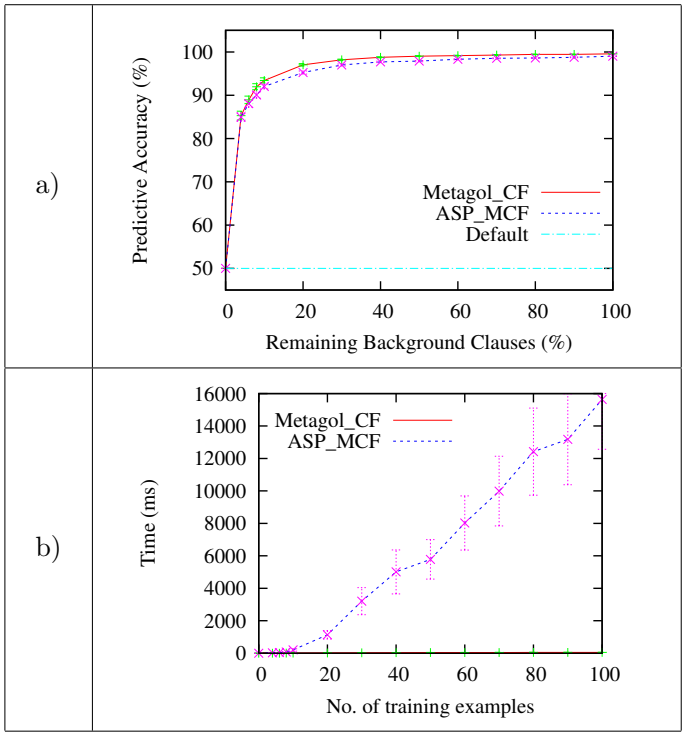


Figure 5.14.: Average (a) predictive accuracies and (b) running times for Null hypothesis 2 (Context-free) on long sequence examples (CFG4).

Null hypothesis 2.2 is refuted with respect to both running time and predictive accuracy. The predictive accuracies of Metagol$_{CF}$ and ASP$_{\mathrm{M}CF}$, have no significant difference on either dataset, as shown by the graphs in Figures 5.13(a) and 5.14(a), since both derive globally optimal solutions. However, Metagol$_{CF}$ has shorter running time due to its procedural bias. Therefore Null hypothesis 2.3 is refuted.

5.4.3. Representation Change

Null Hypothesis 3 Metagol$_{RCF}$ cannot improve performance by changing representation from Regular to Context-Free languages

Materials and Methods

The experiment used RG1 and CFG3 from the previous two experiments. Therefore, there were 400 sampled grammars in total, half being Regular and the other half mostly Context-Free and non-Regular.

We compared Metagol$_{RCF}$ (variable hypothesis space) against Metagol$_{CF}$ (fixed hypothesis space). The predictive accuracies and running time were measured as before. The results were averaged over the 400 grammars.

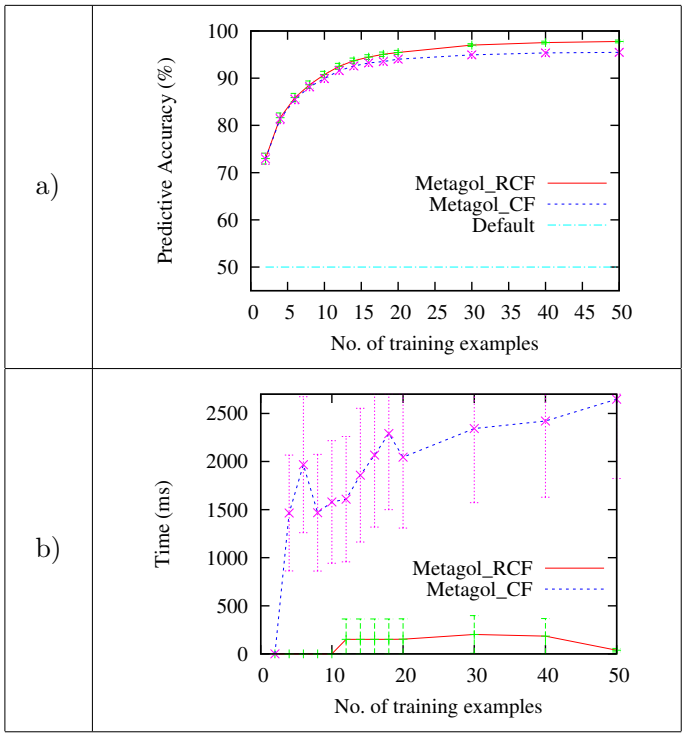


Figure 5.15.: Average (a) predictive accuracies and (b) running times for Null hypothesis 3 (Representation change) on combination of RG dataset1 and CFG dataset3.

Results and Discussion

As shown in Figure 5.15(a), Metagol$_{RCF}$ has slightly higher predictive accuracies than Metagol$_{CF}$. This refutes Null hypothesis 3. The accuracy difference is once more consistent with the Blumer Bound [BEHW89], according to which the error bound decreases with the size of the hypothesis

space. Note also in Figure 5.15(b), that the running times of Metagol$_{CF}$ are significantly higher than that of Metagol$_{RCF}$. This can be explained by the fact that when the target grammar is Regular, Context-Free grammars were still considered.

5.4.4. Learning a simplified natural language grammar

Metagol$_N$ and ASP$_{\mathrm{M}N}$ are two systems resulting from the application of Metagol and ASP$_{\mathrm{M}}$ in learning a simplified natural language grammar. We investigate the following Null hypotheses. MC-TopLog was not included for comparison since its search time was excessive in these learning tasks.

Null Hypothesis 4.1 Metagol$_N$ and ASP$_{\mathrm{M}N}$ cannot learn a simplified natural language grammar.

Null Hypothesis 4.2 Metagol$_N$ cannot outperform ASP$_{\mathrm{M}N}$ on learning a simplified natural language grammar.

Null Hypothesis 4.3 The provision of background knowledge does not improve learning accuracies and efficiency.

Definite Clause Grammar	**Production rules**
$s(S1,S2) \leftarrow np(S1,S3),\ vp(S3,S4),\ np(S4,S2).$	$s \rightarrow s4\ s_1$
$s(S1,S2) \leftarrow np(S1,S3),\ vp(S3,S4),\ np(S4,S5),$	$s_1 \rightarrow s5\ s4$
$prep(S5,S6),\ np(S6,S2).$	$s_1 \rightarrow s5\ s2$
	$s2 \rightarrow s4\ s3$
	$s3 \rightarrow prep\ s4$
$np(S1,S2) \leftarrow det(S1,S3),\ noun(S3,S2).$	$s4 \rightarrow det\ noun$
$np(S1,S2) \leftarrow det(S1,S3),\ adj(S3,S4),\ noun(S4,S2).$	$s4 \rightarrow det\ s6$
	$s6 \rightarrow adj\ noun$
$vp(S1,S2) \leftarrow verb(S1,S2).$	$s5 \rightarrow verb$
$vp(S1,S2) \leftarrow verb(S1,S3),\ prep(S3,S2).$	$s5 \rightarrow verb\ prep$

Figure 5.16.: Target theory for simplified natural language grammar.

Materials and Methods

The training examples come from the same domain considered in [MLTN11] and consist of 50 sentences such as "a ball hits the small dog". Half the examples are positive and half negative, resulting in a default accuracy of 50%. The complete target grammar rules for parsing the training examples are given in Figure 5.16. Each learning task is generated by randomly removing a set of clauses. The left-out clauses become the target to be reconstructed.

For each size of leave-out, we sampled ten times. For each sample, the predictive accuracies were computed by 10-fold cross validation[12]. The results plotted on the figure are averaged over all leave-out samples.

Results and Discussion

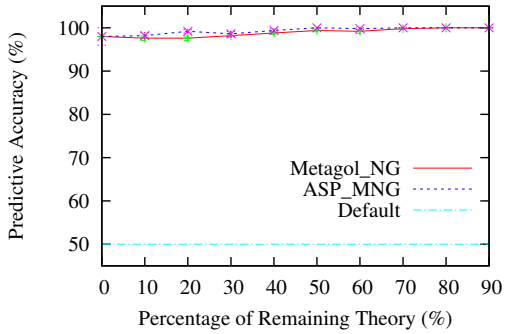


Figure 5.17.: Average predictive accuracies for Null hypothesis 4 on simplified natural language grammars.

The predictive accuracies and running times are plotted in Figures 5.17 and 5.18 respectively. The x-axis corresponds to the percentage of remaining production rules. Specifically, 0% corresponds to the case when $B_A = \emptyset$, while 90% means 9 out of 10 production rules remain. Figure 5.17 shows that the predictive accuracies of both Metagol_N and $\text{ASP}_{\text{M}N}$ are significantly higher than default, therefore Null hypothesis 4.1 is refuted.

Although there is no significant difference between Metagol_N and $\text{ASP}_{\text{M}N}$ in terms of predictive accuracy, $\text{ASP}_{\text{M}N}$ takes much shorter running time than Metagol_N when more than half of the production rules are missing ($x < 50\%$). However, the expanded version for $50\% \leq x \leq 90\%$ in Figure 5.18(b) shows that $\text{ASP}_{\text{M}N}$ becomes slower than Metagol_N when background knowledge is less sparse. Therefore, Null hypothesis 4.2 is refuted since when more than 70% of the production rules remain Metagol_N has significantly shorter running time than $\text{ASP}_{\text{M}N}$ without sacrificing its predictive accuracy. This is due to the procedural bias encoded in Metagol_N.

[12]The size of available examples is 50, therefore not large enough for reserving a subset as test set

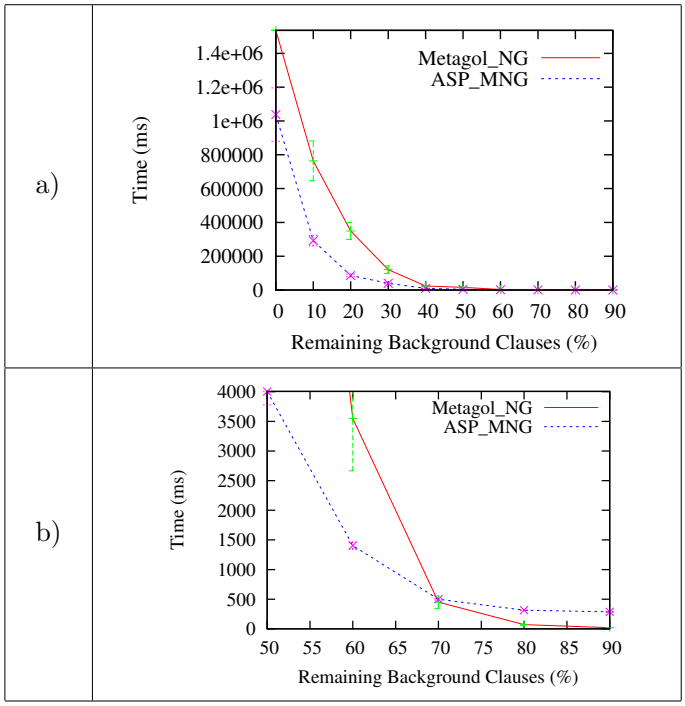


Figure 5.18.: Averaged running time for Null hypothesis 4 on simplified natural language grammars. (a) Full range $[0, 90]$ (b) Partial range $[50, 90]$ but expanded.

The running times of both Metagol$_N$ and ASP$_{\mathrm{M}N}$ decrease dramatically with the increase of background knowledge. The predictive accuracies increase with increasing background knowledge, reaching 100% when the degree of remaining background clauses increases to 70%. Therefore, Null hypothesis 4.3 is refuted.

Figure 5.19 compares the different hypotheses derived by Metagol$_N$ and ASP$_{\mathrm{M}N}$. These are derived when $B_A = \emptyset$. Since both Metagol$_N$ and ASP$_{\mathrm{M}N}$ find an hypothesis which is globally optimal in terms of description length, these hypotheses have identical description length although they are not identical hypotheses. Among all the invented predicates in H_M, s_4 corresponds to np in natural grammars and s_3 is closed to vp. Similarly in H_A, s_3 and $s6$ corresponds to vp and np respectively.

H_M(Metagol)	H_A (ASP_M)
$s \rightarrow s_2\ s_1$	$s \rightarrow s_4\ s_3$
$s_1 \rightarrow prep\ s_4$	$s_3 \rightarrow verb\ s_5$
$s_1 \rightarrow det\ s_5$	$s_3 \rightarrow verb\ s_4$
$s_2 \rightarrow s_4\ s_3$	$s_4 \rightarrow s_6\ s_5$
$s_3 \rightarrow verb$	$s_4 \rightarrow det\ s_5$
$s_3 \rightarrow verb\ s_4$	$s_5 \rightarrow prep\ s_6$
$s_4 \rightarrow det\ s_5$	$s_5 \rightarrow noun$
$s_5 \rightarrow adj\ noun$	$s_6 \rightarrow det\ noun$
$s_5 \rightarrow noun$	$s_6 \rightarrow det\ adj$

Figure 5.19.: Metagol and ASP_M hypotheses for learning a simplified natural language grammar.

5.4.5. Learning a definition of a staircase

The authors of [FS12] have shown that ALEPH can learn a definition of a staircase for a rescue robot from visually-derived data. Figure 5.20 shows an example of the output from ALEPH, which is original given in [FS12]. This kind of definition is not entirely general since it does not involve recursion. We now demonstrate that MIL can be used to learn a general recursive definition of a staircase using predicate invention.

A staircase can be represented by a list of rectangles. Such representation assumes that (1) rectangles adjacent in the list are next to each other in the image; (2) the list is ordered from left to right. For example, Figure 5.21 taken from [FS12] shows a staircase and its range image taken by the 3D depth camera. Each rectangle in the image is associated with a unique colour, whose identifier is given at the top of Figure 5.21. For instance, red colour is assigned identifier 1. Therefore, the staircase in Figure 5.21 can be encoded as $staircase([p1, p3, p4, p5, p6, p8])$. Information from the 3D depth camera indicates that rectangle $p1$ is perpendicular to rectangle$p3$. This can be encoded as a delta rule $delta2(perpendicular, p1, p3)$, where *perpendicular* representing relations between $p1$ and $p3$. The meta-interpreter used in this experiment is a variant of the Context-Free Meta-interpreter from Figure 5.3.

Training examples of staircases and part of the background knowledge are given in Figure 5.22. The resulting hypothesis produced by Metagol is shown in Figure 5.23, where s_1 is an invented predicate, which can be interpreted as *step*. Due to its recursive form, this definition has shorter

$staircase(Planes) \leftarrow$
$member(C, Planes), member(D, Planes), angle(D, C, `0 \pm 15'),$
$member(E, Planes), angle(E, D, ' 90 \pm 15'), angle(E, C, ' 90 \pm 15'),$
$distributed\_along(E, axisX).$
$staircase(Planes) \leftarrow$
$member(C, Planes), member(D, Planes), angle(D, C, `0 \pm 15'),$
$member(E, Planes), member(F, Planes),$
$angle(F, D, `0 \pm 15'), angle(F, C, `0 \pm 15'),$
$dr\_xy(E, F, south)$. % E is on the south of plane F from the XY view of image
$staircase(Planes) \leftarrow$
$n\_of\_parts(Planes, 4)$, % n_of_parts means number of parts in $Planes$.
$member(C, Planes),$
$distributed\_along(C, axisX)$. %C is distributed along X axis.

Figure 5.20.: Non-recursive definition of staircase hypothesised by ALEPH

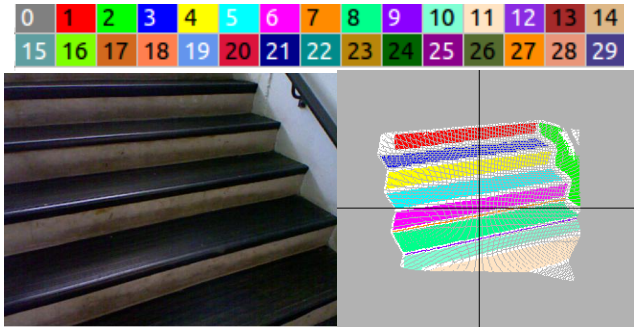


Figure 5.21.: Staircase and its range image with colour legend [FS12]

description length than the one derived by ALEPH (Figure 5.20). In fact, if s_1 is not invented, the output hypothesis would be even shorter like H_{noPI} shown in Figure 5.24. However, due to the form of meta-rules, hypotheses output by Metagol are in Chomskey normal form and limited to two body literals. Therefore s_1 is forced to be introduced.

5.4.6. Robot strategy learning

In AI, planning traditionally involves deriving a sequence of actions which achieves a specific goal from a specific initial situation [RN10]. However, various machine learning approaches support the construction of strategies[13]. Such approaches include the SOAR architecture [Lai08], reinforce-

[13] A strategy is a solution to a problem [RN10]. It is a mapping from a set of initial to a set of goal situations with the problem solved. However, it is not simply a sequence

First-order	Meta-form
Examples	**Examples**
$staircase([p1, p3, p4, p5, p6])$	$prove(staircase, [p1, p3, p4, p5, p6])$
$staircase([p1, p3, p4, p5, p6, p8, p10])$	$prove(staircase, [p1, p3, p4, p5, p6, p8, p10])$
$staircase([p1, p3, p4,])$	$prove(staircase, [p1, p3, p4,])$
Background knowledge	**Background knowledge**
$perpendicular(p1, p3)$.	$delta2(perpendicular, p1, p3)$.
$parallel(p1, p4)$.	$delta2(parallel, p1, p4)$.
$perpendicular(p3, p4)$.	$delta2(perpendicular, p3, p4)$.
$parallel(p3, p5)$.	$delta2(parallel, p3, p5)$.

Figure 5.22.: Training examples and background knowledge for learning the concept of staircase

First-order logic	Production rules
$staircase(Rectangles) \leftarrow s_1(Rectangles)$.	$staircase \rightarrow s_1$
$staircase([X, Y, Z\|Rectangles]) \leftarrow s_1([X, Y, Z]),$ $staircase([Z\|Rectangles])$.	$staircase \rightarrow s_1\ staircase$
$s_1([X, Y, Z]) \leftarrow perpendicular(X, Z),\ parallel(Z, Y)$	$s_1 \rightarrow perpendicular\ parallel$

Figure 5.23.: Recursive definition of staircase hypothesised by MIL. s_1 is an invented predicate corresponding to the concept of *step*

ment learning [SB98], and action learning within ILP [MM97, Ote05, CSIR12].

In this experiment structured strategies are learned which build a stable wall from a supply of bricks. Predicate invention is used for top-down construction of re-usable sub-strategies. Fluents are treated as monadic predicates which apply to a situation, while Actions are dyadic predicates which transform one situation to another.

Materials

Figure 5.25 shows a positive example (a) of a stable wall together with two negative examples (unstable walls) consisting of a column[14] (b) and a wall with insufficient central support (c). Predicates are either *high-level* if defined in terms of other predicates or *primitive* otherwise. High-level

of actions but a contingency plan that specifies what to do depending on the state of world [RN10].

[14] A column is not as stable as a wall built out of brick with offset positions like those in real-world.

a) H_{PI}**(Predicate Invention)**	b) H_{noPI} **(No Predicate Invention)**
$staircase \rightarrow s_1$ $staircase \rightarrow s_1\ staircase$ $s_1 \rightarrow perpendicular\ parallel$	$staircase \rightarrow perpendicular\ parallel$ $staircase \rightarrow perpendicular\ parallel\ staircase$

Figure 5.24.: Staircase definition in the form of production rules: Predicate Invention vs. No Predicate Invention

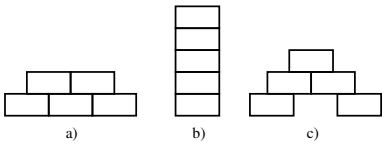


Figure 5.25.: Examples of a) stable wall, b) column and c) non-stable wall

$$
\begin{array}{l}
buildWall(X,Y) \leftarrow a2(X,Y),\ resourceEmpty(Y) \\
buildWall(X,Y) \leftarrow a2(X,Z),\ buildWall(Z,Y) \\
a2(X,Y) \leftarrow fetch(X,Z),\ putOnTopOf(Z,Y)
\end{array}
$$

Figure 5.26.: Column/wall building strategy learned from positive examples. $buildWall/2$ defines an action which transforms state X to state Y. *a2* is an invented predicate which defines another action composed of two subactions: $fecth/2$ and $putOnTop/2$

predicates are learned as Datalog definitions. Primitive predicates are non-Datalog background knowledge which manipulate situations as compound terms.

A wall is represented as a list of lists. Thus the wall shown in Figure 5.25a) can be encoded as $[[2,4],[1,3,5]]$, where each number corresponds to the position of a brick[15] and each sublist corresponds to a row of bricks. The primitive actions are *fetch* and *putOnTopOf*. The arguments of each action are states before and after the action. The primitive fluents are *resourceEmpty*, *offset* and *continuous* (meaning no gap). This is a simplified model, where the state of world is represented by a list containing: (1) object in Robot's hand, (2) number of bricks, (3) previous position where robot put the brick and (4) the wall constructed so far. Therefore a unit clause '$buildWall([empty,0,5,[]],[empty,X,\_,$[[2,4],[1,3,5]]]' represents a positive example which builds a stable wall corresponding to the one shown in Figure 5.25a.

When presented with only positive examples like those in Figure 5.25(a) and (b), Metagol$_D$ learns the recursive strategy shown in Figure 5.26. The invented action $a2$ is decomposed into subactions *fetch* and *putOnTopOf*. The strategy is non-deterministic and repeatedly fetches a brick and puts it on top of others so that it could produce either Figure 5.25a or 5.25b.

[15] Bricks are width 2 and position is a horizontal index.

$buildWall(X,Y) \leftarrow a2(X,Y),\ f1(Y)$
$buildWall(X,Y) \leftarrow a2(X,Z),\ buildWall(Z,Y)$
$a2(X,Y) \leftarrow a1(X,Y),\ f1(Y)$
$a1(X,Y) \leftarrow fetch(X,Z),\ putOnTopOf(Z,Y)$
$f1(X) \leftarrow offset(X),\ continuous(X)$

Figure 5.27.: Stable wall strategy built from positive and negative examples. a1, a2 and f1 are invented predicates. f1 can be interpreted as a post-condition of being stable.

Given negative examples Metagol$_D$ generates the refined strategy shown in Figure 5.27, where the invented action $a2$ tests the invented fluent $f1$. $f1$ can be interpreted as *stable*. This revised strategy will only build stable walls like Figure 5.25a.

Methods

An experiment was conducted to test performance of Metagol$_D$. Training and test examples of walls containing at most 15 bricks were randomly selected with replacement. Training set sizes were $\{2, 4, 8, 16, 32, 64\}$ and the test set size was 1000. Both training and test datasets contain half positive and half negative, thus the default accuracy is 50%. Predictive accuracies and associated learning times were averaged over 10 resamples for each training set size.

Results and discussion

MetagolD's accuracy and learning time plots shown in Figure 5.28 indicate that, consistent with the analysis in Section 5.3.1, Metagol$_D$, given increasing number of randomly chosen examples, produces rapid error reduction while learning time increases roughly linearly.

5.5. Discussions

There are other studies where abduction has been used for predicate invention, e.g.meta-level abduction [IFN10, IDN13] and TAL [Cor12, ABR12]. One important feature of MIL, which makes it distinct from the approach in [IFN10], is that it introduces new predicate symbols which represent

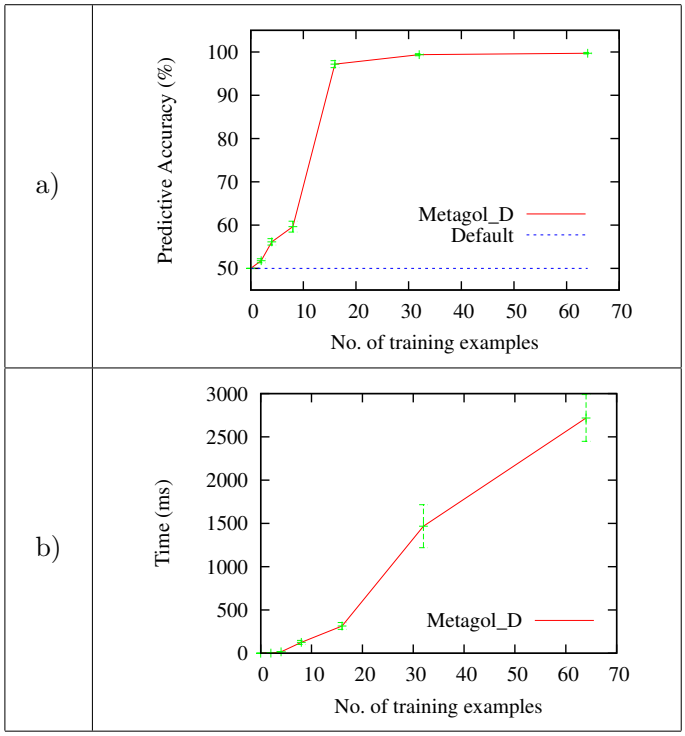


Figure 5.28.: Average a) Predictive accuracies and b) Learning times for robot strategy learning.

relations rather than new objects or propositions. This is critical for the problem of learning grammars in the form of a DCG, as well as challenging applications such as robot strategy learning. MIL is also distinguished from TAL [CRL10] by using meta-interpreters, as there is no meta-interpreter in TAL. Meta-interpreters allows the explicit encoding of meta-level control (e.g. procedural bias), which is key to the high efficiency of Metagol.

For the issues of what arity should be assigned to an invented predicate, MIL leaves them to search. Its procedural bias of "considering shorter hypotheses first" leads to the preference for the one leads to an overall minimal cardinality hypothesis. Although MIL is restricted to monadic and dyadic, H_2^2 can encode predicates with higher arity using the technique called currying [CF68]. This has been demonstrated in [Pan13].

In comparison to previous approaches to predicate invention one might question what is meant by the predicate symbols being *new*. In our case, we

assume a source containing either a finite or an infinite source (e.g. the natural numbers) of uninterpreted predicate symbols. Rather than providing these implicitly in hidden code (as was the case in CIGOL [MB88]), we prefer to have these symbols explicitly defined as part of the Herbrand universe of the meta-interpreter. Abductive hypothesis formation then provides the interpretation for these otherwise uninterpreted symbols.

As a declarative framework, MIL can be directly implemented using ASP. ASP_M does not require preprocess step for generating skeleton rules like that in ASPAL [CRL12, Cor12]. Despite the potential advantage in efficiency, the ASP implementation of MIL does not always outperform the Prolog implementation of MIL, as demonstrated by the experiments of this chapter. This is due to the effective procedural bias in the Prolog implementation.

5.6. Summary

Meta-Interpretive Learning (MIL) [MLPTN13, ML13] is a new framework which uses a Declarative Machine Learning [Rae12] description in the form of a set of Meta-rules, with procedural constraints incorporated within a Meta-Interpreter. The chapter explores the theory, implementation and experimental application of MIL. The language class supported by MIL was originally restricted to Types 2,3 in Chomsky hierarchy [MLPTN13], but has been extended to the Dyadic Datalog fragment H_2^2 [ML13]. This fragment is shown to be Turing expressive in the case of an infinite signature, but decidable otherwise. MIL supports hard tasks such as Predicate Invention and Recursion via abduction with respect to such a Meta-interpreter.

The MIL framework can be implemented using a simple Prolog program or within a more sophisticated solver such as ASP. We have applied these implementations to the problem of inductive inference of grammars, where our experiments indicate that they compete favourably in speed and accuracy with the state of the art ILP system MC-TopLog. When learning logic programs in H_2^2, where the hypothesis space is much larger than that of learning DCG grammars, efficiency of search is achieved by constraining the backtracking search in several ways. For instance, a Knuth-Bendix style total ordering can be imposed over the Herbrand base which requires predicates which are higher in the ordering to be defined in terms of lower ones. Alternatively, an interval inclusion ordering ensures finite termination

of (mutual) recursion in the case that the Herbrand Base is infinitely ascending but finitely descending. Additionally iterative deepening combined with logarithmic bounding of episodes guarantees polynomial-time searches which identify minimal cardinality solutions. Blumer-bound arguments are provided which indicate that search constrained in this way achieves not only speed improvements but also reduction in out-of-sample error.We have applied Metagol$_D$ to the problem of inductively inferring robot plans. In the planning task the Metagol$_D$ implementation used predicate invention to carry-out top-down construction of strategies for building both columns and stable walls. Experimental results indicate that rapid predictive accuracy increase is accompanied by polynomial (near linear) growth in search time with increasing training set sizes.

Finally it is worth noting that a Universal Turing Machine machine can be considered as simply a meta-interpreter incorporated within hardware. In this sense, meta-interpretation is one of, if not the most fundamental concept in Computer Science. Consequently we believe there are fundamental reasons that Meta-Interpretive Learning, which integrates deductive, inductive and abductive reasoning as higher-level operations within a meta-interpreter, will prove to be a flexible and fruitful new paradigm for logic-based Machine Learning, as well as the integration of deductive, inductive and abductive reasoning.

6. Related work

6.1. Logic programs as declarative bias

Considering all the other components (i.e. B, E and H) in ILP are represented by logic programs, having the declarative bias encoded as a logic program leads to the advantages of single-language representation, as discussed in Chapter 1. MOBAL [Mor93] is an ILP system which uses logic programs to encode declarative bias. Such logic programs are referred as rule models in MOBAL. However, rule models are not directly reasoned with background knowledge and examples. In contrast, the top theory first introduced in TopLog [MSTN08] is directly used in the refutation proof of examples, so that the derived hypotheses hold for $B \wedge H \models E$.

As a logic program, a top theory is more expressive than a mode declaration, such as encoding a strong bias about how the predicates should be connected. There do exist other forms of declarative bias that are comparable to the top theory in terms of their expressive power in encoding strong declarative bias. However, they are in the meta-level, such as antecedent description language (ADL) [Coh94] and its extension $\mathcal{D}$LAB [RD97]. In contrast, a top theory is in the object-level as a logic program. This makes it possible for a top theory to be reasoned directly with background knowledge, so that the derived hypotheses can be bound to those cover at least one positive example. A top theory is also similar to Spectre's [BIA94] starting-point theory (an overly general theory to be unfolded), but the top theory makes a clear distinction between terminal and non-terminal predicates. This is a powerful mechanism for distinguishing search control from the object language.

6.2. $\top$-directed approaches

$\top$-directed approaches perform induction and abduction via deductive search. This idea of viewing machine learning as completing deductive proofs by adding facts or rules to an incomplete KB goes back to Wilkins's work on explanation-based apprenticeship learning [Wil88]. However, explanation-based apprenticeship learning made single-clause learning assumption in order to restrict its search space. In addition, explanation-based apprenticeship learning is generate-and-test, rather than test-incorporation via examples like that in $\top$-directed approaches. The approaches in Abductive Logic Programming (ALP) also conduct abduction via deduction [KKT92, Poo87]. However, ALP systems do not have ordered search, but enumerate all consistent hypotheses.

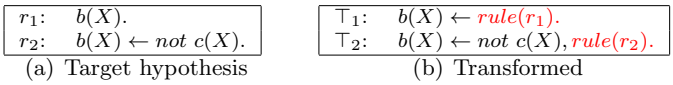
$\top$-directed approaches resemble Explanation-based Generalisation (EBG) [KCM87] in that both approaches find all possible explanations for the seed example first and then construct generalisations based on the derived explanations. However, EBG is essentially deductive learning, while $\top$-directed methods can achieve inductive learning. This is due to the fact that EBG's hypotheses are generalisations of background knowledge, while hypotheses derived by $\top$-directed methods hold for $\top \models H$ and $B \not\models H$.

A $\top$-directed approach called $\top$DHD was first introduced in the TopLog system [MSTN08]. TopLog requires that any predicates appearing in the head of some clause in a top theory $\top$ must not occur in the body of any clauses in B. This is essentially to avoid having clauses in $\top$ to be called by clauses in B. It is this restriction that prevent TopLog from learning multi-clause hypotheses. Thus TopLog is entailment-incomplete like Progol. The $\top$-directed approach was followed up in [Lin09] and this thesis, while TopLog was extended to MC-TopLog and Metagol in order to support multi-clause learning and predicate invention. The $\top$-directed approach was also adopted in TAL [CRL10]. Below are detailed comparisons between TAL and the two $\top$-directed systems introduced in this thesis.

6.2.1. MC-TopLog vs. TAL

TAL transforms an inductive learning task into an equivalent abductive one. Figure 6.1 shows an example of such transformation, which is taken from Chapter 1 of [Cor12]. With such transformation, inducing the non-ground

clause r_1 and r_2 becomes abducing the ground fact $rule(r_1)$ and $rule(r_2)$. Thus it can employ an abductive proof procedural, in particular SLDNFA, so that it supports non-monotonic reasoning [Cor12]. TAL also supports full clauses instead of restricting to definite clauses.



(a) Target hypothesis (b) Transformed

Figure 6.1.: Transformation in TAL [Cor12]

However, the co-generalisation technique introduced in ⊤DTcD has not been considered in TAL. Similar to the fact that TAL can outperform FOIL by bounding the search space to those covering at least one example, co-generalisation can also improve TAL's efficiency by bounding its search space to those covering more than one examples. This is illustrated in the example below.

Example 7 *Consider a learning task where there are five examples. The attributes of the examples are given in the table below. ⊤DTcD derives* $H_1 = \{oberservation(X) \leftarrow a3(X)\}$ *as the only candidate hypothesis, since* H_1 *is the only common generalisation of all positive examples. In contrast, other candidates like* $H_2 = \{oberservation(X) \leftarrow a1(X)\}$ *and* $H_3 = \{oberservation(X) \leftarrow a2(X)\}$ *are within the search space of TAL, although TAL will output* H_1 *as the final hypothesis.*

Training examples	*Background knowledge*			
observation(ex1).	*a1(ex1).*	*a2(ex1).*	*a3(ex1).*	
observation(ex2).	*a1(ex2).*		*a3(ex2).*	*a4(ex2).*
observation(ex3).		*a2(ex3).*	*a3(ex3).*	
observation(ex4).			*a3(ex4).*	*a4(ex4).*
¬observation(ex5).	*a1(ex5).*			

6.2.2. Metagol vs. TAL

Metagol, as an high-order extension of MC-TopLog, is distinct from TAL in terms of encoding the higher-order assumptions explicitly. Specifically, Metagol allows the explicit form of higher-order assumption to be encoded in the form of metarules. Predicate invention is an intrinsically higher-order

logic operation, since introducing new predicates requires instantiations of high-order variables.

6.3. Multi-clause learning

Progol's entailment-incompleteness was first pointed out by Yamamoto [Yam97]. Later this limitation of Progol is theoretically characterised in [RBR04]. In Progol, $\neg H$ is restricted to a set of unit clauses, thus the corresponding H is a single clause. CF-Induction [Ino04a] is a multi-clause learning approach that treats induction as consequence finding. It addressed Progol's entailment-incompleteness by not restricting $\neg H$ to a set of unit clauses. This results in an entailment-complete approach, but the cost is a dramatically increased search space. HAIL [RBR03] addressed Progol's entailment-incompleteness by integrating induction and abduction, while IMPARO [KBR09] address the entailment-incompleteness via a recursive inductive procedure, which is called *induction on failure*.

TAL and the two ILP systems introduced in this thesis address Progol's entailment-incompleteness via a $\top$-directed approach, which is an alternative to Inverse Entailment. The differences between TAL and the two ILP systems introduced in this thesis have been discussed in the last section. HYPER [Bra99] and TILDE [BR98] are another two ILP systems that support multi-clause learning. However, they are neither IE-based nor $\top$-directed. Thus their search spaces are not bound to those satisfying $B \wedge H \models e^+$. In addition, HYPER and TILDE do not support non-observational predicate learning, since they have not integrated abduction with induction. these systems have not been demonstrated to do abduction together with induction.

6.4. Common Generalisation

The idea of common generalisation was first introduced in Reynolds' Least Common Generalisation (LCG) [Rey69] and Plotkin's Least General Generalisation (LGG) [Plo71a]. However, LCG is restricted to atoms, while LGG is restricted to clauses, rather than theories with multiple clauses. Similarly, ProGolem [MSTN10], a recently developed ILP system extending from Golem [MF92] and Progol, also suffers from the entailment-incompleteness

as that in Progol. On the other hand, all the existing entailment-complete methods can only do solo-generalisation. For instance, CF-Induction [Ino04a], XHAIL [Ray09], IMPARO [KBR09] and TAL [CRL10]. Although CF-Induction and XHAIL can generalise multiple examples all at once, their search spaces are not bound to the common generalisations. The latest version of TAL [Cor12] uses all examples to guide its search, since it uses the compression based on all examples as heuristic. However, its seed example remain to be a single example. It has not been extended to multiple examples so that the search space can be directly constrained to common generalisations.

6.5. Predicate invention via abduction

There are studies where abduction has been used for predicate invention. For instance, [IFN10] assumed background knowledge such as the following.

$$\begin{aligned} caused(X,Y) &\leftarrow connected(X,Y). \\ caused(X,Y) &\leftarrow connected(X,Z), caused(Z,Y). \end{aligned}$$

Here the predicates *connected* and *caused* are both *meta-predicates* for object-level propositions g and s. Given multiple observations such as $caused(g,s)$ and $caused(h,s)$ abduction can be used to generate an explanation

$$\exists X(connected(g,X), connected(h,X), connected(X,s))$$

in which X can be thought of as a new propositional predicate. Therefore, [IFN10] is restricted to inventing propositional predicates.

TAL [CRL10], the ILP system mentioned earlier that does induction via abduction, can be extended for predicate invention using placeholders [Cor12]. This is further explored in [ABR12], where the ASP version of TAL, ASPAL [CRL12], is used. Similarly, HYPER [Bra99] is also extended to do predicate invention using placeholds, as demonstrated in [GL08, Bra10]. The approach of using placeholders still does not solve the problem of how to control the dramatically increased hypothesis space. A potential solution based on iterative learning [ACBR13] has been discussed in the future work

session of [ABR12]. Specifically, generating all placeholders first and then iteratively increasing the placeholders input to an ILP system.

Predicate invention is considered for the purpose of theory reformulation and bias shift in [ABR12]. Specifically, theory reformulation is "to identify interesting concepts not directly related to the learning goal that could be used to restructure the program" [ABR12], while bias shift is "to specialise an overgeneral hypothesis and make it consistent with the examples" [ABR12]. Both theory reformulation and bias shift via predicate invention can be accommodated by Metagol$_D$.

Learning a definition of staircase with predicate invention is an example of theory reformulation. As shown in Figure 5.24, without predicate invention, a definition of staircase would have the pattern "*perpendicular parallel*" reoccurring twice. In contrast, Metagol$_D$ suggests a definition, where its invented predicate defines the reoccurring pattern. The resulting hypothesis would be more compressed if the frequency of reoccurrence is high.

Learning Regular grammar is an example of bias shifting. Without inventing predicates corresponding to states, a hypothesised Regular grammar can have only one state as a starting state. Thus the resulting grammar accepting strings with a mixture of 0 and 1 would correspond to a universal grammar, which accept all strings including negative examples. Therefore, Metagol$_D$ shifts its bias by invented new predicates in order specialise a hypothesised grammar. For example, when learning the parity grammar, additional state $q1$ is invented by Metagol$_D$.

6.6. Global optimisation

Finding a hypothesis that achieving maximum compression involves combinatorial search. Therefore many ILP systems restricted to learning definite logic programs use the covering algorithm, where clauses compressed from a *subset* of examples are added to the final H iteratively. There are ILP systems like HYPER [Bra99], in which multiple clauses compressed from the *whole* set of examples are refined together. However, HYPPER performs a best-first search, which will become inefficient when there are many plateau in the search space. In contrast, meta-interpretive learning approaches find globally optimal hypothesis either by iterative deepening search in the case of Prolog implementation or ASP solvers' optimisation component in the

case of ASP implementation.

6.7. Leveraging ASP for ILP

ASP was first used for ILP in [Sak01, Sak05], where the aim is to extend ILP for learning nonmonotonic logic programs. However, the approach in [Sak01, Sak05] has difficulty in generalising from multiple examples, especially when the multiple examples are a mixture of positive and negative examples. ASPAL[CRL12, Cor12] further explores the use of ASP for nonmonotonic ILP. It also leverages ASP solvers' high-performance by delegating the hypothesis search to an ASP solver. The learning process in ASPAL involves two main steps. First, a preprocess is required in ASPAL for generating skeleton rules. Then an ASP solver (CLASP) is applied for hypothesis search.

6.8. Grammatical inference methods

The problem of learning or inferring Regular languages, which can be represented by deterministic finite state automata, has been well studied and efficient automaton-based learning algorithms have existed since the 1950s [Moo56]. Some heuristic approaches to machine learning context-free grammars [VB87, LS00] have been investigated, though the completeness of these approaches is unclear. Although an efficient and complete approach exists for learning context-free grammars from parse trees [Sak92], no comparable complete approach exists in the literature for learning context-free grammars from positive and negative samples of the language. According to a recent survey article learning context-free languages is widely believed to be intractable and the state of the art mainly consists of negative results [dlH05]. There are some positive PAC (probably approximately correct) learning results concerning Regular languages (e.g. [Den01]), but to the best of the author's knowledge, these have not been extended to the context-free case. The difficulty of learning context-free languages arises from a very large search space compared to regular languages.

ILP, among other learning methods, has previously been applied to grammatical inference (e.g. [Bos98]). Although [CP00] has shown that natural language grammars are learnable using ALEPH, its learning setting

avoids predicate invention by assuming all predicates like *np* (noun phrase) are known in the background knowledge. Additionally, the entailment-incompleteness of ALEPH restricts the applicability of the approach.

7. Conclusions and future work

7.1. Conclusions

This thesis considers the use of logic programs as declarative and procedural bias in ILP. The corresponding theory, implementation and experimental application are explored.

7.1.1. Top theory and multi-clause learning

We first consider using a top theory [MSTN08] to encode declarative bias. The resulting ILP system is called MC-TopLog. As a logic program, top theory has the expressivity to encode strong declarative bias. Additionally, being in the object level makes it possible for a declarative bias to be reasoned directly with background knowledge and examples, so that MC-TopLog's search space can be easily constrained to common generalisations. Due to these advantages, MC-TopLog is able to overcome the limitation of entailment-incompleteness, which exists in many ILP systems including Progol. Entailment-incompleteness means the restriction of generalising a single example to a single clause, but not multiple clauses. Therefore it is referred as single-clause learning (SCL) in this thesis. Conversely, entailment-completeness is without such restriction and referred as multi-clause learning (MCL). Multi-clause learning includes non-observational learning and learning recursive definitions without successive examples. The following are demonstrated by the experiments in Chapter 3:

- When learning natural language grammar with sparse background knowledge, where non-observational predicate learning is necessary, MC-TopLog outperforms Progol with significantly higher predictive accuracy. This is due to MC-TopLog's ability to effectively integrate induction and abduction so that non-observational learning is no longer an issue. The experiment also shows the advantage of MC-

TopLog's co-generalisation over its solo-generalisation in terms of efficiency.

- When learning the winning strategy for Nim game, MC-Toplog effectively derives the target hypothesis involving recursion. Thus it has significantly higher predictive accuracies than Progol, which fails to derive any recursive clause from non-successive examples. This demonstrates MC-TopLog's capability of learning recursive definitions even when successive examples are not provided. Additionally, MC-TopLog's co-generalisation is key to the success of this experiment, because it reduces the search space by at least a factor of 10 while solo-generalisation fails in this experiment because of an excessively large search space.

MC-TopLog is also applied to two real-world applications: tomato and predictive toxicology projects. These two projects are both funded by Syngenta, which is an agriculture company. The goals of both projects are to identify metabolic control reactions. The use of ILP in the two real-world problems supported efficient analysis of the biological data. Additionally, interesting hypotheses were produced that are different from what the biologists had stated prior to the machine learning. In both applications, MC-TopLog's hypotheses were also compared against human hypotheses provided by the Syngenta project leaders. It was noted that the human hypotheses were closer in form to the reductionist hypotheses generated by Progol. In several cases the MC-TopLog hypotheses were both more complex and more accurate than the human ones and indicated quite distinct control points within the relevant sub-networks.

Our experiments also show the necessity of multi-clause learning in a real-world application. There do exist datasets in which systems-level hypotheses derived by MCL have significantly higher predictive accuracies than the reductionist ones derived by SCL. On the other hand, MCL does not outperform SCL all the time due to the existence of good approximations to the target hypothesis within SCL's hypothesis space. In this case, it seems not worth applying MCL considering that MCL is much more computationally expensive than SCL. However, for real-world applications where it is unclear whether the unknown target theory or its approximations are within the hypothesis space of the incomplete learner, it is worth trying MCL. Since there

are datasets where neither the target theory nor its approximations exist within the hypothesis space of SCL, thus MCL has the potential to improve the learning results of SCL.

7.1.2. Meta-interpretor and predicate invention

A meta-interpreter is essentially a higher-order top theory. It is extended from top theory by projecting predicate symbols into existentially quantified variables and hypothesis clauses into atomic formulae in a meta-interpreter. This extension allows an ALP or ILP[1] system loaded with a meta-interpreter to become capable of predicate invention. On the other hand, doing predicate invention leads to the challenge of an even larger search space than multi-clause learning because of introducing new predicate symbols. Although meta-interpreters can encode strong declarative bias like top theories, it is not sufficient to deal with this challenge. Therefore in this thesis we consider incorporating procedural bias in a direct implementation of MIL.

We explore the possibility of encoding procedural bias using Prolog's procedural semantics. This turns out to be both feasible and effective, according to our experiments with grammar inference, which requires predicate invention. Specifically, in all of the experiments of learning Regular, Context-free and natural language grammars, Metagol significantly outperforms MC-TopLog[2] in terms of both efficiency and predictive accuracy. The procedural bias is so powerful that in most cases Metagol is not surpassed by ASP_{M}, which is an implementation of MIL using an efficient constraint solver. The procedural bias considered in Metagol are applicable to any concept learning tasks and widely used in concept learning. For example, specific-to-general search and following Occam's razor to consider shorter hypotheses first.

Apart from the advantage of supporting predicate invention, MIL also has the advantages of being a declarative machine learning description. In particular, the goal of finding an hypothesis that satisfies all constraints is taken care by the declarative language that implements MIL. Moreover, declarative implementations of MIL can leverage the high-efficiency of special-purpose solvers. For example, in the experiment of learning nat-

[1]The ILP systems referred here are those that have these two features: (1) capable of doing abduction ; (2) not restricted to single-clause learning.
[2]MC-TopLog is loaded with a meta-interpreter.

ural language grammars, ASP_M is shown to have speed advantage over Metagol when background knowledge is sparse. The optimisation component in ASP is also handy for finding an hypothesis that is globally optimal in terms of description length.

7.1.3. Top theory *vs.* Meta-interpreter

As an extension of top theories, meta-interpreters resemble top theories. In particular, as with top theories, ILP systems loaded with meta-interpreters are capable of learning mutual recursions, as well as integrating induction and abduction. On the other hand, meta-interpreters support predicate invention, which is different from top theories. This subsection summarises the differences between top theories and meta-interpreters from the perspective of encoding declarative and procedural bias.

Declarative bias

Top theories and meta-interpreters are both logic programs, but top theories are in the first-order while meta-interpreters encodes HOL programs despite in the form of FOL. It is this difference that makes meta-interpreters expressive enough to define an hypothesis space that supports predicate invention and higher-order logic learning. Thus meta-interpreters are more expressive than top theories in terms of encoding declarative bias. Being in the higher-order also makes a meta-interpreter more compact than its corresponding first-order top theory. For example, the top theory in Figure 6.1(b), which contains twelve clauses, can be encoded by a meta-interpreter with only four meta-rules.

Procedural bias

As logic programs, both top theories and meta-interpreters can encode procedural bias via Prolog's procedural semantics. However, there are procedural biases not encodable by top theories, since some procedural biases can only be effective via meta-level control. For example, the procedural bias of "consider shorter hypotheses first" can not be captured by the order of clauses or literals in a Prolog program. Meta-interpreters' advantage in encoding procedural bias comes from its separation from hypotheses. Specifically, a top theories $\top$ entails H, as described in Equation 7.1. By

contrast, a meta-interpreter B_M does not entail H, but together with H it can explain positive examples with respect to background knowledge B_A, as described in Equation 7.3. Therefore a meta-interpreter can readily encode any meta-level control over a search space.

Note that B_M in Equation 7.3 explicitly represents inductive bias. It is separated from the traditional background knowledge, which is captured by B_A. This is different from previous ILP framework, (see Equation 7.2 below), where the component of inductive bias is implicit. Therefore, Equation 7.3 from the MIL framework describes the relations among ILP components in a more accurate way. Additionally, explicitly separating inductive bias from other background knowledge makes it a user-definable component, as opposed to implicitly built-in to a system and not modifiable by users.

$$\top \models H \tag{7.1}$$

$$B, H \models E^+ \tag{7.2}$$

$$B_A, B_M, H \models E^+ \tag{7.3}$$

7.2. Future Work

7.2.1. Noise Handling

The current MIL framework assumes there is no noise in the given examples, while this is not necessarily the case in real-world applications. This issue can be addressed by loosing the constraint that a derived hypothesis should explain all the positive while none of the negative. Specifically, loosening this constraint means an example is allowed to be succeeded without proving it either deductively or inductively by adding new clauses. In terms of how many examples are allowed to be skipped in this way, it can be decided by whether the compression is achievable. Loosing this constraint also means enlarging the search space. Since the search involves hypothesising which examples are noise.

7.2.2. Probabilistic MIL

In order to learn probabilistic grammars or deal with uncertainties in real-world application, the MIL framework needs to incorporate probabilities.

Possible representation for such probabilistic MIL includes SLPs [Mug01], Blog [MR06] and Problog [RKT07, dBTvO10]. The advantage of Probablistic MIL over previous Statistical Relational Learning [MP07] approaches is integrating model construction and probability estimation.

7.2.3. Implementing MIL using other declarative languages

Metagol and ASP_M are not the only possible implementations of MIL. They are just two examples of how MIL can be directly implemented using declarative languages. Other declarative languages like Constraint Logic Programming (CLP) [JM94] can be used to implement MIL as well. In particular, the constraint store in CLP can be used to implement a dynamic declarative bias. Dynamic declarative bias means it is not pre-specified, but built up along the search.

7.2.4. Learning declarative and procedural bias

Declarative bias is critical for a learning algorithm. Since a strong declarative bias leads to a smaller hypothesis space, which will lower the error bound according to the Blumer-bound argument. However, this is only the case when the target hypothesis or its approximations are not ruled out by the bias. Thus the question is where does the strong declarative bias come from and how do we guarantee that it still contains the target hypothesis or its approximations. It has been shown that declarative bias can be acquired by learning [Utg86, Coh92, BT07], which is called learning to learn [Utg86]. Although the learning of procedural bias has not been explored before, it can be learned as well since it has been shown in [Sil86, GS81, MUB83] that it is feasible to learn control information for search in other AI tasks.

This thesis proposes the use of logic programs for both declarative and procedural bias, which makes the framework of learning to learn straightforward due to a single-language representation.

7.2.5. Future work for biological applications

The metabolic control reactions hypothesised by MC-TopLog need to be validated. Literature matching is one way, but there are novel hypotheses that can not find support from existing literatures. Experimental tests of

those novel hypotheses would be ideal. Unfortunately, this is practically infeasible due to constraints of time[3] and money. However, there is an indirect way for validating the hypotheses in the tomato project. Since Quantitative Trait Loci (QTLs) [SSR+06] data is available for the tomato project. QTLs are stretches of DNA containing or linked to the genes that underlie a quantitative trait. Although the region of QTL is too large to pinpoint genetic control points, it could be used to see whether the suggested control reactions have their corresponding genes mapped to that regions. In addition, our approach could be generalised to study metabolic perturbations in other systems in response to a range of metabolic perturbations given 'omics' data. The mechanistic insight gained from the hypotheses could be exploited to investigate specific regulatory events or feedback control exerted by the intermediate metabolites.

[3]Doing actual experiments to test hypotheses about metabolic control points in tomato ripening will take years.

Appendices

A. MIL systems

A.1. Metagol$_{CF}$

```
:-use_module(library(lists)).

parse(S,G1,G2,S1,S2,K1,K2) :-
        parse(s(0),S,[],G1,G2,S1,S2,K1,K2).

parse(Q,X,X,G1,G2,S,S,K1,K2) :-
        abduce(acceptor(Q),G1,G2,K1,K2).
parse(Q,[C|X],Y,G1,G2,S1,S2,K1,K2) :-
        skolem(P,S1,S3),
        abduce(delta1(Q,C,P),G1,G3,K3,K2),
        parse(P,X,Y,G3,G2,S3,S2,K3,K2).
parse(Q,X,Y,G1,G2,S1,S2,K1,K2) :-
        skolem(P,S1,S3),
        abduce(delta2(Q,P,C),G1,G3,K1,K3),
        suffix([C|Y],X),[C|Y]\==X,
        parse(P,X,[C|Y],G3,G2,S3,S2,K3,K2).
parse(Q,X,Y,G1,G2,S1,S2,K1,K2) :-
        skolem(P,S1,S3),skolem(R,S1,S3),
        abduce(delta3(Q,P,R),G1,G3,K1,K3),
        suffix(Z,X),Z\==X,
        parse(P,X,Z,G3,G4,S3,S4,K3,K4),
        parse(P,Z,Y,G4,G2,S4,S2,K4,K2).

abduce(X,G,G,K,K) :- member(X,G).
abduce(X,G,[X|G],s(K),K) :- not(member(X,G)).

skolem(s(N),[s(Pre)|SkolemConsts],[s(N),s(Pre)|SkolemConsts]):- N is Pre+1.
skolem(S,SkolemConsts,SkolemConsts):-member(S,SkolemConsts).
```

A.2. Metagol$_D$

```
%=============================================================
% Prover for mono-dyadic definite clauses
prove_d1(P,X,Y,Sig,G1,G2,N,M) :-
        element(Q,Sig), P@>Q,
        abduce(rule3_d1(P,Q),G1,G3,N,N1),
        prove_d1(Q,X,Y,Sig,G3,G2,N1,M).
                % P(X,Y) <- Q(X,Y)
prove_d1(P,X,Y,Sig,G1,G2,N,M) :-
        element(Q,Sig), P@>Q, element(R,Sig), P@>=R,
        object_d1(Z), X @> Z, Z@> Y,
        abduce(rule5_d1(P,Q,R),G1,G3,N,N1),
        prove_d1(Q,X,Z,Sig,G3,G4,N1,N2),
        prove_d1(R,Z,Y,Sig,G4,G2,N2,M).
                % P(X,Y) <- Q(X,Z), P(Z,Y)
prove_d1(P,X,Y,Sig,G1,G2,N,M) :- Y@>X,
        abduce(rule9_d1,G1,G3,N,N1),
        prove_d1(symmetric,P,Sig,G3,G4,N1,N2),
        prove_d1(P,Y,X,Sig,G4,G2,N2,M).
                % P(X,Y) <- symmetric(P), P(Y,X)
prove_d1(P,X,Y,Sig,G1,G2,N,M) :-
        abduce(rule2_d1(P,X,Y),G1,G2,N,M).
                % P(X,Y) <-
prove_d1(P,X,Sig,G1,G2,N,M) :-
        element(Q,Sig), [P,X] @> [Q,X],
        abduce(rule1_d1(P,Q),G1,G3,N,N1),
        prove_d1(Q,X,Sig,G3,G2,N1,M).
                % P(X) <- Q(X)
prove_d1(P,X,Sig,G1,G2,N,M) :-
        abduce(rule0_d1(P,X),G1,G2,N,M).
                % P(X) <-
%=============================================================

abduce(X,G,G,N,N) :- X, !.
abduce(X,G,G,N,N) :- element(X,G), !.
abduce(X,G,[X|G],s(N),N) :- not(element(X,G)), !.

prove_d0(P,X,Sig,G,G,N,N) :-
        prove_d1(P,X,Sig,G,G,N,N), !.   % Deductive proof Monadic
prove_d0(P,X,Sig,G1,G3,N,N1) :-
        prove_d1(P,X,Sig,G1,G3,N,N1). % Abductive proof Monadic
prove_d0(P,X,Y,Sig,G,G,N,N) :-
        prove_d1(P,X,Y,Sig,G,G,N,N), !. % Deductive proof Dyadic
prove_d0(P,X,Y,Sig,G1,G3,N,N1) :-
        prove_d1(P,X,Y,Sig,G1,G3,N,N1). % Abductive proof Dyadic

% Find G that fits all the positive epsisodes
proveall([],_,G,G,N) :- !.
proveall([[P,X]|T],Sig,G1,G2,N) :-
                prove_d0(P,X,Sig,G1,G3,N,N1),
```

```
        write('EXAMPLE:'), pprint([rule0_d1(P,X)]),
        write('HYPOTHESIS: ['), peano(N0,N1), write(N0),
        write(' clauses left]'), pprint(G3),
                proveall(T,Sig,G3,G2,N1).
proveall([[P,X,Y]|T],Sig,G1,G2,N) :-
                prove_d1(P,X,Y,Sig,G1,G3,N,N1),
        write('EXAMPLE:'), pprint([rule2_d1(P,X,Y)]),
        write('HYPOTHESIS: ['), peano(N0,N1), write(N0),
        write(' clauses left]'), pprint(G3),
        proveall(T,Sig,G3,G2,N1).

% Check G fits all the negative epsisode
nproveall([],_,_) :- !.
nproveall([[P,X]|T],Sig,G) :-
        not(prove_d1(P,X,Sig,G,G,0,0)),
        nproveall(T,Sig,G), !.
nproveall([[P,X,Y]|T],Sig,G) :-
        not(prove_d1(P,X,Y,Sig,G,G,0,0)),
        nproveall(T,Sig,G), !.


% Generate a Peano number
peano(0,0) :- !.
peano(N,s(M)) :- not(var(N)), !, N1 is N-1, peano(N1,M), !.
peano(N,s(M)) :- var(N), !, peano(N1,M), N is N1+1, !.

append([],L,L).
append([H|T],L,[H|R]) :- append(T,L,R).

reverse(L1,L2) :- reverse1(L1,[],L2).

reverse1([],L,L) :- !.
reverse1([H|T],[H|R],L) :- reverse1(T,R,L).

element(H,[H|_]).
element(X,[_|T]) :- element(X,T).

% rev_union builds the union of the two signatures in reverse sorted order
rev_union(Sig1,Sig2,Sig3) :-
        append(Sig1,Sig2,Sig4),
        sort(Sig4,Sig5),
        reverse(Sig5,Sig3), !.

% Extract the signature from background atoms
sig_extract([],Sig,Sig) :- !.
sig_extract([Rule|T],Sig,Sig1) :-
        Rule =.. [Rule],
        sig_extract(T,Sig,Sig1), !.
sig_extract([Rule|T],Sig,[P|Sig1]) :-
        arg(1,Rule,P),
        sig_extract(T,Sig,Sig1), !.
```

```
% Learn from sequence of example episodes,
% where theory size is bounded by episode size/2.
learn_seq([],_,B,B) :- !.
learn_seq([E|T],Sig,BK,Hyp) :-
        epsisode(E,Pos,Neg,_), length(Pos,P), length(Neg,N),
        M2 is floor(log(P+N)/log(2)), interval(1,M2,I),
        learn_episode(E,I,Sig,Sig1,BK,Hyp1),
        learn_seq(T,Sig1,Hyp1,Hyp), !.
learn_seq([E|_],_,_,_) :-
        epsisode(E,Pos,Neg,_), length(Pos,P), length(Neg,N),
        M2 is floor(log(P+N)/log(2)),   % Logarithmic clause bound
        write('EPISODE '), write(E),
        write(': NO COMPRESSION FOR CLAUSE BOUND UP TO '),
        write(M2), nl, nl.

interval(Lo,Hi,[Lo|T]) :-
        Lo=<Hi, Lo1 is Lo+1,
        interval(Lo1,Hi,T), !.
interval(_,_,[]).

% Learn from a single example episode and update the signature
learn_episode(Tag,Int,Sig,Sig5,B1,B2) :-
        write('EXAMPLE EPISODE: '), write(Tag), nl, nl,
        epsisode(Tag,Pos,Neg,Sig1),
        rev_union(Sig,Sig1,Sig2),       % Combine signatures
        element(N,Int), peano(N,Lim),
        write('TRY CLAUSE BOUND: '), write(N), nl, nl,
        proveall(Pos,Sig2,B1,B2,Lim), nproveall(Neg,Sig2,B2),
        sig_extract(B2,Sig2,Sig3),      % Extract new signature
        sort(Sig3,Sig4), reverse(Sig4,Sig5),
        write('FINAL HYPOTHESIS FOR EPISODE: '),  write(Tag),
        write(', BOUND: '), write(N),
        pprint(B2), !.
```

A.3. $\text{ASP}_{\mathbf{M}_{CF}}$

```
% Instances
skolem(0..maxNumSkolemConstants).
terminal(0;1).

% Generate: specify the hypothesis space
{acceptor(NT):skolem(NT)}.
{delta1(NT1,T,NT2):skolem(NT1):terminal(T):skolem(NT2)}.
{delta2(NT1,NT2,T):non_terminal(NT1):non_terminal(NT2):terminal(T)}.
{delta3(NT1,NT2,NT3):non_terminal(NT1):non_terminal(NT2):non_terminal(NT3)}.

% Defining Part
parse(ExID,Position,Position,NT):-
        length(ExID,MaxLengh),
        Position=0..MaxLengh,acceptor((NT).
parse(ExID,Position1,Position2,NT1):-
        seqT(ExID,Position1,T),
        delta1(NT1,T,NT2),
        parse(ExID,Position1+1,Position2,NT2).
parse(ExID,Position1,Position2+1,NT1):-
        parse(ExID,Position1,Position2,NT2),
        delta2(NT1,NT2,T),
        seq(ExID,Position2,T).
parse(ExID,Position1,Position3,NT1):-
        delta3(NT1,NT2,NT3),
        parse(ExID,Position1,Position2,NT2),
        parse(ExID,Position2,Position3,NT3).

% Integrity constraint
:- negEx(ExID), length(ExID,MaxLengh), parse(ExID,0,MaxLengh,0)).
:- posEx(ExID), length(ExID,MaxLengh), not parse(ExID,0,MaxLengh,0).

% Optimisation
#minimize [delta1(NT1,T,NT2):non_terminal(NT1):terminal(T):non_terminal(NT2)=1,
delta2(NT1,NT2,T):non_terminal(NT1):non_terminal(NT2):terminal(T)=1,
delta3(NT1,NT2,NT3):non_terminal(NT1):non_terminal(NT2):non_terminal(NT3)=1,
acceptor((NT):non_terminal(NT)= 1].
```

Bibliography

[ABR12] D. Athakravi, K. Broda, and A. Russo. Predicate invention in inductive logic programming. In *2nd Imperial College Computing Student Workshop*, pages 15–21, 2012.

[ACBR13] D. Athakravi, D. Corapi, K. Broda, and A. Russo. Learning through hypothesis renement using answer set programming. In *Proceedings of the 23st International Conference on Inductive Logic Programming*, 2013. Accepted.

[AKMS12] B. Andres, B. Kaufmann, O. Matheis, and T. Schaub. Unsatisfiability-based optimization in clasp. In *Proceedings of the 28th International Conference on Logic Programming*, 2012.

[BCLM+11] D.A. Bohan, G. Caron-Lormier, S.H. Muggleton, A. Raybould, and A. Tamaddoni-Nezhad. Automated discovery of food webs from ecological data using logic-based machine learning. *PloS ONE*, 6(12), 2011.

[BEHW89] A. Blumer, A. Ehrenfeucht, D. Haussler, and M.K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM*, 36(4):929–965, 1989.

[BIA94] H. Boström and P. Idestam-Almquist. Specialisation of logic programs by pruning SLD-trees. In S. Wrobel, editor, *Proceedings of the Fourth Inductive Logic Programming Workshop (ILP94)*, pages 31–48, Bonn, 1994. GDM-studien Nr. 237.

[Bos98] H. Boström. Predicate invention and learning from positive examples only. In *10th European Conference on Machine Learning (ECML-98)*, pages 226–237. Springer, 1998.

[BR98] H. Blockeel and L. De Raedt. Top-down induction of first order logical decision trees. *Artificial Intelligence*, 101(1–2):285–297, 1998.

[Bra86] I. Bratko. *Prolog for artificial intelligence*. Addison-Wesley, London, 1986.

[Bra99] I. Bratko. Refining complete hypotheses in ILP. In *Proceedings of ILP-99*, volume 1634, pages 44–55, Berlin, 1999. Springer-Verlag.

[Bra10] I. Bratko. Discovery of abstract concepts by a robot. In *Proceedings of Discovery Science 2010*, LNAI 6332, pages 372–379, Berlin, 2010. Springer-Verlag.

[Bra11] I. Bratko. *Prolog Programming for Artificial Intelligence*. Addison-Wesley, 2011. Fourth edition.

[BT07] W. Bridewell and L. Todorovski. Learning declarative bias. In *Proceedings of the 17th International Conference on Inductive Logic Programming*, pages 63–77, Berlin, 2007. Springer-Verlag. LNAI 4894.

[Bun86] W. Buntine. Generalised subsumption. In *Proc. of the 7th European Conference on Artificial Intelligence (ECAI-86)*. European Coordinating Committee for Artificial Intelligence, 1986.

[CF68] H. Curry and R. Feys. *Combinatory logic*. Amsterdam, Netherlands: North-Holland, 1968. Volume I.

[Coh92] W. Cohen. Compiling knowledge into an explicit bias. In *Proceedings of the 9th International Conference on Machine Learning*. Morgan Kaufmann, 1992.

[Coh93] W. Cohen. Learnability of restricted logic programs. In S. Muggleton, editor, *Proceedings of the 3rd International Workshop on Inductive Logic Programming (Technical report IJS-DP-6707 of the Josef Stefan Institute, Ljubljana, Slovenia)*, pages 41–72, 1993.

[Coh94] W. Cohen. Grammatically biased learning: Learning logic programs using an explicit antecedent description language. *Artificial Intelligence*, 68:303–366, 1994.

[Cor12] D. Corapi. *Nonmonotonic Inductive Logic Programming as Abductive Search*. PhD thesis, Imperial College London, 2012.

[Cos] V.S. Costa. YAP. http://www.dcc.fc.up.pt/vsc/Yap.

[CP95] W. Cohen and C.D. Page. Polynomial learnability and Inductive Logic Programming: methods and results. *New Generation Computing*, 13:369–409, 1995.

[CP00] J. Cussens and S. Pulman. Experiments in inductive chart parsing. In J. Cussens and S. Dzeroski, editors, *Proceedings of Learning Language in Logic (LLL2000)*, LNAI 1925, pages 143–156. Springer-Verlag, 2000.

[CRL10] D. Corapi, A. Russo, and E. Lupu. Inductive logic programming as abductive search. In *ICLP2010 Technical Communications*, Berlin, 2010. Springer-Verlag.

[CRL12] D. Corapi, A. Russo, and E. Lupu. Inductive logic programming in answer set programming. In *Proceedings of the 21st International Conference on Inductive Logic Programming*, LNAI 7207, pages 91–97, 2012.

[CSIR12] D. Corapi, D. Sykes, K. Inoue, and A. Russo. Probabilistic rule learning in nonmonotonic domains. In *12th International Workshop on Computational Logic in Multi-Agent Systems, (CLIMA XII)*, LNAI 6814, pages 243–258, 2012.

[dBTvO10] G. Van den Broeck, I. Thon, and M. van Otterlo. DTProbLog: A decision-theoretic probabilistic Prolog. In *Proceedings of the Twenty-Fourth National Conference on Artificial Intelligence, AAAI10*, pages 1217–1222, 2010.

[Den01] F. Denis. Learning regular languages from simple positive examples. *Machine Learning*, 44(1):37–66, 2001.

[dlH05] C. de la Higuera. A bibliographical study of grammatical inference. *Pattern Recognition*, 38:1332–1348, 2005.

[DMR93] S. Džeroski, S.H. Muggleton, and S. Russell. Learnability of constrained logic programs. In *Proceedings of the European Conference on Machine Learning*, pages 342–347, London, UK, 1993. Springer-Verlag.

[FCS04] A.R. Fernie, F. Carrari, and L.J. Sweetlove. Respiratory metabolism: glycolysis, the TCA cycle and mitochondrial electron transport. *Current Opinion in Plant Biology*, 7:254–261, 2004.

[FK00] P. A. Flach and A. C. Kakas, editors. *Abductive and Inductive Reasoning*. Pure and Applied Logic. Kluwer, 2000.

[Flo02] C. Florêncio. Consistent identification in the limit of rigid grammars from strings is np-hard. *Grammatical Inference: Algorithms and Applications*, pages 729–733, 2002.

[FP08] A.M. Feist and B.? Palsson. The growing scope of applications of genome-scale metabolic reconstructions using escherichia coli. *Nature Biotechnology*, 26:659–667, 2008.

[FS12] R. Farid and C. Sammut. Plane-based object categorization using relational learning. *ILP2012 MLJ special issue*, 2012. To appear.

[Gel08] M. Gelfond. Answer sets. In F.V. Harmelen, V. Lifschitz, and B. Porter, editors, *Handbook of Knowledge Representation*, pages 285–316. Elsevier, 2008.

[GKKS12] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan and Claypool Publishers, 2012.

[GKNS07] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. clasp: A conflict-driven answer set solver. In Chitta Baral, Gerhard Brewka, and John Schlipf, editors, *Logic Programming and*

Nonmonotonic Reasoning, volume 4483 of *Lecture Notes in Computer Science*, pages 260–265. Springer Berlin / Heidelberg, 2007.

[GL88] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. Kowalski and K. Bowen, editors, *Proceedings of the Fifth International Conference and Symposium of Logic Programming (ICLP88)*, pages 1070–1080. MIT Press, 1988.

[GL08] I. Bratko G. Leban, J. Žabkar. An experiment in robot discovery with ilp. In *Proceedings of the 18th International Conference on Inductive Logic Programming (ILP 2008)*, volume 5194. Springer-Verlag, 2008.

[Gol67] E.M. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.

[GS81] M.R. Genesereth and D. Smith. A learnability model for universal representations. Technical Report Memo HPP-81-6, Stanford University, Stanford, CA, 1981.

[GT07] L Getoor and B. Taskar, editors. *Introduction to Statistical Relational Learning*. MIT Press, Cambridge, Massachusetts, 2007.

[HU79] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata and Formal Languages*. Addison-Wesley, Reading, MA, 1979.

[Hue75] G. Huet. A unification algorithm for typed λ-calculus. *Theoretical Computer Science*, 1(1):27–57, 1975.

[IDN13] Katsumi Inoue, Andrei Doncescu, and Hidetomo Nabeshima. Completing causal networks by meta-level abduction. *Machine Learning*, 91(2):239–277, 2013.

[IFN10] K. Inoue, K. Furukawa, and I. Kobayashiand H. Nabeshima. Discovering rules by meta-level abduction. In L. De Raedt, editor, *Proceedings of the Nineteenth International Conference on Inductive Logic Programming (ILP09)*, pages 49–64, Berlin, 2010. Springer-Verlag. LNAI 5989.

[Ino04a] K. Inoue. Induction as consequence finding. *Machine Learning*, 55:109–135, 2004.

[Ino04b] K Inoue. Induction as consequence finding. *Machine Learning*, 55:109–135, 2004.

[ISI+09] K. Inoue, T. Sato, M. Ishihata, Y. Kameya, and H. Nabeshima. Evaluating abductive hypotheses using an em algorithm on bdds. In *IJCAI-09: Proceedings of the Twenty-first International Joint Conference on Artificial Intelligence*, pages 810–815, San Mateo, CA:, 2009. Morgan-Kaufmann.

[JM94] J. Jaffar and M.J. Maher. Constraint logic programming: a survey. *Journal of Logic Programming*, 19/20:503–582, 1994.

[KB70] D. Knuth and P. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon, Oxford, 1970.

[KBR09] T. Kimber, K. Broda, and A. Russo. Induction on failure: Learning connected Horn theories. In *LPNMR 2009*, pages 169–181, Berlin, 2009. Springer-Verlag.

[KCM87] S.T. Kedar-Cabelli and L.T. McCarty. Explanation-based generalization as resolution theorem proving. In P. Langley, editor, *Proceedings of the Fourth International Workshop on Machine Learning*, pages 383–389, Los Altos, 1987. Morgan Kaufmann.

[KKT92] A.C. Kakas, R.A. Kowalski, and F. Toni. Abductive logic programming. *Journal of Logic and Computation*, 2, 1992.

[KND01] Antonis C. Kakas, Bert Van Nuffelen, and Marc Denecker. A-system: Problem solving through abduction. In *IJCAI*, pages 591–596, 2001.

[Kol65] A.N. Kolmogorov. Three approaches to the quantitative definition of information. *Prob. Inf. Trans.*, 1:1–7, 1965.

[Kow79] R. Kowalski. Algorithm = logic + control. *Communications of the ACM*, 22(7):424–436, 1979.

[Lai08] J. E. Laird. Extending the soar cognitive architecture. *Frontiers in Artificial Intelligence and Applications*, pages 224–235, 2008.

[LCW+11] D. Lin, J. Chen, H. Watanabe, S.H. Muggleton, P. Jain, M. Sternberg, C. Baxter, R. Currie, S. Dunbar, M. Earll, and D. Salazar. Does multi-clause learning help in real-world applications? In *Proceedings of the 21st International Conference on Inductive Logic Programming*, LNAI 7207, pages 221–237, 2011.

[LF01] N. Lavrač and P. Flach. An extended transformation approach to Inductive Logic Programming. *ACM TRANSACTIONS ON COMPUTATIONAL LOGIC*, 2:458–494, 2001.

[Lin09] D. Lin. Efficient, complete and declarative search in inductive logic programming. Master's thesis, Imperial College London, September 2009.

[LMMG73] J. Lighthill, D. Michie, J. McCarthy, and R. Gregory. Lighthill Controversy Debate, 1973. http://media.aiai.ed.ac.uk/Video/Lighthill1973/1973-BBC-Lighthill-Controversy.mov.

[LS00] P. Langley and S. Stromsten. Learning context-free grammars with a simplicity bias. In Ramon Lpez de Mntaras and Enric Plaza, editors, *Machine Learning: ECML 2000*, volume 1810 of *Lecture Notes in Computer Science*, pages 220–228. Springer Berlin / Heidelberg, 2000.

[Lyc] LycoCyc. Solanum lycopersicum database. http://solcyc.solgenomics.net//LYCO/.

[Mal03] D. Malerba. Learning recursive theories in the normal ilp setting. *Fundamenta Informaticae*, 57:39–77, 2003.

[MB88] S.H. Muggleton and W. Buntine. Machine invention of first-order predicates by inverting resolution. In *Proceedings of the 5th International Conference on Machine Learning*, pages 339–352. Kaufmann, 1988.

[MB00] S.H. Muggleton and C.H. Bryant. Theory completion using inverse entailment. In *Proc. of the 10th International Workshop on Inductive Logic Programming (ILP-00)*, pages 130–146, Berlin, 2000. Springer-Verlag.

[MCW+10] S.H. Muggleton, J. Chen, H. Watanabe, S. Dunbar, C. Baxter, R. Currie, J.D. Salazar, J. Taubert, and M.J.E. Sternberg. Variation of background knowledge in an industrial application of ILP. In *Proceedings of the 20th International Conference on Inductive Logic Programming*, pages 158–170, 2010.

[MF92] S.H. Muggleton and C. Feng. Efficient induction of logic programs. In S.H. Muggleton, editor, *Inductive Logic Programming*, pages 281–298. Academic Press, London, 1992.

[Mit77] Tom M. Mitchell. Version spaces: a candidate elimination approach to rule learning. In *Proceedings of the 5th international joint conference on Artificial intelligence - Volume 1*, IJCAI'77, pages 305–310, San Francisco, CA, USA, 1977. Morgan Kaufmann Publishers Inc.

[Mit80] T.M. Mitchell. The need for biases in learning generalizations. Technical Report CBM-TR-117, Department of Computer Science, Rutgers University, 1980.

[Mit82] T.M. Mitchell. Generalisation as search. *Artificial Intelligence*, 18:203–226, 1982.

[Mit97] T. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[ML13] S.H. Muggleton and D. Lin. Meta-interpretive learning of higher-order dyadic datalog: Predicate invention revisited. In *Proceedings of the 23rd International Joint Conference Artificial Intelligence (IJCAI 2013)*, 2013. In Press.

[MLL62] D. Martin, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7):394–397, 1962.

[MLPTN13] S.H. Muggleton, D. Lin, N. Pahlavi, and A. Tamaddoni-Nezhad. Meta-interpretive learning: application to grammatical inference. *Machine Learning*, 2013. Published online.

[MLTN11] S.H. Muggleton, D. Lin, and A. Tamaddoni-Nezhad. MC-Toplog: Complete multi-clause learning guided by a top theory. In *Proceedings of the 21st International Conference on Inductive Logic Programming*, LNAI 7207, pages 238–254, 2011.

[MM97] S. Moyle and S.H. Muggleton. Learning programs in the event calculus. In N. Lavrač and S. Džeroski, editors, *Proceedings of the Seventh Inductive Logic Programming Workshop (ILP97)*, LNAI 1297, pages 205–212, Berlin, 1997. Springer-Verlag.

[Moo56] E.F. Moore. Gedanken-experiments on sequential machines. In C.E. Shannon and J. McCarthy, editors, *Automata Studies*, pages 129–153. Princeton University Press, Princeton, NJ, 1956.

[Mor93] Katharina Morik. Balanced cooperative modeling. *Machine Learning*, 11(2-3):217–235, 1993.

[MP07] S.H. Muggleton and N. Pahlavi. Stochastic logic programs: A tutorial. In L. Getoor and B. Taskar, editors, *Introduction to Statistical Relational Learning*, pages 323–338. MIT Press, 2007.

[MR94] S.H. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19,20:629–679, 1994.

[MR06] B. Milch and S. Russell. First-order probabilistic languages: into the unknown. In Stephen Muggleton Ramon Otero and Alireza Tamaddoni-Nezhad, editors, *Proceedings of the 16th International Conference on Inductive Logic Programming*, volume 4455 of *LNAI*, pages 10–24. SV, 2006.

[MRP+11] S.H. Muggleton, L. De Raedt, D. Poole, I. Bratko, P. Flach, and K. Inoue. ILP turns 20: biography and future challenges. *Machine Learning*, 86(1):3–23, 2011.

[MSTN08] S.H. Muggleton, J. Santos, and A. Tamaddoni-Nezhad. TopLog: ILP using a logic program declarative bias. In *Proceedings of the International Conference on Logic Programming 2008*, LNCS 5366, pages 687–692. Springer-Verlag, 2008.

[MSTN10] S.H. Muggleton, J. Santos, and A. Tamaddoni-Nezhad. ProGolem: a system based on relative minimal generalisation. In *Proceedings of the 19th International Conference on Inductive Logic Programming*, LNCS 5989, pages 131–148. Springer-Verlag, 2010.

[MUB83] T. Mitchell, P. Utgoff, and R. Banerji. Learning by experimentation: Acquiring and refining problem-solving heuristics. In R. Michalski, J. Carbonnel, and T. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*. Tioga, Palo Alto, CA, 1983.

[Mug90] S.H. Muggleton. *Inductive Acquisition of Expert Knowledge*. Addision-Wesley, Wokingham, England, 1990.

[Mug91] S.H. Muggleton. Inductive Logic Programming. *New Generation Computing*, 8(4):295–318, 1991.

[Mug95] S.H. Muggleton. Inverse entailment and Progol. *New Generation Computing*, 13:245–286, 1995.

[Mug96] S.H. Muggleton. Stochastic logic programs. In L. de Raedt, editor, *Advances in Inductive Logic Programming*, pages 254–264. IOS Press, 1996.

[Mug01] S.H. Muggleton. Stochastic logic programs. *Journal of Logic Programming*, 2001. Accepted subject to revision.

[Mug06] S.H. Muggleton. Exceeding human limits. *Nature*, 440(7083):409–410, 2006.

[Mug13] S. H. Muggleton. Alan Turing and the development of Artificial Intelligence. *AI Communications*, 2013. In Press.

[MX11] S.H. Muggleton and C. Xu. Can ILP learn complete and correct game strategies? In *Late-breaking proceedings of ILP*. Imperial College London Press, 2011.

[NCdW97] S-H. Nienhuys-Cheng and R. de Wolf. *Foundations of Inductive Logic Programming*. Springer-Verlag, Berlin, 1997. LNAI 1228.

[NIIR10] Hidetomo Nabeshima, Koji Iwanuma, Katsumi Inoue, and Oliver Ray. Solar: An automated deduction system for consequence finding. *AI Commun.*, 23(2-3):183–203, April 2010.

[NNBC+11] A. Nunes-Nesi, A.L. Bertolo, R.T. Carneiro, W.L Arajo, M.C. Steinhauser, J. Michalska, J. Rohrmann, P. Geigenberger, S.N. Oliver, M. Stitt, F. Carrari, J.K. Rose, and A.R. Fernie. Malate plays a crucial role in starch metabolism, ripening, and soluble solid content of tomato fruit and affects postharvest softening. *Plant Cell*, 23:162–184, 2011.

[NP12] S.H. Muggleton N. Pahlavi. Towards efficient higher-order logic learning in a first-order datalog framework. In *Latest Advances in Inductive Logic Programming*. Imperial College Press, 2012. In Press.

[OGS+99] H Ogata, S Goto, K Sato, W Fujibuchi, H Bono, and M Kanehisa. KEGG: Kyoto Encyclopedia of Genes and Genomes. *Nucl. Acids Res.*, 27(1):29–34, 1999.

[Ote05] R. Otero. Induction of the indirect effects of actions by monotonic methods. In *Proceedings of the Fifteenth International Conference on Inductive Logic Programming (ILP05)*, volume 3625, pages 279–294. Springer, 2005.

[OTP10] J.D. Orth, I. Thiele, and B.? Palsson. What is flux balance analysis? *Nature Biotechnology*, 28:245–248, 2010.

[Pan13] Z. Pan. Meta-interpretive learning on robot strategy. MSc ISO report, Imperial College London, 2013.

[Plo69] G.D. Plotkin. A note on inductive generalisation. In B. Meltzer and D. Michie, editors, *Machine Intelligence 5*, pages 153–163. Edinburgh University Press, Edinburgh, 1969.

[Plo71a] G.D. Plotkin. *Automatic Methods of Inductive Inference*. PhD thesis, Edinburgh University, August 1971.

[Plo71b] G.D. Plotkin. A further note on inductive generalization. In *Machine Intelligence*, volume 6. Edinburgh University Press, 1971.

[Poo87] D. L. Poole. Variables in hypotheses. In *IJCAI-87*, pages 905–908, Los Angeles, CA, 1987. Kaufmann.

[Qui90] J.R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.

[Rae12] L. De Raedt. Declarative modeling for machine learning and data mining. In *Proceedings of the International Conference on Algorithmic Learning Theory*, page 12, 2012.

[Ray09] O. Ray. Nonmonotonic abductive inductive learning. *Journal of Applied Logic*, 7(3):329–340, 2009.

[RBR03] O. Ray, K. Broda, and A. Russo. Hybrid Abductive Inductive Learning: a Generalisation of Progol. In *13th International Conference on Inductive Logic Programming*, volume 2835 of *LNAI*, pages 311–328. Springer Verlag, 2003.

[RBR04] O. Ray, K. Broda, and A. Russo. Generalised kernel sets for inverse entailment. In *20th International Conference on Logic Programming (ICLP2004)*, volume 3132 of *LNAI*, pages 165–179. Springer Verlag, 2004.

[RD97] L. De Raedt and L. Dehaspe. Clausal discovery. *Machine Learning*, 26:99–146, 1997.

[RD06] M Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62:107–136, 2006.

[Rey69] J.C. Reynolds. Transformational systems and the algebraic structure of atomic formulas. In B. Meltzer and D. Michie, editors, *Machine Intelligence 5*, pages 135–151. Edinburgh University Press, Edinburgh, 1969.

[Ris78] J. Rissanen. Modeling by Shortest Data Description. *Automatica*, 14:465–471, 1978.

[RKT07] L. De Raedt, A. Kimmig, and H. Toivonen. Problog: A probabilistic prolog and its applications in link discovery. In R. Lopez de Mantaras and M.M Veloso, editors, *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 804–809, 2007.

[RLD93] L. De Raedt, N. Lavrač, and S. Džeroski. Multiple predicate learning. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, 1993.

[RN10] S.J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, New Jersey, 2010. Third Edition.

[RPP09] H. Resat, L. Petzold, and M.F. Pettigrew. Kinetic modeling of biological systems. *Methods Mol Biol*, 541:311–335, 2009.

[Sak92] Y. Sakakibara. Efficient learning of context-freegrammars from positive structural examples. *Information and Computation*, 97(1):23–60, 1992.

[Sak01] Chiaki Sakama. Learning by answer sets. In Alessandro Provetti and Tran Cao Son, editors, *Answer Set Programming*, 2001.

[Sak05] Chiaki Sakama. Induction from answer sets in nonmonotonic logic programs. *ACM Trans. Comput. Log.*, 6(2):203–231, 2005.

[SB98] R.S. Sutton and A.G. Barto. *Reinforcement learning: An introduction*. Cambridge Univ Press, 1998.

[SB02] I. Salvador and J.M. Benedi. Rna modeling by combining stochastic context-free grammars and n-gram models. *International Journal of Pattern Recognition and Artificial Intelligence*, 16(3):309–316, 2002.

[Sha83] E.Y. Shapiro. *Algorithmic program debugging.* MIT Press, 1983.

[Sil86] B. Silver. *Meta-level inference : representing and learning control information in artificial intelligence.* Studies in computer science and artificial intelligence. Elsevier Science Publishers, Amsterdam, 1986.

[Sol64] R.J. Solomonoff. A formal theory of inductive inference. *Information and Control*, 7:376–388, 1964.

[SS86] L. Sterling and E. Shapiro. *The art of Prolog: advanced programming techniques.* MIT-Press, Cambridge, MA, 1986.

[SSR+06] N. Schauer, Y. Semel, U. Roessner, A. Gur, I. Balbo, F. Carrari, and T. Pleban et al. Comprehensive metabolic profiling and phenotyping of interspecific introgression lines for tomato improvement. *Nat Biotechnol*, 24(4):447–454, 2006.

[Sto74] M. Stone. Cross-validatory choice and assessment of statistical predictions. *J. Royal Statistical Society*, 36:111–133, 1974.

[Sto95] A. Stolcke. An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics*, 21(2):165–201, 1995.

[syn] Syngenta Ltd. http://www.syngenta.com/en/index.html.

[Tar77] S-A Tarnlund. Horn clause computability. *BIT Numerical Mathematics*, 17(2):215–226, 1977.

[TNBRM11] A. Tamaddoni-Nezhad, D. Bohan, A. Raybould, and S.H. Muggleton. Machine learning a probabilistic network of ecological interactions. In *Proceedings of the 21st International Conference on Inductive Logic Programming*, LNAI 7207, pages 332–346, 2011.

[TNCKM06] A. Tamaddoni-Nezhad, R. Chaleil, A. Kakas, and S.H. Muggleton. Application of abductive ILP to learning metabolic network inhibition from temporal data. *Machine Learning*, 64:209–230, 2006. DOI: 10.1007/s10994-006-8988-x.

[Tur50] A. Turing. Computing machinery and intelligence. *Mind*, 59(236):435–460, 1950.

[Utg86] P.E. Utgoff. *Machine Learning of Inductive Bias*. Kluwer, Boston, MA, 1986.

[Val84] L.G. Valiant. A theory of the learnable. *Communications of the ACM*, 27:1134–1142, 1984.

[VB87] K. Vanlehn and W. Ball. A version space approach to learning context-free grammars. *Machine Learning*, 2:39–74, 1987.

[WCC+10] C.L. Waterman, R.A. Currie, L.A. Cottrell, J. Dow, J. Wright, C.J. Waterfield, and J.L. Griffin. An integrated functional genomic study of acute phenobarbital exposure in the rat. *BMC Genomics*, 11(9), 2010.

[Wil88] David C. Wilkins. Knowledge base refinement using apprenticeship learning techniques. In Howard E. Shrobe, Tom M. Mitchell, and Reid G. Smith, editors, *AAAI*, pages 646–653. AAAI Press / The MIT Press, 1988.

[Yam97] A. Yamamoto. Which hypotheses can be found with inverse entailment? In N. Lavrač and S. Džeroski, editors, *Proceedings of the Seventh International Workshop on Inductive Logic Programming*, pages 296–308. Springer-Verlag, Berlin, 1997. LNAI 1297.

[ZSM05] T. Zhang, H. Sipma, and Z. Manna. The decidability of the first-order theory of Knuth-Bendix order. In *Automated Deduction–CADE-20*, pages 738–738. Springer, 2005.